MIS in Light Transport

YASH PATEL

The goal of this project is to implement the techniques we have learned so far to the study of light transport (more information about what precisely this means in the background section below). The focus in this project will be almost entirely towards applications, with the vast majority of time and effort expected to be allocated to understanding how to translate theoretical concepts from class to code in light transport and writing/debugging the corresponding code. The particular element that will be of focus will be multiple importance sampling. Metropolis MCMC techniques is also provided as a brief theoretical presentation, but the implementation only includes MIS. All the code for the project is written in C++ and is available at: https://github.com/yashpatel5400/raytrace-montecarlo
Note: ALL the diagrams and derivations are all from the **fantastic** resource that is "Physically based rendering: From theory to implementation." See the references for the full citation.

1 BACKGROUND

1.1 Motivation

Light transport has been one of the crowning achievements from years of development in the subfield of computer graphics. Computer graphics is specifically the field involved with making computers render 3D environments, with the main use cases being the rendering in CGI in movies, animated movies, and video games. In fact, any company heavily involved within this space will have their own, in-house "renderer" that does precisely this, with the most well known being Pixar's Renderman software. Computer graphics has gotten so incredibly realistic that it is oftentimes **very** hard to even know something that was **ONLY ever** generated on a computer and did not actually come from recording.

Fig. 1. Computer graphics rendering is capable of producing absolutely **stunning** scenes, such as the one generated here. Realize that **NOTHING** in the above picture came from the "real world": this is **ALL** be generated in a computer!

1.2 Ray Tracing Setup

Scenes consist of four components: geometry, materials, lights, and a virtual camera. Scene geometry are the "objects in the scene," which could include any number of things, from tables to animals to imaginary creatures. In implementation, scene elements are defined as combination of geometric primitives, usually taken to be triangles, spheres, prisms, and the like. Naturally, in the real world, objects don't just differ by their structure: they also differ by their materials. Different materials interact with light in different ways: that is, they differ in how they reflect and transmit light. This is discussed in far greater mathematical detail later. In fact, **most** of the complication of sampling in rendering comes from these very interactions between light and object materials.

Another key component of scenes are the sources of light. There are a number of forms of light sources, such as directional spotlights, global sunlight, or point light sources, but these are not of great importance here. The final component of a scene is a virtual camera. This represents the point of view we wish to render the scene from. In other words, we wish to produce a picture corresponding to "how the scene would look if viewed from some point of view of interest." These components are well illustrated in the following figure:

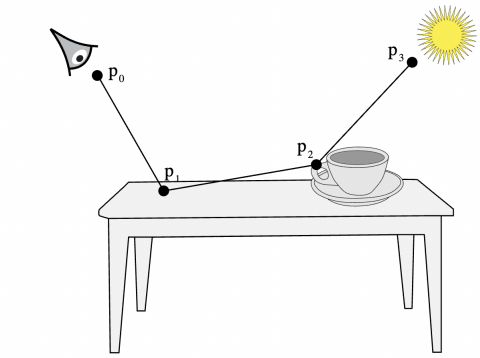


Fig. 2. This is the basic set of elements when rendering: geometry in the scene (the table and mug here), corresponding materials for those objects (not shown), light sources (the sun), and a desired camera POV (the eye)

Of course, now the question becomes **how** do we render this scene from that camera POV, which is where the technical intrigue comes in. In the "real world," light rays originate from some sources (typically the sun or lightbulbs) and bounce around your environment until they finally make it to your eye, at which point your brain interprets the raw signals to produce what you call "vision." A very natural starting point would be to simulate this precise process in your computer, with your brain being swapped out with a computer program. In other words, an initial approach can be imagined as following the rays of light from the light sources as they bounce off of the elements of the scene until they reach the camera position. The main issue with this naive approach is that **almost all** of the rays of light would **never** hit the camera! This makes the naive approach totally computationally infeasible to use.

So instead, we actually turn the situation around and shoot rays **out of** the camera! That is, for each pixel in the final render, we shoot out rays from that position that correspond to the light rays that **would have** ended up hitting that pixel (see https://www.youtube.com/watch?v=frLwRLS_ZR0 for a great visualization of this).

Here is where the main complication comes from: there are **infinitely many** rays that actually hit that particular pixel on the camera (or your eye for that matter)! If you follow the ray from your eye to the first object in that direction (consider the table in the above figure), you'll realize that **that** ray could have come from

infinitely many different initial rays, since light bounces off of that table in a way dictated by the material of that table. With that basic setup, we now dive into the mathematical formalism. There are two pieces of the mathematical formalism:

(1) **Material BRDFs/BSDFs:** This describes the **local** interaction of how a **single** light ray interacts probabilistically with a material. Importance sampling over this distribution is a **very** common strategy
(2) **Global path integration:** This is the **global** formalization of how we combine paths together to produce average results of a single point. Notice that we can do sampling over the space of **paths** directly, which is where Metropolis sampling is typically used

1.3 Material BRDFs

When light hits an object, it actually scatters in all directions off of that surface, with a probability distribution defined by the properties of that surface. These probability distributions are referred to as BRDFs (bidirectional reflectance distribution functions). In general, we are interested in looking at the distribution of outgoing rays of light $\omega_o \in \mathbb{R}^3$ for particular choices of incoming rays of light ω_i, shown in the figure below:

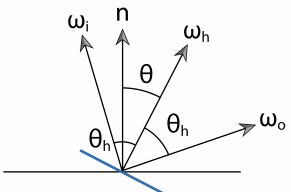


Fig. 3. For a surface, we are interested in looking at the BRDF to describe the distribution over ω_o for fixed ω_i

In general, a BRDF can be expressed as:

$$f_r(\omega_i, \omega_f) = \frac{dL_r(\omega_r)}{dE_r(\omega_r)} = \frac{dL_r(\omega_r)}{L_i(\omega_i)cos(\theta_i)d\omega_i}$$

Where L, E are the radiance and irradiance of a material. The radiance is how much light is **emitted** by a surface per unit surface area and irradiance simply how much is **received** per unit surface area. One key detail to keep in mind is what **precise space** ω lives in. The answer is that $\omega \in S^2$, which is the space of angles that span the unit sphere. ω is measured in **steradians**: this is the extension of the concept of how radians are defined for lengths over the unit circle to surface areas over the unit sphere. Therefore, f_r maps $S^2 \times S^2 \rightarrow \mathbb{R}$.

The question now becomes **how** are such BRDFs obtained? That is, what specific property of materials causes their interactions with light to differ so much? The answer comes from investigating their microscopic structure. While a surface may feel smooth running your hand along it, the surface is littered with what are called **microfacets**, which are simply the microscopic bumps on the material surface. **These microfacets** are what result in the macroscopic interactions such objects have with light. Therefore, BRDF functions are actually produced by modelling these microfacets. This construction of microfacet models is actually itself a rather intricate subfield that we limit consideration of in this study. Using a particular microfacet model known Torrance-Sparrow gives us a BRDF of the following form:

$$f_r(\omega_i, \omega_o) = \frac{D(\omega_h)G(\omega_o, \omega_i)(1 - F_r(\omega_o))}{((\omega_o \cdot \omega_h) + \eta(\omega_i \cdot \omega_h))^2} \frac{|\omega_i \cdot \omega_h||\omega_o \cdot \omega_h|}{cos(\theta_o)cos(\theta_i)}$$

Where $F_r(\omega)$ is the proportion of light that is reflected and D, G are themselves (rather involved) functions of ω. For brevity, I only show the form of $D(\omega_h)$:

$$D(\omega_h) = \frac{e^{-tan(\theta_h)/\alpha^2}}{\pi\alpha^2 cos^4(\theta_h)}$$

Where α is a parameter that characterizes a particular material (meaning brick would have some particular value of α and plastic another). Notice that this is conceptually (and mathematically) where importance sampling greatly improves convergence of the computation of integrals: much of rays that contribute to the final color of an object comes from some subset of the space of incoming rays, meaning it is best to concentrate samples from that region. For examples of how some basic materials differ in BRDF structure, see the below figure

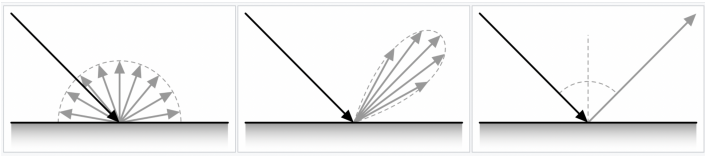


Fig. 4. Here we see the difference between three **very** common material types: the left one is a diffuse material (such as wood), which scatters light equally in all directions; the middle a metallic surface (such as steel or iron), which has a directional preference, but still has some degree of scattering; and the right a mirror that purely reflects light in some direction

1.4 Material BTDFs

Notice we have **totally neglected** a large feature of light in our discussions thus far! Whenever light interacts with any not-opaque material, it is partially **transmitted** through the material, with the light rays bending in accordance to the index of refraction of the material. Therefore, to model **this** phenomenon, we have a **very** similar expression known as the **BTDF** (bidirectional transmittance distribution function). Once again, in particular cases, this function can have very complicated forms, but the general form is as follows:

$$f_t(\omega_i, \omega_o) = \frac{\eta_o^2}{\eta_i^2}(1 - F_r(\omega_i))\frac{\delta(\omega_i - T(\omega_o, n))}{|\cos(\theta_i)|}$$

Where T is a transmission vector and $F_r(\omega)$ is once again the proportion of light reflected, meaning $1 - F_r(\omega_i)$ is that transmitted. The sum interactions of light ray at the surface of an object can be described as:

$$f(\omega_i, \omega_o) = f_r(\omega_i, \omega_o) + f_t(\omega_i, \omega_o)$$

This is often referred to as the **BSDF** (bidirectional scattering distribution function). In probability parlance, this defines a joint probability distribution over $(\omega_i, \omega_o) \in S^2 \times S^2$.

