Hybrid Radiosity/Monte Carlo Methods

Peter Shirley

1 Introduction

Monte Carlo methods refer to any method that uses random numbers to get an approximate answer to a problem. In computer graphics Monte Carlo techniques can be used to perform radiosity calculations and can be used in distribution ray tracing for effects such as soft shadows and motion blur. Although these notes are for a course on radiosity, they have a great deal of material from a previous Siggraph 93 global illumination course notes that also dealt with distribution ray tracing, although the radiosity material has been expanded.

In these notes I will cover the basics of both *Monte Carlo simulation*, where a physical system is modeled, *Monte Carlo integration*, where random numbers are used to approximate integrals, and *Quasi-Monte Carlo integration*, where non-random numbers are used. This discussion will cover the general techniques, and will use global illumination problems as examples.

One appeal of using Monte Carlo methods is that they are easy to design and use. However, it is not so easy to design a *good* Monte Carlo method, where the computation can be completed to the desired accuracy relatively quickly. Here both cleverness and some analytic skills are required. Fortunately, the analytic skills are fairly narrow in scope, so many of them can be covered in this short tutorial. A more formal discussion of Monte Carlo simulation can be found in the neutron transport literature (e.g. [47]) and an extremely current survey of Monte Carlo integration for practical applications can be found in the survey article by Spanier and Maize[48].

2 Background and Terminology

Before getting to the specifics of Monte Carlo techniques, we need several definitions, the most important of which are: *continuous random variable*, *probability density function*, *expected value*, and *variance*. This section is meant as a review, and those unfamiliar with these terms should consult an elementary probability theory book (particularly the sections on continuous, rather than discrete, random variables).

Loosely speaking, a *continuous random variable* x is a scalar or vector quantity that 'randomly' takes on some value from a continuous space S, and the behavior of x is entirely described by the distribution of values it takes. This distribution of values can be quantitatively described by the *probability density function*, p, associated with x (the relationship is denoted $x \sim p$). If x ranges over some region S, then the probability that x will take on a value in some subregion $S_i \subset S$ is given by the integral:

$$Prob(x \in S_i) = \int_{x \in S_i} p(x) d\mu_x \quad (p : S \to \Re^1) \tag{1}$$

Here $Prob(\textit{event})$ is the probability that *event* is true, so the integral is the probability that x takes on a value in the region S_i. The measure μ_x is the measure on our probability space. In graphics S is often an area ($d\mu = dA = dxdy$), or a set of directions (points on a unit sphere: $d\mu = d\omega = \sin\theta d\theta d\phi$). Loosely speaking, the probability density function describes the relative likelihood of a random variable taking a certain value; if $p(x_1) = 6.0$ and $p(x_2) = 3.0$, then a random variable with density p is twice as likely to have a value "near" x_1 than it it to have a value near x_2. The density p has two characteristics:

$$p(x) \geq 0 \quad \text{(Probability is nonnegative)} \tag{2}$$

$$\int_{x\in S} p(x) d\mu_x = 1 \quad (x \text{ has a value in } S) \tag{3}$$

As an example, the *canonical* random variable ξ takes on values between zero and one with uniform probability (here *uniform* simply means each value for ξ is equally likely). This implies that:

$$f(\xi) = \begin{cases} 1 & \text{if } 0 \leq \xi \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

The space over which ξ is defined is simply the interval $[0, 1]$. The probability that ξ takes on a value in a certain interval $[a, b]$ is:

$$Prob(a < \xi < b) = \int_a^b 1 dx = b - a$$

As a two dimensional example, a random variable α is a uniformly distributed random variable on the disk of radius R. Since it is uniform, we know that $p(\alpha)$ is some constant. From Equation 3 we can deduce that $p(\alpha) = 1/(\pi R^2)$. The measure on the disk is just area. This means that the probability that α is in a certain subset S_1 of the disk is just:

$$Prob(\alpha \in S_1) = \int_{\alpha \in S_1} \frac{1}{\pi R^2} dA_\alpha$$

This is all very abstract. To actually use this information we need the integral in a form we can evaluate. Suppose S_i is the portion of the disk closer to the center than the perimeter. If we convert to polar coordinates, then α is represented as a (r, θ) pair, and S_1 is where $r < R/2$. The differential area dA becomes $r dr d\theta$. This leads to:

$$Prob(r < \frac{R}{2}) = \int_0^{2\pi} \int_0^{\frac{R}{2}} \frac{1}{\pi R^2} r dr d\theta = 0.25$$

The average value that a real function f of a one dimensional random variable will take on is called its *expected value*, $E(f(x))$:

$$E(f(x)) = \int_{x\in S} f(x)p(x)d\mu_x$$

Note that the expected value of a one dimensional random variable can be calculated by letting $f(x) = x$. The expected value has a surprising and useful property: the expected value of the sum of two random variables is the sum of the expected values of those variables ($E(f(x) + g(y)) = E(f(x)) + E(g(y))$). This property holds *whether or not* the variables are independent! Since the sum of two random variables is itself a random variable, this principle generalizes. As an example of expectation, consider random points on the disk of radius R. What is the expected distance r from the center of the disk?

$$E(r) = \int_0^{2\pi} \int_0^R \frac{1}{\pi R^2} r^2 dr d\theta = \frac{2R}{3}$$

The *variance*, $var(x)$, of a one dimensional random variable is the expected value of the square of the difference between x and $E(x)$:

$$var(x) = E(\left[x - E(x)\right]^2) = E(x^2) - \left[E(x)\right]^2$$

The expression $E(\left[x - E(x)\right]^2)$ is more useful for thinking intuitively about variance, while the algebraically equivalent expression $E(x^2) - \left[E(x)\right]^2$ is usually convenient for calculations. The variance of a sum of random variables is the sum of the variances *if the variables are independent*. This summation property of variance is one of the reasons it is frequently used in analysis of probabilistic models. The square root of the variance is called the *standard deviation*, σ, which gives some indication of absolute deviation from the expected value.

Many problems involve sums of independent random variables x_i, where the variables share a common density f. Such variables are said to be *independent identically distributed* random variables. When the sum is divided by the number of variables, we get an estimate of $E(x)$:

$$E(x) \approx \frac{1}{N} \sum_{i=1}^{N} x_i$$

This idea can be generalized to the *Law of Large Numbers*:

$$Prob\left[E(x) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} x_i\right] = 1$$

This idea leads naturally to the idea of Monte Carlo estimation of integrals.

3 Monte Carlo Simulation

For some physical processes, we have statistical models of behavior at a microscopic level from which we attempt to derive an analytic model of behavior. For example, we often think of a luminaire (a light emitting object) as emitting a very large number of random photons (really pseudo-photons that obey geometric, rather than physical, optics) with certain probability density functions controlling the wavelength and direction of the photons. From this a physicist might use statistics to derive an analytic model of how the luminaire distributes its energy in terms of the directional properties of the probability density functions. However, if we are not interested in forming a general model, but instead want to know about the empirical behavior of a luminaire in a particular environment, we can just numerically simulate the behavior of the luminaire. This is particularly useful if we only care about the effects of the luminaire, and are not trying to derive any high level understanding of its behavior. To do this we computationally "emit" photons from the luminaire and keep track of where the photons go. This simple method is from a family of techniques called *Monte Carlo Simulation* and can be a very easy, though often slow, way to numerically solve physics problems. In this section simulation techniques are discussed, and methods for improving their efficiency are presented.

The first thing that you might try in generating a highly realistic image is to actually track simulated photons until they hit some computational camera plane or were absorbed. This would be very inefficient, but would certainly produce a correct image, although not necessarily while you were alive. In practice, very few Monte Carlo simulations model the full physical process. Instead, an *analog* process is found that is easier to simulate, but retains all the *important* behavior of the original physical process. One of the difficult parts of finding an analog process is deciding what effects are important.

An analog process that is almost always employed in graphics is to replace photons with set wavelengths with power carrying beams that have values across the entire spectrum. If photons are retained as an aspect of the model, then an obvious analog process is one where photons whose wavelengths are outside of the region of spectral sensitivity of the film do not exist.

Several researchers (e.g. [3]) have used Monte Carlo simulation of a simple analog of optics, where only Lambertian and specular surfaces are used. A lambertian surface is one with several simple properties. First, its radiance at any wavelength does not vary with viewing angle. Second, this radiance varies linearly according to the total incident power per unit area and the reflectance of the surface. Quantitatively this can be written:

$$L(\lambda) = \frac{\rho_d(\lambda)\Phi_{incoming}(\lambda)}{\pi A} \tag{4}$$

where $L(\lambda)$ is the spectral radiance at wavelength λ, $\rho_d(\lambda)$ is the reflectance of the surface at λ, $\Phi_{incoming}(\lambda)$ is the incident power per unit wavelength at λ, and A is the area of the surface being illuminated. What makes the Lambertian surface attractive, is that if we can figure out how much light is hitting it (irrespective of where the light comes from), then we know its radiance for all viewing directions. Note that ρ_d is the reflectance, not the BRDF, of the surface. The BRDF is the reflectance divided by π.

