Monte Carlo Ray Tracing

Siggraph 2003 Course 44

Tuesday, July 29, 2003

Organizer

Henrik Wann Jensen
University of California, San Diego

Lecturers

James Arvo
University of California, Irvine

Phil Dutre
Katholieke Universiteit Leuven

Alexander Keller
Universität Kaiserslautern

Henrik Wann Jensen
University of California, San Diego

Art Owen
Stanford University

Matt Pharr
NVIDIA

Peter Shirley
University of Utah

Abstract

This full day course will provide a detailed overview of state of the art in Monte Carlo ray tracing. Recent advances in algorithms and available compute power have made Monte Carlo ray tracing based methods widely used for simulating global illumination. This course will review the fundamentals of Monte Carlo methods, and provide a detailed description of the theory behind the latest techniques and algorithms used in realistic image synthesis. This includes path tracing, bidirectional path tracing, Metropolis light transport, irradiance caching and photon mapping.

Course Syllabus

8:30 Introduction and Welcome
Henrik Wann Jensen

8:40 Why Monte Carlo Ray Tracing?
Peter Shirley

The global illumination problem
Monte Carlo as a general tool

9:00 Fundamentals of Monte Carlo Integration
Peter Shirley

An overview of Monte Carlo integration techniques.
Rejection methods
Importance sampling
Stratified sampling Hammersley points
Arbitrary-edge discrepancy
Applications of quasirandom techniques to distribution ray tracing
Spectral sampling techniques

9:30 A Recipe for Sampling Algorithms
James Arvo

Sampling of special geometries and reflection models
Russian roulette

10:00 Break

10:30 Direct Illumination
Peter Shirley
Sampling of light sources
Special types of light sources
Efficient sampling of many light sources

11:00 Variance Reduction Techniques
James Arvo

Combined estimators
Hybrid sampling methods.

11:00 Quasi-Monte Carlo Techniques

Art Owen

Good sample distributions
Efficient sampling of high-dimensional spaces

12:00 Lunch

2:15 The Rendering Equation

Philip Dutre

The path integral formulation
Path tracing
Russian Roulette
Adjoint techniques
Bidirectional transport
Bidirectional path tracing

3:00 Quasi-Monte Carlo Techniques II

Alexander Keller

Algorithms for (randomized) quasi-Monte Carlo sample points
Variance reduction by quasi-Monte Carlo points
Benefits of correlated sampling
How to boost a ray tracer by quasi-Monte Carlo

4:00 Break

4:15 Metropolis Sampling

Matt Pharr

One dimensional setting
Motion blur
Metropolis light transport

5:00 Biased Monte Carlo Ray Tracing

Henrik Wann Jensen

Biased vs. consistent methods
Filtering Techniques
Irradiance Caching
Photon Mapping

6:00 Conclusion and Questions

Contents

1 Introduction *by Henrik Wann Jensen* **11**

1.1 Purpose of this Course 12
1.2 Prerequisites 12
1.3 Acknowledgements 12

2 Fundamentals of Monte Carlo Integration *by Peter Shirley* **13**

2.1 Background and Terminology 13
2.1.1 One-dimensional Continuous Probability Density Functions 13
2.1.2 One-dimensional Expected Value 14
2.1.3 Multi-dimensional Random Variables 15
2.1.4 Variance 16
2.1.5 Estimated Means 17
2.2 Monte Carlo Integration 17
2.2.1 Quasi-Monte Carlo Integration 19
2.2.2 Multidimensional Monte Carlo Integration 20
2.3 Choosing Random Points 21
2.3.1 Function inversion 22
2.3.2 Rejection 24
2.3.3 Metropolis 25
2.4 Monte Carlo Simulation 27
2.5 Density Estimation 28

3 Direct Lighting via Monte Carlo Integration *by Peter Shirley* **29**

3.1 Mathematical framework 29
3.2 Sampling a spherical luminaire 31
3.3 Non-diffuse Luminaries 34
3.4 Direct Lighting from Many Luminaires 35

3.4.1 Constant α_i 36
3.4.2 Linear α_i 37

4 Stratified Sampling of 2-Manifolds *by Jim Arvo* **39**
4.1 Introduction 39
4.2 A Recipe for Sampling Algorithms 41
4.3 Analytic Area-Preserving Parametrizations 46
4.3.1 Sampling Planar Triangles 46
4.3.2 Sampling the Unit Disk 47
4.3.3 Sampling the Unit Hemisphere 48
4.3.4 Sampling a Phong Lobe 49
4.3.5 Sampling Spherical Triangles 49
4.4 Sampling Projected Spherical Polygons 54
4.4.1 The Cumulative Marginal Distribution 56
4.4.2 The Sampling Algorithm 57

5 Combining Sampling Strategies *by Jim Arvo* **63**
5.1 Introduction 63
5.2 Using Multiple PDFs 64
5.3 Possible Weighting Functions 66
5.4 Obvious is also Nearly Optimal 68

6 Quasi-Monte Carlo Sampling *by Art B. Owen* **69**
6.1 Crude Monte Carlo 70
6.2 Stratification 72
6.3 Multiple Stratification 74
6.4 Uniformity and Discrepancy 77
6.5 Digital Nets and Related Methods 80
6.6 Integration Lattices 84
6.7 Randomized Quasi-Monte Carlo 86
6.8 Padding and Latin Supercube Sampling 87

7 Monte Carlo Path Tracing *by Pat Hanrahan* **89**
7.1 Solving the Rendering Equation 89
7.2 Monte Carlo Path Tracing 96
7.3 Random Walks and Markov Chains 98
7.4 Adjoint Equations and Importance Sampling 103

8 The Rendering Equation and Path Tracing *by Philip Dutre* **109**
8.1 Formulations of the rendering equation 109
8.2 Importance function 111
8.3 Path formulation 114
8.4 Simple stochastic ray tracing 115
8.5 Russian Roulette 115
8.6 Indirect Illumination 116
8.6.1 Hemisphere sampling 117
8.6.2 Importance sampling 118
8.6.3 Overview 118

9 Metropolis Sampling *by Matt Pharr* **121**
9.1 Overview 123
9.1.1 Detailed Balance 123
9.1.2 Expected Values 125
9.2 One Dimensional Setting 126
9.2.1 Mutations and Transition Functions 127
9.2.2 Start-up bias 128
9.2.3 Initial Results 129
9.2.4 Ergodicity 131
9.2.5 Mutations via PDFs 131
9.3 Motion Blur 133
9.3.1 Basic Mutation Strategies 135
9.3.2 Re-normalization 139
9.3.3 Adapting for large variation in f 139
9.3.4 Color 141
9.4 Metropolis Light Transport 143
9.4.1 Path Mutations 144
9.4.2 Pixel Stratification 146
9.4.3 Direct Lighting 147
9.4.4 Participating Media 147

10 Biased Techniques *by Henrik Wann Jensen* **149**
10.1 Biased vs. Unbiased 149
10.2 Filtering Techniques 151
10.3 Adaptive Sampling 153

10.4 Irradiance Caching 153
10.5 Photon Mapping 156
10.5.1 Pass 1: Photon Tracing 156
10.5.2 The Radiance Estimate 157
10.5.3 Pass 2: Rendering 159

References **162**

Chapter 1

Introduction

By Henrik Wann Jensen

Realistic image synthesis is increasingly important in areas such as entertainment (movies, special effects and games), design, architecture and more. A common trend in all these areas is to request more realistic images of increasingly complex models. Monte Carlo ray tracing based techniques are the only methods that can handle this complexity. Recent advances in algorithms and compute power has made Monte Carlo ray tracing the natural choice for most problems. This is a significant change from just a few years back when the (finite element) radiosity method was the prefered algorithm for most graphics researchers.

Monte Carlo ray tracing has several advantages over finite element methods. A recent list from [33] includes:

- Geometry can be procedural
- No tessellation is necessary
- It is not necessary to precompute a representation for the solution
- Geometry can be duplicated using instancing
- Any type of BRDF can be handled
- Specular reflections (on any shape) are easy
- Memory consumption is low
- The accuracy is controlled at the pixel/image level

- Complexity has empirically been found to be $O(\log N)$ where N is number of scene elements. Compare this with $O(N \log N)$ for the fastest finite element methods [12].

In addition one might add that Monte Carlo ray tracing methods can be very easy to implement. A basic path tracing algorithm which has all of the above advantages is a relatively straightforward extension to ray tracing.

The main problem with Monte Carlo ray tracing is variance seen as noise in the rendered images. This noise can be eliminated by using more samples. Unfortunately the convergence of Monte Carlo methods is quite slow, and a large number of samples can be necessary to reduce the variance to an acceptable level. Another way of reducing variance is to try to be more clever; a large part of this course material is devoted to techniques and algorithms for making Monte Carlo ray tracing more efficient.

1.1 Purpose of this Course

The purpose of this course is to impart upon the attendies a thorough understanding of the principles of Monte Carlo ray tracing methods, as well as a detailed overview of the most recently developed methods.

1.2 Prerequisites

The reader is expected to have a good working knowledge of ray tracing and to know the basics of global illumination. This includes knowledge of radiometric terms (such as radiance and flux) and knowledge of basic reflection models (such as diffuse, specular and glossy).

1.3 Acknowledgements

Funding for the authors of these notes include DARPA DABTB63-95-C0085 and an NSF Career Award (CCR9876332).

Chapter 2

Fundamentals of Monte Carlo Integration

By Peter Shirley

This Chapter discusses *Monte Carlo integration*, where random numbers are used to approximate integrals. First some basic concepts from probability are reviewed, and then they are applied to numerically estimate integrals. The problem of estimating the direct lighting at a point with arbitrary lighting and reflection properties is then discussed. The next Chapter applies Monte Carlo integration to the direct light problem in ray tracing.

2.1 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* (pdf), *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).

2.1.1 One-dimensional Continuous Probability Density Functions

Loosely speaking, a *continuous random variable* x is a scalar or vector quantity that "randomly" takes on some value from the real line $\mathbb{R} = (-\infty, +\infty)$. 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 $x \sim p$). The probability that x will take on a value in some interval $[a, b]$ is given by the integral:

$$\text{Probability}(x \in [a, b]) = \int_a^b p(x)dx. \tag{2.1}$$

Loosely speaking, the probability density function p 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:

$$p(x) \geq 0 \quad \text{(Probability is nonnegative)}, \tag{2.2}$$

$$\int_{-\infty}^{+\infty} p(x)dx = 1 \quad (\text{Probability}(x \in \mathbb{R}) = 1). \tag{2.3}$$

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

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

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 $[a, b] \in [0, 1)$ is:

$$\text{Probability}(a \leq \xi \leq b) = \int_a^b 1dx = b - a.$$

2.1.2 One-dimensional Expected Value

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

$$E(f(x)) = \int f(x)p(x)dx.$$

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:

$$E(x + y) = E(x) + E(y),$$

for random variables x and y. Because functions of random variables are themselves random variables, this linearity of expectation applies to them as well:

$$E(f(x) + g(y)) = E(f(x)) + E(g(y)).$$

An obvious question is whether this property holds if the random variables being summed are correlated (variables that are not correlated are called *independent*). This linearity property in fact does hold *whether or not* the variables are independent! This summation property is vital for most Monte Carlo applications.

2.1.3 Multi-dimensional Random Variables

The discussion of random variables and their expected values extends naturally to multidimensional spaces. Most graphics problems will be in such higher-dimensional spaces. For example, many lighting problems are phrased on the surface of the hemisphere. Fortunately, if we define a measure μ on the space the random variables occupy, everything is very similar to the one-dimensional case. Suppose the space S has associated measure μ, for example S is the surface of a sphere and μ measures area. We can define a pdf $p : S \mapsto \mathbb{R}$, and if x is a random variable with $x \sim p$, then the probability that x will take on a value in some region $S_i \subset S$ is given by the integral:

$$\text{Probability}(x \in S_i) = \int_{S_i} p(x) d\mu \tag{2.4}$$

Here Probability$($*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.

In graphics S is often an area ($d\mu = dA = dxdy$), or a set of directions (points on a unit sphere: $d\mu = d\omega = \sin\theta d\theta d\phi$). As an example, a two dimensional random variable α is a uniformly distributed random variable on a disk of radius R. Here *uniformly* means uniform with respect to area, e.g., the way a bad dart player's hits would be distributed on a dart board. Since it is uniform, we know that $p(\alpha)$ is some constant. From Equation 2.3, and the fact that area is the appropriate measure, we can deduce that $p(\alpha) = 1/(\pi R^2)$. This means that the probability that α is in a certain subset S_1 of the disk is just:

$$\text{Probability}(\alpha \in S_1) = \int_{S_1} \frac{1}{\pi R^2} dA.$$

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 $r < R/2$. Note that just because α is uniform does not imply that $theta$ or r are necessarily uniform (in fact, $theta$ is, and r is not uniform). The differential area dA becomes $r\, dr\, d\theta$. This leads to:

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

The formula for expected value of a real function applies to the multidimensional case:

$$E(f(x)) = \int_S f(x) p(x) d\mu,$$

Where $x \in S$ and $f : S \mapsto \mathbb{R}$, and $p : S \mapsto \mathbb{R}$ For example, on the unit square $S = [0,1] \times [0,1]$ and $p(x,y) = 4xy$, the expected value of the x coordinate for $(x,y) \sim p$ is:

$$\begin{aligned} E(x) &= \int_S f(x,y) p(x,y) dA \\ &= \int_0^1 \int_0^1 4x^2 y\, dx\, dy \\ &= \frac{2}{3} \end{aligned}$$

Note that here $f(x,y) = x$.

2.1.4 Variance

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

$$V(x) \equiv E([x - E(x)]^2).$$

Some algebraic manipulation can give the non-obvious expression:

$$V(x) = E(x^2) - [E(x)]^2 .$$

The expression $E([x - E(x)]^2)$ is more useful for thinking intuitively about variance, while the algebraically equivalent expression $E(x^2) - [E(x)]^2$ 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 expected absolute deviation from the expected value.

2.1.5 Estimated Means

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

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

As N increases, the variance of this estimate decreases. We want N to be large enough that we have confidence that the estimate is "close enough". However, there are no sure things in Monte Carlo; we just gain statistical confidence that our estimate is good. To be sure, we would have to have $n = \infty$. This confidence is expressed by *Law of Large Numbers*:

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

2.2 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 [27, 72, 26, 98].

As discussed earlier, given a function $f : S \mapsto \mathbb{R}$ and a random variable $x \sim p$, we can approximate the expected value of $f(x)$ by a sum:

$$E(f(x)) = \int_{x \in S} f(x)p(x)d\mu \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{2.5}$$

Because the expected value can be expressed as an integral, the integral is also approximated by the sum. The form of Equation 2.5 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:

$$\int_{x \in S} g(x)d\mu \approx \frac{1}{N}\sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)}. \tag{2.6}$$

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 (g and p should have a 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 2.5 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:

$$var\left(\sum_{i=1}^{N} \frac{g(x_i)}{p_i(x_i)}\right) = \sum_{i=1}^{N} var\left(\frac{g(x_i)}{p_i(x_i)}\right). \tag{2.7}$$

It can be shown that the variance of stratified sampling is never higher than unstratified if all strata have equal measure:

$$\int_{S_i} p(x)d\mu = \frac{1}{N}\int_{S} p(x)d\mu.$$

The most common example of stratified sampling in graphics is jittering for pixel sampling [14].

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	$1/4$	$21.3N^{-3}$	70

Table 2.1: Variance for Monte Carlo Estimate of $\int_0^4 x\,dx$

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

$$I = \int_0^4 x\,dx = 8. \tag{2.8}$$

The great impact of the shape of the function p on the variance of the N sample estimates is shown in Table 2.1. 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 2.1 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.

2.2.1 Quasi-Monte Carlo Integration

Although distribution ray tracing is usually phrased as an application of Equation 2.6, many researchers replace the ξ_i with more evenly distributed (quasi-random) samples (e.g. [13, 53]). This approach can be shown to be sound by analyzing decreasing error in terms of some discrepancy measure [99, 97, 53, 67] 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, 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.

