Path Tracing Notes

Alberto Morcillo Sanz

November 27, 2023

1 Introduction

Brief notes of Monte Carlo path tracing, explaining the math behind it in order code it in any programming language.

2 Rendering equation

The rendering equation is given by the following expression:

$$L_o\left(p, w_o\right) = L_e\left(p, w_o\right) + \int_\Omega f_r\left(p, w_i, w_o\right) L_i\left(p, w_i\right) \cos\theta dw_i$$

Where $\cos\theta = w_i \cdot n$ if w_i and n are normalized.

- The function L_o measures the outgoing radiance of a point p with a direction w_o.
- The function L_e measures the emitted radiance of a point p with a direction w_o.
- The function L_i measures the incoming radiance to a point p from a direction w_i.
- The function f_r is known as BRDF (bidirectional reflective distribution function) that scales the incoming radiance based on the surface's material properties
- Ω is the hemisphere aligned with the normal vector n

Note that the product is the Hadamard product or element-wise product

2.1 Solving the rendering equation recursively

Each L_i of a point is the L_o of other point as well. So the original equation may be written like:

$$L_o\left(p_1, w_{o_1}\right) = L_e\left(p_1, w_{o_1}\right) + \int_\Omega f_r\left(p_1, w_{i_1}, w_{o_1}\right) \left[L_e\left(p_2, w_{o_2}\right) + \int_\Omega f_r\left(p_2, w_{i_2}, w_{o_2}\right) \cdots \left(w_{i_2} \cdot n_2\right) dw_{i_2}\right] \left(w_{i_1} \cdot n_1\right) dw_{i_1}$$

As L_e does not depend on w_i we can take it out of the integral like:

$$L_o = Le + \int_\Omega f_r L_i \Rightarrow L_o = Le + \int_\Omega f_r L_e + \int_\Omega f_r \int_\Omega f_r L_i \Rightarrow L_o = Le + \int_\Omega f_r L_e + \int_\Omega f_r \int_\Omega f_r L_e + \int_\Omega f_r \int_\Omega f_r \int_\Omega f_r L_i \cdots$$

We can simply write this sum of integrals in the form of Neumann Series:

$$L_i = L_e + TL_e + T^2 L_e + T^3 L_e \ldots = \sum_{m=0}^{\infty} T^m L_e$$

3 Monte Carlo estimator

As the previous equation does not have an analytic solution, we have to find an approximation. One common way to do it is using the Monte Carlo method, so the equation:

$$L_o(p, w_o) = L_e(p, w_o) + \int_\Omega f_r(p, w_i, w_o) L_i(p, w_i)(w_i \cdot n) dw_i$$

can be approximated using the following estimator:

$$\hat{L}_o(p, w_o) = L_e(p, w_o) + \frac{1}{N} \sum_{i=0}^{N} \frac{f_r(p, w_i, w_o) L_i(p, w_i)(w_i \cdot n)}{p(w_i)}$$

3.1 Probability density function

$p(w_i)$ is the probability density function of the ray output in the direction w_i
As $p(w_i)$ is constant since all rays have the same probability of exiting in any direction from the hemisphere (the solid angle goes from 0 to 2π):

$$\int_\Omega p(w_i) dw_i = \int_0^{2\pi} p(w_i) dw_i = 1 \Rightarrow p(w_i) \int_0^{2\pi} dw_i = 1 \quad \therefore p(w_i) = \frac{1}{2\pi}$$

So finally we have the following expression:

$$\hat{L}_o(p, w_o) = L_e(p, w_o) + \frac{2\pi}{N} \sum_{i=0}^{N} f_r(p, w_i, w_o) L_i(p, w_i)(w_i \cdot n)$$