As an example of an analog process, the *illumination ray tracing* of Arvo[3] assumed photons traveled as bundles with a spectral distribution. He further assumed that these bundles were attenuated when reflecting from a specular surface. Like almost all graphics programs, his also assumed that the optical properties of the scene were constant within the time interval the picture represented, and that this time interval was very large relative to the speed it takes light to travel any distances in the scene. This last assumption, usually taken for granted, makes it possible to treat light as moving instantaneously within our programs. Finally, Arvo assumed that diffuse surfaces can be broken into zones whose radiances are described by the power incident to them (i.e. they obey Equation 4).

These assumptions allowed Arvo to trace power carrying rays and mark each zone with the accumulated power. Once the simulation was over, the radiance of each zone could be calculated using Equation 4. Although we often think of this as being a brute force physical simulation, it is important to remember that this is really the simulation of an analog process where all wavelengths follow the same paths, and time dependent behavior can be ignored.

The trickiest part of implementing Arvo's simulation method is tracking the power through the environment. An natural choice for tracking the power is to use ray tracing. However, it is not so obvious how many rays to send, or where to send them. This question has been examined in a fairly sophisticated way in [17], but even for a simple implementation the answer is non-obvious. The number of rays that must be sent is "enough". This depends on how much noise is acceptable in an image, and how small the zones are. In [41] it is argued that the number of rays should be linearly proportional to the number of zones, so doubling the number of zones implies that the number of rays should also be doubled. A visual example of this argument is shown in Figure 1 where an

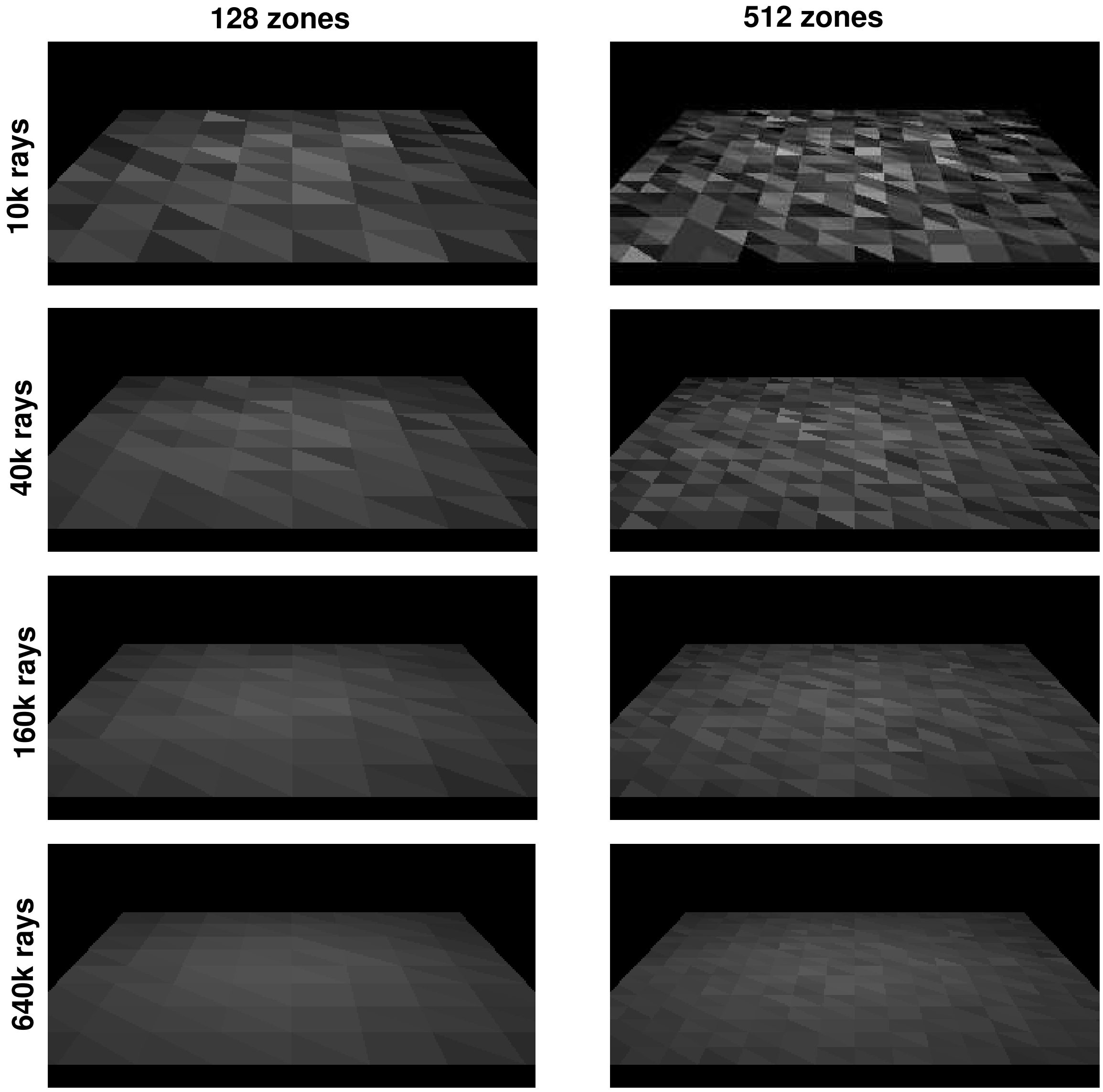

Figure 1: Noise reduction as the number of energy bundles increases. Note that the number of bundles needed is approximately inversely related to the surface area of each zone.

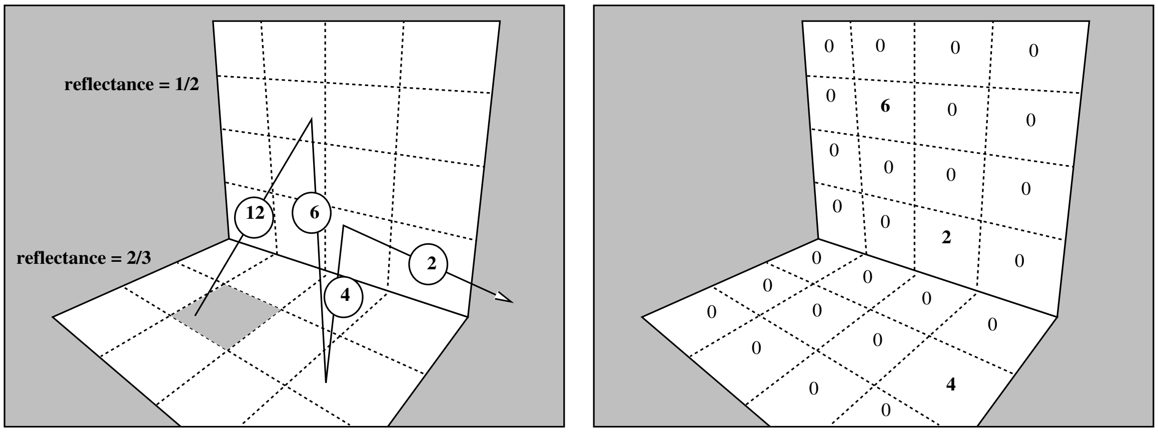

Figure 2: Reflection Simulation. The patch on the floor is a luminaire and emits a "photon" with 12 watts of power. Each reflection damps some of the power and scatters the photon according to a diffuse (cosine) distribution. On the right is the outgoing power from each patch after the photon leaves the environment. The emitted power is also stored for each patch but is not shown.

environment with four times as many zones seems to require four times as many rays for the same level of accuracy as the environment with fewer zones[1]. The other detail, where the rays should be sent, is easier. The rays should be generated randomly with the same distribution as the emitted power of the luminaire. Generating rays sets with such directional distributions is discussed in Appendix A. The rays should also be sent from points distributed on the surface of the luminaire. To do this, first choose a random point on the luminaire surface, and then choose a random direction based on the surface normal at that point.

Arvo's method can be extended to a radiosity[13] method by letting the Lambertian zones interreflect light[29, 1, 2, 38]. This is really just a ray tracing variant of the progressive refinement radiosity method[7]. In this method, reflectance ($\rho_{d,i}$) and emitted power ($\Phi_{e,i}$) of the ith zone are known, and the reflected power ($\Phi_{r,i}$) ($\Phi_{r,i} = \rho_d \Phi_{incoming,i}$) is unknown. If we solve for $\Phi_{r,i}$, then we can find Φ_i, the total power coming from the ith patch.

Once the total power of each patch is found, it can be converted to radiance using Equation 4. These radiance values can then be interpolated to form a smooth appearance[8].

We first set our estimate of Φ_i to be $\Phi_{e,i}$ for all i. For each surface i that has non-zero $\Phi_{e,i}$, we can shoot a set of n_i energy packets each carrying a power of Φ_i^e/n_i. When a packet with power Φ hits a surface j, we can add $\rho_{d,j}\Phi$ for our estimate of Φ_j, and reflect a new energy packet with power $\rho_{d,j}\Phi$. This energy packet will bounce around the environment until it is depleted to a point where truncation is used. This basic energy packet tracing technique has been used in Heat Transfer[19, 12, 50], Illumination Engineering[49], and Physics[47, 21].