The relative advantages of Monte Carlo versus QMC for graphics is still an open question. QMC does have better convergence subject to certain conditions, but these conditions are often not true in graphics. Also, there are rarely enough samples taken in practice for the asymptotic analysis to apply. To further complicate matters, QMC sometimes produces aliasing. However, this aliasing is sometimes not visually objectionable in often looks better than the noise produced by traditional Monte Carlo. For more information on this topic, see the recent work of Alexander Keller.

2.2.2 Multidimensional Monte Carlo Integration

Applying Equation 2.6 to multidimensional integrals is straightforward, except that choosing the multidimensional sampling points can be more involved than in the one dimensional case.

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 a single two dimensional variable x:

$$I = \int_{[-1,1]^2} f(x)dA.$$

Applying Equation 2.6 to this gives us:

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)},$$

where each x_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:

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

This is really no different than the form above, except that we see the explicit components of x_i to be (x_i, y_i).

If our integral is over the 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.

2.3 Choosing Random Points

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.

There are several ways to generate such non-uniform points, and this section reviews the three most often used: function inversion, rejection, and Metropolis.

2.3.1 Function inversion

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 cumulative probability distribution function $P(x)$:

$$\text{Probability}(\alpha < x) = P(x) = \int_{x_{min}}^{x} f(x')d\mu \tag{2.9}$$

To get α_i we simply transform ξ_i:

$$\alpha_i = P^{-1}(\xi_i) \tag{2.10}$$

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:

$$Prob(r < r_0 \text{ and } \theta < \theta_0) = F(r_0, \theta_0) = \int_{0}^{\theta_0} \int_{0}^{r_0} \frac{r dr d\theta}{\pi R^2} = \frac{\theta r^2}{2\pi R^2}$$

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:

$$p(\theta, \phi) = \frac{n+1}{2\pi} \cos^n \theta \tag{2.11}$$

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:

$$P(\theta, \phi) = \int_0^{\phi} \int_0^{\theta} p(\theta', \phi') \sin \theta' d\theta' d\phi' \tag{2.12}$$

The $\sin \theta'$ term arises because on the sphere $d\omega = \sin \theta d\theta d\phi$. 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 2.11. 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 some unit normal vector $\vec{N}$ (as opposed to the z axis). To do this we can first convert the angles to a unit vector a:

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

We can then transform $\vec{a}$ to be an $\vec{a}'$ with respect to ψ by multiplying $\vec{a}$ by a rotation matrix R ($\vec{a}' = R\vec{a}$). 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 $\vec{u} = (u_x, u_y, u_z)$, $\vec{v} = (v_x, v_y, v_z)$, $\vec{w} = (w_x, w_y, w_z)$, form a basis (an orthonormal set of unit vectors where $\vec{u} = \vec{v} \times \vec{w}$, $\vec{v} = \vec{w} \times \vec{u}$, and $\vec{w} = \vec{u} \times \vec{v}$) with the constraint that $\vec{w}$ is aligned with $\vec{N}$:

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

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

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

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

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 $\vec{a}$ simplifies to

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

2.3.2 Rejection

A *rejection* method chooses points according to some simple distribution and rejects some of them so that they are in a more complex distribution. There are several scenarios where rejection is used, and we show several of these by example.

Suppose we want uniform random points within the unit circle. We can first choose uniform random points $(x, y) \in [-1, 1]^2$ and reject those outside the circle. If the function $r()$ returns a canonical random number, then the procedure for this is:

```
done = false
while (not done)
    x = -1 + 2*r()
    y = -1 + 2*r()
    if (x*x + y*y < 1)
       done = true
end while
```

If we want a random number $x \sim p$ and we know that $p : [a, b] \mapsto \mathbb{R}$, and that for all x, $p(x) < m$, then we can generate random points in the rectangle $[a, b] \times [0, m]$ and take those where $y < p(x)$:

```
done = false
while (not done)
    x = a + r()*(b-a)
    y = r()*m
    if (y <  p(x))
       done = true
end while
```

A variant of the last methods is common because we can often deal more easily with boxes than spheres. To pick a random unit vector with uniform directional distribution, we first pick a random point in the unit sphere and then treat that point as a direction vector by taking the unit vector in the same direction:

```
done = false
while (not done)
    x = -1 + 2*r()
    y = -1 + 2*r()
    z = -1 + 2*r()
    if ((length2 =x*x + y*y +z*z) <  1)
       done = true
end while
length = sqrt(length2)
x /= length
y /= length
z /= length
```

2.3.3 Metropolis

The *Metropolis* method uses random *mutations* to produce a set of samples with a desired density. This concept is used extensively in the *Metropolis Light Transport* algorithm described later in Chapter 9. Suppose we have a random point x_0 in a domain S. Further, suppose for any point x we have a way to generate random $y \sim p_x$. We use the marginal notation $p_x(y) \equiv p(x \rightarrow y)$ to denote this density function. Now suppose we let x_1 be a random point in S selected with underlying density $p(x_0 \to x_1)$. We generate x_2 with density $p(x_1 \to x_0)$ and so in. In the limit where we generate an infinite number of samples, it can be proven that the samples with have some underlying density determined by p regardless of the initial point x_0.

Now suppose we want to chose p such that the underlying density of samples we converge to is proportional to a function $f(x)$ where f is a non-negative function with domain S. Further, suppose we can evaluate f but we have little or no additional knowledge about its properties (such functions are common in graphics). Also suppose we have the ability to make "transitions" from x_i to x_{i+1} with underlying density function $t(x_i \to x_{i+1})$. To add flexibility, further suppose we add the potentially non-zero probability that x_i transitions to itself, i.e., $x_{i+1} = x_i$. We phrase this as generating a potential candidate $y \sim t(x_i \to y)$ and "accepting" this candidate (i.e., $x_{i+1} = y$) with probability $a(x_i \to y)$ and rejecting it (i.e., $x_{i+1} = x_i$) with probability $1 - a(x_i \to y)$. Note that the sequence $x_0, x_1, x_2, \ldots$ will be a random set, but there will be some correlation among samples. They will still be suitable for Monte Carlo integration or density estimation, but analyzing the variance of those estimates is much more challenging.

Now suppose that given a transition function $t(x \to y)$ and a function $f(x)$ we want to mimic the distribution of, can we use $a(y \to x)$ such that the points are distributed in the shape of f, or more precisely:

$$\{x_0, x_1, x_2, \ldots\} \sim \frac{f}{\int_s f}$$

It turns out this can be forced by making sure the x_i are *stationary* in some strong sense. If you visualize a huge collection of sample points x, you want the "flow" between two points to be the same in each direction. If we assume the density of points near x and y are proportional to $f(x)$ and $f(y)$ respectively, the the flow in the two directions as follows should be the same:

$$\begin{cases} \text{flow}(x \to y) & = k f(x) t(x \to y) a(x \to y) \\ \text{flow}(y \to x) & = k f(y) t(y \to x) a(y \to x) \end{cases}$$

where k is some positive constant. Setting these two flows constant gives a constraint on a:

$$\frac{a(y \to x)}{a(x \to y)} = \frac{f(x) t(x \to y)}{f(y) t(y \to x)}.$$

Thus if either $a(y \to x)$ or $a(x \to y)$ is known, so is the other. Making them bigger improves the chance of acceptance, so the usual technique is to set the larger of the two to 1.

An awkward part of using the Metropolis sample generation technique is that it is hard to estimate how many points are needed before the set of points is "good". Things are accelerated if the first n points are discarded, although choosing n wisely is non-trivial. Weights can be added if a truly unbiased distribution is desired, as shown later in the context of Metropolis Light Transport.

2.4 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 macroscopic 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 to predict 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 behavior of a particular luminaire in a particular environment, we can just numerically simulate the behavior of the luminaire. 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.

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.

2.5 Density Estimation

Often in graphics we have a set of random points $\{x_0, x_1, \ldots, x_{n-1}\}$ on some domain S and we wish to infer a plausible underlying density function $p(x)$. These problems usually arise in photon tracing applications, although one can also view Metropolis Light Transport's final screen reconstruction as density estimation in screen space.

If the underlying form of the density is known, e.g., $p(x) = ax + b$ on the interval $[0, 1]$ then this becomes a classic problem based on error metric such as a least squares fit. This is known as *parametric density estimation* because the density is a known parametric form.

In graphics we rarely have a parametric form for p. However, we do assume p is smooth and a variety of techniques then exist estimating a smooth p that is representative of the points.

Chapter 3

Direct Lighting via Monte Carlo Integration

By Peter Shirley

In this chapter we apply Monte Carlo Integration to compute the lighting at a point from an area light source in the presence of potential occluders. As the number of samples in the pixel becomes large, the number if samples on the light will become large as well. Thus the shadows will become smooth. Much of the material in this chapter is from the book *Realistic Ray Tracing* and is used with permission from the publisher AK Peters.

3.1 Mathematical framework

To calculate the direct light from one *luminaire* (light emitting object) onto a diffuse surface, we solve the following equations:

$$L(x) = L_e(x) + \frac{R(x)}{\pi} \int_{\text{all } \vec{\omega}'} L_e(x, \vec{\omega}') \cos\theta \, d\omega, \tag{3.1}$$

where $L(x)$ is radiance (color) of x, $L_e(x)$ is the light emitted at x, $R(x)$ is the reflectance of the point, $\vec{\omega}'$ is the direction the light is incident from, and θ is the angle between the incident light and the surface normal. Suppose we wanted to restrict this integral to a domain of one luminaire. Instead of "all $\vec{\omega}'$" we would need to integrate over just the directions toward the luminaire. In practice this can be hard (the projection of a polygon onto the hemisphere is a spherical polygon,

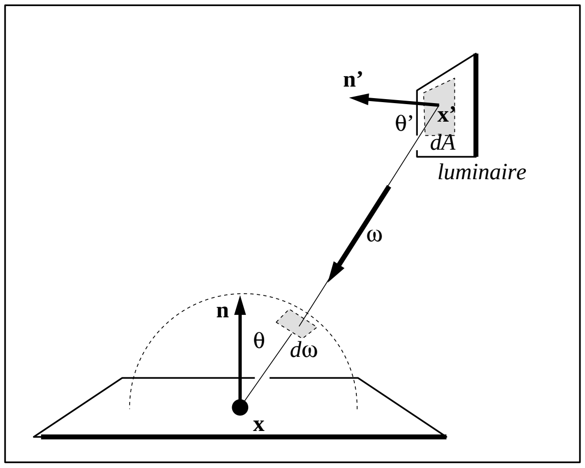


Figure 3.1: Integrating over the luminaire. Note that there is a direct correspondence between dx, the differential area on the luminaire, and $d\omega$, the area of the projection of dx onto the unit sphere centered at x.

and the projection of a cylinder is stranger still). So we can change variable to integrate over just area (Figure 3.1). Note that the differential relationship exists:

$$d\omega = \frac{dA\ \cos\theta'}{\|x' - x\|^2}. \tag{3.2}$$

Just plugging those relationships into Equation 3.1 gives:

$$L(x) = L_e(x) + \frac{R(x)}{\pi} \int_{\text{all } x'} L_e(x') \cos\theta \frac{dA\ \cos\theta'}{\|x' - x\|^2}.$$

There is an important flaw in the equation above. It is possible that the points x and x' cannot "see" each other (there is a shadowing object between them). This can be encoded in a "shadow function" $s(x, x')$ which is either one or zero depending on whether or not there is a clear line of sight between x and x'. This gives us the equation:

$$L(x) = L_e(x) + \frac{R(x)}{\pi} \int_{\text{all } x'} L_e(x') \cos\theta \frac{s(x, x')dA\ \cos\theta'}{\|x' - x\|^2}. \tag{3.3}$$

If we are to sample Equation 3.3, we need to pick a random point x' on the surface of the luminaire with density function p (so $x' \sim p$). Just plugging into the Monte Carlo equation with one sample gives:

$$L(x) \approx L_e(x) + \frac{R(x)}{\pi} L_e(x') \cos\theta \frac{s(x, x') \cos\theta'}{p(x')\|x' - x\|^2}. \tag{3.4}$$

If we pick a uniform random point on the luminaire, then $p = 1/A$, where A is the area of the luminaire. This gives:

$$L(x) \approx L_e(x) + \frac{R(x)}{\pi} L_e(x') \cos\theta \frac{A\, s(x,x') \cos\theta'}{\|x' - x\|^2}. \tag{3.5}$$

We can use Equation 3.5 to sample planar (e.g. rectangular) luminaires in a straightforward fashion. We simply pick a random point on each luminaire. The code for one luminaire would be:

```
spectrum directLight( $x$, $\vec{n}$)
pick random point $x$' with normal vector $\vec{n}'$ on light
$\vec{d} = (x' - x)$
if ray $x + t\vec{d}$ hits at $x'$ then
    return $AL_e(x')(\vec{n} \cdot \vec{d})(-\vec{n}' \cdot \vec{d})/\|\vec{d}\|^4$
else
    return 0
```

The above code needs some extra tests such as clamping the cosines to zero if they are negative. Note that the term $\|\vec{d}\|^4$ comes from the distance squared term and the two cosines, e.g., $\vec{n} \cdot \vec{d} = \|\vec{d}\| \cos\theta$ because $\vec{d}$ is not necessarily a unit vector.

Several examples of soft shadows are shown in Figure 3.2.

3.2 Sampling a spherical luminaire

Although a sphere with center c and radius r can be sampled using Equation 3.5, this will yield a very noisy image because many samples will be on the back of the sphere, and the $\cos\theta'$ term varies so much. Instead we can use a more complex $p(x')$ to reduce noise. The first nonuniform density we might try is $p(x') \propto \cos\theta'$. This turns out to be just as complicated as sampling with $p(x') \propto \cos\theta'/\|x' - x\|^2$, so we instead discuss that here. We observe that sampling on the luminaire this way is the same as using a density constant function $q(\vec{\omega}') = \text{const}$ defined in the space of directions subtended by the luminaire as seen from x. We now use a coordinate system defined with x at the origin, and a right-handed orthonormal basis with $\vec{w} = (c - x)/\|c - x\|$, and $\vec{v} = (\vec{w} \times \vec{n})/\|(\vec{w} \times \vec{n})\|$ (see Figure 3.3). We also define (α, ϕ) to be the azimuthal and polar angles with respect to the *uvw* coordinate system.