1.5 Light Transport Equation

We now return to the overall question of light transport, which we recall involves capturing the contributions of the infinitely many rays that bounced to hit the point of interest. The key mathematical formalism is to express this problem as an infinite dimensional path integral problem. Since this is of central importance to the problem, we provide a brief derivation of this central equation. Start with a particular ray of light, which we denote by (p, ω), p being the endpoint and ω being the direction. The power transmitted by light in a system must be conserved. Therefore, the outbound light from a particular point must be the sum total of any that surface is itself producing and what it is reflecting. Mathematically (remember L is the radiance or emitted light):

$$L_o(p, \omega_o) = L_e(p, \omega_e) + \int_{S^2} f(p, \omega_i, \omega_o) L_i(p, \omega_i) |cos(\theta_i)| d\omega_i \tag{1}$$

Where S^2 is the surface integral over the point we are considering and $f(r, \omega_i, \omega_o)$ is the distribution of rays coming to said surface (which relates back to the BRDF/BTDF discussion we had before). This formulation may give a **false sense** of simplicity! There is **no** analytic or close to straightforward way of calculating this. This is because we have simply hidden the dependence of the scene geometry in the $L_i(p, \omega_i)$ function, which will become more clear as we expand this expression.

We now rewrite L_i in a way that shows this explicitly and also parallels how the implementation works. Notice that this inbound L_i **must** have come from some other point in the scene, by conservation of energy. That is, there must have been some other point p' whose **outbound** radiance is **precisely** what is measured at $L_i(p)$. In other words, $L_o(p', \omega) = L_i(p, \omega')$. We now introduce a "ray-tracing" function $t(p, \omega)$ that returns the **first** point of scene geometry that is intersected starting from p and tracing the direction ω, shown below:

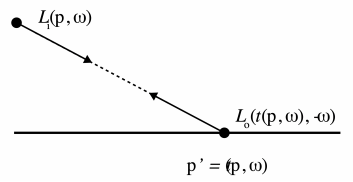


Fig. 5. Formulating the equations with L_i totally obscures the relation to **where** such relevant rays are coming from! By introducing L_o, we make the dependence much more clear and explicit and have the additional benefit of it directly pertaining to the eventual implementation

Therefore, we rewrite this original formulation as:

$$L_o(p, \omega_o) = L_e(p, \omega_e) + \int_{S^2} f(p, \omega_i, \omega_o) L_o(t(p, \omega_i), -\omega_i) |cos(\theta_i)| d\omega_i$$

We now expand this out to obtain the path integral formulation. We introduce a bit of new notation: instead of considering a solid angle ω, we now consider a fixed path $p' \rightarrow p$, which has direct correspondence to the solid angle formulation (simply by following that corresponding solid angle "out" to reach the other point p'). In other words, we can now denote $L_o(p, \omega_i)$ as $L_o(p' \rightarrow p)$. Notice that we can similarly notate the BSDF interaction that initially produced such a p' as $f(p', \omega_i, \omega_o) = f(p'' \rightarrow p' \rightarrow p)$, as depicted in the below diagram:

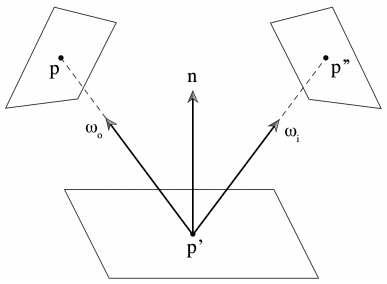


Fig. 6. Here we have an illustration of the origin of the rewrite $f(p', \omega_i, \omega_o) = f(p'' \rightarrow p' \rightarrow p)$

From here, it is common to bundle the visibility of the scene geometry into a function $V(p'', p')$ and the factors related to the change of variable from solid angle to area into a single "geometric" factor $G(p'' \leftrightarrow p')$:

$$G(p'' \leftrightarrow p') = V(p'' \leftrightarrow p')\frac{|cos(\theta)||cos(\theta')|}{||p - p'||^2}$$

With these notations introduced, we are left with:

$$L(p' \rightarrow p) = L_e(p' \rightarrow p) + \int_{\mathcal{A}} f(p'' \rightarrow p' \rightarrow p)L(p'' \leftrightarrow p')G(p'' \leftrightarrow p')dA(p'')$$

Notice that this final term is **recursively** related to the original formulation! So, we can expand this ad infinitum. Specifically, we can see that this can be written as:

$$L(p_1 \rightarrow p) = L_e(p_1 \rightarrow p) + \int_{\mathcal{A}} f(p_2 \rightarrow p_1 \rightarrow p)G(p_2 \leftrightarrow p_1)L(p_2 \leftrightarrow p_1)dA(p_2)$$

$$= L_e(p_1 \rightarrow p) + \int_{\mathcal{A}} f(p_2 \rightarrow p_1 \rightarrow p)G(p_2 \leftrightarrow p_1)\left(L_e(p_2 \rightarrow p_1) + \int_{\mathcal{A}} f(p_3 \rightarrow p_2 \rightarrow p_1)G(p_3 \leftrightarrow p_2)L(p_3 \leftrightarrow p_2)dA(p_3)\right)dA(p_2)$$

$$= L_e(p_1 \rightarrow p) + \int_{\mathcal{A}} L_e(p_2 \rightarrow p_1)f(p_2 \rightarrow p_1 \rightarrow p)G(p_2 \leftrightarrow p_1)dA(p_2) +$$

$$\int_{\mathcal{A}}\int_{\mathcal{A}} f(p_3 \rightarrow p_2 \rightarrow p_1)f(p_2 \rightarrow p_1 \rightarrow p)G(p_3 \leftrightarrow p_2)G(p_2 \leftrightarrow p_1)L(p_3 \leftrightarrow p_2)dA(p_3))dA(p_2)$$

Now clearly, this process **NEVER** stops! That is to say, we have an **infinite** number of path integrals that need to be solved in the actual case of true physical simulation of what light is doing. Another thing to notice is how to **semantically** interpret each of these separate integrals: these correspond to paths of increasing length, with the first being a path consisting of a single bounce, the next with two bounces, and so on ad infinitum. In other words, we can take a sum over all possible path lengths (where the nth term in this sum is **exactly** the nth term in the above expansion):

$$L(p_1 \rightarrow p) = \sum_{n=1}^{\infty} P(\bar{p}_n)$$

Now, referring to the above, we see that the amount a light path of length n actually contributes is dependent on **all** the BSDFs of the material hit along the path. In other words, we can define an overall throughput of said path as:

$$T(\bar{p}_n) = \prod_{i=1}^{n-1} f(p_{i+2} \rightarrow p_{i+1} \rightarrow p_i)G(p_{i+1} \leftrightarrow p_i)$$

Meaning each of these paths can be defined as:

$$P(\bar{p}_n) = \int_{\mathcal{A}}\int_{\mathcal{A}}...\int_{\mathcal{A}} L_e(p_n \rightarrow p_{n-1})T(\bar{p}_n)dA(p_2)...dA(p_{n-1})dA(p_n)$$

This is the final equation we wish to consider in calculating $L(p_1 \rightarrow p)$, which leads us posing the question of how to calculate this seemingly intractable integral. Unsurprisingly, we turn to Monte Carlo for this.