This method, which I call *reflection simulation* (see Figure 2), is problematic in that each reflection is followed by a ray intersection test to find the next surface hit. The later reflections will carry a relatively small amount of power, so tracing these later rays is somewhat wasteful in the sense that we have bad 'load-balancing': some rays do more work than others. One solution to this problem is to replace the reflection model with an analog model where light is absorbed and

[1]The number of rays sent can be thought of as the number of photons tracked in a certain time interval. The number of rays will be proportional to the time a "shutter" is open. Once the "exposure" is long enough, the noise will not be objectionable.

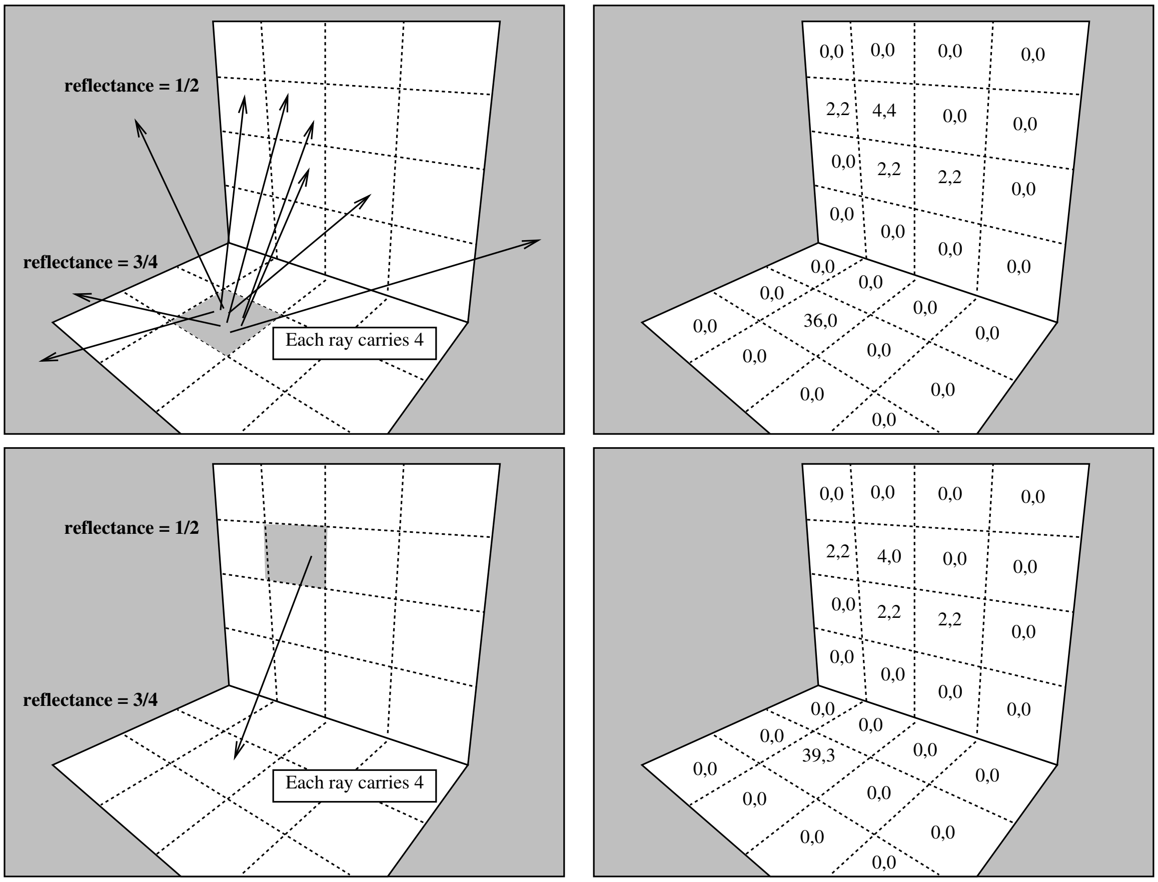

Figure 3: Absorb and reemit. The patch on the floor has 36 units of power that it has not distributed. Each patch has tow numbers, the total reflected power (left of pair), and the power that still needs to be sent (right of pair).

immediately reemitted (after attenuation by the reflectance) (see Figure 3). A scene where light is absorbed and reemitted in this way looks similar to a scene where light is reflected, so solving for the transport in either model will hopefully yield a similar solution. The difference is that now light may strike one side of a zone and later be reemited on the other side of the zone. This can give rise to objectional artifacts if the zone is partially in a dark area and partially in a light area[2] (e.g. goes under a door between the outside and a dark room). To solve for this absorb and reemit model, we can again send power in bundles from light sources. When a bundle carrying power Φ hits a surface j, the absorbed power that will later be reemitted by surface j can be increased by $\rho_{d,j}\Phi$. After each light source emits its power, reflective surfaces can, in turn, emit their absorbed power. The efficiency of this method is best if surfaces with the greatest amount of power send their power first.

The reason that we have the freedom to let the zones emit in any order we choose is that our analog has lost its time dependence. We are lucky the speed of light is so fast! There are two points which are crucial to the implementation of this progressive refinement method. The first is that the number of rays emitted from a certain zone is proportional to the power being emitted in that

[2]Thanks to Dani Lischinski for pointing this out. Earlier versions of this document said that absorb and reemit was asymptotically equivalent to the photon tracking model.

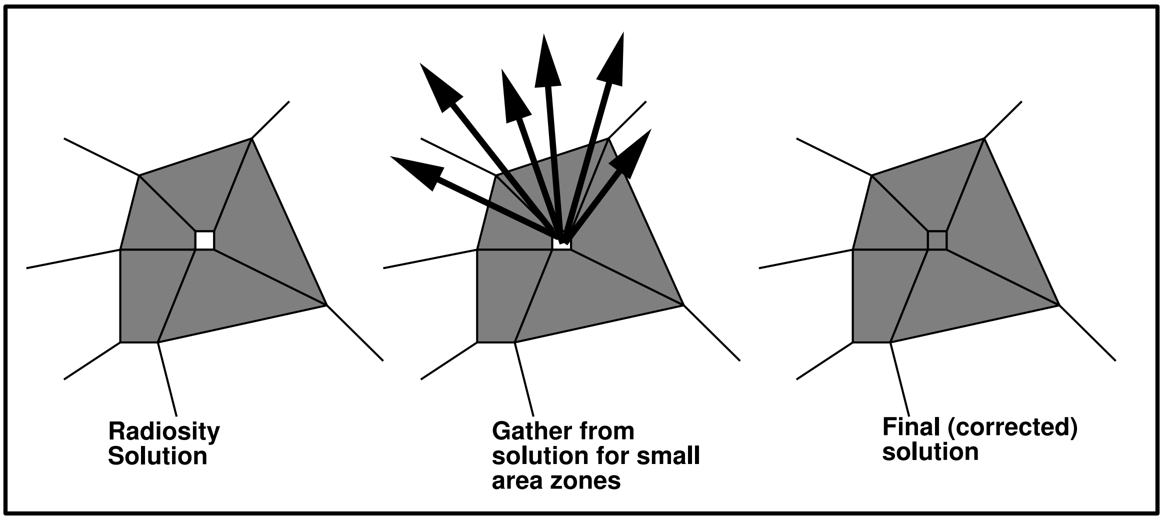

Figure 4: Zones with small areas have their radiance recalculated more accurately in a postprocess.

iteration (each ray carries approximately the same amount of power). The other is that, unlike in [7], the zone with the most power is not searched for, or the time complexity of the method will increase from $O(N \log N)$ to $O(N^2)$, where N is the number of zones[41]. This problem can be avoided if a heap or similar structure is used to make the search for maximum $O(\log N)$ rather than $O(N)$. A more detailed discussion of the implementation of Monte Carlo radiosity can be found in [40].

The biggest problem with these Monte Carlo radiosity methods is that small zones will be undersampled and will have large errors, or enough rays will be sent that the large area zones are oversampled. This is only a problem in scenes with a large range of zone areas, but this is not uncommon. One way to get around this problem (that I have not yet tried) is to do a "gather" on small zones in the scene after the first radiosity solution is done. The radiance of the zone is simply its reflectance times the average radiance "seen" by the gather rays provided they are sent in a cosine distribution. This idea is illustrated in Figure 4.

This simple simulation method could also be used for diffuse transmission, in a manner similar to that of Rushmeier and Torrance[35]. Some of the simulation techniques discussed earlier can be extended to non-diffuse reflection types. The most important application is to scenes that include specular surfaces, but glossy surfaces are sometimes desirable too.

The simplest method of including specular reflection in a zonal calculation is the *image method*[51, 35]. In the image method, a specular surface is replaced by a hole into a virtual environment. This method works only for planar mirrors, but performs very well for environments that have one important specular surface like a mirror or highly polished floor. Malley extended his Monte Carlo power transport method to account for zonal transport by specular surfaces[29]. He did this by allowing power carrying rays to reflect off specular surfaces as shown in Figure 5. The colors of specular surfaces can be determined in the viewing phase by standard ray tracing. Sillion and Puech used a similar technique to account for specular reflection, and included subdivision strategies for sampling more heavily where ray paths diverged[46].