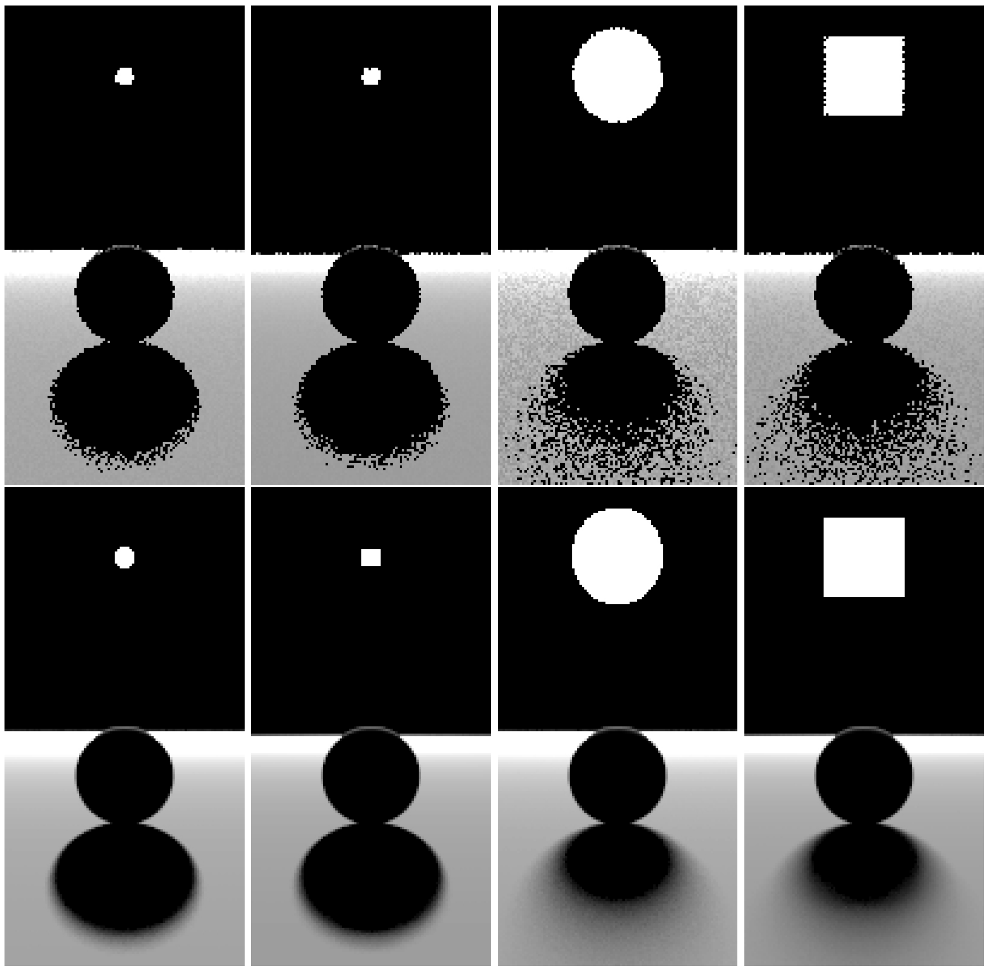


Figure 3.2: Various soft shadows on a backlit sphere with a square and a spherical light source. Top: one sample. Bottom: 100 samples. Note that the shape fof the light source is less important than its size in determining shadow appearance.

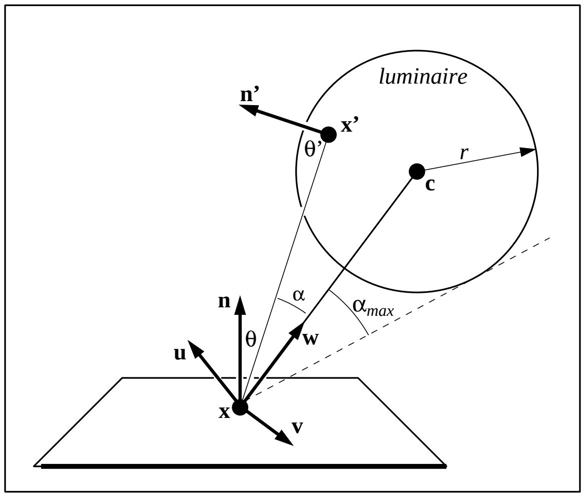


Figure 3.3: Geometry for spherical luminaire.

The maximum α that includes the spherical luminaire is given by:

$$\alpha_{\text{max}} = \arcsin\left(\frac{r}{\|x-c\|}\right) = \arccos\sqrt{1-\left(\frac{r}{\|x-c\|}\right)^2}.$$

Thus a uniform density (with respect to solid angle) within the cone of directions subtended by the sphere is just the reciprocal of the solid angle $2\pi(1-\cos\alpha_{\text{max}})$ subtended by the sphere:

$$q(\omega) = \frac{1}{2\pi\left(1-\sqrt{1-\left(\frac{r}{\|x-c\|}\right)^2}\right)}.$$

And we get

$$\begin{bmatrix} \cos\alpha \\ \phi \end{bmatrix} = \begin{bmatrix} 1-\xi_1+\xi_1\sqrt{1-\left(\frac{r}{\|x-c\|}\right)^2} \\ 2\pi\xi_2 \end{bmatrix}.$$

This gives us the direction to x'. To find the actual point, we need to find the first point on the sphere in that direction. The ray in that direction is just ($x+t\vec{a}$), where $\vec{a}$ is given by:

$$\vec{a} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} \cos\phi\sin\alpha \\ \sin\phi\sin\alpha \\ \cos\alpha \end{bmatrix}.$$

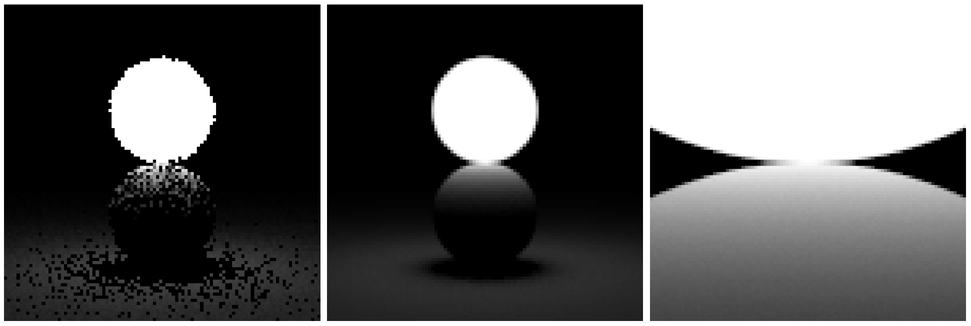


Figure 3.4: A sphere with $L_e = 1$ touching a sphere of reflectance 1. Where they touch the reflective sphere should have $L(x) = 1$. Left: one sample. Middle: 100 samples. Right: 100 samples, close-up.

We must also calculate $p(x')$, the probability density function with respect to the area measure (recall that the density function q is defined in solid angle space). Since we know that q is a valid probability density function using the ω measure, and we know that $ddir = dA(x') \cos\theta' / \|x' - x\|^2$, we can relate any probability density function $q(\vec{\omega}')$ with its associated probability density function $p(x')$:

$$q(\vec{\omega}') = \frac{p(x') \cos\theta'}{\|x' - x\|^2}. \tag{3.6}$$

So we can solve for $p(x')$:

$$p(x') = \frac{\cos\theta'}{2\pi \|x' - x\|^2 \left(1 - \sqrt{1 - \left(\frac{r}{\|x-c\|}\right)^2}\right)}.$$

A good debugging case for this is shown in Figure 3.4. For further details on sampling the sphere with p see the article by Wang [92].

3.3 Non-diffuse Luminaries

There is no reason the brightness of the luminaire cannot vary with both direction and position. It can vary with position if the luminaire is a television. It can vary with direction for car headlights and other directional sources. Nothing need change from the previous sections, except that $L_e(x')$ must change to $L_e(x', \vec{\omega}')$. The simplest way to vary the intensity with direction is to use a phong-like pattern with respect to the normal vector $\vec{n}'$. To keep the total light output independent of exponent, you can use the form:

$$L_e(x', \vec{\omega}') = \frac{(n+1)E(x')}{2\pi} cos^{(n-1)}\theta',$$

where $E(x')$ is the *radiant exitance* (power per unit area) at point x', and n is the phong-exponent. You get a diffuse light for $n = 1$.

3.4 Direct Lighting from Many Luminaires

Traditionally, when N_L luminaires are in a scene, the direct lighting integral is broken into N_L separate integrals [14]. This implies at least N_L samples must be taken to approximate the direct lighting, or some bias must be introduced (as done by Ward where small value samples are not calculated [93]). This is what you should probably do when you first implement your program. However, you can later leave the direct lighting integral intact and design a probability density function over all N_L luminaires.

As an example, suppose we have two luminaires, l_1 and l_2, and we devise two probability functions $p_1(x')$ and $p_2(x')$, where $p_i(x') = 0$ for x' not on l_i and $p_i(x')$ is found by a method such as one of those described previously for generating x' on l_i. These functions can be combined into a single density over both lights by applying a weighted average:

$$p(x') = \alpha p_1(x') + (1-\alpha)p_2(x'),$$

where $\alpha \in (0, 1)$. We can see that p is a probability density function because its integral over the two luminaires is one, and it is strictly positive at all points on the luminaires. Densities that are "mixed" from other densities are often called *mixture densities* and the coefficients α and $(1-\alpha)$ are called the *mixing weights* [82].

To estimate $L = (L_1 + L_2)$, where L is the direct lighting and L_i is the lighting from luminaire l_i, we first choose a random canonical pair (ξ_1, ξ_2), and use it to decide which luminaire will be sampled. If $0 \leq \xi_1 < \alpha$, we estimate L_1 with e_1 using the methods described previously to choose x' and to evaluate $p_1(x')$, and we estimate L with e_1/α. If $\xi_1 \geq \alpha$ then we estimate L with $e_2/(1-\alpha)$. In either case, once we decide which source to sample, we cannot use (ξ_1, ξ_2) directly because we have used some knowledge of ξ_1. So if we choose l_1 (so $\xi_1 < \alpha$), then we choose a point on l_1 using the random pair $(\xi_1/\alpha, \xi_2)$. If we sample l_2 (so $\xi_1 \geq \alpha$), then we use the pair $((\xi_1 - \alpha)/(1 - \alpha), \xi_2)$. This way a collection of stratified samples will remain stratified in some sense. Note that it is to our advantage to have ξ_1 stratified in one dimension, as well as having the pair (ξ_1, ξ_2) stratified in two dimensions, so that the l_i we choose will be stratified over many (ξ_1, ξ_2) pairs, so some multijittered sampling method may be helpful (e.g [7]).

This basic idea used to estimate $L = (L_1 + L_2)$ can be extended to N_L luminaires by mixing N_L densities

$$p(x') = \alpha_1 p_1(x') + \alpha_2 p_2(x') + \cdots + \alpha_{N_L} p_{N_L}(x'), \tag{3.7}$$

where the α_i's sum to one, and where each α_i is positive if l_i contributes to the direct lighting. The value of α_i is the probability of selecting a point on the l_i, and p_i is then used to determine which point on l_i is chosen. If l_i is chosen, the we estimate L with $e_i/\alpha i$. Given a pair (ξ_1, ξ_2), we choose l_i by enforcing the conditions

$$\sum_{j=1}^{i-1} \alpha_j \ < \xi_1 < \ \sum_{j=1}^{i} \alpha_j.$$

And to sample the light we can use the pair (ξ_1', ξ_2) where

$$\xi_1' = \frac{\xi_1 - \sum_{j=1}^{i-1} \alpha_j}{\alpha_i}.$$

This basic process is shown in Figure 3.5. It cannot be over stressed that it is important to “reuse” the random samples in this way to keep the variance low, in the same way we use stratified sampling (jittering) instead of random sampling in the space of the pixel To choose the point on the luminaire l_i given (ξ_1', ξ_2), we can use the same types of p_i for luminaires as used in the last section. The question remaining is what to use for α_i.

3.4.1 Constant α_i

The simplest way to choose values for α_i was proposed by Lange [45] (and this method is also implied in the figure on page 148 of [36]), where all weights are made equal: $\alpha_i = 1/N_L$ for all i. This would definitely make a valid estimator because the α_i sum to one and none of them is zero. Unfortunately, in many scenes this estimate would produce a high variance (when the L_i are very different as occurs in most night “walkthroughs”).

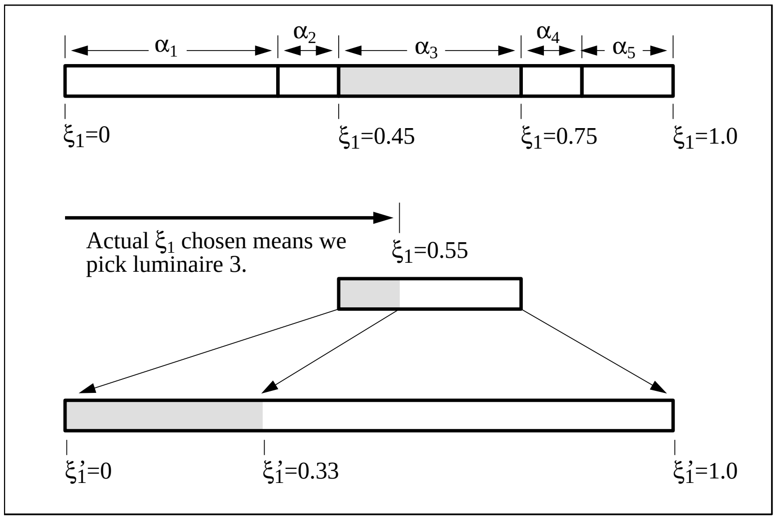


Figure 3.5: Diagram of mapping ξ_1 to choose l_i and the resulting remapping to new canonical sample ξ_1'.

3.4.2 Linear α_i

Suppose we had perfect p_i defined for all the luminaires. A zero variance solution would then result if we could set $\alpha_i \propto L_i$, where L_i is the contribution from the ith luminaire. If we can make α_i approximately proportional to L_i, then we should have a fairly good estimator. We call this the *linear method* of setting α_i because the time used to choose one sample is linearly proportional to N_L, the number of luminaires.

To obtain such α_i we get an estimated contribution e_i at x by approximating the rendering equation for l_i with the geometry term set to one. These e_is (from all luminaires) can be directly converted to α_i by scaling them so their sum is one:

$$\alpha_i = \frac{e_i}{e_1 + e_2 + \cdots + e_{N_L}}. \tag{3.8}$$

This method of choosing α_i will be valid because all potentially visible luminaires will end up with positive α_i. We should expect the highest variance in areas where shadowing occurs, because this is where setting the geometry term to one causes α_i to be a poor estimate of α_i.

Implementing the linear α_i method has several subtleties. If the entire luminaire is below the tangent plane at x, then the estimate for e_i should be zero. An easy mistake to make is to set e_i to zero if the center of the luminaire is below the horizon. This will make α_i take the one value that is not allowed: an incorrect zero. Such a bug will become obvious in pictures of spheres illuminated by luminaires that subtend large solid angles, but for many scenes such errors are not noticeable (the figures in [68] had this bug, but it was not noticeable). To overcome this problem, we make sure that for a polygonal luminaires all of its vertices are below the horizon before it is given a zero probability of being sampled. For spherical luminaires, we check that the center of the luminaire is a distance greater than the sphere radius under the horizon plane before it is given a zero probability of being sampled.

Chapter 4

Stratified Sampling of 2-Manifolds

By Jim Arvo

4.1 Introduction

Monte Carlo techniques arise in image synthesis primarily as a means to solve integration problems. Integration over domains of two or higher dimensions is ubiquitous in image synthesis; indirect global illumination is expressed as an integral over all paths of light, which entails numerous direct illumination problems such as the computation of form factors, visibility, reflected radiance, subsurface scattering, and irradiance due to complex or partially occluded luminaires, all of which involve integration.

The Monte Carlo method stems from a very natural and immediate connection between integration and *expectation*. Every integral, in both deterministic and probabilistic contexts, can be viewed as the expected value (mean) of a random variable; by averaging over many samples of the random variable, we may thereby approximate the integral. However, there are infinitely many random variables that can be associated with any given integral; of course, some are better than others.

One attribute that makes some random variables better than others for the purpose of integration is the ease with which they can be sampled. In general, we tend to construct Monte Carlo methods using only those random variables with convenient and efficient sampling procedures. But there is also a competing attribute. One of the maxims of Monte Carlo integration is that the probability density function of the random variable should mimic the integrand as closely as possible. The closer the match, the smaller the variance of the random variable, and the more reliable (and efficient) the estimator. In the limit, when the samples are generated with a density that is exactly proportional to the (positive) integrand, the variance of the estimator is identically zero [64]. That is, a single sample delivers the exact answer with probability one.

Perhaps the most common form of integral arising in image synthesis is that expressing either irradiance or reflected radiance at a surface. In both cases, we must evaluate (or approximate) an integral over solid angle, which is of the form

$$\int_S f(\vec{\omega})(\vec{\omega} \cdot \vec{n})\, d\omega, \tag{4.1}$$

where S is subset of the unit sphere, $\vec{\omega}$ is a unit direction vector, and $\vec{n}$ is the surface normal vector, or over surface area, which is of the form

$$\int_A f(x) \frac{(x \cdot \vec{n})(x \cdot \vec{n}')}{||x||^2}\, dx, \tag{4.2}$$

where A is a surface and x is a point in $\mathbb{R}^3$. Technically, these integrals differ only by a change of variable that results from the pullback of surface differentials to solid angle differentials [1]. The function f may represent the radiance or emissive power of a luminaire (in the case of irradiance) or it may include a BRDF (in the case of reflected radiance). In all cases, visibility may be included in the function f, which can make the integrand arbitrarily discontinuous, thereby drastically reducing the effectiveness of standard numerical quadrature methods for computing the integral.