1.6 Monte Carlo Path Tracing

The natural question is **how** can we possibly calculate the behemoth that is:

$$L(p_1 \rightarrow p) = \sum_{n=1}^{\infty} P(\bar{p}_n)$$

It suffices to determine how to estimate $P(\bar{p}_n)$ for any choice n, since then we can just add these up (curtailing the sum at some large, finite value). A very naive way of doing so might be to uniformly at random n pieces of geometry from the scene and construct a path between such things in hopes of getting a Monte Carlo estimate of $P(\bar{p}_n)$. However, this is totally hopeless. First off, most realistic scenes have counts of geometry in the hundreds of **billions** and sometimes **trillions**, making the choosing of light paths at random totally unusable, since almost all light paths will lack end-to-end visibility, which will render the contribution of such a path to 0. The repetitive bouncing of rays also concentrates the angle of the relevant paths, which makes any uniform sampling strategy necessarily high variance. Note that there are **very** apparent visual ramifications of choosing a sampling method with high variance, shown in the below figure:

Fig. 7. An example of a render using a strategy that has high variance is shown on the left, where there are many speckles. In computer graphics parlance, these are referred to as "fireflies." These represent points where the estimator has not yet converged to any meaningful answer and is typically only resolved with a significantly increased computation time

What we do instead is incremental sampling. That is, we construct the path starting by performing a sampling of the first BSDF we encounter, then following that ray out to the next encountered object, sample its BSDF, and so on n times to sample a ray of length n. Remember that the goal is to estimate the integral:

$$P(\bar{p}_n) = \int_{\mathcal{A}} \int_{\mathcal{A}} \dots \int_{\mathcal{A}} L_e(p_n \rightarrow p_{n-1}) T(\bar{p}_n) dA(p_2) \dots dA(p_{n-1}) dA(p_n)$$

Recall that importance sampling weighting simply involves (for a single variable):

$$\int f(x)dx = \int \frac{f(x)}{h(x)} h(x)dx$$

In other words, we need to reweight our sampled light ray by the reciprocal of the sampling probability. First observe that, in order to do sampling in the space of S^2, which is more convenient in practice, it is necessary to reformulate the previous result back into solid angles. Relating the probabilities gives us:

$$p_A(p_i) = p_\omega(p_{i+1} - p_i)\frac{|cos(\theta_i)|}{||p_{i+1} - p_i||^2}$$

This comes from the fact that there is a direct geometric relationship between differential solid angles and the corresponding angles, given by the equation below and corresponding figure:

$$d\omega = \frac{dAcos(\theta)}{r^2}$$

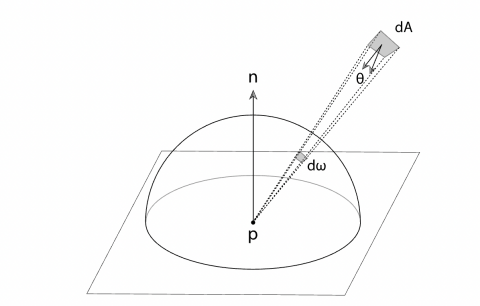


Fig. 8. The solid angle is clearly directly relatable to the differential vector, as given by the standard transformation to spherical coordinates given here

Notice that the “geometric factor" defined has cancellation with this Monte Carlo weighting term:

$$\frac{f(p_{i+2} \to p_{i+1} \to p_i)G(p_{i+1} \leftrightarrow p_i)}{p_\omega(p_{i+2} - p_{i+1})\frac{|cos(\theta_i)|}{||p_{i+1} - p_i||^2}} = \frac{f(p_{i+2} \to p_{i+1} \to p_i)V(p_{i+1} \leftrightarrow p_i)\frac{|cos(\theta_i)||cos(\theta_{i+1})|}{||p_i - p_{i+1}||^2}}{p_\omega(p_{i+2} - p_{i+1})\frac{|cos(\theta_i)|}{||p_{i+1} - p_i||^2}} = \frac{f(p_{i+2} \to p_{i+1} \to p_i)V(p_{i+1} \leftrightarrow p_i)|cos(\theta_{i+1})|}{p_\omega(p_{i+2} - p_{i+1})}$$

We can additionally simplify this by observing that the **only** samples that are considered in the final evaluation are those that have end-to-end visibility, meaning that all such samples **must** have $V(p_{i+1} \leftrightarrow p_i) = 1$. Removing this term results in the final simplified expression for the weighting of **each bounce** in the integrand:

$$\frac{f(p_{i+2} \to p_{i+1} \to p_i)|cos(\theta_{i+1})|}{p_\omega(p_{i+2} - p_{i+1})}$$

Combining this with the actual function of interest, therefore, gives us the **almost** final importance weighted Monte Carlo samples we will collect:

$$L_e(p_n \to p_{n-1})\left(\prod_{i=1}^{n-1}\frac{f(p_{i+2} \to p_{i+1} \to p_i)|cos(\theta_{i+1})|}{p_\omega(p_{i+2} - p_{i+1})}\right) \tag{2}$$

This equation above is **not necessarily** the actual final form is that, in sampling, sometimes we want the final bounce to be sampled over a distribution of the **light sources**. In other words, we take $n - 1$ bounces randomly through the scene (sampled based on the BSDFs) and sample the last over the distribution of light sources, since there is an implicit assumption that this ray of light must have **originated** from some light source in our scene.

That is, we assume the ray did not simply appear out of the ether, meaning the last bounce must connect it to the source. Because of this, the geometry factor in this bounce does **not** cancel out as it does in the others. Denoting the distribution over light sources as $p_A(p_i)$, we get the final expression by pulling out the $n-1$st term from the product and reciprocal weighting it by $p_A(p_i)$ instead:

$$\frac{L_e(p_n \to p_{n-1}) f(p_{n+1} \to p_n \to p_{n-1}) G(p_n \leftrightarrow p_{n-1})}{p_A(p_i)} \left(\prod_{i=1}^{n-2} \frac{f(p_{i+1} \to p_i \to p_{i-1}) |cos(\theta_i)|}{p_\omega(p_{i+1} - p_i)} \right)$$

1.7 Bidirectional Ray Tracing

There is **just** one more piece of background before we can discuss Metropolis Light Transport: **bidirectional** ray tracing. Notice that, in the setup discussed so far, **all** the paths that have no visibility along the final bounce to the light source are discarded in the sampling. This means a scene where the main light source of the scene is occluded from the geometry would be nearly **impossible** to render with our current sampling strategy, since almost all the rays will end up hitting said occlusion, as shown below

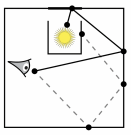


Fig. 9. Case where the light source is occluded from most of rays as they bounce around the scene, which would result in poor convergence rates for naive, unidirectional ray tracing. Bidirectional ray tracing shines in this case

While this seems like a totally contrived example, it actually shows up very often in (animated) movies! Think about where there is a scene that has the characters in a room that is illuminated primarily by a light in an adjacent room. For this reason, we extend the framework of ray tracing as follows. One key observation is that the rendering equation is totally decomposable. That is, instead of viewing all the rays as ending up on the light sources, we can shoot rays **out of** the light source, bounce that around the scene according to the BSDFs, and then have **that** bounced ray be our light source! In other words, we would construct a sequence $p_0, p_1, ..., p_t, q_t, q_{t-1}, ...q_0$, where the $\{p_i\}$ trace **away from the camera** as we have been considering so far and the $\{q_i\}$ trace away from the **light source**! By doing the ray tracing partially from the light sources, we reduce the number of samples that need to be rejected due to occlusions and hence reduce the variance of the estimator.

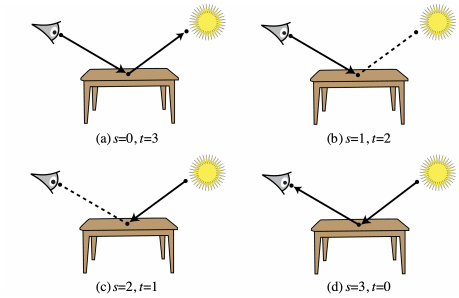


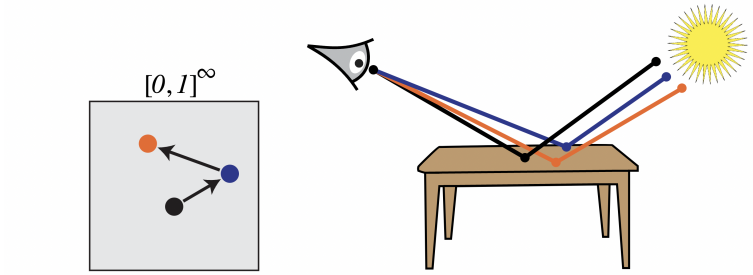
Fig. 10. Here are different ways of tracing the **same** scene. Notice that bidirectional ray tracing imposes **no constraint** on how much of the total path is contributed by the light source trace vs. the camera trace, meaning we can choose any subset of the path length n to be our camera source length.

Notice that the sample weighting follows a very similar scheme to that derived in the previous section for forward rendering. For this reason, we do not derive it explicitly here, although the slight changes (for the initial light ray projecting distribution for instance) need to be accounted for in the weighting.

1.8 Metropolis Light Transport

This leads us to what has become one of the crowning jewels in computer graphics: Metropolis Light Transport. There are many incarnations of MLT, some of which are far too complex to consider in this setting. We only consider the simplest of these: Primary Sample Space MLT (PSSMLT). To motivate the formulation of this technique, suppose we have the difficult-to-render scene discussed above, where the primary light source for the scene is largely occluded. In that case, suppose we are using the uncorrelated sampling method proposed in a bidirectional ray tracer. In this case, if we happen to get lucky and **finally** do find a ray through sampling, the very next ray is once again randomly sampled! In other words, we are starting to search for rays as if we have no knowledge of what a successful light path looks like. This is **extremely** computationally wasteful! PSSMLT is a step in the direction of alleviating that issue. As a subpoint, notice that PSSMLT is **not** a general purpose replacement for bidirectional tracing! It **only** has advantages in scenes where the light source is largely occluded or coming from a narrow sampling band.