Any non-diffuse reflectors can have zonal values, as long as each incoming power packet adds to a power *distribution function* that will be reemitted. In the viewing stage, this distribution can be queried with results depending on viewer position. The distribution functions could be stored in a

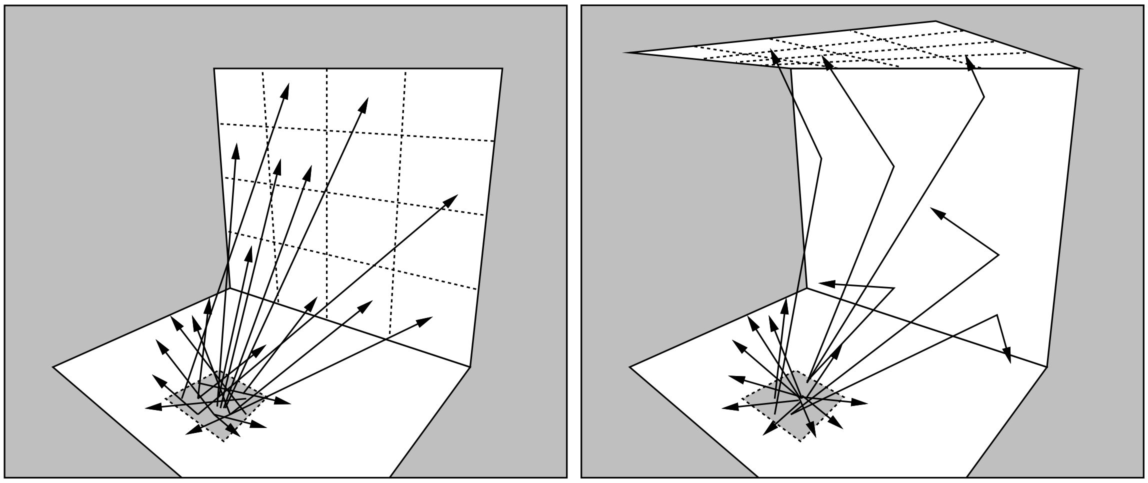

Figure 5: Monte Carlo emission of energy with and without specular reflection.

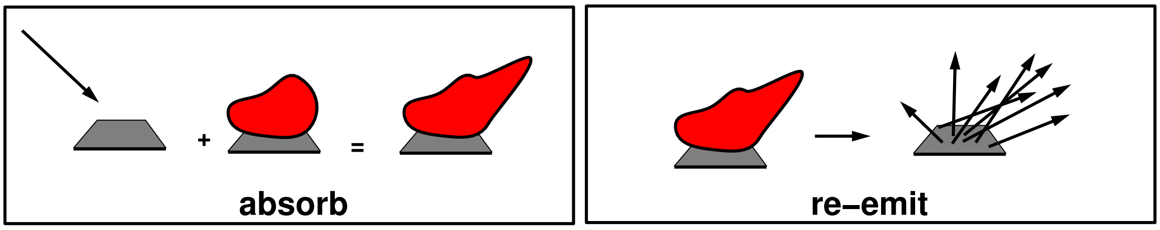

Figure 6: Absorb and re-emit strategy requires directional distribution at each zone and a way to directionally shoot power to directions where the accumulated distribution is large.

Hemicube as done by Immel et al.[20], as spherical harmonics as done in [5, 45], or in hemispherical tables as done in [14, 36, 42]. The Monte Carlo method could be used by generating outgoing power rays according to the shape of the unemitted power function as shown in Figure 6. My experience has been that non-diffuse radiosity does not work well for near mirrors because more zones are needed (the surfaces have detail visible in them) and each zone needs a larger table to represent the complicated outgoing power distribution.

4 Monte Carlo Integration

In this section the basic Monte Carlo solution methods for definite integrals are outlined. These techniques are then straightforwardly applied to certain integral problems. All of the basic material of this section is also covered in several of the classic Monte Carlo texts. This section differs by being geared toward classes of problems that crop up in Computer Graphics. Readers interested in a broader treatment of Monte Carlo techniques should consult one of the classic Monte Carlo texts[16, 44, 15, 57].

From Section 2 we saw that for a function f and a random variable $x \sim p$, we can approximate the expected value of $f(x)$ by a sum:

$$E(f(x)) = \int_{x \in S} f(x)p(x)d\mu_x \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{5}$$

Because the expected value can be expressed as an integral, the integral is also approximated by the sum. The form of Equation 5 is a bit awkward; we would usually like to approximate an integral of a single function g rather than a product fp. We can get around this by substituting $g = fp$ as the integrand:

$$\int_{x \in S} g(x) d\mu_x \approx \frac{1}{N} \sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)} \tag{6}$$

For this formula to be valid, p must be positive where g is nonzero.

So to get a good estimate, we want as many samples as possible, and we want the g/p to have a low variance (similar shape). Choosing p intelligently is called importance sampling, because if p is large where g is large, there will be more samples in important regions. Equation 5 also shows the fundamental problem with Monte Carlo integration: *diminishing return*. Because the variance of the estimate is proportional to $1/N$, the standard deviation is proportional to $1/\sqrt{N}$. Since the error in the estimate behaves similarly to the standard deviation, we will need to quadruple N to halve the error.

Another way to reduce variance is to partition S, the domain of the integral, into several smaller domains S_i, and evaluate the integral as a sum of integrals over the S_i. This is called stratified sampling. Normally only one sample is taken in each S_i (with density p_i), and in this case the variance of the estimate is:

$$var\left(\sum_{i=1}^{N} \frac{g(x_i)}{p_i(x_i)}\right) = \sum_{i=1}^{N} var\left(\frac{g(x_i)}{p_i(x_i)}\right) \tag{7}$$

The most common example of stratified sampling in graphics is jittering for pixel sampling[11].

As an example of the Monte Carlo solution of an integral I set $g(x)$ to be x over the interval (0, 4):

$$I = \int_0^4 x\, dx = 8 \tag{8}$$

The great impact of the shape of the function p on the variance of the N sample estimate is shown in Table 1. Note that the variance is lessened when the shape of p is similar to the shape of g. The variance drops to zero if $p = g/I$, but I is not usually known or we would not have to resort to Monte Carlo. One important principle illustrated in Table 1 is that stratified sampling is often *far* superior to importance sampling. Although the variance for this stratification on I is inversely proportional to the cube of the number of samples, there is no general result for the behavior of variance under stratification. There are some functions where stratification does no good. An example is a white noise function, where the variance is constant for all regions. On the other hand, most functions will benefit from stratified sampling because the variance in each subcell will usually be smaller than the variance of the entire domain.

4.1 Quasi-Monte Carlo Integration

Although distribution ray tracing is usually phrased as an application of Equation 6, many researchers replace the ξ_i with more evenly distributed (quasi-random) samples (e.g. [10, 30]). This approach can be shown to be sound by analyzing decreasing error in terms of some discrepancy measure[58, 56, 30, 39] rather than in terms of variance. However, it is often convenient to develop a sampling strategy using variance analysis on random samples, and then to turn around and use non-random,

method	*sampling function*	*variance*	*samples needed for standard error of 0.008*
importance	$(6-x)/(16)$	$56.8N^{-1}$	887,500
importance	$1/4$	$21.3N^{-1}$	332,812
importance	$(x+2)/16$	$6.3N^{-1}$	98,437
importance	$x/8$	0	1
stratified	uniform	$21.3N^{-3}$	70

Table 1: Variance for Monte Carlo Estimate of $\int_0^4 x\,dx$

but equidistributed samples in an implementation. This approach is almost certainly correct, but its justification and implications have yet to be explained.

For example, when evaluating a one dimensional integral on $[0, 1]$ we could use a set of N uniformly random sample points $(x_1, x_2, \cdots, x_N)$ on [0, 1] to get an approximation:

$$\int_0^1 f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

Interestingly, we can replace the points $(x_1, x_2, \cdots, x_N)$ with a set of non-random points $(y_1, y_2, \cdots, y_N)$, and the approximation will still work. If the points are too regular, then we will have aliasing, but having correlation between the points (e.g. using one dimension Poisson disk sampling), does not invalidate the estimate (merely the Monte Carlo argument used to justify the approximation!). In some sense, this quasi-Monte Carlo method can be thought of as using the equidistributed points to estimate the height of f. This does not fit in with the traditional quadrature approaches to numerical integration found in most numerical analysis texts (because these texts focus on one-dimensional problems), but is no less intuitive once you are used to the idea.

4.2 Multidimensional Monte Carlo Integration

Applying Equation 6 to multidimensional integrals is straightforward, except that choosing the multidimensional sampling points can be more involved than in the one dimensional case. More specifics on this can be found in Appendix A.

As an example in two dimensions, suppose we want to integrate some function f on the origin centered square $[-1, 1]^2$. This can be written down as a integral over one two dimensional variable $\mathbf{x}$:

$$I = \int_{\mathbf{x}\in[-1,1]^2} f(\mathbf{x})dA_{\mathbf{x}}$$

Applying Equation 6 to this gives us:

$$I \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}$$

Where each $\mathbf{p}_i$ is a two dimensional point distributed according to a two dimensional density p. We can convert to more explicit Cartesian coordinates and have a form we are probably more comfortable with:

$$I = \int_{y=-1}^{1}\int_{x=-1}^{1} f(x,y)dxdy \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i,y_i)}{p(x_i,y_i)}$$

This is really no different than the form above, except that we see the explicit components of $\mathbf{x}_i$ to be (x_i, y_i).