To apply Monte Carlo integration most effectively in image synthesis, we seek sampling algorithms that match the geometries and known light distributions found in a simulated scene to the extent possible. Consequently, a wide assortment of sampling algorithms have been developed for sampling both the surfaces of and the solid angles subtended by various scene geometries [71] so that both forms of the integral above can be accommodated. In this chapter we will see how to construct random variables for specific geometries: that is, random variables whose range coincides with some bounded region of the plane or some bounded surface in $\mathbb{R}^3$, and whose probability density function is constant. For instance, we will see how to generate uniformly distributed samples over both planar and spherical triangles, and projected spherical polygons.

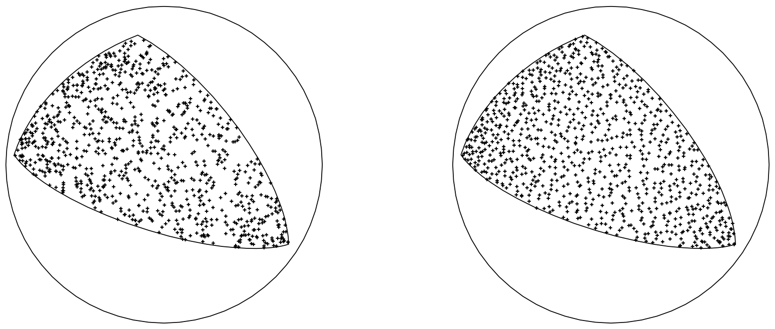


Figure 4.1: *A spherical triangle with uniform samples (left) and stratified samples (right). Both sets of samples were generated using an area-preserving parametrization for spherical triangles, which we derive below.*

All of the sampling algorithms that we construct are based on mappings from the unit square, $[0, 1] \times [0, 1]$, to the regions or surfaces in question that preserve uniform sampling. That is, uniformly distributed samples in the unit square are mapped to uniformly distributed samples in the range. Such mappings also preserve *stratification*, also known as *jitter sampling* [13], which means that uniform partitionings of the unit square map to uniform partitionings of the range. The ability to apply stratified sampling over various domains is a great advantage, as it is often a very effective variance reduction technique. Figure 4.1 shows the result of applying such a mapping to a spherical triangle, both with and without stratified (jittered) sampling. Figure 4.2 shows the result of applying such a mapping to a projected spherical polygon, so that the samples on the original spherical polygon are "cosine-distributed" rather than uniformly distributed.

All of the resulting algorithms depend upon a source of uniformly distributed random numbers in the interval $[0, 1]$, which we shall assume is available from some unspecified source: perhaps "drand48," or some other pseudo-random number generator.

4.2 A Recipe for Sampling Algorithms

Although there is a vast and mature literature an Monte Carlo methods, with many texts describing how to derive sampling algorithms for various geometries and density functions (see, for example, Kalos and Whitlock [37], Spanier and Gel-

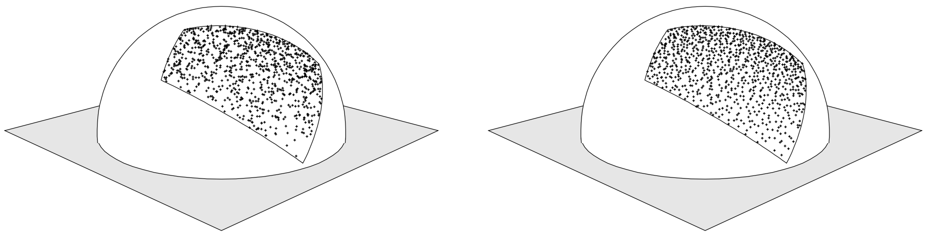


Figure 4.2: *A projected spherical polygon with uniform samples (left) and stratified samples (right). Projection onto the plane results in more samples near the north pole of the sphere than near the equator. Both sets of samples were generated using an area-preserving parametrization for spherical polygons, which we derive below.*

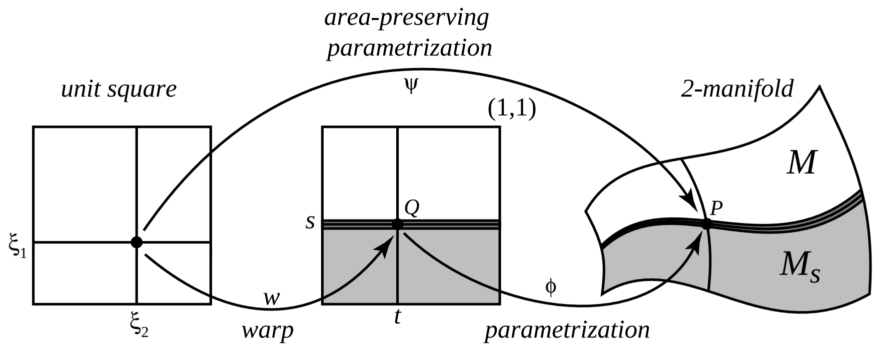


Figure 4.3: *An arbitrary parametrization ϕ for a 2-manifold $\mathcal{M}$ can be converted into an area-preserving parametrization, which is useful for uniform and stratified sampling, by composing it with a warping function. The warping function can be derived directly from ϕ by following a precise procedure.*

bard [77], or Rubinstein [64]), these treatments do not provide step-by-step instructions for deriving the types of algorithms that we frequently require in computer graphics. In this section we present a detailed "recipe" for how to convert an arbitrary parametrization $\phi : [0,1]^2 \to \mathcal{M}$, from the unit square to a 2-manifold, into an *area-preserving* parametrization, $\psi : [0,1]^2 \to \mathcal{M}$. That is, a mapping ψ with the property that

$$\mathbf{area}(A) = \mathbf{area}(B) \implies \mathbf{area}(\psi[A]) = \mathbf{area}(\psi[B]), \tag{4.3}$$

for all $A, B \in [0,1] \times [0,1]$. Such mappings are used routinely in image synthesis to sample surfaces of luminaires and reflectors. Note that ψ may in fact shrink or magnify areas, but that all areas undergo exactly the same scaling; hence, it is area-preserving in the strictest sense only when $\mathbf{area}(\mathcal{M}) = 1$. A parametrization with this property will allow us to generate uniformly distributed and/or stratified samples over $\mathcal{M}$ by generating samples with the desired properties on the unit square (which is trivial) and then mapping them onto $\mathcal{M}$. We shall henceforth consider area-preserving parametrizations to be synonymous with *sampling algorithms*.

Let $\mathcal{M}$ represent a shape that we wish to generate uniformly distributed samples on; in particular, $\mathcal{M}$ may be any 2-manifold with boundary in $\mathbb{R}^n$, where n is typically 2 or 3. The steps for deriving a sampling algorithm for $\mathcal{M}$ are summarized in Figure 4.4. These steps apply for all dimensions $n \geq 2$; that is, $\mathcal{M}$ may be a 2-manifold in any space.

Step 1 requires that we select a smooth bijection ϕ from $[0,1] \times [0,1]$ to the 2-manifold $\mathcal{M}$. Such a function is referred to as a *parametrization* and its inverse is called a *coordinate chart*, as it associates unique 2D coordinates with (almost) all points of $\mathcal{M}$. In reality, we only require ϕ to be a bijection *almost everywhere*; that is, on all but a set of measure zero, such as the boundaries of $\mathcal{M}$ or $[0,1] \times [0,1]$. In theory, any smooth bijection will suffice, although it may be impractical or impossible to perform some of the subsequent steps in Figure 4.4 symbolically (particularly step 4) for all but the simplest functions.

Step 2 defines a function $\sigma : [0,1]^2 \to \mathbb{R}$ that links the parametrization ϕ to the notion of surface area on $\mathcal{M}$. More precisely, for any region $A \subseteq [0,1]^2$, the function σ satisfies

$$\int_A \sigma \;=\; \mathbf{area}(\phi[A]). \tag{4.10}$$

That is, the integral of σ over any region A in the 2D parameter space is the surface area of the corresponding subset of $\mathcal{M}$ under the mapping ϕ. Equation (4.4) holds for all $n \geq 2$. For the typical cases of $n = 2$ and $n = 3$, however, the function σ can be expressed more simply. For example, if $\mathcal{M}$ is a subset of $\mathbb{R}^2$ then

$$\sigma(s,t) \;\equiv\; \det\bigl(D_{(s,t)}\phi\bigr)\,, \tag{4.11}$$

where $D_{(s,t)}\phi$ is 2×2 Jacobian matrix of ϕ at the point (s,t). On the other hand, if $\mathcal{M}$ is a subset of $\mathbb{R}^3$, then

$$\sigma(s,t) \;\equiv\; \|\, \phi_s(s,t) \times \phi_t(s,t)\, \|\,, \tag{4.12}$$

which is a convenient abbreviation for equation (4.4) that holds only when $n = 3$, as the two partial derivatives of ϕ are vectors in $\mathbb{R}^3$ in this case. Non-uniform sampling can also be accommodated by including a weighting function in the definition of σ in step 2.

1. Select a *parametrization* ϕ for the 2-manifold $\mathcal{M} \subset \mathbb{R}^n$. That is, select a smooth bijection (diffeomorphism) $\phi : [0,1]^2 \to \mathcal{M}$.

2. Define the function $\sigma : [0,1]^2 \to \mathbb{R}$ by

$$\sigma \;\equiv\; \sqrt{(\phi_s \cdot \phi_s)(\phi_t \cdot \phi_t) - (\phi_s \cdot \phi_t)^2} \tag{4.4}$$

   where $\phi_s = \left(\frac{\partial \phi_1}{\partial s}, \ldots, \frac{\partial \phi_n}{\partial s}\right)$ is the vector of partial derivatives.

3. Define two cumulative distribution functions on $[0,1]$ by

$$F(s) \;\equiv\; \frac{\int_0^1 \int_0^s \sigma(u,v)\,du\,dv}{\int_0^1 \int_0^1 \sigma(u,v)\,du\,dv} \tag{4.5}$$

$$G_s(t) \;\equiv\; \frac{\int_0^t \sigma(s,v)\,dv}{\int_0^1 \sigma(s,v)\,dv} \tag{4.6}$$

4. Invert the two cumulative distribution functions

$$f(z) \;\equiv\; F^{-1}(z) \tag{4.7}$$

$$g(z_1, z_2) \;\equiv\; G^{-1}_{f(z_1)}(z_2) \tag{4.8}$$

5. Define the new parametrization $\psi : [0,1]^2 \to \mathcal{M}$ by

$$\psi(z_1, z_2) \;\equiv\; \phi(f(z_1), g(z_1, z_2)). \tag{4.9}$$

   Thus, the mapping $(z_1, z_2) \mapsto (s,t) = (f(z_1), g(z_1, z_2))$ defines the warping function that converts the original parametrization ϕ into the area-preserving parametrization ψ.

Figure 4.4: *Five steps for deriving an area-preserving parametrization ψ from $[0,1] \times [0,1]$ to any bounded 2-manifold $\mathcal{M} \subset \mathbb{R}^n$, beginning from an arbitrary parametrization ϕ for $\mathcal{M}$. The function ψ is suitable for stratified sampling of $\mathcal{M}$. Step 2 simplifies in the common two- and three- dimensional cases. Step 4 is often the only impediment to finding a closed-form expression for the area-preserving parametrization.*

Step 3 can often be carried out without the aid of an explicit expression for σ. For example, the cumulative distributions can often be found by reasoning directly about the geometry imposed by the parametrization rather than applying formulas (4.4), (4.5) and (4.6), which can be tedious. Let $\mathcal{M}_s$ denote the family of sub-manifolds of $\mathcal{M}$ defined by the first coordinate of ϕ. That is,

$$\mathcal{M}_s \;=\; \phi\Big[\; [0,s] \times [0,1] \;\Big]\,. \tag{4.13}$$

See Figure 4.3. It follows from the definition of F and equation (4.10) that

$$F(s) \;=\; \frac{\mathbf{area}(\mathcal{M}_s)}{\mathbf{area}(\mathcal{M})}, \tag{4.14}$$

which merely requires that we find an expression for the surface area of $\mathcal{M}_s$ as a function of s. Similarly, by equation (4.7) we have

$$s \;=\; f\left(\frac{\mathbf{area}(\mathcal{M}_s)}{\mathbf{area}(\mathcal{M})}\right). \tag{4.15}$$

Thus, f is the map that recovers the parameter s from the fractional area of the sub-manifold $\mathcal{M}_s$. Equation (4.15) can be more convenient to work with than equation (4.14), as it avoids an explicit function inversion step. While $G_s(\cdot)$ and $g(\cdot,\cdot)$ do not admit equally intuitive interpretations, they can often be determined from the general form of σ, since many of the details vanish due to normalization. A good example of how this can be done is provided by the area-preserving parametrization derived for spherical triangles, which we discuss below.

Step 4 above is the only step that is not purely mechanical, as it involves function inversion. When this step can be carried out symbolically, the end result is a closed-form area-preserving transformation ψ from $[0,1]^2$ to the manifold $\mathcal{M}$. Closed-form expressions are usually advantageous, both in terms of simplicity and efficiency. Of the two inversions entailed in step 4, it is typically equation (4.5) that is the more troublesome, and frequently resists symbolic solution. In such a case, it is always possible to perform the inversion numerically using a root-finding method such as Newton's method; of course, one must always weigh the cost of drawing samples against the benefits conferred by the resulting importance sampling and stratification. When numerical inversion is involved, the area-preserving transformation is less likely to result in a net gain in efficiency.

The steps outlined in Figure 4.4 generalize very naturally to the construction of volume-preserving parametrizations for arbitrary k-manifolds. For any $2 \le k \le n$, step 3 entails a sequence of k cumulative distribution functions, each dependent upon all of its predecessors, and step 4 requires the cascaded inversion of all k distributions, in the order of their definition. Step 5 entails a k-way function composition. In the remainder of these notes, we will consider only the case where $k = 2$ and $n \in \{2, 3\}$; that is, we will only consider the problem of generating samples over 2-manifolds (surfaces) in $\mathbb{R}^2$ or $\mathbb{R}^3$.

4.3 Analytic Area-Preserving Parametrizations

In this section we will apply the "recipe" given in Figure 4.4 to derive a number of useful area-preserving parametrizations. Each will be expressed in closed form since the functions F and G_s will be invertible symbolically; however, in the case of spherical triangles it will not be trivial to invert F.

4.3.1 Sampling Planar Triangles

As a first example of applying the steps in Figure 4.4, we shall derive an area-preserving parametrization for an arbitrary triangle ABC in the plane. We begin with an obvious parametrization from $[0, 1] \times [0, 1]$ to a given triangle in terms of barycentric coordinates. That is, let

$$\phi(s, t) = (1 - s)A + s(1 - t)B + stC. \tag{4.16}$$

It is easy to see that ϕ is a smooth mapping that is bijective except when $t = 0$, which is a set of measure zero. Since the codomain of ϕ is $\mathbb{R}^2$, σ is simply the Jacobian of ϕ. After a somewhat tedious computation, we obtain

$$\text{det}(D\phi) = 2cs, \tag{4.17}$$

where c is the area of the triangle. From equations (4.5) and (4.6) we obtain

$$F(s) \;=\; s^2 \quad \text{and} \quad G_s(t) \;=\; t. \tag{4.18}$$

In both cases the constant c disappears due to normalization. These functions are trivial to invert, resulting in

$$f(z) \;=\; \sqrt{z} \quad \text{and} \quad g(z_1, z_2) \;=\; z_2. \tag{4.19}$$

```
SamplePlanarTriangle( real $\xi_1$, real $\xi_2$ )
    Compute the warping function $(\xi_1, \xi_2) \mapsto (s, t)$.
        $s \leftarrow \sqrt{\xi_1}$;
        $t \leftarrow \xi_2$;
    Plug the warped coords into the original parametrization.
        $\mathbf{P} \leftarrow (1-s)A + s(1-t)B + stC$;
        return P;
        end
```

Figure 4.5: *Algorithm for computing an area-preserving parametrization of the triangle with vertices A, B, and C. This mapping can be used for uniform or stratified sampling.*

Finally, after function composition, we have

$$\begin{aligned} \psi(z_1, z_2) &= \phi(f(z_1), g(z_1, z_2)) \\ &= (1-\sqrt{z_1})A + \sqrt{z_1}(1-z_2)B + \sqrt{z_1}z_2C. \end{aligned} \tag{4.20}$$

Figure 4.5 shows the final algorithm. If ξ_1 and ξ_2 are independent random variables, uniformly distributed over the interval $[0, 1]$, then the resulting points will be uniformly distributed over the triangle.