As brief preface to posing the MLT formulation, we must reformulate the sampling procedure. Realize that, by default, sampling is happening over the space of paths. However, this space is totally non-Euclidean: we can only sample 2D subsets of $\mathbb{R}^3$ that correspond to object surfaces, which clearly does not resemble a Euclidean space. We need to figure out a way to convert this to a Euclidean domain to proceed through the sampling. We can rethink the sampling path $p_0 \rightarrow p_1 \rightarrow ... \rightarrow p_n ... (\rightarrow \infty)$ as actually a draw from an infinite distribution $X_1, X_2, ..., X_n, ...$, where each $X_i \in [0, 1]$. One key implementation detail that makes this possible is that material surfaces are **always** parameterized as normalized coordinates system $[0, 1]^2$ (referred to as u, v coordinates in computer graphics parlance). This means that a path can be modelled as a sampling from $[0, 1]^\infty$, as shown below:



(a) Primary sample space view (b) Path space view

Fig. 11. There is a direct equivalence between the Euclidean domain of the infinite dimensional hypercube and the non-Eucliean path domain, with the former lending itself well to MLT.

The idea with MLT, therefore, is to sample this space as we would typically with bidirectional ray tracing initially. However, for subsequent samples, we have two “mutations" that we can introduce. The first is a “large mutation" that is used to escape islands in probability space. This consists of totally replacing the current components of $\{X_i\}$ and resampling from scratch using bidirectional ray tracing. In other words, “large mutations" are no

different than a standard sampling run for a bidirectional ray tracer. The other is just to make a “small mutation," which involves moving the $\{X_i\}$ incrementally in the hypercube space, thereby allowing a local exploration of the ray space.

Therefore, the final version of PSSMLT consists of a bidirectional ray tracer (itself using MIS for sampling bounces through the BSDFs in the scene) and doing incremental adjustments on those rays in the hypercube sampling space to occasionally allow correlated samples for local light exploration.

2 IMPLEMENTATION

With that background in place, we turn to what was actually implemented for the project. As mentioned in the abstract, all the code for the project was written in C++. As with most examples in engineering, there is often a disparity between the mathematical formulation in theory and that employed in practice, with the transformation between the two oftentimes being totally non-obvious. Such is certainly the case here, and we therefore present the concrete steps that were taken to implement this project along with code snippets. The results from this development are presented in the next section.

2.1 Ray Generation

As mentioned in the background section, the generation of rays from the camera is the basic operation necessary to perform ray tracing, with the other basical components being intersection logic for fundamental geometry and scattering for the corresponding object material. Generating rays involves projecting out from a virtual camera. All cameras, in their simplest forms, can be imagined as being pinhole cameras, which have an infinitely small hole through which light passes and an image plane onto which such rays accumulate. In real cameras, such an image plane is precisely determined by where the CCD sensor is located. Of course, in real cameras, there are a number of other components, most notably the lens, which itself adds distortion and a number of other geometric complications. Luckily for us, when doing rendering in software, we can simply imagine the idealized case and totally ignore the realistic necessity of camera lenses.

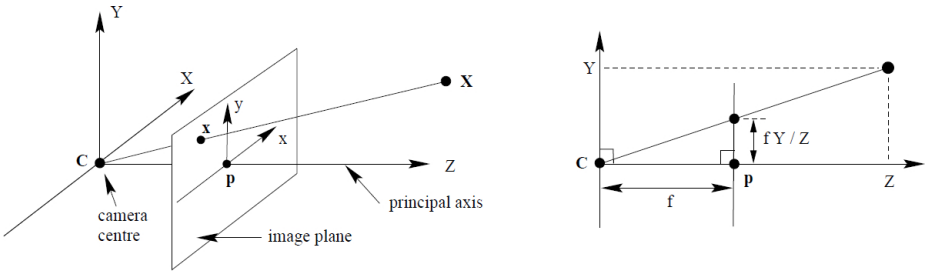


Fig. 12. The pinhole camera is an idealization of how cameras project 3D geometry from the real world to the plane of the camera onto which pictures are “taken." This, however, serves as the basis for virtualized cameras.

In the case of virtual cameras, we have to take this and do the opposite, meaning we generate rays by doing precisely the opposite of the projection of these incoming light rays. Simply looking at the above figure, we see there are two similar triangles that define the projection onto a cameras as:

$$x' = \frac{fx}{z}, y' = \frac{fy}{z}$$

Where f is the focal length, (x, y, z) is the 3D position of a point, and (x', y') is the 2D image projection. Here, we have to construct the **ray** starting from a position (x', y'). For technical reasons dealing with camera geometry,

the pixels on the image do not precisely line up with those on the CCD, so rays are generated as a ratio of these. For an image that is $Im_w \times Im_h$, CCD of size $CCD_w \times CCD_h$, and each pixel (i, j):

$$\overrightarrow{r} = ray(0, (\frac{CCD_w}{Im_w} i, \frac{CCD_h}{Im_h} j, f))$$

Where we are using the notation that will be employed for the remainder of this discussion that $\overrightarrow{r} = (o, d), o \in \mathbb{R}^3, d \in \mathbb{R}^3, ||d|| = 1$, where o is the ray origin and d is the ray direction. From here, we can project into the scene and check for geometry intersection, from which all the sampling that is of interest arises.

2.2 Scene Geometry

There are two types of geometry included in this ray tracer: spheres and planes. Keep in mind there are two stages in ray tracing (that, as mentioned previously, happen theoretically for path lengths of indefinite size): ray intersection with scene geometry and scattering. The former is **only** dependent on **where** things are located in the scene whereas the latter depends also on the material of said objects, which we present in the following section.

2.2.1 Geometry: Sphere. Spheres are defined fully by a center and radius: $(\mathbf{c}, r)$, with $c \in \mathbb{R}^3$. For checking the intersection of a generic ray $\overrightarrow{r} = (o, d)$, simply notice we wish to have $o + td$ satisfy $||c - (o + td)||^2 = r^2$. This is simply a quadratic:

$$t^2 d \cdot d + 2td \cdot (o - c) + (o - c) \cdot (o - c) - r^2 = 0$$

Meaning the corresponding discriminant is:

$$(d \cdot (o - c))^2 - 4(d \cdot d)((o - c) \cdot (o - c) - r^2)$$

The nature of the discriminant determines **whether** there is an intersection with the sphere, with a single and multiple leading being values ≥ 0. We are **always** interested in finding the **first** intersection of the ray with a piece of geometry in the scene, so we take the smallest $t \geq 0$ that satisfies the equation.

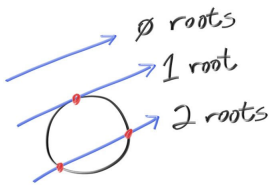


Fig. 13. The discriminant of the sphere solution determines the intersection of a ray with a sphere, from which we have the point that will serve for scattering.

2.2.2 Geometry: Planes. Intersection with planes is even simpler than with spheres. For an axis aligned plane, simply take the axis i that is fixed with a corresponding value c and simply do $t = (c - o_i)/d_i$, from which $o + t * d$ clearly gives the intersection point. For finite planes, simply check whether such a point lies within the bounds. For generic planes, notice that any plane can be imagined as a plane with a rotation about an axis. Further notice that, instead of adding the complication of doing intersection logic with the rotated plane, we can simply rotate the rays in the opposite direction! We can then take the typical intersection logic with this inverse rotated ray, namely:

$$x' = x\cos(\theta) - y\sin(\theta), y' = x\sin(\theta) + y\cos(\theta)$$

2.3 Material BRDFs

For this ray tracer, three material types were implemented: Lambertian (diffuse), metallic, and dieletric (glass). Recall that the basic mental image for these materials is illustrated in Figure 4. Here, we present how the scattering for each one was implemented. Notice that, for any surface, the sampling can be formulated as being a sampling over the unit sphere aligned with the normal at the point of intersection:

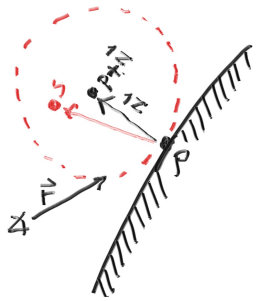


Fig. 14. Scattering in general involves sampling a unit sphere aligned tangent to the point of intersection, with a normal aligned with the sphere radius

2.3.1 BRDF: Lambertian. Almost any surface can be modelled as a combination of an idealized Lambertian (diffuse) surface material combined with some metallic (glossy) component. As a result, Lambertian surfaces are of great importance to understand. Notice a diffuse surface can be mathematically formulated as a surface that has no preferential direction for scattering from the surface, meaning scattering is done uniformly at random across all possible outbound directions. Notice that sampling from a unit sphere can be done naively simply by sampling uniformly in $[0, 1]^3$ and only retaining those points that end up inside the sphere. The more complicated derivations for sampling explicit PDFs is derived later in the paper when we introduce MIS and its role in ray tracing.