If our integral is over the disk of radius R, nothing really changes, except that the sample points must be distributed according to some density on the disk. This is why Monte Carlo integration is relatively easy: once the sample points are chosen, the application of the formula is always the same.

For a more complicated example, we look at the four dimensional integral for the form factor between two surfaces S_1 and S_2:

$$F_{12} = \frac{1}{A_1} \int_{\mathbf{x}_1 \in S_1} \int_{\mathbf{x}_2 \in S_2} \frac{g(\mathbf{x}_1, \mathbf{x}_2) \cos\theta_1 \cos\theta_2 dA_1 dA_2}{\pi ||\mathbf{x}_1 - \mathbf{x}_2||^2}$$

The sampling space is the four dimensional space $S_1 \times S_2$. A four dimensional point in this space is just an ordered pair $(\mathbf{x}_1, \mathbf{x}_2)$, where $\mathbf{x}_1$ is a point on S_1 and $\mathbf{x}_2$ is a point on S_2. The simplest way to proceed is to choose our four dimensional sample point as a pair of uniformly random points from each surface. The probability density function for this is the constant $1/(A_1 A_2)$, because $A_1 A_2$ is the four dimensional volume of the space, and this value just enforces Equation 3. If we use only one sample we have the estimate:

$$F_{12} \approx A_2 \frac{g(\mathbf{x}_1, \mathbf{x}_2) \cos\theta_1 \cos\theta_2}{\pi ||\mathbf{x}_1 - \mathbf{x}_2||^2}$$

A ray would be sent to evaluate the geometry term g. If many samples were taken, we could increase our accuracy. Notice that the shape of the surfaces was never explicitly used. This formula is valid whenever we have a method to choose random points from a shape!

4.3 Weighted Averages

We often have integrals that take the form of a strictly positive weighted average of a function:

$$I = \int_{x \in S} w(x) f(x) d\mu_x$$

where w is a weighting function with unit volume. To solve this by Equation 6, the optimal choice for the probability function is $p(x) = Cw(x)f(x)$, but as is often pointed out, this choice requires us to already know the value of I. Instead, people often either choose uniform p, or set $p(x) = w(x)$[10, 32, 27].

An example of a weighted average often used is pixel filtering. The color of a pixel $I(i, j)$ can be expressed as an integral:

$$I(i, j) = \int_{\mathbf{p} \in S} w(\mathbf{p}) L(\mathbf{p}) dA_{\mathbf{p}} \tag{9}$$

where $\mathbf{p}$ is a point on the viewport (or filmplane if a camera model is used), $L(\mathbf{p})$ is the radiance seen through the viewport at $\mathbf{p}$, and S is the non-zero region of the filter function w.

Rewriting with the assumption that the same origin-centered weighting function is used for every pixel yields the estimator :

$$I(i, j) \approx \frac{1}{N} \sum_{k=1}^{N} \frac{w(x_k, y_k) L(i + 0.5 + x_k, j + 0.5 + y_k)}{p(x_k, y_k)} \tag{10}$$

This assumes a coordinate system where a pixel (i, j) has unit area and is centered at $(i+0.5, j+0.5)$ as suggested by Heckbert[18].

Once a w is chosen for filtering, implementation is straightforward provided that w is strictly positive (as it must be if negative pixel colors are disallowed). But how do we choose *non-uniform* random points? As discussed in Appendix A, sample points can be chosen uniformly from $[0, 1]^2$ and then a warping transformation can be applied to distribute the points according to w[44, 39, 27].

For several practical and theoretical reasons[37] we have chosen the width 2 weighting function that is non-zero on $(x, y) \in [-1, 1]^2$:

$$w(x, y) = (1 - |x|)\,(1 - |y|) \tag{11}$$

We generate random points with density equal to w by applying a transformation to a uniform random pair $(r_1, r_2) \in [0, 1]^2$. The transformed sample point is just $(t(r_1), t(r_2))$, where the transformation function t is:

$$t(u) = \begin{cases} -0.5 + \sqrt{2u} & \text{if } u < 0.5 \\ 1.5 - \sqrt{2(1-u)} & \text{if } u \geq 0.5 \end{cases}$$

An important detail is that we do not really use uniform (r_1, r_2), but instead use jittered or an otherwise better distributed set of points. After warping, we still have a better than random distribution.

Another example of a weighted average is the radiance of a point $\mathbf{x}$ on a Lambertian surface:

$$L(\mathbf{x}) = \rho_d(\mathbf{x}) \int_{\text{incoming } \psi'} \frac{1}{\pi} L(\mathbf{x}, \psi') \cos\theta d\omega'$$

Where $L(\mathbf{x}, \psi')$ is the incoming radiance seen at point $\mathbf{x}$ coming from direction ψ'. Again, we might be more comfortable with the explicit form:

$$L(\mathbf{x}) = \rho_d(\mathbf{x}) \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\frac{\pi}{2}} \frac{1}{\pi} L(\theta, \phi) \cos\theta \sin\theta d\theta d\phi$$

The $\sin\theta$ term arises because the measure is solid angle (area on the unit sphere: $d\omega = dA = \sin\theta d\theta d\phi$). To solve this we just need to choose a random direction ψ to sample with a distribution according to the density function $\cos\theta/\pi$. This gives the estimator:

$$L(\mathbf{x}) = \rho_d(\mathbf{x}) L(\mathbf{x}, \psi)$$

This makes it easy to figure out the color of the ground in the midwest: it's the weighted average of the color of the sky times the reflectance of the ground!

4.4 Multidimensional Quasi-Monte Carlo Integration

Suppose we want to numerically estimate the value of an integral I on $[0, 1]^2$:

$$I = \int_0^1 \int_0^1 f(x, y) dx dy$$

For pure Monte Carlo we might use a set of uniform random points $(x_i, y_i) \in [0, 1]^2$ and estimate I to be the average of $f(x_i, y_i)$. For stratified sampling we might partition $[0^1]$ into several equal-area

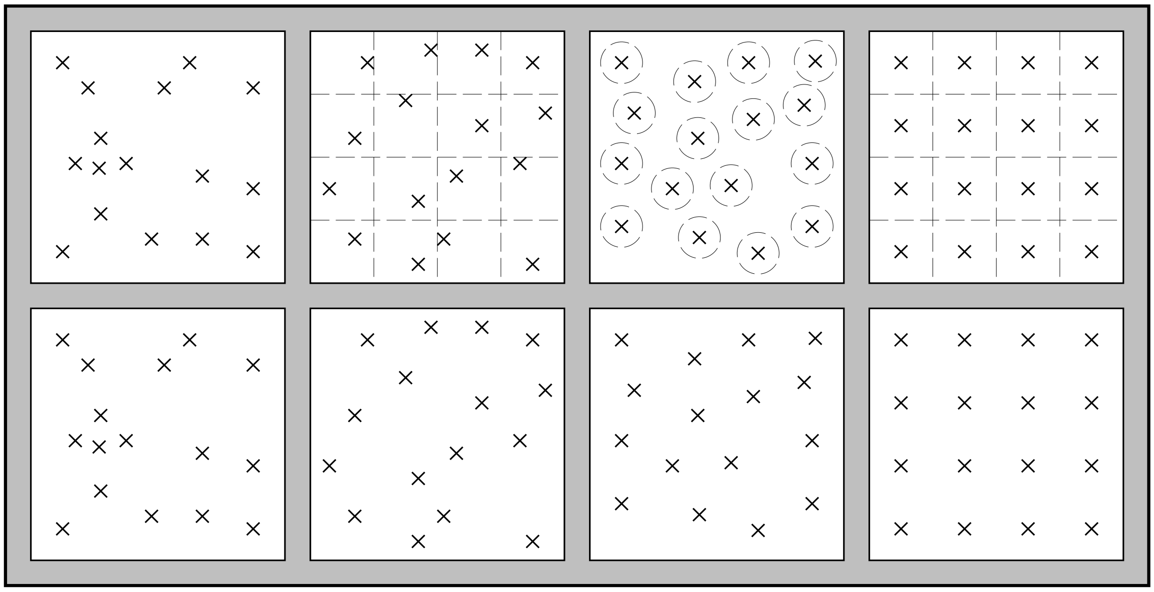

Figure 7: Random, Jittered, Dart-throwing, Regular.

rectangles and take one sample (x_i, y_i) in each rectangle and again average $f(x_i, y_i)$. Interestingly, we might also use "Poisson-disk" sampling to generate the points, or even just use points on a regular grid. No matter which of these point distributions (shown in Figure 7) we use, the estimate of I is the average of $f(x_i, y_i)$. Interestingly, only when we use one of the first two patterns are we doing Monte Carlo integration. With Poission-disk (dart-throwing), the samples are correlated, and in regular sampling they are deterministic.

As in the one-dimensional case, we can replace the random sample points with nay set of samples that are in some sense uniform, and this is just quasi-Monte carlo integration. There is a rich literature on this topic, but Mitchell has indicated that the graphics community will not be able to find many useful answers there, because the patterns that are used in that literature are deterministic, which causes aliasing in images[31].