4.3.2 Sampling the Unit Disk

Next, we derive an area-preserving parametrization for a unit-radius disk D in the plane, centered at the origin. We start with the parametrization from $[0, 1] \times [0, 1]$ to D given by

$$\phi(s, t) = s\,[\,\cos(2\pi t)\mathbf{x} + \sin(2\pi t)\mathbf{y}\,]\,, \tag{4.21}$$

where **x** and **y** are the orthogonal unit vectors in the plane. Again, ϕ is a smooth mapping that is bijective except when $s = 0$. Computing the Jacobian of ϕ, we obtain

$$\det(D\phi) \;=\; 2\pi s. \tag{4.22}$$

The remaining steps proceed precisely as in the case of the planar triangle; in fact, the distributions F and G turn out to be identical. Thus, we obtain

$$\begin{aligned} \psi(z_1, z_2) &= \phi(\sqrt{z_1}, z_2) \\ &= \sqrt{\xi_1}\,[\cos(2\pi\xi_2)\mathbf{x} + \sin(2\pi\xi_2)\mathbf{y}]\,. \end{aligned} \tag{4.23}$$

The resulting algorithm for sampling the unit disk is exactly analogous to the algorithm shown in Figure 4.5 for sampling planar triangles.

4.3.3 Sampling the Unit Hemisphere

As a first example of applying the steps in Figure 4.4 to a surface in $\mathbb{R}^3$, we shall derive the well-known area-preserving parametrization for the unit-radius hemisphere centered at the origin. First, we define a parametrization using spherical coordinates:

$$\phi(s,t) = \begin{bmatrix} \sin\left(\frac{\pi s}{2}\right)\cos(2\pi t) \\ \sin\left(\frac{\pi s}{2}\right)\sin(2\pi t) \\ \cos\left(\frac{\pi s}{2}\right) \end{bmatrix}. \tag{4.24}$$

Here the parameter s defines the polar angle and t defines the azimuthal angle. Since the codomain of ϕ is $\mathbb{R}^3$, we can apply equation (4.12). Since

$$\phi_s(s,t) \times \phi_t(s,t) = \pi^2 \begin{bmatrix} \cos\left(\frac{\pi s}{2}\right)\cos(2\pi t) \\ \cos\left(\frac{\pi s}{2}\right)\sin(2\pi t) \\ -\sin\left(\frac{\pi s}{2}\right) \end{bmatrix} \times \begin{bmatrix} -\sin\left(\frac{\pi s}{2}\right)\sin(2\pi t) \\ \sin\left(\frac{\pi s}{2}\right)\cos(2\pi t) \\ 0 \end{bmatrix},$$

we obtain

$$\begin{aligned} \sigma(s,t) &= \|\, \phi_s(s,t) \times \phi_t(s,t) \,\| \\ &= \pi^2 \sin\left(\frac{\pi s}{2}\right). \end{aligned} \tag{4.25}$$

It then follows easily that

$$F(s) \;=\; 1 - \cos\left(\frac{\pi s}{2}\right) \quad \text{and} \quad G_s(t) \;=\; t,$$

which are trivial to invert, resulting in

$$f(z) \;=\; \frac{2\cos^{-1}(1-z)}{\pi} \quad \text{and} \quad g(z_1,z_2) \;=\; z_2.$$

Composing f and g with ϕ results in

$$\psi(z_1,z_2) = \begin{bmatrix} \sqrt{z_1(2-z_1)}\,\cos(2\pi z_2) \\ \sqrt{z_1(2-z_1)}\,\sin(2\pi z_2) \\ 1 - z_1 \end{bmatrix}. \tag{4.26}$$

Here the s coordinate of the parametrization simply selects the z-plane from $z = 1$ and $z = 0$, while the t coordinate parameterizes the resulting circle in the z-plane. The form of ψ can be simplified somewhat by substituting $1 - z_1$ for z_1, which does not alter the distribution.

4.3.4 Sampling a Phong Lobe

Now suppose that we wish to sample the hemisphere according to a Phong distribution rather than uniformly; that is, with a density proportional to the cosine of the polar angle to a power. To do this we simply include a weighting function in the definition of σ given in equation (4.25). That is, we let

$$\sigma(s,t) \;=\; \pi^2 \sin\!\left(\frac{\pi s}{2}\right) \cos^k\!\left(\frac{\pi s}{2}\right), \tag{4.27}$$

where k is the Phong exponent. It follows that

$$F(s) \;=\; \cos^{k+1}\!\left(\frac{\pi s}{2}\right) \qquad \text{and} \qquad G_s(t) \;=\; t,$$

which implies that

$$f(z) \;=\; \frac{2}{\pi} \cos^{-1} z^{\frac{1}{n+1}} \qquad \text{and} \qquad g(z_1, z_2) \;=\; z_2.$$

It follows that

$$\psi(z_1, z_2) = \begin{bmatrix} \sqrt{1 - z_1^{\frac{2}{k+1}}}\,\cos(2\pi z_2) \\ \sqrt{1 - z_1^{\frac{2}{k+1}}}\,\sin(2\pi z_2) \\ z_1^{\frac{1}{k+1}} \end{bmatrix}. \tag{4.28}$$

4.3.5 Sampling Spherical Triangles

We shall now derive an area-preserving parametrization for an arbitrary spherical triangle, which is significantly more challenging than the cases we've considered thus far. Let T denote the spherical triangle with vertices **A**, **B**, and **C**, as shown in Figure 4.6. Such a triangle can be parameterized by using the first coordinate s to select the edge length b_s, which in turn defines sub-triangle $\mathrm{T}_s \subset \mathrm{T}$, and the second coordinate t to select a point along the edge $\mathbf{BC}_s$, as shown in Figure 4.6. This parameterization can be expressed as

$$\phi(s,t) \;=\; \mathbf{slerp}(\,\mathbf{B},\, \mathbf{slerp}(\,\mathbf{A}, \mathbf{C}, s\,),\, t\,), \tag{4.29}$$

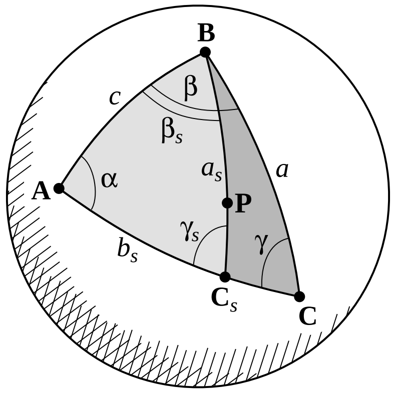


Figure 4.6: *Parameter s controls the edge length b_s, which determines the vertex* $\mathbf{C}_s$*, and consequently sub-triangle* T_s*. Parameter t then selects a point* **P** *along the arc between* $\mathbf{C}_s$ *and* **B**. *Not shown is the length of the arc* **AC**, *which is b.*

where $\mathbf{slerp}(\mathbf{A}, \mathbf{C}, s)$ is the *spherical linear interpolation* function that generates points along the great arc connecting **A** and **C** (according to arc length) as s varies from 0 to 1. The **slerp** function can be defined as

$$\mathbf{slerp}(\mathbf{x}, \mathbf{y}, s) \;=\; \mathbf{x}\cos(\theta s) + [\,\mathbf{y}\,|\,\mathbf{x}\,]\sin(\theta s), \tag{4.30}$$

where $\theta = \cos^{-1}\mathbf{x}\cdot\mathbf{y}$, and $[\,\mathbf{y}\,|\,\mathbf{x}\,]$ denotes the normalized component of the vector **y** that is orthogonal to the vector **x**; that is

$$[\,\mathbf{y}\,|\,\mathbf{x}\,] \;\equiv\; \frac{(\mathbf{I} - \mathbf{x}\mathbf{x}^{\mathrm{T}})\,\mathbf{y}}{||(\mathbf{I} - \mathbf{x}\mathbf{x}^{\mathrm{T}})\,\mathbf{y}\,||}, \tag{4.31}$$

where **x** is assumed to be a unit vector. From the above definition of ϕ it is now possible to derive the function σ using equation (4.12). We find that σ is of the form

$$\sigma(s,t) \;=\; h(s)\sin(\,a_s t\,) \tag{4.32}$$

for some function h, where a_s is the length of the moving edge $\mathbf{BC}_s$ as a function of s. The exact nature of h is irrelevant, however, as it will not be needed to compute F, and it is eliminated from G_s by normalization. Thus, we have

$$F(\,s\,) \;=\; \frac{\mathbf{area}(\,\mathrm{T}_s\,)}{\mathbf{area}(\mathrm{T})} \tag{4.33}$$

$$G_s(\,t\,) \;=\; \frac{1 - \cos(\,a_s t\,)}{1 - \cos a_s}. \tag{4.34}$$

It follows immediately from inversion of equation (4.34) that

$$g(z_1, z_2) \;=\; \frac{1}{a_{z_1}} \cos^{-1}\left[1 - z_2(1 - \cos a_{z_1})\right]. \tag{4.35}$$

However, solving for f, which is the inverse of F, is not nearly as straightforward. Our approach will be to derive an expression for the function f directly, using equation (4.15), rather than starting from F and inverting it. To do this, we require several elementary identities from spherical trigonometry. Let $\mathcal{A}$ denote the surface area of the spherical triangle T with vertices **A**, **B** and **C**. Let a, b, and c denote the edge lengths of T,

$$\begin{aligned} a &= \cos^{-1}\ \mathbf{B}\cdot\mathbf{C}, \\ b &= \cos^{-1}\ \mathbf{A}\cdot\mathbf{C}, \\ c &= \cos^{-1}\ \mathbf{A}\cdot\mathbf{B}, \end{aligned}$$

and let α, β, and γ denote the three internal angles, which are the dihedral angles between the planes containing the edges. See Figure 4.6. Listed below are a few well-known identities for spherical triangles:

$$\mathcal{A} = \alpha + \beta + \gamma - \pi \tag{4.36}$$

$$\frac{\sin\alpha}{\sin a} = \frac{\sin\beta}{\sin b} = \frac{\sin\gamma}{\sin c} \tag{4.37}$$

$$\cos\alpha = -\cos\beta\,\cos\gamma \;+\; \sin\beta\,\sin\gamma\,\cos a \tag{4.38}$$

$$\cos\beta = -\cos\gamma\,\cos\alpha \;+\; \sin\gamma\,\sin\alpha\,\cos b \tag{4.39}$$

$$\cos\gamma = -\cos\beta\,\cos\alpha \;+\; \sin\beta\,\sin\alpha\,\cos c \tag{4.40}$$

Each of these identities will be employed in deriving area-preserving parametrizations, either for spherical triangles or projected spherical polygons, which will be described in the following section. Equation (4.36) is known as Girard's formula, equation (4.37) is the spherical law of sines, and equations (4.38), (4.39), and (4.40) are spherical cosine laws for angles [6].

Our task will be to construct $f : [0,1] \to \mathbb{R}$ such that $f(\mathcal{A}_s/\mathcal{A}) = s$, where the parameter $s \in [0,1]$ selects the sub-triangle T_s and consequently determines the area $\mathcal{A}_s$. Specifically, the sub-triangle T_s is formed by choosing a new vertex $\mathbf{C}_s$ on the great arc between **A** and **C**, at an arc length of $b_s = sb$ along the arc from **A**, as shown in Figure 4.6. The point **P** is finally chosen on the arc between **B** and $\mathbf{C}_s$, according to the parameter t.

```
SampleSphericalTriangle( real $\xi_1$, real $\xi_2$ )
    Use one random variable to select the new area.
        $\mathcal{A}_s \leftarrow \xi_1 \mathcal{A}$;
    Save the sine and cosine of the angle $\Delta$.
        $p \leftarrow \sin(\mathcal{A}_s - \alpha)$;
        $q \leftarrow \cos(\mathcal{A}_s - \alpha)$;
    Compute the pair $(u, v)$ that determines $\sin\beta_s$ and $\cos\beta_s$.
        $u \leftarrow q - \cos\alpha$;
        $v \leftarrow p + \sin\alpha \cos c$;
    Compute the $s$ coordinate as normalized arc length from $\mathbf{A}$ to $\mathbf{C}_s$.
        $s \leftarrow \frac{1}{b} \cos^{-1}\left[\frac{(v\,q \,-\, u\,p)\, \cos\alpha \,-\, v}{(v\,p \,+\, u\,q)\, \sin\alpha}\right]$;
    Compute the third vertex of the sub-triangle.
        $\mathbf{C}_s \leftarrow \mathbf{slerp}(\mathbf{A}, \mathbf{C}, s)$;
    Compute the $t$ coordinate using $\mathbf{C}_s$ and $\xi_2$.
        $t \leftarrow \frac{\cos^{-1}\left[1 - \xi_2(1 - \mathbf{C}_s \cdot \mathbf{B})\right]}{\cos^{-1} \mathbf{C}_s \cdot \mathbf{B}}$;
    Construct the corresponding point on the sphere.
        $\mathbf{P} \leftarrow \mathbf{slerp}(\mathbf{B}, \mathbf{C}_s, t)$;
        return P;
        end
```

Figure 4.7: *An area-preserving parametrization for an arbitrary spherical triangle* **ABC**. *This procedure can be easily optimized to remove the inverse cosines used to compute the warped coordinates s and t, since the* $\mathbf{slerp}$ *function uses the cosine of its scalar argument.*

To find the parameter s that corresponds to the fractional area $\mathcal{A}_s/\mathcal{A}$, we first solve for $\cos b_s$ in terms of $\mathcal{A}_s$ and various constants associated with the triangle. From equations (4.36) and (4.39) we have

$$
\begin{aligned}
\cos b_s &= \frac{\cos\gamma_s \,\cos\alpha \,+\, \cos\beta_s}{\sin\gamma_s \,\sin\alpha} \\
&= \frac{-\cos(\mathcal{A}_s - \alpha - \beta_s)\,\cos\alpha \,+\, \cos\beta_s}{-\sin(\mathcal{A}_s - \alpha - \beta_s)\,\sin\alpha} \\
&= \frac{\cos(\Delta - \beta_s)\,\cos\alpha \,-\, \cos\beta_s}{\sin(\Delta - \beta_s)\,\sin\alpha},
\end{aligned}
\qquad (4.41)
$$

where we have introduced $\Delta \equiv \mathcal{A}_s - \alpha$. We now eliminate β_s to obtain a function that depends only on area and the fixed parameters: in particular, we shall construct a function of only Δ, α, and c. We accomplish this by using spherical trigonometry to find expressions for both $\sin\beta_s$ and $\cos\beta_s$. From equation (4.36) and plane trigonometry it follows that

$$\cos\gamma_s \;=\; -\cos(\Delta - \beta_s) \;=\; \sin\Delta\sin\beta_s - \cos\Delta\cos\beta_s. \tag{4.42}$$

Combining equation (4.42) with equation (4.40) we have

$$(\cos\Delta \;-\; \cos\alpha)\,\cos\beta_s \;+\; (\sin\Delta \;+\; \sin\alpha\,\cos c)\,\sin\beta_s \;=\; 0. \tag{4.43}$$

Consequently, $\sin\beta_s = -ru$ and $\cos\beta_s = rv$ where

$$\begin{aligned} u &\equiv \cos\Delta \;-\; \cos\alpha, \\ v &\equiv \sin\Delta \;+\; \sin\alpha\,\cos c, \end{aligned}$$

and r is a common factor that cancels out in our final expression, so it is irrelevant. Simplifying equation (4.41) using these new expressions for $\sin\beta_s$ and $\cos\beta_s$, we obtain an expression for $\cos b_s$ in terms of Δ, u, v, and α. It then follows that

$$s \;=\; \frac{1}{b}\,\cos^{-1}\left[\frac{(\,v\cos\Delta - u\sin\Delta\,)\,\cos\alpha \;-\; v}{(\,v\sin\Delta + u\cos\Delta\,)\,\sin\alpha}\right], \tag{4.44}$$

since $s = b_s/b$. Note that $\cos b_s$ determines b_s, since $0 < b_s < \pi$, and that b_s in turn determines the vertex $\mathbf{C}_s$. The algorithm shown in Figure 4.7 computes an area-preserving map from the unit square onto the triangle T; it takes two variables ξ_1 and ξ_2, each in the unit interval, and returns a point $\mathbf{P} \in \mathrm{T} \subset \mathbb{R}^3$. If ξ_1 and ξ_2 are uniformly distributed random variables in $[0, 1]$, the algorithm will produce a random variable $\mathbf{P}$ that is uniformly distributed over the surface of the spherical triangle T.