2.3.2 BRDF: Metallic. In the case of metallic surfaces, **most** of the light is scattered in a particular direction, but there is some degree of scattering about that preferred direction. Clearly, this can be imagined as a perfect reflection with some amount of noise. Reflection for a direction d about a normal n is simply $d + 2n(n \cdot d)$, as illustrated in this diagram:

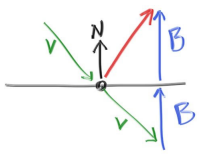


Fig. 15. Reflection about the normal of a surface is simply $d + 2n(n \cdot d)$, where n is the normal of the surface

The additional “noise" term can be obtained simply by adding in the simple spherical sampling described in the previous section with some weighting term α, as illustrated here:

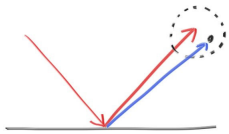


Fig. 16. Scattering off of metallic surfaces is pure reflection combined with uniform spherical scattering

2.3.3 BRDF: Dielectric. The case of dielectric modelling is a fair bit more involved than the previous two parts. Unlike metallic or Lambertian surfaces, glass objects have the additional property that light can travel **through** them. This is an additional layer of complication beyond the standard reflection properties described in the previous two sections. For this reason, we have to take an aside to discuss how the refraction phenomena described in the background actually come into play in practice. From physics, we know about refraction:

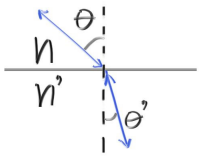


Fig. 17. Refraction is the main secondary phenomenon seen in glass that gives its characteristic look in the presence of light

The equations dictating the bending of light from refraction are:

$$\sin(\theta') = \eta/\eta' \sin(\theta)$$

Where the material that corresponds to the "inside" vs. "outside" depends on whether the light is travelling from outside the object to inside it or through the object to the outside. For air and glass, the $\eta = 1, \eta' = 1.5$ respectively if travelling from air to glass.

Finally, while glass does have this refraction phenomenon, it also has reflection, meaning the final interaction is a linear combination of these two effects. The **exact** ratio is an extremely complicated mess, so most people use an approximation known as "Schlick's approximation" in practice. This is:

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos(\theta))^5, R_0 = \left(\frac{\eta_1 - \eta_2}{\eta_1 + \eta_2}\right)^2$$

Where θ is the angle between n and d and $R(\theta)$ the percent of incident light that gets reflected.

2.4 Basic Integration

As we segue to the discussion about importance sampling, recall that the goal is to estimate the intractable rendering equation:

$$L_o(p, \omega_o) = L_e(p, \omega_e) + \int_{S^2} f(p, \omega_i, \omega_o) L_i(p, \omega_i) |cos(\theta_i)| d\omega_i$$

Reframing this in terms of concrete variables, we can see that this is actually:

$$color = \int s(dir) A(dir) color(dir) d(dir)$$

Where we are looking at the color of a particular point in space by integrating all the directions light could have come from to hit that point and adding the “flux" of color coming from all directions on the local hemisphere of light weighted by their corresponding probabilities. Note that intractability of this integral still exists despite the simplified notation, once again because the dependence on scene geometry is buried in $color(dir)$.

Nonetheless, we can use the incremental path building as described in Eqn 2 from the background coupled with the concrete methods presented in the last couple sections to get an estimate. Notice that, in calculating this integral, we can turn to importance sampling to estimate it. In particular, we are free to choose the sampling distribution of X, so long as we correspondingly normalize out the value of the integrand by the sampling distribution PDF. That is, we simply wish to find $\mathbb{E}_X\left[\frac{sAc}{p}\right]$, where $X \sim P$ is a distribution over the space of directions, since:

$$\mathbb{E}_X\left[\frac{sAc}{p}\right] = \int p(dir)\frac{s(dir)A(dir)c(dir)}{p(dir)}d(dir) = \int s(dir)A(dir)c(dir)d(dir)$$

However, notice that if we simply take $X \sim S$, we simply get $\mathbb{E}_X\left[Ac\right]$! In other words, if we directly sample with the scattering distribution, we don’t even need to know the value of the PDFs! This makes it a very natural choice as a base version of a ray tracer, where the algorithm is simply as follows: for each pixel, generate N rays, each traced through the scene following the corresponding material scattering distributions with the color of the ray being an incremental sum of the objects on the path if the path terminates in a light source, and average these N values to give the final color for that pixel. Light sources do not scatter light and are seen as sinks for the projected rays (for forward tracing). Note that this means, in the simplest case without importance sampling, we are sampling all directions equally often, whether they contribute significantly to the final color of the pixel or not! This is extremely wasteful, largely because much of the base color can be computed by sampling the direction of the light more often than other directions, with those other directions filling in nuances in the coloring of the scene.

2.5 Light Importance Sampling

At last, we get to the concrete realization of the importance sampling that is used in ray tracers. As alluded to in the previous section, it is very inefficient to use the vanilla version of ray tracing. Even though it is guaranteed to converge in theory, the number of rays required to eliminate blurriness greatly exceeds that it might were we to sample light rays in a more intelligent fashion. For importance sampling, we specifically focus on the Lambertian material case, where the scattering (and hence, sampling without MIS) is taken uniformly at random. The issue with this is that, for such directions, $A(dir)c(dir)$ may be quite small or may in fact be 0 if the particular path we attempt to trace turns out to not terminate in a light source! The idea, therefore, is to instead have an biased possibility for sampling directly from the light source.

Notice, however, we cannot simply change the sampling without explicitly computing the PDF of this sampling distribution in $\mathbb{E}_X\left[\frac{sAc}{p}\right]$! Recall that we can get away with ignoring the explicit calculation of s, p in the base case only because $s = p$, which is not true in general! We must now derive both the sampling **and** explicit PDF for this reason for each of the special regions in the scene we wish to sample from explicitly.

The imposed sampling PDF is **totally** dependent on the geometry on the scene, so we must go through the details of the contents of the scene now. In our scene, which there are pictures of in the following (Results) section, there is a single rectangular light, a glass ball, and a Lambertian grey box all within a larger box (feel free to peek ahead to the results section to actually see what this looks like!). In the computer graphics community, this is well-known as the “Cornell Box."

For this scene, there are two directions we wish to bias towards in sampling: the light and the glass ball. The bias towards the light is for the obvious reasons mentioned, where it ensures the path contributes to the final

sum and also gives greater weight than those paths that undergo several scattering events before making their ways to the light. The glass ball is for more nuanced reasons: glass introduces a number of rather intricate light bending patterns known as caustics, which are difficult to reproduce in software by default. For this reason, preferentially firing rays towards the glass object gives greater opportunity for such patterns to emerge in the simulation. Therefore, we have three PDFs and sampling techniques to develop: random directions, the light source, and the glass ball. Notice that, if we do sampling from the light source with probability α, from the glass ball with probability β, and randomly with $1 - \alpha - \beta$, the overall PDF can be described as:

$$\mathbb{P}(dir) = \alpha\mathbb{P}_{light}(dir) + \beta\mathbb{P}_{ball}(dir) + (1 - \alpha - \beta)\mathbb{P}_{random}(dir) \tag{3}$$

This framework can be extended indefinitely, meaning it can be similarly defined for any choice of constructed scene with bias placed as so desired by the engineer.

2.5.1 MIS: Random Directions. While we **did** previously define the distribution of scattering randomly by that on the sphere, this formulation does not lend itself well to making corresponding derivations of the PDFs for the light and glass ball. Therefore, rather than using the uniform spherical scattering that was previously introduced, we switch to a **hemispheric** scattering for Lambertian surfaces now, since it makes all the geometry for the derivations of the other PDFs much more tractable. One very interesting subpoint is that this previously used uniform sampling on the sphere (starting tangent to the sphere) is equivalent to **cosine** weighted sampling on the hemisphere (starting from the center of the sphere corresponding to the hemisphere)! For this reason, we derive the corresponding sampling procedure for the cosine-weighted directions PDF

We derive this explicitly due to its importance for our use case. Note that we wish to have $\mathbb{P}[dir] = cos(\theta)/\pi$. The issue is simply in taking this desired distribution over the directions and finding a correspondence to variables that we can directly sample. Notice that, if we consider this scattering direction d in spherical coordinates with $\rho = 1$ for a unit sphere, we can explicitly construct formulas for angles that can be directly sampled:

$$f_1(\phi) = \frac{1}{2\pi}, f_2(\theta) = 2\pi\frac{\cos(\theta)}{\pi}\sin(\theta) = \sin(2\theta)$$

Notice that we can obtain both of these uniformly at random by doing the standard "inverse CDF" transformation. That is, for uniform variables $u_1, u_2 \sim Unif[0, 1]$, we have:

$$F_1(\phi) = \frac{\phi}{2\pi}, F_2(\theta) = \frac{1}{2}(1 - \cos(2\theta)) = 1 - cos^2(\theta)$$
$$\implies F_1^{-1}(\phi) = u_1 = \frac{\phi}{2\pi} \implies \phi = 2\pi u_1$$
$$F_2^{-1}(\theta) = u_2 = 1 - \cos^2(\theta) \implies \cos(\theta) = \sqrt{1 - u_2}$$

Where we simply leave $\cos(\theta)$ as is, since all expressions in the following derivation use $\cos(\theta)$ instead of directly using θ. Putting this back into rectalinear coordinates, we have:

$$d_x = \cos(\phi)\sin(\theta) = \cos(2\pi u_1)2\sqrt{u_2}, d_y = \sin(\phi)\cos(\theta) = \sin(2\pi u_1)2\sqrt{u_2}, d_z = \cos(\theta) = \sqrt{1 - u_2}$$

This gives us how to do **sampling** for random directions. Notice that, since these assume an axis aligned hemisphere, for the actual sampling in the code, we have to do a local change of basis to align with the normal vector at the point of intersection. In this transformed coordinate system we do the above sampling to get the direction of the scattered ray.

