1 Introduction

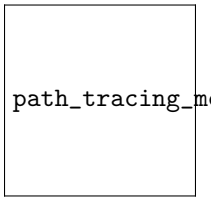
While stochastic simulation has many applications and ideologies, the goal in any simulation study is to evaluate estimators $\hat{\theta}$, of one or more parameters

$$\theta = E\phi(X) \tag{1}$$

where function $\phi(X)$ encodes the quantity or relationship we aim to evaluate on some domain X [**?**]. In a statistical application, $\phi(X)$ is some probabilistic model with input *random variable* X, where we wish to evaluate approximately how this model behaves. Alternatively, $\phi(X)$ may be a completely deterministic function but is impossible or inefficient to evaluate analytically, hence we aim to approximate its value through simulation.
In this chapter we delve into the latter case, and see how stochastic simulation is used in *path tracing* to efficiently model the deterministic but complex nature of light particles in computer graphics.

2 Path Tracing Model

Figure 1: Model of Path Tracing



As seen in Fig.() our *path tracing* simulation is modeled as a 3D space with a single light emitter e. That emitter projects rays with an *incidence* direction vector $\boldsymbol{\omega_i}$ onto multiple planes A at position vector $\boldsymbol{x}$. The ray is then reflected with an *outgoing* direction vector $\boldsymbol{\omega_o}$, this path is defined as $(\boldsymbol{\omega_i} \to \boldsymbol{x} \to \boldsymbol{\omega_o})$. The intensity of any given path is represented as $L_i(\boldsymbol{\omega_i} \to \boldsymbol{x})$ on incidence and $L_o(\boldsymbol{x} \to \boldsymbol{\omega_o})$ on outgoing, where $L_o = L_e$ at the emitter.

At each reflection, the rays intensity is scaled down based on geometry and material variables. For example, it is scaled by 1 in a mirrors perfect reflection and 0 in a true black's full absorption. However, in this chapter to reduce complexity we will just assume this scaling to be calculated based on some scattering term $g(\boldsymbol{\omega_i}, \boldsymbol{\omega_o})$.
Hence, for a single light path from position $\boldsymbol{x}$ in direction $\boldsymbol{\omega_o}$, we have

$$L_o^{single}(\boldsymbol{x} \to \boldsymbol{\omega_o}) = L_e(\boldsymbol{x} \to \boldsymbol{\omega_o}) + L_i^{single}(\boldsymbol{\omega_i} \to \boldsymbol{x}) g(\boldsymbol{\omega_i}, \boldsymbol{\omega_o}) \tag{2}$$

where if $\boldsymbol{x}$ is a light source then $L_i = 0$ as we are at the emitter and hence $L_o = L_e$. Otherwise, if $\boldsymbol{x}$ is on a surface A then $L_e = 0$ assuming surface doesn't emit light. Starting at the emitter L_e this can be solved recursively all the way to solve for L_o or vice versa.

However, this is only for a single path, any given L_o will be the cumulation of many different incidence rays at position $\boldsymbol{x}$. Thus, if we want the total contribution to L_o from all possible incidence paths we must integrate over all possible incidence light paths and their surface positions $A(\boldsymbol{x_i})$,

$$L_o(\boldsymbol{x} \to \boldsymbol{\omega_o}) = L_e(\boldsymbol{x} \to \boldsymbol{\omega_o}) + \int_A L_i(\boldsymbol{\omega_i} \to \boldsymbol{x}) g(\boldsymbol{\omega_i}, \boldsymbol{\omega_o})\, dA(\boldsymbol{x_i}). \tag{3}$$

Once again, starting at the source we can solve each integral recursively to solve for L_o.

Now imagine, we solve for the $L_o(\boldsymbol{x} \to \boldsymbol{\omega_o})$ that intersects with some virtual pixel after passing through a pinhole lense Fig(). It is clear, we now know what intensity to display that pixel at to produce realistic lighting of position x. Repeating this process for all pixels, gives a perfect global illumination image of some virtual scene. This *rendering equation* has no known closed-form solution and requires using *Monte Carlo*