4.5 Direct Lighting

In this section the famous direct lighting calculation is discussed. Even if radiosity is used, it can often be used only for the indirect component and the direct component can be done using the machinery of this section.

The rendering equation can be written down in two basic ways. It can be written down in terms of all directions visible to $\mathbf{x}$ (as in [20]):

$$L(\mathbf{x}, \psi) = \int_{\text{incoming } \psi'} \rho_{bd}(\mathbf{x}, \psi, \psi') L(\mathbf{x}, \psi') \cos\theta d\omega' \tag{12}$$

or it can written down as an integral over all surfaces (as in [22]):

$$L(\mathbf{x}, \psi) = \int_{\text{all } \mathbf{x}'} g(\mathbf{x}, \mathbf{x}') \rho_{bd}(\mathbf{x}, \psi, \psi') L(\mathbf{x}', \psi') \cos\theta \frac{dA' \cos\theta'}{\|\mathbf{x}' - \mathbf{x}\|^2} \tag{13}$$

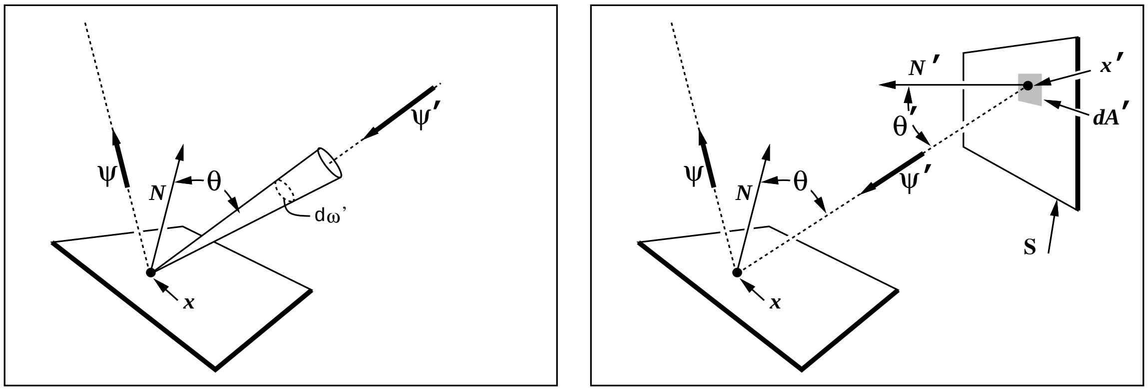

Figure 8: Definitions for the rendering equations.

When Equation 12 is used, we can view $\rho_{bd}(\mathbf{x}, \psi, \psi') \cos\theta$ as a weighting function and sample according to it. Because there is some energy absorbed by a surface, this gives us the estimator:

$$L(\mathbf{x}, \psi) \approx R(\mathbf{x}, \psi) L(\mathbf{x}, \xi) \tag{14}$$

where ξ is a random direction with density proportional to $\rho_{bd}(\mathbf{x}, \psi, \psi') \cos\theta$. The reflectivity term is simply:

$$R(\mathbf{x}, \psi) = \int_{\text{incoming } \psi'} \rho_{bd}(\mathbf{x}, \psi, \psi') \cos\theta d\omega$$

For an ideal specular surface, the ξ will always be the ideal reflection direction. For a dielectric, ξ can be chosen randomly between reflected and transmitted directions[4], or it can be split into two integrals as is done in a Whitted-style ray tracer[55]. For a diffuse surface, ξ will follow a cosine distribution: $p(\psi') = \cos\theta/\pi$.

When Equation 13 is used, the sampling takes place over all surfaces in the environment. In practice, only the direct lighting is calculated, so the integration space becomes all luminaire surfaces. This can be split into one integral for each surface[10], or can be viewed as a single sampling space[27, 43]. To simplify this discussion, we will assume only one luminaire, so the sampling space is just a single surface. Looking at Equation 13, an ideal estimator for diffuse luminaires would result if we sampled according to the density:

$$p(x') = Cg(\mathbf{x}, \mathbf{x}')\rho_{bd}(\mathbf{x}, \psi, \psi') \cos\theta \frac{\cos\theta'}{\|\mathbf{x}' - \mathbf{x}\|^2}$$

where C is a normalization constant. In practice, this isn't practical because the geometry term g and the BRDF ρ_{bd} can be very difficult to characterize. Instead, many researchers[11, 22] sample uniformly within the solid angle subtended by the luminaire, which yields:

$$p(x') = C' \frac{\cos\theta'}{\|\mathbf{x}' - \mathbf{x}\|^2} \tag{15}$$

Even this must only be approximated for polygonal luminaires[52], but can be exactly applied for spherical luminaires[24, 52]. If Equation 15 is used to choose points on the luminaire, then radiance can be estimated to be:

$$L(\mathbf{x}, \psi) \approx g(\mathbf{x}, \mathbf{x}')\rho_{bd}(\mathbf{x}, \psi, \psi') L(\mathbf{x}', \psi') \cos\theta\omega \tag{16}$$

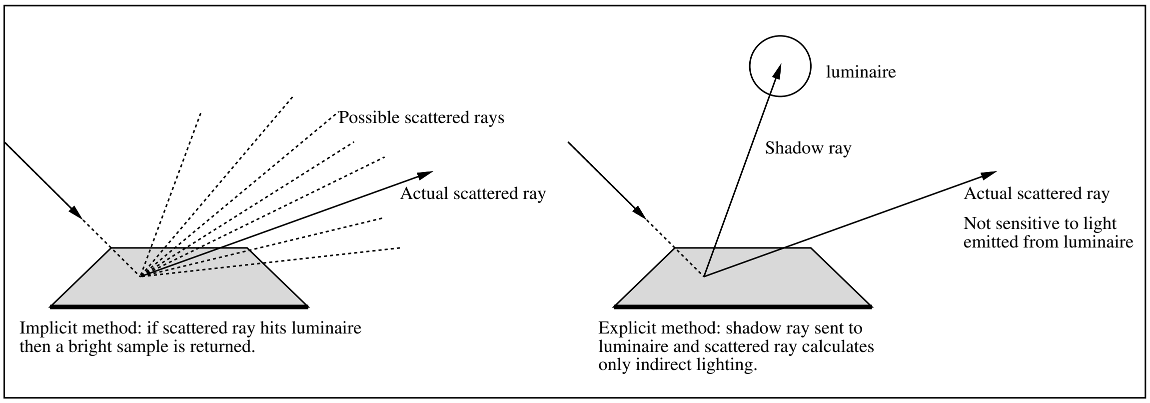

Figure 9: Implicit and explicit lighting calculation.

where ω is the total solid angle subtended by the luminaire as seen by $\mathbf{x}$.

We call the use of Equation 12 an *implicit* direct lighting calculation because any scattered ray that hits a luminaire will account for light from that luminaire. The use of Equation 13 is an *explicit* direct lighting calculation because each luminaire is explicitly queried using shadow rays (see Figure 9). Which should be used, an implicit or explicit direct lighting calculation? Clearly, the implicit method must be used for perfect mirrors, because that method implicitly evaluates the delta function BRDF. For a diffuse surface, the explicit method is usually used for direct lighting, and the implicit method is used only for indirect lighting[22, 54, 27]. To decide which method to use, variance should be analyzed, but the general rule is that specular surfaces should be dealt with using the implicit calculation and diffuse surfaces are treated explicitly.

If indirect lighting is to be added, then the surfaces that use the explicit direct lighting calculation can calculate indirect lighting implicitly with a scattered reflection ray[22]. This method, called *path tracing*, just recursively applies the direct lighting calculation and adds indirect lighting. If you implement this, be sure not to double count the indirect lighting!

5 Hybrid Methods

Many methods use some combination of view-dependent and view-independent methods. There are three basic approaches that have been used:

1. Generate a radiosity solution and view with ray tracing.
2. Generate a radiosity solution and use only for indirect lighting. Use ray tracing for direct lighting.
3. Generate a radiosity solution on a *low resolution* environment and use this in the viewing phase.

In method 1 the ray tracing is really just to accurately capture specular effects[51] and the radiosity phase my or may not include specular transport[29, 46] or directional diffuse transport[36, 42, 45]. Any problems with the meshing in high gradient areas will be very obvious in method 1, so some form of discontinuity meshing should be used[28].

In method 2 the fine detail caused by shadows (see Figure 10) is handled in the direct phase and the indirect lighting is handled by some precomputed values[38, 6, 25]. Ward's Radiance

Figure 10: Combined, indirect, and direct lighting. Note that the the sharp shading changes are in the direct component
.

Figure 11: Left: Low-resolution radiosity solution. Middle: Rushmeier solution. Right: Path tracing.

program[54, 53] is in the second family although the indirect information is calculated on the fly and cached, and the mesh is implicit.

In method 3, a zonal solution is carried out on a low-resolution version of the scene, and this is used as sources for gather phases at each pixel[34, 33, 26]. This in some sense is a generalization of the patch and elements approach[9]. The application of this technique and brute-force path tracing[22] is shown in Figure 11. The Rushmeier method ran eight times faster because it did not have to recursively fire rays. On complex scenes this advantage will only grow.