3.2 Casting rays

The way of solving the previous equation is casting a ray from the camera for each pixel. When the ray intersects a surface it bounces with a random direction. The rougher the surface is, the new ray direction will be more random. The smoother the surface is, the new ray direction will tend to be the reflected direction (like a mirror). To do this we use a linear interpolation:

$$lerp(u, v, t) = u + t(v - u) \quad \text{where} \quad u, v \in \mathbf{R^3}$$

Keep in mind that the reflect function is defined as:

$$r(\omega_o, n) = \omega_o - n(2n \cdot \omega_o)$$

```
// Generate diffuseDir (random direction: x,y,z between 0 and 1)
Vector3 diffuseDir(
    normal.x + (2 * static_cast<float>(rand()) / RAND_MAX - 1),
    normal.y + (2 * static_cast<float>(rand()) / RAND_MAX - 1),
    normal.z + (2 * static_cast<float>(rand()) / RAND_MAX - 1)
);

// Generate specularDir
Vector3 specularDir = reflect(wo, normal);

// Interpolate between diffuseDir and specularDir depending on the roughness
Vector3 wi = lerp(diffuseDir, specularDir, (1.0f - material.roughness));

// Flip wi if it is on the opposite side of the normal
if (wi.dot(normal) < 0.0f)
    wi = wi * -1.0f;

Line newRay(nearestIntersection, wi);
```

$w_i \cdot n < 0$ means that the direction w_i is in the sphere but not in the hemisphere so we need to take the opposite direction or calculate a new one.

We can create new rays from the intersection point (N samples) in order to generate more paths instead of bouncing only one ray. So for each bounce we cast N new rays. This is the key of path tracing.

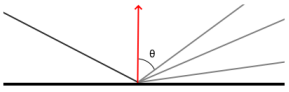

Figure 1: incoming and outgoing rays

4 BRDF (bidirectional reflective distribution function)

The bidirectional reflective distribution function is a function that defines how light is reflected at an opaque surface. Physically realistic BRDFs have additional properties:

- Positivity: $f_r(p, w_i, w_o) \geq 0$
- Obeying Helmholtz reciprocity: $f_r(p, w_i, w_o) = f_r(p, w_o, w_i)$
- Conserving energy: $\forall \omega_i \int_\Omega f_r dw_o \leq 1$ the outgoing energy is never greater than the incoming energy except if the material emits light

4.1 BRDF definition

In phisically based rendering, f_r is a BRDF which is usually a vector function which returns a color although in some cases it can be a scalar function which scales the incoming light. There are different BRDF, even you can model your own one. In this case, we are going to use the Cook-Torrance BRDF, which is one of the most used ones.

4.2 Cook-Torrance

It simulates how light behaves using two distinct approaches, distinguishing between diffuse reflection and specular reflection. The concept revolves around the simulated material reflecting a specific quantity of light in various directions (Lambert) and another portion in a specular manner, akin to a mirror. Consequently, the Cook-Torrance BRDF doesn't entirely substitute the previous model. Instead, we can precisely define the extent of radiance diffused and the amount reflected in a specular fashion, tailoring the simulation to the characteristics of the material in question. Thus the Cook-Torrance BRDF is defined as:

$$f_r = k_d f_{lambert} + k_s f_{cook-torrance}$$

$f_{lambert}$ is the refracted light (diffuse; light penetrating and exiting the material) and $f_{cook-torrance}$ is the reflected light (specular).
k_d and k_s are the ratios or the amount of diffuse and specular light. Due to energy conservation $k_d + k_s \leq 1$

```
Vector3 kS = fresnelSchlick(max(halfwayVector.dot(wo), 0.0f), specular);
Vector3 kD = Vector3(1.0f, 1.0f, 1.0f) - kS;
// Multiply kD by the inverse metalness such that only non-metals
// have diffuse lighting, or a linear blend if partly metal (pure metals
// have no diffuse light).
kD *= (1.0f - material.metallic);
```