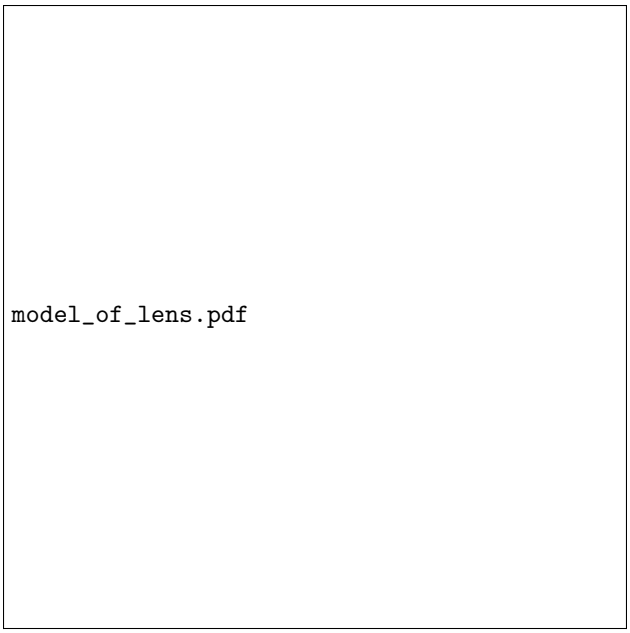


Figure 2: Shows the path of light into a pixel through a lens.

integration to evaluate. As shown in Eq.(**??**) Monte Carlo integration has variance that reduces proportional to $1/N$ irrespective of the dimensions of an integral. For this reason it is the best way to approximate for L_o in our path tracing simulation. [**?**]

3 Monte Carlo Integration

From Eq.(**??**) it is not immediately evident how a stochastic simulation can be used to evaluate for the integral $L_o(\boldsymbol{x} \to \boldsymbol{\omega_o})$ on the domain A.
If we let some parameter θ be the integral of $f(\boldsymbol{x})$ over domain $D \subset \mathbb{R}^d$, where $\boldsymbol{x} = (x_1, \ldots, x_d)$, then

$$\theta = \int_D f(\boldsymbol{x})\, d\boldsymbol{x}.$$

However, if we transform it to be an expected value for some random variable

$$= \int_D \frac{f(\boldsymbol{x})}{p_X(\boldsymbol{x})} p_X(\boldsymbol{x})\, d\boldsymbol{x}$$

where p_X is the p.d.f. of $\boldsymbol{X} = (X_1, \ldots, X_d)$ and $p_X(\boldsymbol{x}) > 0$ when $f(\boldsymbol{x}) \neq 0$, thus

$$= E\left[\frac{f(\boldsymbol{X})}{p_X(\boldsymbol{X})}\right]. \tag{4}$$

It can be seen that this is in the form of Eq.(**??**), where θ is the parameter we are trying to solve for. Thus, in the same way we would for a stochastic model, we can solve this through random sampling by taking the sample mean $\hat{\theta}$ as an unbiased estimator

$$\hat{\theta} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\boldsymbol{X}_i)}{p_X(\boldsymbol{X}_i)} \tag{5}$$

where $\hat{\theta} \to \theta$ as $N \to \infty$.
And thus it becomes clear that we can solve Eq.(**??**) by taking $N \to \infty$ random samples of the incidence rays with direction $\boldsymbol{\omega_i}$, from the random variable Ω.

$$L_o(\boldsymbol{x} \to \boldsymbol{\omega_o}) = \frac{1}{N} \sum_{i=1}^{N} \frac{L_i(\boldsymbol{\omega_i} \to \boldsymbol{x}) g(\boldsymbol{\omega_i}, \boldsymbol{\omega_o})}{p_\Omega(\boldsymbol{\omega_i})} \tag{6}$$

Additionally, because it can be seen that L_i can be solved by recursively applying Eq.(**??**) until $L_i = L_e$ after K_i steps, we have

$$= \frac{1}{N} \sum_{i=1}^{N} \prod_{k=1}^{K_i} \left(\frac{1}{M} \sum_{j=1}^{M} \frac{g(\boldsymbol{\omega_{k,j}}, \boldsymbol{\omega_{k-1,j}})}{p_\Omega(\boldsymbol{\omega_{k,j}})} \right) L_e \tag{7}$$

where k is our k^{th} recursion and j is our j^{th} sample at the k^{th} recursion. Here we are assuming that we have the same sample size M and p.d.f. p_Ω at each recursion, however even if they were different this would still be an un-bias estimator. In fact, often more complex algorithms adjust these variables proactively to reduce variance.
Here we let $M = 1$, which despite increasing the variance, reduces computational requirements and makes the model easier to visualise.

$$L_o(\vec{x}, \boldsymbol{\omega_o}) = \frac{1}{N} \sum_{i=1}^{N} \prod_{k=1}^{K_i} \left(\frac{g(\boldsymbol{\omega_k}, \boldsymbol{\omega_{k-1}})}{p_\Omega(\boldsymbol{\omega_k})} \right) L_e \tag{8}$$