Currently I am using method 3 exclusively. I used to use method 2 (see [38]) but I found it cumbersome to have to mesh all diffuse objects. The beauty of method 3 is that it works even if the high resolution environment is an on-demand procedural model, it is easy to code, and that there are no smoothing issues. The radiosity solution does not have to look good! However, there are a number of open questions related to method 3:

- How should directional diffuse surfaces be handled?
- How should nearly specular surfaces be handled?
- How should caustics be handled?

- Should a hierarchy of various low-resolution environments be used?
- How should the low-resolution environment be created?

6 Conclusion

I hope that this tutorial has revealed the elegance and simplicity of Monte Carlo methods. This elegance and simplicity allow the modeling and solution of many problems with very few assumptions. However, these benefits come with the price of long execution times. If you need speed, use other techniques, or supplement Monte Carlo techniques with other methods. A good example of this combined strategy is Ward's *Radiance* program described in the conference procedings[53].

When designing new Monte Carlo methods, we usually think in terms of variance reduction. Work by Arvo and Kirk[4, 23] has detailed that this can be a non-trivial and sometimes counter intuitive process. To add to the confusion, we usually use quasi-random sampling, so the variance calculations are only an approximation. In the end I find that developing a theory using straight Monte Carlo assumptions, and then adding to it using intuition works the best for me. The most common heuristic I use is that every sample should do about the same amount of work. This is intuitively related to importance sampling, because the way to come close to this is to try to give every sample the same weight (load balancing).

The real key to a successful Monte Carlo method is the design of the probability density functions used to generate the samples. This is where your efforts should be concentrated. There is a tendency to think that your work is done (and the computer's starts!) once you have chosen to use a Monte Carlo method, but the very freedom to choose any density function dooms us to look for a *better* choice!

7 Acknowledgments

Many people have helped me understand the material in these notes, and an incomplete list of those I owe thanks to is Willie Hunt, Kelvin Sung, Frederick Jansen, Greg Ward, Claude Puech, William Kubitz, Don Mitchell, James Kajiya, Holly Rushmeier, Andrew Glassner, Kenneth Chiu, Dani Lischinski, Jim Arvo, John Wallace, Atul Jain, Randy Bramley, and Changyaw Wang. Special thanks to Paul Heckbert that found many errors in a previous version.

A Generating Random Numbers With Non-Uniform Densities

We often want to generate sets of random or pseudorandom points on the unit square for applications such as distribution ray tracing. There are several methods for doing this such as jittering and Poisson disk sampling. These methods give us a set of N reasonably equidistributed points on the unit square: (u_1, v_1) through (u_N, v_N).

Sometimes, our sampling space may not be square (e.g. a circular lens), or may not be uniform (e.g. a filter function centered on a pixel). It would be nice if we could write a mathematical transformation that would take our equidistributed points (u_i, v_i) as input, and output a set of points in our desired sampling space with our desired density. For example, to sample a camera lens, the transformation would take (u_i, v_i) and output (r_i, θ_i) such that the new points were approximately equidistributed on the disk of the lens.

If the density is a one dimensional $f(x)$ defined over the interval $x \in [x_{min}, x_{max}]$, then we can generate random numbers α_i that have density f from a set of uniform random numbers ξ_i, where $\xi_i \in [0, 1]$. To do this we need the probability distribution function $P(x)$:

$$Prob(\alpha < x) = P(x) = \int_{x_{min}}^{x} f(x')d\mu_{x'} \tag{17}$$

To get α_i we simply transform ξ_i:

$$\alpha_i = P^{-1}(\xi_i) \tag{18}$$

where P^{-1} is the inverse of P. If P is not analytically invertible then numerical methods will suffice because an inverse exists for all valid probability distribution functions.

For example, to choose random points x_i that have the density $p(x) = 3x^2/2$ on $[-1, 1]$, we see that $P(x) = (x^3+1)/2$, and $P^{-1}(x) = \sqrt[3]{2x-1}$, so we can "warp" a set of canonical random numbers $(\xi_1, \cdots, \xi_N)$ to the properly distributed numbers $(x_1, \cdots, x_N) = (\sqrt[3]{2\xi_1 - 1}, \cdots, \sqrt[3]{2\xi_N - 1})$. Of course, this same warping function can be used to transform "uniform" Poisson disk samples into nicely distributed samples with the desired density.

If we have a random variable $\alpha = (\alpha_x, \alpha_y))$ with two dimensional density (x, y) defined on $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ then we need the two dimensional distribution function:

$$Prob(\alpha_x < x \text{ and } \alpha_y < y) = F(x, y) = \int_{y_{min}}^{y} \int_{x_{min}}^{x} f(x', y')d\mu(x', y')$$

We first choose an x_i using the marginal distribution $F(x, y_{max})$, and then choose y_i according to $F(x_i, y)/F(x_i, y_{max})$. If $f(x, y)$ is separable (expressible as $g(x)h(y)$), then the one dimensional techniques can be used on each dimension.

For example, suppose we are sampling uniformly from the disk of radius R, so $p(r, \theta) = 1/(\pi R^2)$. The two dimensional distribution function is:

$$Prob(r < r_0 \theta < \theta_0) = F(r_0, \theta_0) = \int_0^{\theta_0} \int_0^{r_0} \frac{r dr d\theta}{\pi R^2} = \frac{\theta r^2}{2\pi R^2}$$

This means that a canonical pair (ξ_1, ξ_2) can be transformed to a uniform random point on the disk: $(r, \theta) = (R\sqrt{\xi_1}, 2\pi\xi_2)$.

To choose random points on a triangle defined by vertices p_0, p_1, and p_2, a more complicated analysis leads to the transformation $u = 1 - \sqrt{1 - \xi_1}$, $v = (1 - u)\xi_2$, and the random point p will is:

$$p = p_0 + u(p_1 - p_0) + v(p_2 - p_0).$$

To choose reflected ray directions for zonal calculations or distributed ray tracing, we can think of the problem as choosing points on the unit sphere or hemisphere (since each ray direction ψ can be expressed as a point on the sphere). For example, suppose that we want to choose rays according to the density:

$$p(\theta, \phi) = \frac{n+1}{2\pi} \cos^n \theta \tag{19}$$

Where n is a Phong-like exponent, θ is the angle from the surface normal and $\theta \in [0, \pi/2]$ (is on the upper hemisphere) and ϕ is the azimuthal angle ($\phi \in [0, 2\pi]$). The distribution function is:

$$P(\theta, \phi) = \int_0^{\phi} \int_0^{\theta} p(\theta', \phi') \sin\theta' d\theta' d\phi' \tag{20}$$

The $\cos\theta'$ term arises because on the sphere $d\omega = \cos\theta d\theta d\phi$. When the marginal densities are found, p (as expected) is separable and we find that a (ξ_1, ξ_2) pair of canonical random numbers can be transformed to a direction by:

$$(\theta, \phi) = (\arccos((1 - r_1)^{\frac{1}{n+1}}), 2\pi r_2)$$

One nice thing about this method is that a set of jittered points on the unit square can be easily transformed to a set of jittered points on the hemisphere with a distribution of Equation 19. If n is set to 1 then we have a diffuse distribution needed for a Monte Carlo zonal method.

For a zonal or ray tracing application, we choose a scattered ray with respect to to some unit normal vector $\vec{\mathbf{N}}$ (as opposed to the z axis). To do this we can first convert the angles to a unit vector a:

$$\vec{\mathbf{a}} = (\cos\phi\sin\theta, \sin\phi\sin\theta, \cos\theta)$$

We can then transform $\vec{\mathbf{a}}$ to be an $\vec{\mathbf{a}}'$ with respect to ψ by multiplying $\vec{\mathbf{a}}$ by a rotation matrix R ($\vec{\mathbf{a}}' = R\vec{\mathbf{a}}$). This rotation matrix is simple to write down:

$$R = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}$$

where $\vec{\mathbf{u}} = (u_x, u_y, u_z)$, $\vec{\mathbf{v}} = (v_x, v_y, v_z)$, $\vec{\mathbf{w}} = (w_x, w_y, w_z)$, form a basis (an orthonormal set of unit vectors where $\vec{\mathbf{u}} = \vec{\mathbf{v}} \times \vec{\mathbf{w}}$, $\vec{\mathbf{v}} = \vec{\mathbf{w}} \times \vec{\mathbf{u}}$, and $\vec{\mathbf{w}} = \vec{\mathbf{u}} \times \vec{\mathbf{v}}$) with the constraint that $\vec{\mathbf{w}}$ is aligned with $\vec{\mathbf{N}}$:

$$\vec{\mathbf{w}} = \frac{\vec{\mathbf{N}}}{|\vec{\mathbf{N}}|}$$

To get $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$, we need to find a vector $\vec{\mathbf{t}}$ that is not collinear with $\vec{\mathbf{w}}$. To do this simply set $\vec{\mathbf{t}}$ equal to $\vec{\mathbf{w}}$ and change the smallest magnitude component of $\vec{\mathbf{t}}$ to one. The $\vec{\mathbf{u}}$ and $\vec{\mathbf{v}}$ follow easily:

$$\vec{\mathbf{u}} = \frac{\vec{\mathbf{t}} \times \vec{\mathbf{w}}}{|\vec{\mathbf{t}} \times \vec{\mathbf{w}}|}$$

$$\vec{\mathbf{v}} = \vec{\mathbf{w}} \times \vec{\mathbf{u}}$$

As an efficiency improvement, you can avoid taking trigonometric functions of inverse trigonometric functions (e.g. $\cos\arccos\theta$). For example, when $n = 1$ (a diffuse distribution), the vector $\vec{\mathbf{a}}$ simplifies to

$$\vec{\mathbf{a}} = (\cos(2\pi\xi_1)\sqrt{\xi_2}, \sin(2\pi\xi_1)\sqrt{\xi_2}, \sqrt{1 - \xi_2})$$

References

[1] John M. Airey and Ming Ouh-young. Two adaptive techniques let progressive radiosity outperform the traditional radiosity algorithm. Technical Report TR89-20, University of North Carolina at Chapel Hill, August 1989.