The procedure in Figure 4.7 explicitly warps the coordinates (ξ_1, ξ_2) into the coordinates (s, t) in such a way that the resulting parametrization is area-preserving. If implemented exactly as shown, the procedure performs a significant amount of unnecessary computation. Most significantly, all of the inverse cosines can be eliminated by substituting the equation (4.30) for the $\mathbf{slerp}$ function and then simplifying [3]. Also, $\cos\alpha$, $\sin\alpha$, $\cos c$, and $[\,\mathbf{C}\,|\,\mathbf{A}\,]$, which appears in the expression for $\mathbf{slerp}(\mathbf{A}, \mathbf{C}, s)$, need only be computed once per triangle rather than once per sample.

Results of the algorithm are shown in Figure 4.1. On the left, the samples are identically distributed, which produces a pattern equivalent to that obtained by rejection sampling; however, each sample is guaranteed to fall within the triangle. The pattern on the right was generated by partitioning the unit square into a regular grid and choosing one pair (ξ_1, ξ_2) uniformly from each grid cell, which corresponds to stratified sampling. The advantage of stratified sampling is evident in the resulting pattern; the samples are more evenly distributed, which generally reduces the variance of Monte Carlo estimates based on these samples. The sampling algorithm can be applied to spherical polygons by decomposing them into triangles and performing stratified sampling on each component independently, which is analogous to the method for planar polygons described by Turk [84]. This is one means of sampling the solid angle subtended by a polygon. We discuss another approach in the following section.

4.4 Sampling Projected Spherical Polygons

In this section we will see an example in which the inversion of the F function can not be done symbolically; consequently, we will resort to either approximate inversion, or inversion via a root finder.

The dot product $\vec{\omega} \cdot \vec{n}$ appearing in equation (4.1) is the ubiquitous "cosine" factor that appears in nearly every illumination integral. Since it is often infeasible to construct a random variable that mimics the full integrand, we settle for absorbing the cosine term into the sampling distribution; this compromise is a useful special case of *importance sampling*. In this section we address the problem of generating stratified samples over the solid angle subtended by arbitrary polygons, while taking the cosine weighting into account, as shown in Figure 4.2. The combination of stratification and importance sampling, even in this relatively weak form, can significantly reduce the variance of the associated Monte Carlo estimator [3, 64].

We now describe a new technique for Monte Carlo sampling of spherical polygons with a density proportional to the cosine from a given axis which, by Nusselt's analogy, is equivalent to uniformly sampling the projection of the spherical polygon onto the $z = 0$ plane. The technique handles polygons directly, without first partitioning them into triangles, and is ideally suited for stratified sampling. The Jacobian of the bijection from the unit square to the polygon can be made arbitrarily close to the cosine density, making the statistical bias as close to zero as desired. After preprocessing a polygon with n vertices, which can be done in $O(n^2 \log n)$

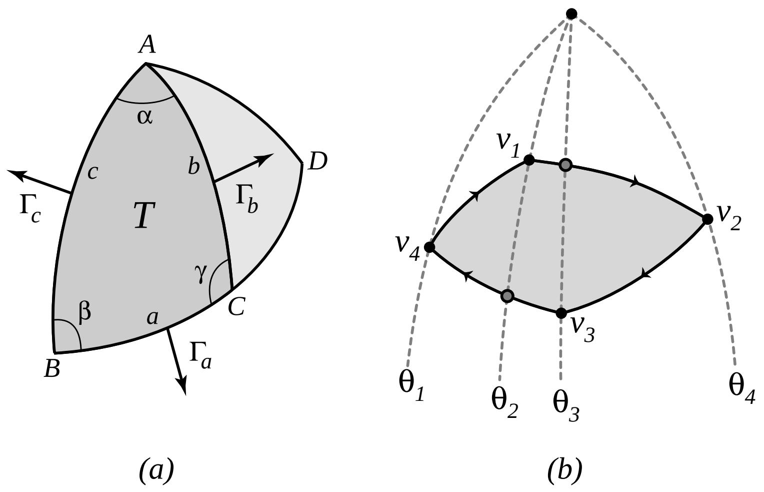


Figure 4.8: *(a) Spherical triangle T. We consider the projected area of triangle T as a function of α, keeping vertices A and B, and angle β fixed. (b) Partitioning a spherical polygon by great circles passing through the poles and the vertices.*

time, each sample can be generated in $O(n)$ time.

Let $\mathcal{P}$ denote a spherical polygon. To help in defining the mapping $\psi : [0,1]^2 \to \mathcal{P}$, we first derive several basic expression that pertain to spherical triangles. Let T be a spherical triangle, and consider the family of sub-triangles shown in Figure 4.8a, where the unit vectors A and B and the internal angle β are all fixed, but the internal angle α is allowed to sweep from 0 to the last vertex. Our task in this section is to express a (which is the length of edge BC) and $\cos b$ as functions of α. From equation (4.38) we have

$$\cos a \;=\; \frac{\cos\alpha + \cos\beta\cos\gamma}{\sin\beta\sin\gamma}. \tag{4.45}$$

Let Γ_a, Γ_b, and Γ_c denote the outward unit normals for each edges of the triangle, as shown in Figure 4.8a. Then $\cos\gamma = -\Gamma_a \cdot \Gamma_b$, where Γ_b can be expressed as

$$\Gamma_b \;=\; -\Gamma_c \cos\alpha + (\Gamma_c \times A)\sin\alpha. \tag{4.46}$$

Also, from equation (4.37) we have $\sin\gamma = \sin c\, \sin\alpha / \sin a$. Therefore,

$$a(\alpha) \;=\; \tan^{-1}\left(\frac{\sin\alpha}{c_1\cos\alpha - c_2\sin\alpha}\right), \tag{4.47}$$

where the constants c_1 and c_2 are given by

$$c_1 = \frac{(\Gamma_a \cdot \Gamma_c)\cos\beta + 1}{\sin\beta\,\sin c}, \quad c_2 = \frac{(\Gamma_a \cdot \Gamma_c \times A)\cos\beta}{\sin\beta\,\sin c}. \tag{4.48}$$

These constants depend only on the fixed features of the triangle, as the vectors Γ_a and Γ_c do not depend on α. It is now straightforward to find $\cos b$ as a function of α, which we shall denote by $z(\alpha)$. Specifically,

$$z(\alpha) \;=\; (B \cdot \mathbf{N}) \cos a(\alpha) \;+\; (D \cdot \mathbf{N}) \sin a(\alpha), \tag{4.49}$$

where D is a point on the sphere that is orthogonal to B, and on the great circle through B and C. That is,

$$D \;=\; (\mathbf{I} - BB^{\mathrm{T}})C. \tag{4.50}$$

We now show how to sample an arbitrary spherical polygon according to a cosine distribution. The function $a(\alpha)$ will be used to invert a cumulative marginal distribution over the polygon, as a great arc sweeps across the polygon, while $z(\alpha)$ will be used to sample one-dimensional vertical slices of the polygon.

4.4.1 The Cumulative Marginal Distribution

We break the problem of computing the bijection $\psi : [0,1]^2 \to \mathcal{P}$ into two parts. First, we define a sequence of sub-polygons of $\mathcal{P}$ in much the same way that we parameterized the triangle T above; that is, we define $\mathcal{P}(\theta)$ to be the intersection of $\mathcal{P}$ with a lune [1] whose internal angle is θ, and with one edge passing through an extremal vertex of $\mathcal{P}$. Next we define a *cumulative marginal distribution* $F(\theta)$ that gives the area of polygon $\mathcal{P}(\theta)$ projected onto the plane orthogonal to **N**, which is simply the cosine-weighted area of $\mathcal{P}$. Then F is a strictly monotonically increasing function of θ. By inverting this function we arrive at the first component of our sampling algorithm. That is, if ξ_1 is a uniformly distributed random variable in $[0,1]$, and if $\hat{\theta}$ is given by

$$\hat{\theta} \;=\; F^{-1}(\rho\,\xi_1), \tag{4.51}$$

then $\hat{\theta}$ defines the great circle from which to draw a sample.

To find F, we first consider the spherical triangle T and its family of sub-triangles. The projected area of the triangle T, which we denote by ρ, follows immediately from Lambert's formula for computing the irradiance from a polygonal luminaire [2]. That is

$$\rho \;=\; -(a\Gamma_a + b\Gamma_b + c\Gamma_c) \cdot \mathbf{N}, \tag{4.52}$$

[1] A *lune* is a spherical triangle with exactly two vertices, which are antipodal.

where Γ_a, Γ_b, and Γ_c are outward normals of the triangle T, as shown in Figure 4.8a. If we now constrain T to be a *polar triangle*, with vertex A at the pole of the hemisphere ($A = \mathbf{N}$), then ρ becomes a very simple function of α. Specifically,

$$\rho(\alpha) \;=\; -a(\alpha)\,(\Gamma_a \cdot \mathbf{N}), \tag{4.53}$$

where $\Gamma_a \cdot \mathbf{N}$ is fixed; this follows from the fact that both Γ_b and Γ_c are orthogonal to $\mathbf{N}$. Equation (4.53) allows us to easily compute the function $F(\alpha)$ for any collection of spherical polygons whose vertices all lie on the lune with vertices at A and $-A$ as shown in Figure 4.10, where we restrict our attention to the *positive* or upper half of the lune. Thus,

$$F(\theta) \;=\; \sum_{i=1}^{k} \eta_i\, a_i(\theta - \theta_k), \tag{4.54}$$

for $\theta \in [\theta_k, \theta_{k+1}]$, where the constants $\eta_1, \eta_2, \ldots, \eta_k$ account for the slope and orientation of the edges; that is, edges that result in clockwise polar triangles are positive, while those forming counter-clockwise triangles are negative.

We now extend $F(\theta)$ to a general spherical polygon $\mathcal{P}$ by slicing $\mathcal{P}$ into lunes with the above property; that is, we partition $\mathcal{P}$ into smaller polygons by passing a great arc through each vertex, as shown in Figure 4.8b. Then for any spherical polygon, we can evaluate $F(\theta)$ exactly for any value of θ by virtue of equation (4.47). The resulting function F is a piecewise-continuous strictly monotonically increasing function with at most $n - 2$ discontinuities, where n is the number of vertices in the polygon. See Figure 4.9. This function is precisely the *cumulative marginal distribution function* that we must invert to perform the first stage of cosine-weighted sampling. Because it is monotonically increasing, its inverse is well-defined.

4.4.2 The Sampling Algorithm

Given two variables ξ_1 and ξ_2 in the interval $[0, 1]$ we will compute the corresponding point $\mathbf{P} = \psi(\xi_1, \xi_2)$ in the polygon $\mathcal{P}$. We use ξ_1 to determine an angle $\hat{\theta}$, as described above, and ξ_2 to select the height $\hat{z}$ according to the resulting *conditional density* defined along the intersection of the polygon $\mathcal{P}$ and the great circle at $\hat{\theta}$.

To compute $\hat{\theta}$ using equation (4.51), we proceed in two steps. First, we find the lune from which $\hat{\theta}$ will be drawn. This corresponds to finding the integer k such

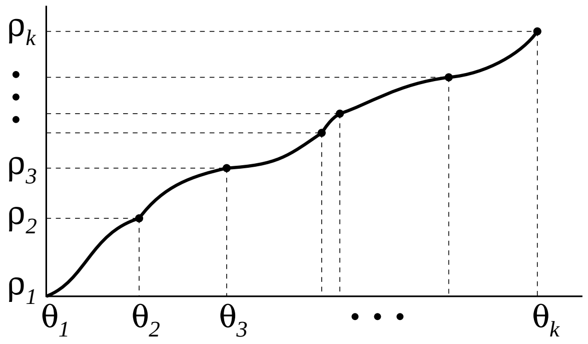


Figure 4.9: *The cumulative marginal distribution function F as a function of the angle θ. At each value of θ_i, the abscissa is the form factor from the origin to the polygon that is within the range $[\theta_1, \theta_i]$. This function is strictly monotonically increasing, with at most $n - 2$ derivative discontinuities, where n is the number of vertices in the polygon. The fluctuations in F have been greatly exaggerated for purposes of illustration.*

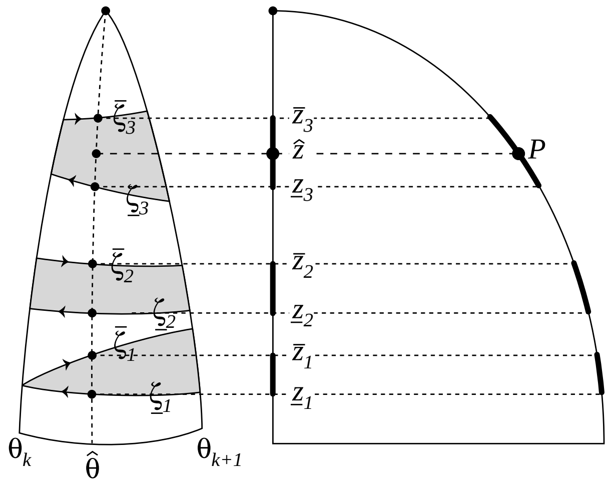


Figure 4.10: *On the left is an illustration of a single lune with a collection of arcs passing through it, and the points at which a great circle at $\hat{\theta}$ intersects them. On the right is a cross-section of the circle, showing the heights $\underline{z}_1$, $\overline{z}_1$, corresponding to these intersection points.*

that

$$\frac{\rho_k}{\rho_{tot}} \;\le\; \xi_1 \;\le\; \frac{\rho_{k+1}}{\rho_{tot}}. \tag{4.55}$$

Next, we must invert F as it is defined on this interval. Given the nature of F, as defined in equations (4.47) and (4.54), it is unlikely that this can be done symbolically in general, so we seek a numerical approximation. This is the *only* step in the algorithm which is not computed exactly; thus, any bias that is introduced in the sampling is a result of this step alone.

Approximate numerical inversion is greatly simplified by the nature of F within each lune. Since F is extremely smooth and strictly monotonic, we can approximate F^{-1} directly to high accuracy with a low-order polynomial. For example, we may use

$$F^{-1}(x) \;\approx\; a + bx + cx^2 + dx^3, \tag{4.56}$$

where we set

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \;=\; V^{-1} \begin{bmatrix} \theta_k \\ \theta_k + \delta_1 \\ \theta_k + \delta_2 \\ \theta_{k+1} \end{bmatrix}. \tag{4.57}$$

Here V is the Vandermonde matrix formed from $F(\theta_k)$, $F(\theta_k + \delta_1)$, $F(\theta_k + \delta_2)$, and $F(\theta_{k+1})$. Coupling this approximation with a Newton iteration can, of course, compute the function inverse to any desired numerical accuracy, making the bias effectively zero. However, such a high degree of accuracy is not warranted for a typical Monte Carlo simulation.