Intuitively, this can be thought of as starting at some position $\boldsymbol{x}$ with an outgoing radiance L_o. We then sample random variable Ω with p.d.f. p_Ω for a random incidence direction $\boldsymbol{\omega_i}$. We then backwards propagate through this path, repeating the process at every interaction with a surface, until we reach the light source L_e. We then calculate the total scaling of L_e by the scattering terms $g(\boldsymbol{\omega_k}, \boldsymbol{\omega_{k-1}})$, and being divided by $p_\Omega(\boldsymbol{\omega_k})$ to remove bias at every step. This scaled value of L_e is one sample of L_o, where the sample mean is an estimator for L_o that converges as $N \to \infty$. This can be clearly seen as the recursive sampling in Eq.(**??**).

3.1 Importance Sampling

While an estimator $\hat{\theta}$ is certainly unbias for any $p_X(x)$ as seen in Eq.(**??**), it does not mean that the variance remains constant regardless of what distribution we choose to sample with. In fact, it is apparent that our choice of p_X weights our sample for values with a higher probability and thus with an optimal choice of p.d.f we can reduce the variance.
Evaluating the variance of $\hat{\theta}$ from Eq.(**??**), we have

$$var\, \hat{\theta} = \frac{1}{N} var \left(\frac{f(\boldsymbol{X})}{p_X(\boldsymbol{X})} \right) \tag{9}$$

$$= \frac{1}{N} \left[\; E\left(\frac{f^2(\boldsymbol{X})}{p_X^2(\boldsymbol{X})} \right) - E\left(\frac{f(\boldsymbol{X})}{p_X(\boldsymbol{X})} \right)^2 \; \right] \tag{10}$$

$$= \frac{1}{N} \left[\; E\left(\frac{f^2(\boldsymbol{X})}{p_X^2(\boldsymbol{X})} \right) - \theta^2 \; \right]. \tag{11}$$

It can be clearly seen in Eq.(**??**), that as $p_x(\boldsymbol{x})$ more closely resembles $|f(\boldsymbol{x})|$ the variance reduces. In fact, Jensen's inequality allows us to derive that our minimum variance occurs for $p_X(\boldsymbol{x}) \propto |f(\boldsymbol{x})|$ [**?**].
Fig() shows the clear advantage of a good sample distribution.

Unsurprisingly, this is termed *importance sampling* as we sample where we expect to get the most contribution to our final estimator.

Naturally, this can be applied to path tracing in a variety of ways. One obvious method is to sample with a distribution similar to the scattering term $g(\boldsymbol{\omega_i} \to \boldsymbol{\omega_o})$ from section **??**. Intuitively, this makes sense as sampling in areas with more light will provide a greater contribution to our final image as shown in Fig().

Another, basic method of importance sampling is deciding when to terminate a path. By terminating paths based on a p.d.f. that relates to the absorption probability of light, we prevent wasted resources on long sample paths.
Fig() shows path tracer algorithms with different levels of importance sampling for the same run time.

4 Simulating Random Variables

While it is clear that we can estimate complex integrals via estimating samples, the question of how to actually create these sample distributions still remains unanswered.

4.1 Generating Random Numbers

In simulation theory we define randomness as, events which are non-deterministic, independent and also follow a known distribution [**?**]. Hence, the first step in simulating any random variable, is to generate a sequence of *independent and identically distributed* random numbers.

True Random Numbers These are generated numbers that are in every essence truly non-deterministic and independent. The only known method to generate such numbers is to measure inherently entropic natural phenomena. This can include atomic decay, or the random state of internal hardware as measured by **HAVEGE** [**?**]. However, no matter what the generator, in the case of simulation which requires computational algorithms, outsourcing RNG is far to limiting to be viable.

4.1.1 Pseudorandom Numbers

The caveat is that we only need random numbers to seem random within the framework of our model, but the methods of generation can in fact be completely algorithmic. These deterministic generators are what are commonly referred to as *pseudorandom number generators* or PNG [**?**].

Linear Congruential Generators By creating a *true* random seed x_0 through **HAVEGE**, we can recursively use the algorithm denoted LCG$[m, a, c]$ or equivalently

$$x_i \equiv (ax_{i-1} + c) \pmod{m}, \quad \text{with} \quad 0 \leq x_i < m. \tag{12}$$

Thus, producing a *Lehmer* sequence of uniformly distributed numbers with period $k \leq m$. We define parameters a, c and m as the *multiplier*, *increment* and *modulus* respectively, where in a *multiplicative Congruential generator* the increment $c = 0$. [**?**][**?**]

A popular choice is *the Minimal Standard* LCG$[2^{31} - 1, 16807, 0]$, which despite being deterministic can be seen in fig() to be an i.i.d. $X \sim U(a, b)$ random variable [**?**].
For every generator, there will exist applications in which the determinism of the generator is exploited to introduce bias. However, often this can be mitigated by the correct choice of parameters.