On the flip side, to get the PDF of a **given** ray direction, we can simply use the fact that $\mathbb{P}[dir] = cos(\theta)/\pi$. For a normalized direction d and normal n, $cos(\theta) = d \cdot n$, meaning $\mathbb{P}[dir] = d \cdot n/\pi$.

2.5.2 MIS: Light. The derivation for sampling from the light follows a very similar derivation, where there are two components: sampling and a PDF calculation. The light in the scene is a (finite) axis aligned plane, which simplifies the derivations that follow. As a result, unlike random hemispheric scattering, sampling from the light is **super** straightforward! Simply choose a point p uniformly at random on the surface of the axis and draw a vector $p - o$ from the current intersection point to get that scattering direction. The PDF calculation is only slightly more involved than the simple hemispheric scattering case. Notice that we wish to consider the PDF of sampling from a small piece of area from the light source, as illustrated below:

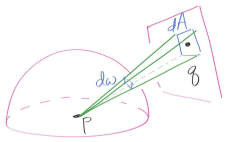


Fig. 18. The goal is to formulate an analytic expression for the PDF of sampling directions with respect to the light source, which has to do with the projections of small sampled pieces of areas onto the local hemispheres considered in the random scattering case previously.

From this above diagram, we see that:

$$d\omega = \frac{dA\cos(\theta)}{d(o,q)^2}$$

Where θ is once again the angle between the normal and this sampled direction, which happens to also be the vector connecting o, q. $d(o,q)$ is simply the Euclidean distance between the two. We wish to have $\mathbb{P}[d\omega] = \mathbb{P}[dA]$, that is to have the same density for sampling a patch of the hemisphere as the corresponding patch of the light that projects onto it. The corresponding probabilities are directly related to the sampling we are explicitly doing: $\mathbb{P}[d\omega] = \mathbb{P}[dir]d\omega$ and $\mathbb{P}[dA] = \mathbb{P}[q]dA$, where q is the connecting point we directly sample on the light. Therefore, we wish to have:

$$\mathbb{P}[d\omega] = \mathbb{P}[dA] \implies \mathbb{P}[dir]d\omega = \mathbb{P}[dir]\frac{dA\cos(\theta)}{d(o,q)^2} = \mathbb{P}[q]dA$$
$$\implies \mathbb{P}[dir] = \frac{d(o,q)^2}{A\cos(\theta)}$$

This gives the sampling PDF for any given ray of light, as desired. Therefore, simply by computing the distance from the scattering point to a point on the light source and knowing the area of the light source, we can find the PDF of a particular direction! Note that there is a subtlety here: if a ray of light **never** intersects the source of light, the PDF for that direction is 0.

2.5.3 MIS: Glass Ball. The final part of the derivation is getting sampling from the glass ball in the scene. Observe that sampling from a ball is equivalent to sampling a direction uniformly from the enveloping cone:

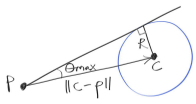


Fig. 19. To sample from the glass ball, we wish to sample and compute the PDF of choosing a ray in enveloping cone of the sphere. This means, rather than weighting directions uniformly in the range possible in the hemisphere, we do so in the supported range by the enveloping cone.

This means the derivation provided in the random hemispheric directions extends naturally here, namely:

$$u_1 = \frac{\phi}{2\pi}$$
$$u_2 = \int 2\pi f(t)\sin(t)dt = 2\pi C(1-\cos(\theta))$$
$$\implies \cos(\theta) = 1 - \frac{u_2}{2\pi C}$$

Where $f(t)$ is a yet unknown weighting for sampling, which was previously the cosine weighting density. Since we know that $u_1 = 2$ should correspond to the extreme value for theta, we know that it should correspond to $\cos(\theta_{max})$ from the above diagram. This means we have:

$$\cos(\theta) = 1 + u_2(\cos(\theta_{max}) - 1)$$

Which gives us the sampling as before:

$$d_x = \cos(\phi)\sin(\theta) = \cos(2\pi u_1)2\sqrt{1-z^2}, d_y = \sin(\phi)\cos(\theta) = \sin(2\pi u_1)2\sqrt{1-z^2}, d_z = \cos(\theta) = 1+u_2(\cos(\theta_{max})-1)$$

For an explicit value of $\cos(\theta_{max})$, notice that we have a triangle in the previous diagram, which gives:

$$\cos(\theta_{max}) = \sqrt{1 - \frac{R^2}{||c-p||^2}}$$

The **final** part of this derivation is finding the corresponding PDF of this sampling distribution. Realize that the PDF is uniform across the angles that are in the support. Therefore, we need to compute the total solid angle that is spanned by this cone, where we find:

$$solidAngle = \int_0^{2\pi}\int_0^{\theta_{max}} \sin(\theta)d\theta d\phi = 2\pi(1-\cos(\theta_{max}))$$

This means the PDF is simply:

$$\frac{1}{2\pi(1-\cos(\theta_{max}))}$$

Once again, as with the light discussion in the previous section, if the ray being considered does **not** intersect the sphere, then the corresponding PDF is simply 0.

2.6 Final Summary

Therefore, the final form of the algorithm looks as follows: for each pixel, shoot out N rays, each of perhaps differing numbers of scattering events, with each scatter following a sampling that is split between random hemispheric scattering, being directed towards the light, and being directed towards the glass ball, weight each sample by the corresponding sampling PDFs, and average them to get the final results. These results are shown in the following section.

3 RESULTS

Here, we present the results of running the ray tracer, with and without importance sampling for the Cornell Box. An initial scene of random balls was generated simply to test the basic functionality, but the crux of the results are the Cornell Box:

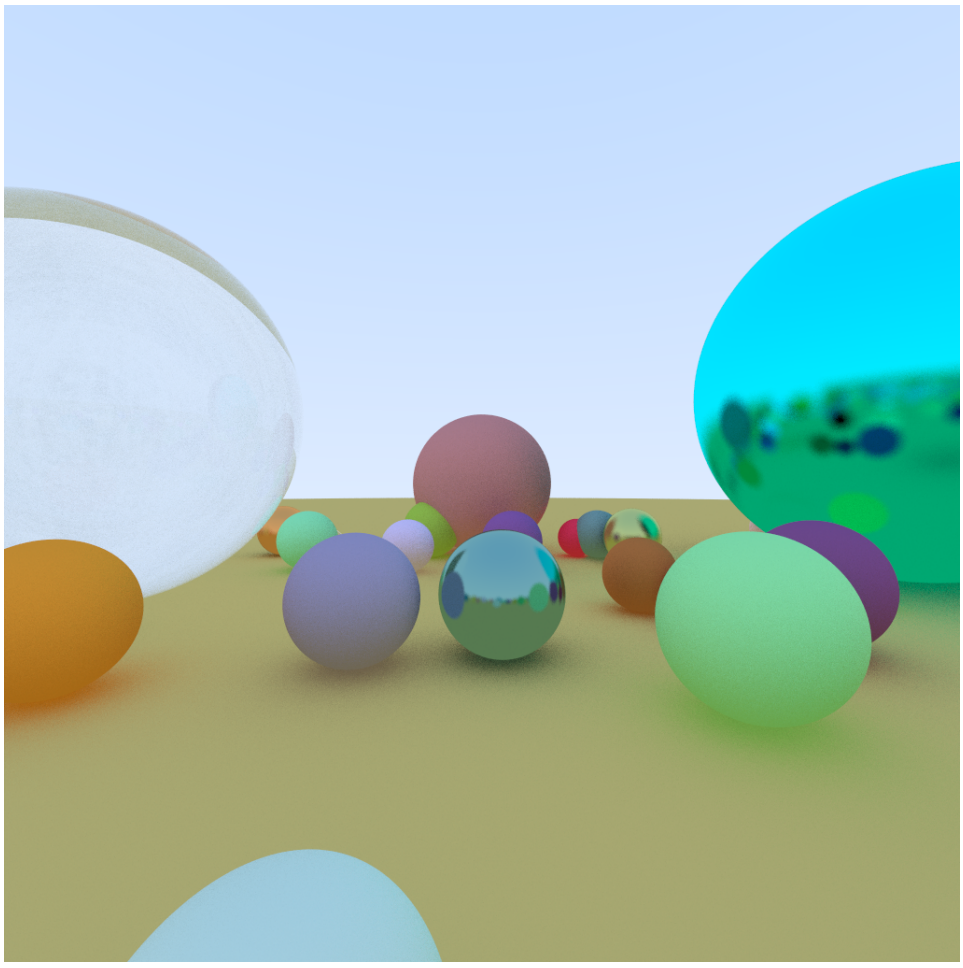


Fig. 20. This is a very basic, single globally lit scene rendered with **my** ray tracer! This was simply used as a baseline to ensure core functionality of the renderer worked

Recall that we have a couple knobs we can tune for determining how we wish to do sampling in the scene, specifically for choosing **where** in the scene to sample from. In particular, remember we have:

$$\alpha \mathbb{P}_{light}(dir) + \beta \mathbb{P}_{ball}(dir) + (1 - \alpha - \beta)\mathbb{P}_{random}(dir)$$

In the case of $\alpha = \beta = 0$, we get the original case of no-MIS, and for $\alpha = 1, \beta = 0$, we have **only** sampling from the light source. We demonstrate how the behaviors improve by doing MIS, and how specifically that manifests in the results below.