[2] John M. Airey, John H. Rohlf, and Frederick P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics*, 24(1):41–50, 1990. ACM Workshop on Interactive Graphics Proceedings.

[3] James Arvo. Backward ray tracing. *Developments in Ray Tracing*, pages 259–263, 1985. ACM Siggraph '85 Course Notes.

[4] James Arvo and David Kirk. Particle transport and image synthesis. *Computer Graphics*, 24(3):63–66, August 1990. ACM Siggraph '90 Conference Proceedings.

[5] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflectance functions from surface bump maps. *Computer Graphics*, 21(4):273–282, July 1987. ACM Siggraph '87 Conference Proceedings.

[6] Shenchang Eric Chen, Holly Rushmeier, Gavin Miller, and Douglass Turner. A progressive multi-pass method for global illumination. *Computer Graphics*, 25(4):165–174, July 1991. ACM Siggraph '91 Conference Proceedings.

[7] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics*, 22(4):75–84, August 1988. ACM Siggraph '88 Conference Proceedings.

[8] Michael F. Cohen and Donald P. Greenberg. The hemi-cube: a radiosity solution for complex environments. *Computer Graphics*, 19(3):31–40, July 1985. ACM Siggraph '85 Conference Proceedings.

[9] Micheal F. Cohen, Donald P. Greenberg, David S. Immel, and Philip J. Brock. An efficient radioisty approach for realistic image synthesis. *IEEE Computer Graphics and Applications*, 6(2):26–35, 1986.

[10] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.

[11] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(4):165–174, July 1984. ACM Siggraph '84 Conference Proceedings.

[12] R. C. Corlett. Direct monte carlo calculation of radiative heat transfer in vacuum. *Journal of Heat Transfer*, pages 376–382, November 1966.

[13] Cindy M. Goral, Kenneth E. Torrance, and Donald P. Greenberg. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(4):213–222, July 1984. ACM Siggraph '84 Conference Proceedings.

[14] David Edward Hall. An analysis and modification of shao's radiosity method for computer graphics image synthesis. Master's thesis, Department of Mechanical Engineering, Georgia Institute of Technology, March 1990.

[15] John H. Halton. A retrospective and prospective of the monte carlo method. *SIAM Review*, 12(1):1–63, January 1970.

[16] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Wiley, New York, N.Y., 1964.

[17] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics*, 24(3):145–154, August 1990. ACM Siggraph '90 Conference Proceedings.

[18] Paul S. Heckbert. What are the coordinates of a pixel? In Andrew Glassner, editor, *Graphics Gems*. Academic Press, New York, NY, 1990.

[19] J. R. Howell and M. Perlmutter. Monte carlo solution of thermal transfer through radiant media between gray walls. *Journal of Heat Transfer*, pages 116–122, February 1964.

[20] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. *Computer Graphics*, 20(4):133–142, August 1986. ACM Siggraph '86 Conference Proceedings.

[21] Theodore M. Jenkins, Walter R. Nelson, and Alessandro Rindi, editors. *Monte Carlo Transport of Electrons and Photons*. Plenum Press, New York, N.Y., 1988.

[22] James T. Kajiya. The rendering equation. *Computer Graphics*, 20(4):143–150, August 1986. ACM Siggraph '86 Conference Proceedings.

[23] David Kirk and James Arvo. Unbiased sampling techniques for image sysnthesis. *Computer Graphics*, 25(4):153–156, July 1991. ACM Siggraph '91 Conference Proceedings.

[24] David Kirk and James Arvo. Unbiased variance reduction for global illumination. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[25] A. Kok and F. Jansen. Source selection for the direct lighting calculation in global illumination. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[26] Arjan F. Kok. Grouping of patches in progressive radiosity. In *Proceedings of the Fourth Eurographics Workshop on Rendering*, pages 221–231, 1993.

[27] Brigitta Lange. The simulation of radiant light transfer with stochastic ray-tracing. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[28] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Compining hierarchical radiosity and discontinuity meshing. *Computer Graphics*, pages 199–208, August 1993. ACM Siggraph '93 Conference Proceedings.

[29] Thomas J. V. Malley. A shading method for computer generated images. Master's thesis, University of Utah, June 1988.

[30] Don P. Mitchell. Spectrally optimal sampling for distribution ray tracing. *Computer Graphics*, 25(4), July 1991. Siggraph '91 Conference Proceedings.

[31] Don P. Mitchell. Ray tracing and irregularities of distribution. In *Proceedings of the Third Eurographics Workshop on Rendering*, pages 61–70, 1992.

[32] Werner Purgathofer. A statistical method for adaptive stochastic sampling. *Computers & Graphics*, 11(2):157–162, 1987.

[33] Holly Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. In *Graphics Interface '93*, pages 227–236, May 1993.

[34] Holly E. Rushmeier. *Realistic Image Synthesis for Scenes with Radiatively Participating Media*. PhD thesis, Cornell University, May 1988.

[35] Holly E. Rushmeier and Kenneth E. Torrance. Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Transaction on Graphics*, 9(1):1–27, January 1990.

[36] Bertrand Le Saec and Christophe Schlick. A progressive ray-tracing-based radiosity with general reflectance functions. In *Proceedings of the Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 103–116, June 1990.

[37] Peter Shirley. *Physically Based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, November 1990.

[38] Peter Shirley. A ray tracing algorithm for global illumination. In *Graphics Interface '90*, pages 205–212, May 1990.

[39] Peter Shirley. Discrepancy as a quality measure for sampling distributions. In *Eurographics '91*, pages 183–193, September 1991.

[40] Peter Shirley. Radiosity via ray tracing. In James Arvo, editor, *Graphics Gems 2*. Academic Press, New York, NY, 1991.

[41] Peter Shirley. Time complexity of monte carlo radiosity. In *Eurographics '91*, pages 459–466, September 1991.

[42] Peter Shirley, Kelvin Sung, and William Brown. A ray tracing framework for global illumination. In *Graphics Interface '91*, pages 117–128, June 1991.

[43] Peter Shirley and Changyaw Wang. Direct lighting by monte carlo integration. In *Proceedings of the Second Eurographics Workshop on Rendering*, 1991.

[44] Y. A. Shreider. *The Monte Carlo Method*. Pergamon Press, New York, N.Y., 1966.

[45] Francois Sillion, James Arvo, Stephen Westin, and Donald Greenberg. A global illumination algorithm for general reflection distributions. *Computer Graphics*, 25(4):187–196, July 1991. ACM Siggraph '91 Conference Proceedings.

[46] Francois Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. *Computer Graphics*, 23(3):335–344, July 1989. ACM Siggraph '89 Conference Proceedings.

[47] Jerome Spanier and Ely M. Gelbard. *Monte Carlo Principles and Neutron Transport Problems*. Addison-Wesley, New York, N.Y., 1969.

[48] Jerome Spanier and Earl H. Maize. Quasi-random methods for estimating integrals using relatively small samples. *SIAM Review*, 36(1):18–44, March 1994.

[49] Dan Stanger. Monte carlo procedures in lighting design. *Journal of the Illumination Engineering Society*, pages 14–25, July 1984.

[50] J. S. Toor and R. Viskanta. A numerical experiment of radiant heat interchange by the monte carlo method. *International Journal of Heat and Mass Transfer*, 11:883–897, 1968.

[51] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A two-pass solution to the rendering equation: a synthesis of ray tracing and radiosity methods. *Computer Graphics*, 21(4):311–320, July 1987. ACM Siggraph '87 Conference Proceedings.

[52] Changyaw Wang. Physically correct direct lighting for distribution ray tracing. In David Kirk, editor, *Graphics Gems 3*. Academic Press, New York, NY, 1992.

[53] Gregory J. Ward. The radiance lighting simulation and rendering system. *Computer Graphics*, 28(2), July 1994. ACM Siggraph '94 Conference Proceedings.

[54] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics*, 22(4):85–92, August 1988. ACM Siggraph '88 Conference Proceedings.

[55] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[56] H. Wozniakowski. Average case complexity of multivariate integration. *Bulliten (New Series) of the American Mathematical Society*, 24(1):185–193, January 1991.

[57] Sidney J. Yakowitz. *Computational Probability and Simulation*. Addison-Wesley, New York, N.Y., 1977.

[58] S. K. Zeremba. The mathematical basis of monte carlo and quasi-monte carlo methods. *SIAM Review*, 10(3):303–314, July 1968.