Once the angle $\hat{\theta}$ has been computed using ξ_1, we then compute $\hat{z}$ using ξ_2. This corresponds to sampling $\hat{z}$ according to the *conditional density function* corresponding to the choice of $\hat{\theta}$. This conditional density function is defined on the intervals

$$[\underline{z}_1, \overline{z}_1] \cup [\underline{z}_2, \overline{z}_2] \cup \cdots \cup [\underline{z}_n, \overline{z}_n],$$

which correspond to the intersection points shown in Figure 4.10. These intervals are computed using equation (4.49). The conditional density is proportional to z^2 within these intervals, which distributes the samples vertically according to the cosine of the angle from the pole. The most costly part of sampling according to this density is normalization. We define

$$Z_j \;\equiv\; \sum_{i=1}^{j} \left(\overline{z}_i^2 - \underline{z}_i^2 \right). \tag{4.58}$$

Then Z_n is the normalization constant. The random variable ξ_2 then selects the interval by finding $1 \le \ell \le n$ such that

$$\frac{Z_{\ell-1}}{Z_n} \;\le\; \xi_2 \;\le\; \frac{Z_\ell}{Z_n} \tag{4.59}$$

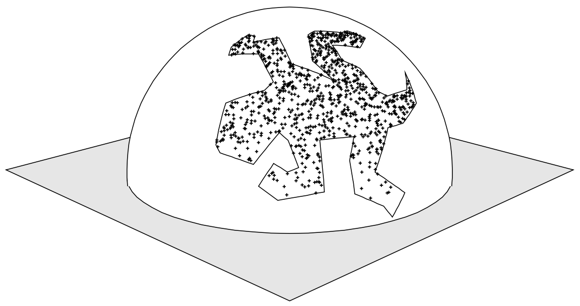


Figure 4.11: *A non-convex spherical polygon with cosine-weighted samples generated with the proposed mapping.*

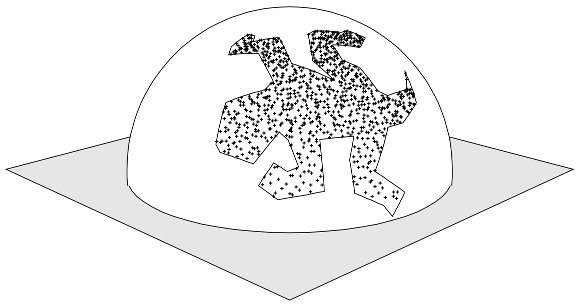


Figure 4.12: *A non-convex spherical polygon with cosine-weighted and stratified samples generated with the proposed mapping.*

where $Z_0 \equiv 0$. Finally, the height of $\mathbf{P} = \psi(\xi_1, \xi_2)$ is

$$\hat{z} \;=\; \sqrt{\xi_2 - Z_{\ell-1} + \underline{z}_\ell}, \tag{4.60}$$

and the point itself is

$$\mathbf{P} \;=\; (\omega \cos\hat{\theta},\, \omega \sin\hat{\theta},\, \hat{z}), \tag{4.61}$$

where $\omega = \sqrt{1 - \hat{z}^2}$.

The algorithm described above also works for spherical polygons $\mathcal{P}$ that surround the pole of the sphere. In this case, each lune has an odd number of segments crossing it, and $\overline{z}_n = 1$ must be added to the list of heights defined by each $\hat{\theta}$ in sampling from the conditional distribution.

The algorithm described above is somewhat more costly than the algorithm for uniform sampling of spherical triangles [3] for two reasons: 1) evaluating piecewise continuous functions requires some searching, and 2) the cumulative marginal distribution cannot be inverted exactly. Furthermore, the sampling algorithm requires some preprocessing to make both of these operations efficient and accurate.

Pre-processing includes partitioning the polygon into lunes, computing the constants c_1 and c_2 defined in equation (4.48) for each resulting edge, and sorting the line segments within each lune into increasing order. In the worst case, there may be $n-2$ lunes, with $\Omega(n)$ of them containing $\Omega(n)$ segments. Thus, creating them and sorting them requires $O(n^2 \log n)$ in the worst case. For convex polygons, this drops to $O(n)$, since there can be only two segments per lune.

Once the pre-processing is done, samples can be generated by searching for the appropriate $[\theta_k, \theta_{k+1}]$ interval, which can be done in $O(\log n)$ time, and then sampling according to the conditional distribution, which can be done on $O(n)$ time. The latter cost is the dominant one because all of the intervals must be formed for normalization. Therefore, in the worst case, the cost of drawing a sample is $O(n)$; however, for convex polygons this drops to $O(\log n)$.

Figure 4.2 shows 900 samples in a spherical quadrilateral, distributed according to the cosine distribution. Note that more of the samples are clustered near the pole than the horizon. Stratification was performed by mapping "jittered" points from the unit square onto the quadrilateral. Figures 4.11 and 4.12 show 900 samples distributed according to the cosine density within a highly non-convex spherical polygon. These samples were generated without first partitioning the polygon into triangles. In both of the test cases, the cumulative marginal distribution function F is very nearly piecewise linear, and its inverse can be computed to extremely high accuracy with a piecewise cubic curve.

Chapter 5

Combining Sampling Strategies

By Jim Arvo

5.1 Introduction

In this chapter we explore the idea of constructing effective random variables for Monte Carlo integration by combining two or more simpler random variables. For instance, suppose that we have at our disposal a convenient means of sampling the solid angle subtended by a luminaire, and also a means of sampling a brdf; how are these to be used in concert to estimate the reflected radiance from a surface? While each sampling method can itself serve as the basis of an importance sampling scheme, in isolation neither can reliably predict the shape of the resulting integrand. The problem is that the shape of the brdf may make some directions "important" (i.e. likely to make a large contribution to the integral) while the luminaire, which is potentially orders of magnitude brighter than the indirect illumination, may make other directions "important." The question that we shall address is how to construct an importance sampling method that accounts for all such "hot spots" by combining available sampling methods, but without introducing statistical bias. The following discussion closely parallels the work of Veach [90], who was the first to systematically explore this idea in the context of global illumination.

To simplify the discussion, let us assume that we are attempting to approximate some quantity $\mathcal{I}$, which is given by the integral of an unknown and potentially ill-behaved function f over the domain D:

$$\mathcal{I} \;=\; \int_D f(x)\, dx. \tag{5.1}$$

For instance, f may be the product of incident radiance (direct and indirect), a reflectance function, and a visibility factor, and D may be the either a collection of surfaces or the hemisphere of incident directions; in cases such as these, $\mathcal{I}$ may represent reflected radiance. In traditional *importance sampling*, we select a *probability density function* (pdf) p over D and rewrite the integral as

$$\mathcal{I} \;=\; \int_D \left[\frac{f(x)}{p(x)}\right] p(x)\,dx \;=\; \left\langle \frac{f(\mathbf{X})}{p(\mathbf{X})} \right\rangle, \tag{5.2}$$

where $\mathbf{X}$ denotes a random variable on the domain D distributed according to the pdf p, and $\langle\cdot\rangle$ denotes the *expected value* of a random variable. The second equality in equation (5.2) is simply the definition of expected value. It follows immediately that the *sample mean* of the new random variable $f(\mathbf{X})/p(\mathbf{X})$ is an estimator for $\mathcal{I}$; that is, if

$$\mathcal{E} \;=\; \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{X}_i)}{p(\mathbf{X}_i)}, \tag{5.3}$$

for $N \geq 1$, where $\mathbf{X}_1, \ldots, \mathbf{X}_N$ are iid (independent identically distributed) random variables, each distributed according to the pdf p, then $\langle\mathcal{E}\rangle = \mathcal{I}$. Consequently, $\mathcal{E} \approx \mathcal{I}$, and the quality of the approximation can be improved by increasing N, the number of samples, and/or by increasing the similarity between the original integrand f and the pdf p. Since evaluating $f(\mathbf{X}_i)$ is potentially very costly, we wish to pursue the second option to the extent possible. This is precisely the rationale for importance sampling.

5.2 Using Multiple PDFs

Now let us suppose that we have k distinct pdfs, $p_1, p_2, \ldots, p_k$, that each mimic some potential "hot spot" in the integrand; that is, each concentrates samples in a region of the domain where the integrand may be relatively large. For instance, p_1 may sample according to the brdf, concentrating samples around specular directions of glossy surfaces, while $p_2, \ldots, p_k$ sample various luminaires or potential specular reflections. Let us further suppose that for each p_i we draw N_i iid samples, $\mathbf{X}_{i,1}, \mathbf{X}_{i,2}, \ldots, \mathbf{X}_{i,N_i}$, distributed according to p_i. Our goal is to combine them into an estimator $\mathcal{E}$ that has several desirable properties. In particular, we wish to ensure that

1. $\langle\mathcal{E}\rangle \;=\; \mathcal{I}$,

2. $\mathcal{E}$ is relatively easy to compute,

3. var$(\mathcal{E})$ is small.

That is, we wish to have the expected value of $\mathcal{E}$ match the actual value of the integral, $\mathcal{I}$, to pay a low computational price for drawing each sample, and to reduce the variance of $\mathcal{E}$ as much as possible, thereby reducing the number of samples required to attain a reliable approximation. The first requirement ensures that the estimator is *unbiased*. Unbiased estimators have the highly desirable property that they allow us to converge to the exact answer by taking a sufficiently large number of samples. As a general rule, this is the first property that any random variable designed for Monte Carlo integration should possess [40, 41].

As we shall see, there is a large family of functions ϕ that meet property 1, leaving much flexibility in choosing one that meets both properties 2 and 3. We begin by identifying such a class of estimators, then imposing the other constraints. First, consider an estimator of the form

$$\mathcal{E}_\phi \;\equiv\; \sum_{i=1}^{k}\sum_{j=1}^{N_i} \phi_i(\mathbf{X}_{i,j}) f(\mathbf{X}_{i,j}), \tag{5.4}$$

for some suitable choice of the functions ϕ_i. That is, let us allow a different function ϕ_i to be associated with the samples drawn from each pdf p_i, and also allow the weight of each sample to depend on the sample itself. Equation (5.4) is extremely general, and also reasonable, as we can immediately ensure that $\mathcal{E}_\phi$ is unbiased by constraining the functions ϕ_i to be of the form

$$\phi_i(x) \;\equiv\; \frac{w_i(x)}{N_i\, p_i(x)}, \tag{5.5}$$

where for all $x \in D$ and $1 \le i \le k$, the *weighting functions* w_i satisfy

$$w_i(x) \ge 0, \tag{5.6}$$

$$w_1(x) + \cdots + w_k(x) = 1. \tag{5.7}$$

To see that the resulting estimator is unbiased, regardless of the choice of the weighting functions w_i, provided that they satisfy constraints (5.6) and (5.7), let us define $\mathcal{E}_w$ to be the estimator $\mathcal{E}_\phi$ where the ϕ_i are of the form shown in equation (5.5), and observe that

$$
\begin{aligned}
\langle \mathcal{E}_w \rangle &= \left\langle \sum_{i=1}^{k} \sum_{j=1}^{N_i} \phi_i(\mathbf{X}_{i,j}) f(\mathbf{X}_{i,j}) \right\rangle \\
&= \sum_{i=1}^{k} \sum_{j=1}^{N_i} \langle \phi_i(\mathbf{X}_{i,j}) f(\mathbf{X}_{i,j}) \rangle \\
&= \sum_{i=1}^{k} \sum_{j=1}^{N_i} \int_D \phi_i(x) f(x) p_i(x)\, dx \\
&= \sum_{i=1}^{k} \int_D \frac{w_i(x)}{p_i(x)} f(x) p_i(x)\, dx \\
&= \int_D f(x) \left[\sum_{i=1}^{k} w_i(x) \right] dx \\
&= \int_D f(x)\, dx \\
&= \mathcal{I}.
\end{aligned}
$$

Thus, by considering only estimators of the form $\mathcal{E}_w$, we may henceforth ignore property 1 and concentrate strictly on selecting the weighting functions w_i so as to satisfy the other two properties.

5.3 Possible Weighting Functions

In some sense the most obvious weighting functions to employ are given by

$$
w_i(x) \equiv \frac{c_i p_i(x)}{q(x)}, \tag{5.8}
$$

where

$$
q(x) \equiv c_1 p_1(x) + \cdots + c_k p_k(x), \tag{5.9}
$$

is a pdf obtained by taking a convex combination of the original pdfs; that is, the constants c_i satisfy $c_i \geq 0$ and $c_1 + \cdots + c_k = 1$. Clearly, these w_i are positive and sum to one at each x; therefore the resulting estimator is unbiased, as shown above. This particular choice is "obvious" in the sense that it corresponds exactly to classical importance sampling based on the pdf defined in equation (5.9), when a very natural constraint is imposed on $N_1, \ldots, N_k$. To see this, observe that

$$\begin{aligned} \mathcal{E}_w &= \sum_{i=1}^{k} \sum_{j=1}^{N_i} \frac{w_i(\mathbf{X}_{i,j})}{N_i\, p_i(\mathbf{X}_{i,j})} f(\mathbf{X}_{i,j}) \\ &= \sum_{i=1}^{k} \left[\frac{1}{N_i} \sum_{j=1}^{N_i} \frac{c_i}{q(\mathbf{X}_{i,j})} f(\mathbf{X}_{i,j}) \right] \\ &= \sum_{i=1}^{k} \frac{c_i}{N_i} \left[\sum_{j=1}^{N_i} \frac{f(\mathbf{X}_{i,j})}{q(\mathbf{X}_{i,j})} \right]. \end{aligned} \tag{5.10}$$

Now, let $N = N_1 + \cdots + N_k$ be the total number of samples, and let us further assume that the samples have been partitioned among the pdfs $p_1, \ldots, p_k$ in proportion to the weights $c_1, \ldots, c_k$, that is, with $N_i = c_i N$. Then the ratio c_i / N_i is constant, and equation (5.10) simplifies to

$$\mathcal{E}_w = \frac{1}{N} \sum_{i=1}^{k} \sum_{j=1}^{N_i} \frac{f(\mathbf{X}_{i,j})}{q(\mathbf{X}_{i,j})}. \tag{5.11}$$

Note that in equation (5.11) all samples are handled in exactly the same manner; that is, the weighting of the samples does not depend on i, which indicates the pdfs they are distributed according to. This is precisely the formula we would obtain if we began with q as our pdf for importance sampling. Adopting Veach's terminology, we shall refer to this particular choice of weighting functions as the *balance heuristic* [90]. Other possibilities for the weighting functions, which are also based on convex combinations of the original pdfs, include

$$w_i(x) = \begin{cases} 1 & \text{if } c_i p_i(x) = \max_j c_j p_j(x) \\ 0 & \text{otherwise} \end{cases}, \tag{5.12}$$

and

$$w_i(x) = c_i p_i^m(x) \left[\sum_{j=1}^{k} c_j p_j^m(x) \right]^{-1}, \tag{5.13}$$

for some exponent $m \geq 1$. Again, we need only verify that these weighting functions are non-negative and sum to one for all x to verify that they give rise to unbiased estimators. Note, also, that each of these strategies is extremely simple to compute, thus satisfying property 2 noted earlier.