Period It is generally accepted, that for linear methods an upper bound for the sample size is $n = \sqrt{k}$, where k is the period of the PNG [**?**]. Thus methods to find and then maximise the period of a generator are necessary. For any LCG$[m, a, c = 0]$ the maximal period is bounded by the *Euler totient function* $\phi(m)$, occurring only when a and m are coprime. [**?**]. Furthermore, it is evident from Eq.(**??**) that the generator repeats for the smallest positive k such that $x_i = x_{i+k}$. Thus by solving recursively from x_i till x_{i+k} we have

$$a^k \equiv 1 \bmod m, \tag{13}$$

where $k = \phi(m)$, if a and m are coprime. [**?**]

Thus in the minimal standard LCG, because $a = 16807$ and $m = 2^{31}$ are coprime, our maximal period is $k = \phi(2^{31}) = 2^{31} - 1$, which is large enough for most sample sizes.

Independence through Dimensions Additionally, we must ensure independence is maintained through the dimensions d of our application. In our path tracer Eq.(**??**), every sample path of k_i recursions has dimensions of at least $d = 2K_i$. That means we need a PNG such that biases do not accumulate over every bounce in our simulation. We can clearly see in fig(), that despite *the Minimal Standard* having parameters that maximise the period, the distribution of d-tuples $x_n = (x_n, x_{n+1} \dots x_{n+d})$ have a clear correlation in $d = 2$ [**?**].
This is a known weakness of LCGs as while they are efficient they have poor dimensionality. For this reason path tracers such as ours commonly use the *Mersenne Twister* generator, which has period $2^{19937} - 1$, more than the number of photons that has ever existed, and dimensions $d = 623$ potentially allowing for samples with upwards of 300 bounces [**?**].

Fig() shows the comparison of the *Minimal Standard* and the *Mersenne Twister* in path tracing over equal sample size.

4.2 Generating Random Variables

The Inverse Transformation Method Naturally once we have found a suitably un-bias linear generator we must simulate the desired distribution.

Let U be a uniform (0,1) random variable. Then for the monotone c.d.f. F with inverse F^{-1}, we define the random variable X as

$$X = F^{-1}(U). \tag{14}$$

Thus

$$\begin{aligned} F_X(a) &= P\{X \leq a\} \\ &= P\{F^{-1}(U) \leq a\} \\ &= P\{U \leq F(a)\} \\ &= F(a) \quad [\textbf{?}]. \end{aligned} \tag{15}$$

For a diffuse surface in path tracing, it is common to importance sample a cosine distribution to better simulate how Lambertian reflectance occurs. Fig() shows how a linear generator can generate a cosine transformation.

It should be noted that the theory for these equations assumes U is a true random variable. Hence, when using PNGs we must test our simulated random variable to be sure it is distributed correctly [**?**]. In the case where independence has been confirmed, a given sample sequence that passes a significantly rigorous good-fit test against p.d.f. p_X of X, can from then on be taken to be a sample from random variable X [**?**].

Kolmogrov-Smirnov Test The Glivenko-Cantelli theorem states that, for a random sample $X_1, \dots, X_n$ from some unknown c.d.f. $F_X(x)$, and our *sample c.d.f.* $F_N(x)$, defined as

$$F_N(x) = \frac{1}{N}(\text{number of } X_i \text{ less than or equal to } x) \ , \tag{16}$$

then

$$\lim_{N\to\infty} P\left[\sup_x |F_N(x) - F_X(x)| > \epsilon\right] = 0 \ . \tag{17}$$

Thus showing for large N the maximum deviation $\sup_x |F_N(x) - F_X(x)|$, denoted simply as D_N, between the *true* function F_X and our sample c.d.f. F_N is small for all x. Using this quality Kolmogrov proved and Smirnov later formalised that for any continuous distribution F_X

$$\lim_{N\to\infty} P(\sqrt{N}\ D_N \leq x) = H(x) \tag{18}$$

where the Kolmogrov distribution function $H(x)$ is tabulated. Thus remarkably this Kolmogrov-Smirnov test allows a sample of any distribution and any sufficiently large sample size (at least 35) to be evaluated against $H(x)$. [**?**]

Fig() shows the comparison of the *sample c.d.f.* $F_N(x)$ of our generated cosine random variable from Fig() against the Kolmogrov distribution function $H(x)$.