4.2.1 Fresnel approximation and energy ratios

The Fresnel equation describes the ratio of light that gets reflected over the light that gets refracted, which varies over the angle we're looking at a surface.
We can approximate this equation using the Fresnel-Schlick approximation:

$$F_{Schlick}(h, v, F_0) = F_0 + (1 - F_0)\left[1 - (h \cdot v)\right]^5$$

As $cos\theta = h \cdot v = h \cdot \omega_o$ we can also define the Fresnel-Schlick approximation as:

$$F_{Schlick}(h, cos\theta) = F_0 + (1 - F_0)\left[1 - cos\theta\right]^5$$

F_0 represents the base reflectivity of the surface, which we calculate using the indices of refraction:

$$F_0 = \left(\frac{\eta_1 - \eta_2}{\eta_1 + \eta_2}\right)^2$$

Then, the k_d and k_s energy ratios can be defined as:

$$k_s = F_{Schlick}(h, w_o, F_0), \quad k_s \in \mathbf{R^3}$$
$$k_d = 1 - k_s, \quad k_d \in \mathbf{R^3}$$

Where h is the halfway vector, v is the direction of the viewer and F_0 is the surface's response at normal incidence at a 0 degree angle as if looking directly onto the surface.

$h = \frac{l+v}{||l+v||}$. In this case $h = \frac{w_i+w_o}{||w_i+w_o||}$

Fresnel equation table
Some of the more common values listed below as taken from Naty Hoffman's course notes:

Material	F_0 **(Linear)**	F_0 **(sRGB)**
Water	(0.02, 0.02, 0.02)	(0.15, 0.15, 0.15)
Plastic / Glass (Low)	(0.03, 0.03, 0.03)	(0.21, 0.21, 0.21)
Plastic High	(0.05, 0.05, 0.05)	(0.24, 0.24, 0.24)
Glass (high) / Ruby	(0.08, 0.08, 0.08)	(0.31, 0.31, 0.31)
Diamond	(0.17, 0.17, 0.17)	(0.45, 0.45, 0.45)
Iron	(0.56, 0.57, 0.58)	(0.77, 0.78, 0.78)
Copper	(0.95, 0.64, 0.54)	(0.98, 0.82, 0.76)
Gold	(1.00, 0.71, 0.29)	(1.00, 0.86, 0.57)
Aluminium	(0.91, 0.92, 0.92)	(0.96, 0.96, 0.97)
Silver	(0.95, 0.93, 0.88)	(0.98, 0.97, 0.95)

4.2.2 Diffuse light

Diffuse light is the refracted light. Light penetrating and exiting the material. It is basically the color of the material. (The division by π is a convention that helps normalize the diffuse reflectance in the Lambertian model. It ensures that the amount of reflected light is consistent with the spherical distribution of incident light, taking into account the average fraction of light contributing to diffuse reflection on a surface).

$f_{lambert} = \frac{c}{\pi}$ where c is the albedo or surface color.

4.2.3 Specular light

Specular light is the reflected light. Metal materials does not have diffuse lighting, only specular. So the more metal a material is, the reflected color tends to be similar to the incoming light, whereas the more plastic (dia-electric) it is, the reflected color tends to be the albedo color.

$$f_{cook-torrance} = \frac{DFG}{4\left(\omega_o \cdot n\right)\left(\omega_i \cdot n\right)}$$

D is the normal distribution function, F is the Fresnel equation and G is the geometry function.

Normal distribution function
The normal distribution function D statistically approximates the relative surface area of microfacets exactly aligned to the (halfway) vector h according to the roughness of the matrial α. We'll be using the Trowbridge-Reitz GGX:

$$D(n, h, \alpha) = \frac{\alpha^2}{\pi\left[\left(n \cdot h\right)^2\left(\alpha^2 - 1\right) + 1\right]^2}$$

Geometry function
The geometry function statistically approximates the relative surface area where its micro surface-details overshadow each other, causing light rays to be occluded.

$$G(n, \omega_o, k) = \frac{n \cdot \omega_o}{\left(n \cdot \omega_o\right)(1 - k) + k}$$

Where k is a remapping of α based on whether we're using the geometry function for either direct lighting or IBL lighting. $k_{direct} = \frac{(\alpha+1)^2}{8}$ or $k_{IBL} = \frac{\alpha^2}{2}$.