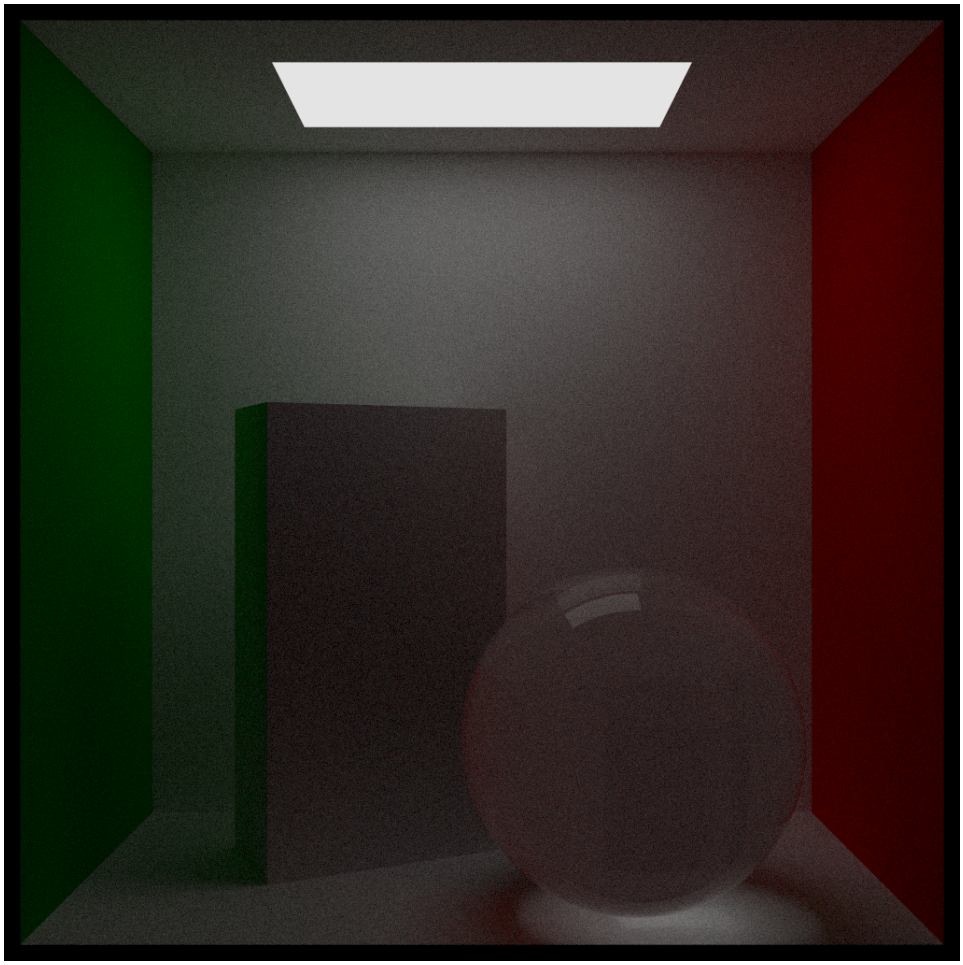


Fig. 21. This is the scene rendered **without** MIS ($\alpha = \beta = 0$). Notice that it captures a great deal of the desired details in the lighting of the scene, particularly in the reflection of the walls onto the respective pieces of geometry in the scene, both visible in the green left side of the prism and the general green “halo" in the back left corner. This is similarly visible in the red highlights on the left side of the glass ball. On the other hand, clearly the picture is quite grainy, even having rendered with 1000 samples per pixel, which takes roughly two hours to render on a Macbook Air. This graininess is what we sought to fix with MIS, in the same number of samples per pixel.

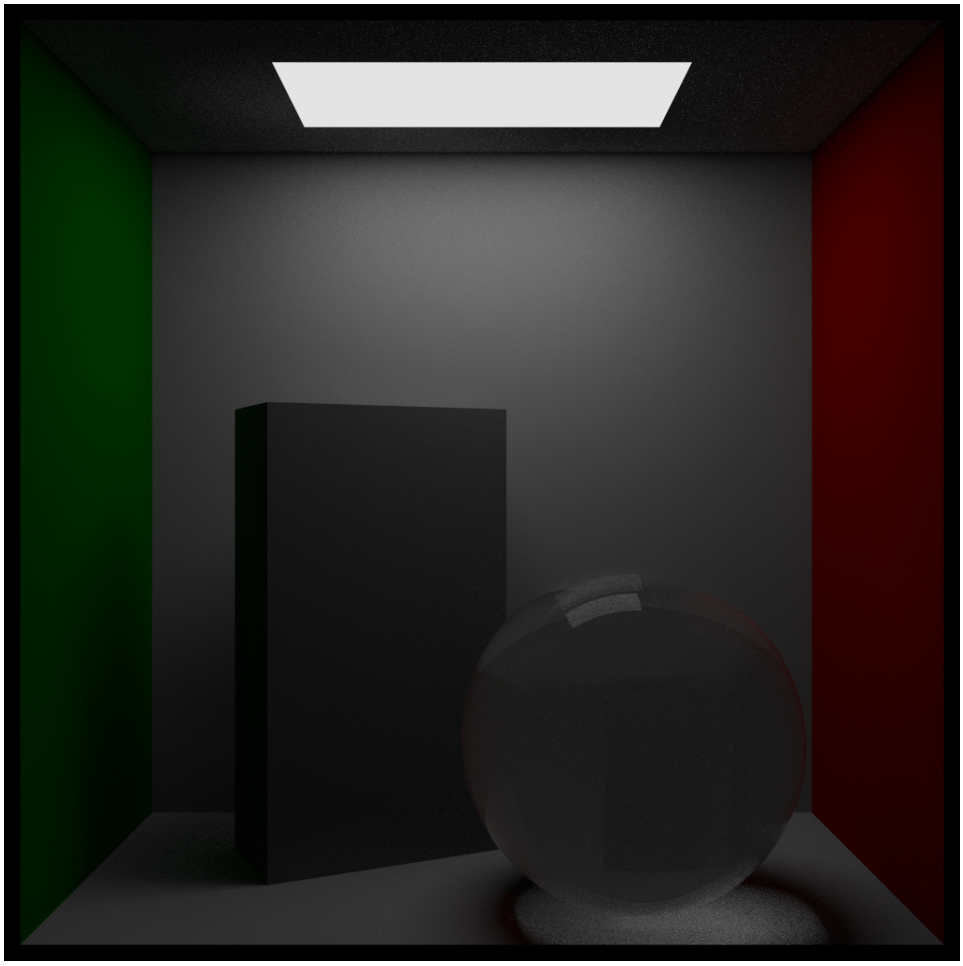


Fig. 22. Here, we **only** sample from the light source to contrast the result with the previous results. Notice that, since there is no random scattering, two large differences emerge: the image is far crisper but all the nuances of scattering are no longer in the picture. The latter point is most noticeable in the lack of green highlighting on the left side of the box and the disappearance of the "halo" in that area. Similarly, there is no red reflection through the glass ball. To alleviate this total loss of fidelity in reconstruction, we wish to do the full MIS, as shown in the following result. This result was also obtained using 1000 samples per pixel.

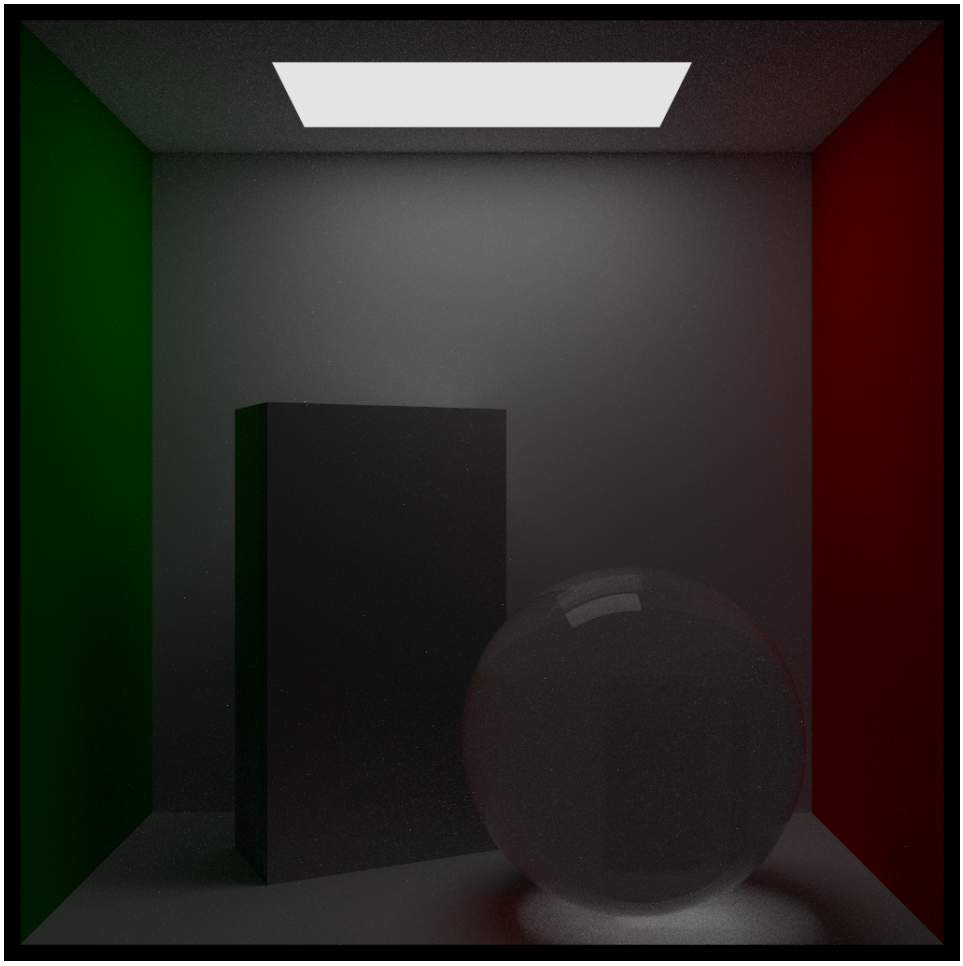


Fig. 23. Here, we use $\alpha = 0.4, \beta = 0.3$, meaning we wish to mix both the vanilla flat rendering in the pure light source sampling case with the nuanced version in the hemispheric scattering case. Here, we in fact do get both desired features, in that we have a crisp final render, while also retaining the green halo effect near the box and the red reflectance through the glass ball. This, therefore, demonstrates how MIS can be used to greatly accelerate convergence of the render of a scene, where we used 1000 samples per pixel once again here.

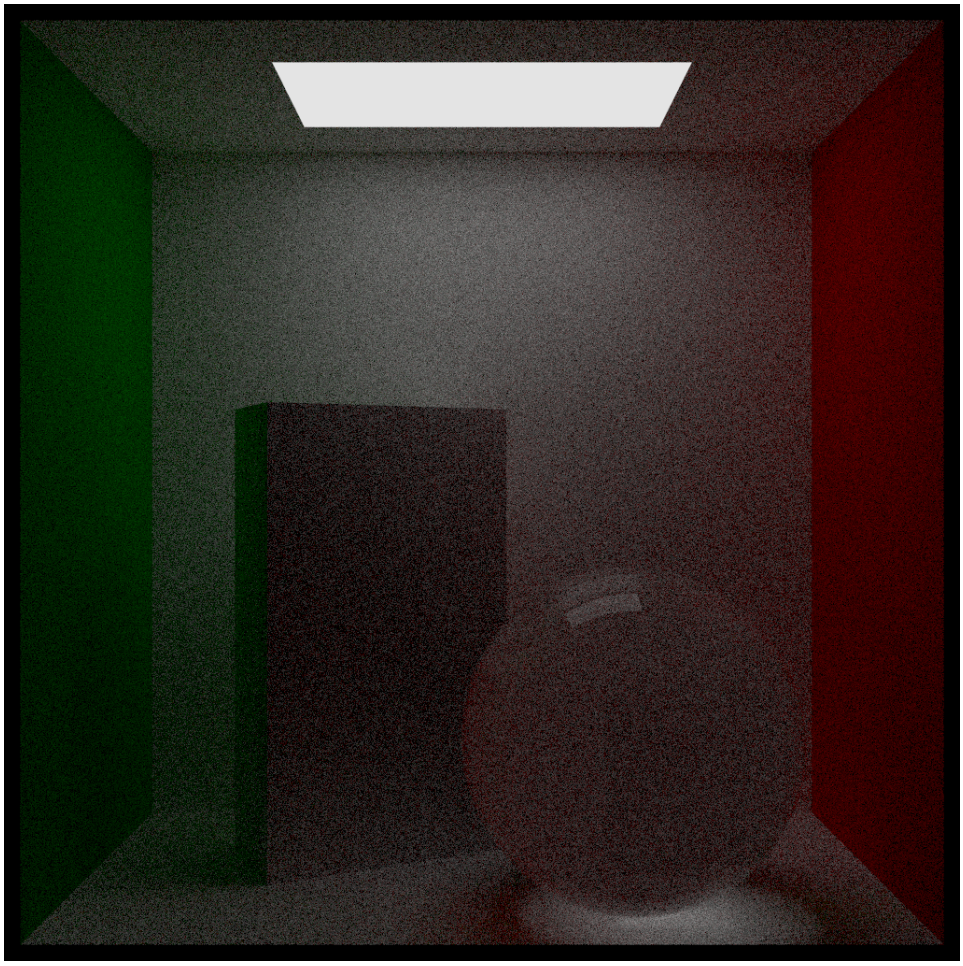


Fig. 24. We can re-run the above experiments with lower samples per pixel to highlight the rate of convergence even more. Here, we are doing **no MIS** with 50 samples per pixel and see a **huge** amount of grain in the rendered image.

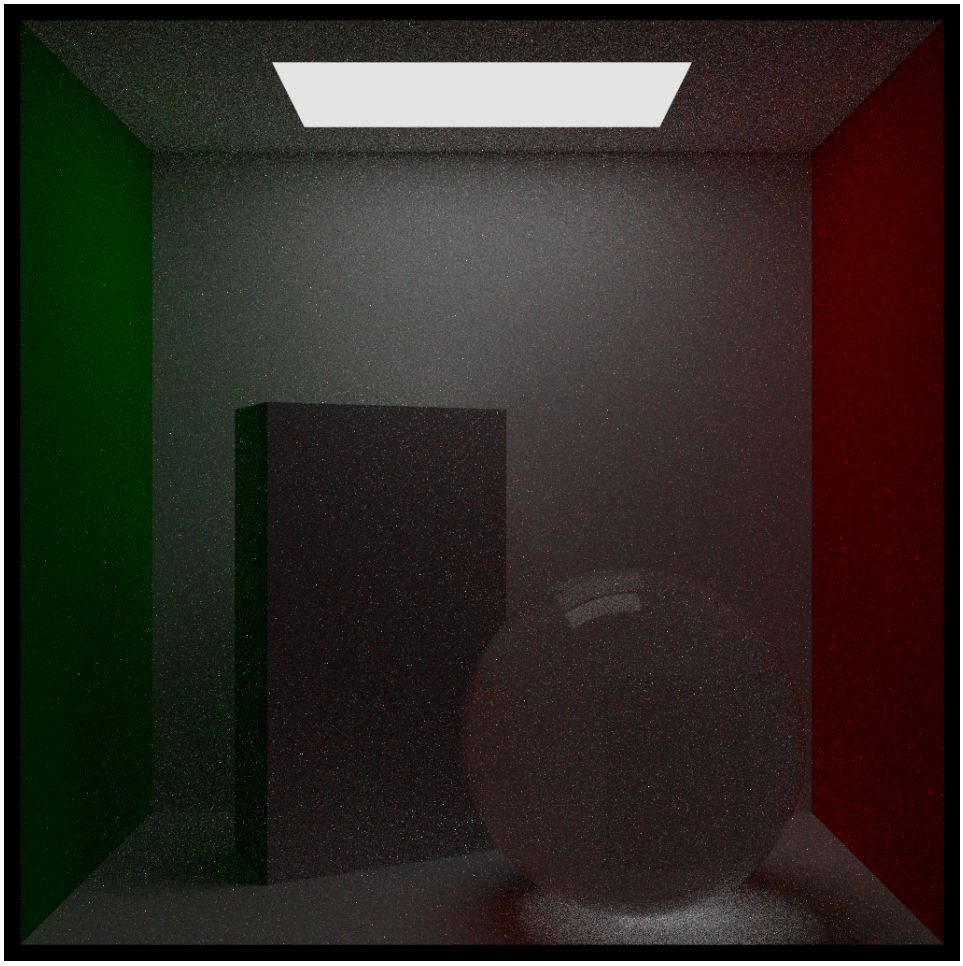


Fig. 25. On the other hand, running **with MIS** with 50 samples per pixel and produces a fairly reasonable image. This demonstrates how much faster the convergence is using an MIS approach and what utility it provides for artists doing protyping renders for films.

4 CONCLUSION

Overall, the use of MIS **greatly** improves the performance of ray tracing, by prioritizing directions that are believed to either have slower convergence due to complex light interactions or those believed to have strong impact on the final color. By combining PDFs appropriately, crisp images can be rendered in reasonable periods of time. Further work can extend the developed ray tracer into the realms of bidirectional tracing and Metropolis sampling of paths.

5 REFERENCES

[1] Pharr, M., Jakob, W., & Humphreys, G. (2016). *Physically based rendering: From theory to implementation.* Morgan Kaufmann.
[2] Shirley, P. (2018) *Ray Tracing in One Weekend.* raytracing.github.io/books/RayTracingTheRestOfYourLife.html

[3] Shirley, P. (2018)*Ray Tracing: The Next Week*. raytracing.github.io/books/RayTracingTheRestOfYourLife.html
[4] Shirley, P. (2018) *Ray Tracing: The Rest of Your Life*. raytracing.github.io/books/RayTracingTheRestOfYourLife.html