To effectively approximate the geometry we need to take account of both the view direction (geometry obstruction) and the light direction vector (geometry shadowing). We can take both into account using Smith's method:

$$G(n, \omega_o, \omega_i, k) = G(n, \omega_o, k)G(n, \omega_i, k)$$

4.3 Energy absorption

Depending on the material with which the ray intersects, it absorbs more or less energy. So there is a new parameter for each material called absorption that will allow us to scale the radiance.

```
Vector3 kS = fresnelSchlick(max(halfwayVector.dot(wo), 0.0f), specular);
Vector3 kD = Vector3(1.0f, 1.0f, 1.0f) - kS;
kD *= (1.0f - material.metallic);

float energyAbsorption = 1.0f - material.absorption
Vector3 fr = (kD.elementWiseProduct(diffuse) + specular) * energyAbsorption
```

5 Final equation

With every component of the Cook-Torrance BRDF described, we can include the physically based BRDF into the now final reflectance equation. Substituting $f_r = k_d f_{lambert} + k_s f_{cook-torrance}$ we get:

$$L_o\left(p, w_o\right) = L_e\left(p, w_o\right) + \int_\Omega \left[k_d \frac{c}{\pi} + k_s \frac{DFG}{4\left(\omega_o \cdot n\right)\left(\omega_i \cdot n\right)}\right] L_i\left(p, w_i\right)\left(w_i \cdot n\right) dw_i$$

This equation is not fully mathematically correct however. You may remember that the Fresnel term F represents the ratio of light that gets reflected on a surface. This is effectively our ratio k_s, meaning the specular (BRDF) part of the reflectance equation implicitly contains the reflectance ratio k_s. Given this, our final final reflectance equation becomes:

$$L_o(p, w_o) = L_e(p, w_o) + \int_\Omega \left[k_d \frac{c}{\pi} + \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} \right] L_i(p, w_i)(w_i \cdot n) dw_i$$

Solving the previous equation using Monte Carlo and the Cook Torrance BRDF we get the following estimator:

$$\hat{L}_o(p, w_o) = L_e(p, w_o) + \frac{2\pi}{N} \sum_{i=0}^{N} \left[k_d \frac{c}{\pi} + \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} \right] L_i(p, w_i)(w_i \cdot n)$$

6 Summary

- For each pixel throw a ray from the camera.
- If it intersects nothing, $Lo = (0, 0, 0)$ or $Lo =$ sky color
- If it intersects with an object. Solve the rendering equation using the Monte Carlo estimator. For each intersection, throw N rays (their direction depend on the roughness of the material intersected). And for each ray, solve the rendering equation. Keep in mind that $L_o^{t+1} = L_i^t$. It is a recursive problem that iterates a number of bounces.
- The recursive equation would look like this (It is the same as the recursive integral equation defined in page 1, but now using Monte Carlo in order to solve those integrals):

$$\hat{L}_{o1} = L_{e1} + \frac{2\pi}{N} \sum_{i=0}^{N} f_{r1} L_{i1} (w_{i1} \cdot n_1) = L_{e1} + \frac{2\pi}{N} \sum_{i=0}^{N} f_{r1} \left(L_{e2} + \frac{2\pi}{N} \sum_{i=0}^{N} f_{r2} L_{i2} (w_{i2} \cdot n_2) \right) (w_{i1} \cdot n_1) =$$

$$= L_{e1} + \frac{2\pi}{N} \sum_{i=0}^{N} f_{r1} \left[L_{e2} + \frac{2\pi}{N} \sum_{i=0}^{N} f_{r2} \left(L_{e3} + \frac{2\pi}{N} \sum_{i=0}^{N} f_{r3} \cdots (w_{i3} \cdot n_3) \right) (w_{i2} \cdot n_2) \right] (w_{i1} \cdot n_1)$$

- Consider bidirectional path tracing in order to get better results. Some times, depending on the environment, reaching a light is quite difficult, so that all the scene would be much darker (or even completely black) than it should be. So one way to solve this is not only throwing rays from the camera, but also from the light source and joining them in the same path.

7 Conclusions

Monte Carlo path tracing is noisy. Using many samples and bounces will make the image look smoother but it will take much computational time.
A good choice would be to denoise the image after generating it, this way there is no need on generating that many rays.