5.4 Obvious is also Nearly Optimal

Let $\widehat{\mathcal{E}}_w$ denote an estimator that incorporates the balance heuristic, and let $\mathcal{E}_w$ be an estimator with any other valid choice of weighting function. Veach has shown [90] that

$$\operatorname{var}\left(\widehat{\mathcal{E}}_w\right) \;\le\; \operatorname{var}(\mathcal{E}_w) + \mathcal{I}^2 \left[\frac{1}{N_{\text{min}}} - \frac{1}{N}\right], \tag{5.14}$$

where $N_{\text{min}} = \min_i N_i$. Inequality (5.14) indicates that the variance of the estimator $\widehat{\mathcal{E}}_w$ compares favorably with the optimal strategy, which would be infeasible to determine in any case. In fact, as the number of samples of the least-sampled pdf approaches to infinity, the balance heuristic approaches optimality.

Fortunately, the balance heuristic is also extremely easy to apply; it demands very little beyond the standard requirements of importance sampling, which include the ability to generate samples distributed according to each of the original pdfs p_i, and the ability to compute the density of a given point x with respect to each of the original pdfs [41]. This last requirement simply means that for each $x \in D$ and $1 \le i \le k$, we must be able to evaluate $p_i(x)$. Thus, $\widehat{\mathcal{E}}_w$ satisfies all three properties noted earlier, and is therefore a reasonable heuristic in itself for combining multiple sampling strategies.

Chapter 6

Quasi-Monte Carlo Sampling

By Art B. Owen

In Monte Carlo (MC) sampling the sample averages of random quantities are used to estimate the corresponding expectations. The justification is through the law of large numbers. In quasi-Monte Carlo (QMC) sampling we are able to get a law of large numbers with deterministic inputs instead of random ones. Naturally we seek deterministic inputs that make the answer converge as quickly as possible. In particular it is common for QMC to produce much more accurate answers than MC does. Keller [39] was an early proponent of QMC methods for computer graphics.

We begin by reviewing Monte Carlo sampling and showing how many problems can be reduced to integrals over the unit cube $[0,1)^d$. Next we consider how stratification methods, such as jittered sampling, can improve the accuracy of Monte Carlo for favorable functions while doing no harm for unfavorable ones. Method of multiple-stratification such as Latin hypercube sampling (n-rooks) represent a significant improvement on stratified sampling. These stratification methods balance the sampling points with respect to a large number of hyperrectangular boxes. QMC may be thought of as an attempt to take this to the logical limit: how close can we get to balancing the sample points with respect to every box in $[0,1)^d$ at once? The answer, provided by the theory of discrepancy is surprisingly far, that the result produces a significant improvement compared to MC. This chapter concludes with a presentation of digital nets, integration lattices and randomized QMC.

6.1 Crude Monte Carlo

As a frame of reference for QMC, we recap the basics of MC. Suppose that the average we want to compute is written as an integral

$$I = \int_{\mathcal{D}} f(x)q(x)dx. \tag{6.1}$$

The set $\mathcal{D} \subseteq \mathbb{R}^d$ is the domain of interest, perhaps a region on the unit sphere or in the unit cube. The function q is a probability density function on $\mathcal{D}$. That is $q(x) \geq 0$ and $\int_{\mathcal{D}} q(x)dx = 1$. The function f gives the quantity whose expectation we seek: I is the expected value of $f(x)$ for random x with density q on $\mathcal{D}$.

In crude Monte Carlo sampling we generate n independent samples $x_1, \dots, x_n$ from the density q and estimate I by

$$\hat{I} = \hat{I}_n = \frac{1}{n}\sum_{i=1}^{n} f(x_i). \tag{6.2}$$

The strong law of large numbers tells us that

$$\Pr\big(\lim_{n\to\infty} \hat{I}_n = I\big) = 1. \tag{6.3}$$

That is, crude Monte Carlo always converges to the right answer as n increases without bound.

Now suppose that f has finite variance $\sigma^2 = \mathrm{Var}(f(x)) \equiv \int_{\mathcal{D}}(f(x)-I)^2 q(x)dx$. Then $E((\hat{I}_n - I)^2) = \sigma^2/n$ so the root mean square error (RMSE) of MC sampling is $O(1/\sqrt{n})$. This rate is slow compared to that of classical quadrature rules (Davis and Rabinowitz [16]) for smooth functions in low dimensions. Monte Carlo methods can improve on classical ones for problems in high dimensions or on discontinuous functions.

A given integration problem can be written in the form (6.1) in many different ways. First, let p be a probability density on $\mathcal{D}$ such that $p(x) > 0$ whenever $q(x)|f(x)| > 0$. Then

$$I = \int_{\mathcal{D}} f(x)q(x)dx = \int_{\mathcal{D}} \frac{f(x)q(x)}{p(x)} p(x)dx$$

and we could as well sample $x_i \sim p(x)$ and estimate I by

$$\hat{I}_p = \hat{I}_{n,p} = \frac{1}{n}\sum_{i=1}^{n} \frac{f(x_i)q(x_i)}{p(x_i)}. \tag{6.4}$$

The RMSE can be strongly affected, for better or worse, by this re-expression, known as importance sampling. If we are able to find a good p that is nearly proportional to fq then we can get much better estimates.

Making a good choice of density p is problem specific. Suppose for instance, that one of the components of x describes the angle $\theta = \theta(x)$ between a ray and a surface normal. The original version of f may include a factor of $\cos(\theta)^\eta$ for some $\eta > 0$. Using a density $p(x) \propto q(x)\cos(\theta)^\eta$ corresponds to moving the cosine power out of the integrand and into the sampling density.

We will suppose that a choice of p has already been made. There is also the possibility of using a mixture of sampling densities p_j as with the balance heuristic of Veach and Guibas [88, 89]. This case can be incorporated by increasing the dimension of x by one, and using that variable to select j from a discrete distribution.

Monte Carlo sampling of $x \sim p$ over $\mathcal{D}$ almost always uses points from a pseudo-random number generator simulating the uniform distribution on the interval from 0 to 1. We will take this to mean the uniform distribution on the half-open interval $[0, 1)$. Suppose that it takes d^* uniform random variables to simulate a point in the d dimensional domain $\mathcal{D}$. Often $d^* = d$ but sometimes $d^* = 2$ variables from $[0, 1)$ can be used to generate a point within a surface element in $d = 3$ dimensional space. In other problems we might use $d^* > d$ random variables to generate a p distributed point in $\mathcal{D} \subseteq \mathbb{R}^d$. Chapter 4 describes general techniques for transforming $[0, 1)^d$ into $\mathcal{D}$ and provides some specific examples of use in ray tracing. Devroye [17] is a comprehensive reference on techniques for transforming uniform random variables into one's desired random objects.

Suppose that a point having the $U[0, 1)^{d^*}$ distribution is transformed into a point $\tau(x)$ having the density p on $\mathcal{D}$. Then

$$I = \int_{\mathcal{D}} \frac{f(x)q(x)}{p(x)} p(x)dx = \int_{[0,1)^{d^*}} \frac{f(\tau(x))q(\tau(x))}{p(\tau(x))} dx \equiv \int_{[0,1)^{d^*}} f^*(x)dx \tag{6.5}$$

where f^* incorporates the transformation τ and the density q. Then I is estimated by

$$\hat{I} = \frac{1}{n} \sum_{i=1}^{n} \frac{f(\tau(x_i))q(\tau(x_i))}{p(\tau(x_i))} = \frac{1}{n} \sum_{i=1}^{n} f^*(x_i) \tag{6.6}$$

where x_i are independent $U[0, 1)^{d^*}$ random variables.

Equation (6.5) expresses the original MC problem (6.1) as one of integrating a function f^* over the unit cube in d^* dimensions. We may therefore reformulate the problem as finding $I = \int_{[0,1)^d} f(x)dx$. The new d is the old d^* and the new f is the old f^*.

6.2 Stratification

Stratified sampling is a technique for reducing the variance of a Monte Carlo integral. It was originally applied in survey sampling (see Cochran [10]) and has been adapted in Monte Carlo methods, Fishman [21]. In stratified sampling, the domain of x is written as a union of strata $\mathcal{D} = \bigcup_{h=1}^{H} \mathcal{D}_h$ where $\mathcal{D}_j \bigcap \mathcal{D}_k = \emptyset$ if $j \neq k$. An integral is estimated from within each stratum and then combined. Following the presentation in chapter 6.1, we suppose here that $\mathcal{D} = [0,1)^d$.

Figure 6.1 shows a random sample from the unit square along with 3 alternative stratified samplings. The unit cube $[0,1)^d$ is very easily partitioned into box shaped strata like those shown. It is also easy to sample uniformly in such strata. Suppose that $a, c \in [0,1)^d$ with $a < c$ componentwise. Let $U \sim U[0,1)^d$. Then $a + (c - a)U$ interpreted componentwise is uniformly distributed on the box with lower left corner a and upper right corner c.

In the simplest form of stratified sampling, a Monte Carlo sample $x_{h1}, \dots x_{hn_h}$ is taken from within stratum $\mathcal{D}_h$. Each stratum is sampled independently, and the results are combined as

$$\hat{I}_{\text{STRAT}} = \hat{I}_{\text{STRAT}}(f) = \sum_{h=1}^{H} \frac{|\mathcal{D}_h|}{n_h} \sum_{i=1}^{n_h} f(x_{hi}), \tag{6.7}$$

where $|\mathcal{D}_h|$ is the volume of stratum $\mathcal{D}$.

For any $x \in [0,1)^d$ let $h(x)$ denote the stratum containing x. That is $x \in \mathcal{D}_{h(x)}$. The mean and variance of f within stratum h are

$$\mu_h = |\mathcal{D}_h|^{-1} \int_{\mathcal{D}_h} f(x)dx, \quad \text{and,} \tag{6.8}$$

$$\sigma_h^2 = |\mathcal{D}_h|^{-1} \int_{\mathcal{D}_h} (f(x) - \mu_h)^2 dx \tag{6.9}$$

respectively. We can write $E(\hat{I}_{\text{STRAT}})$ as:

$$\sum_{h=1}^{H} \frac{|\mathcal{D}_h|}{n_h} \sum_{i=1}^{n_h} E(f(x_{hi})) = \sum_{h=1}^{H} |\mathcal{D}_h| \mu_h = \sum_{h=1}^{H} \int_{\mathcal{D}_h} f(x)dx = I,$$

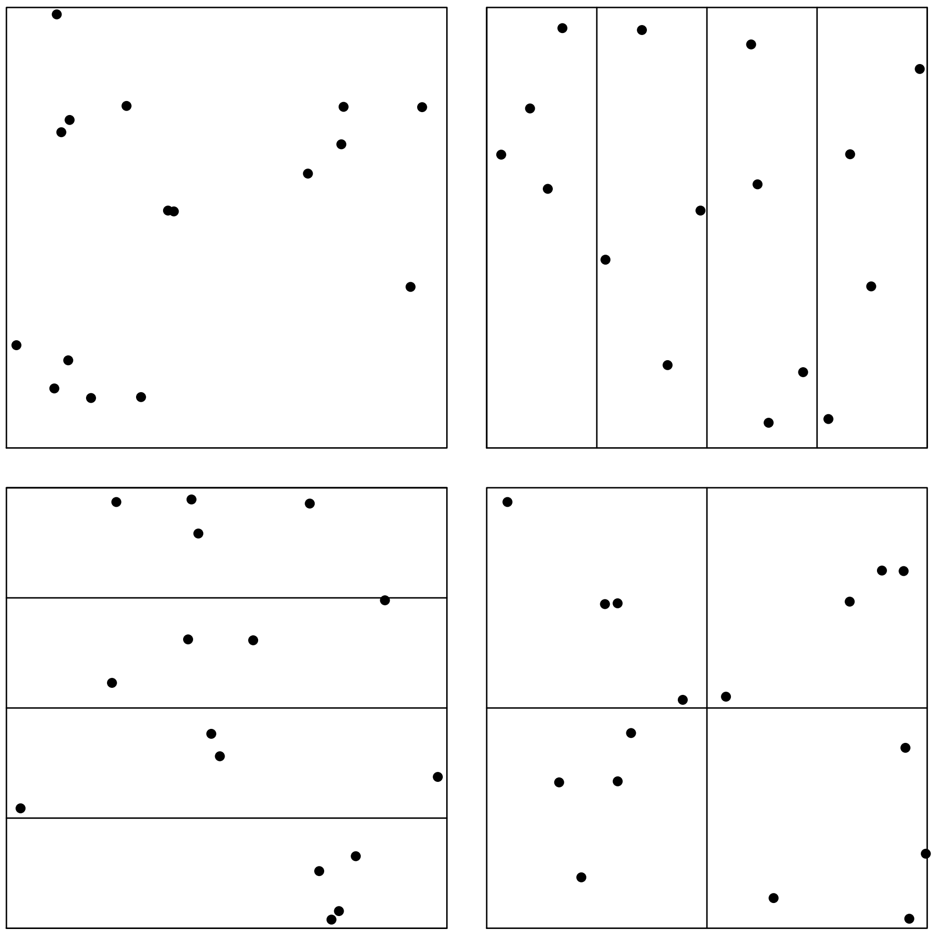


Figure 6.1: The upper left figure is a simple random sample of 16 points in $[0, 1)^2$. The other figures show stratified samples with 4 points from each of 4 strata.

so that stratified sampling is unbiased.

The variance of stratified sampling depends on the allocation of sample size n_h to strata. We will suppose that n_h is allocated proportionally, so that $n_h = n|\mathcal{D}_h|$ for the total sample size n. First we note that when $x \sim U[0, 1)^d$, then $h(x)$ is a random variable taking the value ℓ with probability $|\mathcal{D}_\ell|$. Then from a standard variance formula

$$\sigma^2 = \text{Var}(f(x)) = E(\text{Var}(f(x) \mid h(x))) + \text{Var}(E(f(x) \mid h(x))) \tag{6.10}$$

$$= \sum_{h=1}^{H} |\mathcal{D}_h|\sigma_h^2 + \sum_{h=1}^{H} |\mathcal{D}_h|(\mu_h - I)^2, \tag{6.11}$$

so that σ^2 is a sum of contributions from within and between strata. Now

$$\text{Var}(\hat{I}_{\text{STRAT}}) = \sum_{h=1}^{H} \frac{|\mathcal{D}_h|^2}{n_h}\sigma_h^2 = \frac{1}{n}\sum_{h=1}^{H} |\mathcal{D}_h|\sigma_h^2 \le \frac{\sigma^2}{n}, \tag{6.12}$$

from (6.10).

Equation (6.12) shows that stratified sampling with proportional allocation does not increase the variance. Proportional allocation is not usually optimal. Optimal allocations take $n_h \propto |\mathcal{D}_h|\sigma_h$. If estimates of σ_h are available they can be used to set n_h, but poor estimates of σ_h could result in stratified sampling with larger variance than crude MC. We will assume proportional allocation.

A particular form of stratified sampling is well suited to the unit cube. Haber [24] proposes to partition the unit cube $[0, 1)^d$ into $H = m^d$ congruent cubical regions and to take $n_h = 1$ point from each of them. This stratification is known as jittered sampling in graphics, following Cook, Porter and Carpenter [14].

Any function that is constant within strata is integrated without error by $\hat{I}_{\text{STRAT}}$. If f is close to such a function, then f is integrated with a small error. Let $\bar{f}$ be the function defined by $\bar{f}(x) = \mu_{h(x)}$, and define the residual $f_{\text{RES}}(x) = f(x) - \bar{f}(x)$. This decomposition is illustrated in Figure 6.2 for a function on $[0, 1)$. The error $\hat{I}_{\text{STRAT}} - I$ reduces to the stratified sampling estimate of the mean of f_{RES}. Stratified sampling reduces the Monte Carlo variance from $\sigma^2(f)/n$ to $\sigma^2(f_{\text{RES}})/n$.

6.3 Multiple Stratification

Suppose we can afford to sample 16 points in $[0, 1)^2$. Sampling one point from each of 16 vertical strata would be a good strategy if the function f depended primarily on the horizontal coordinate. Conversely if the vertical coordinate is the more important one, then it would be better to take one point from each of 16 horizontal strata.

It is possible to stratify both ways with the same sample, in what is known as Latin hypercube sampling (McKay, Beckman and W. J. Conover [51]) or n-rooks

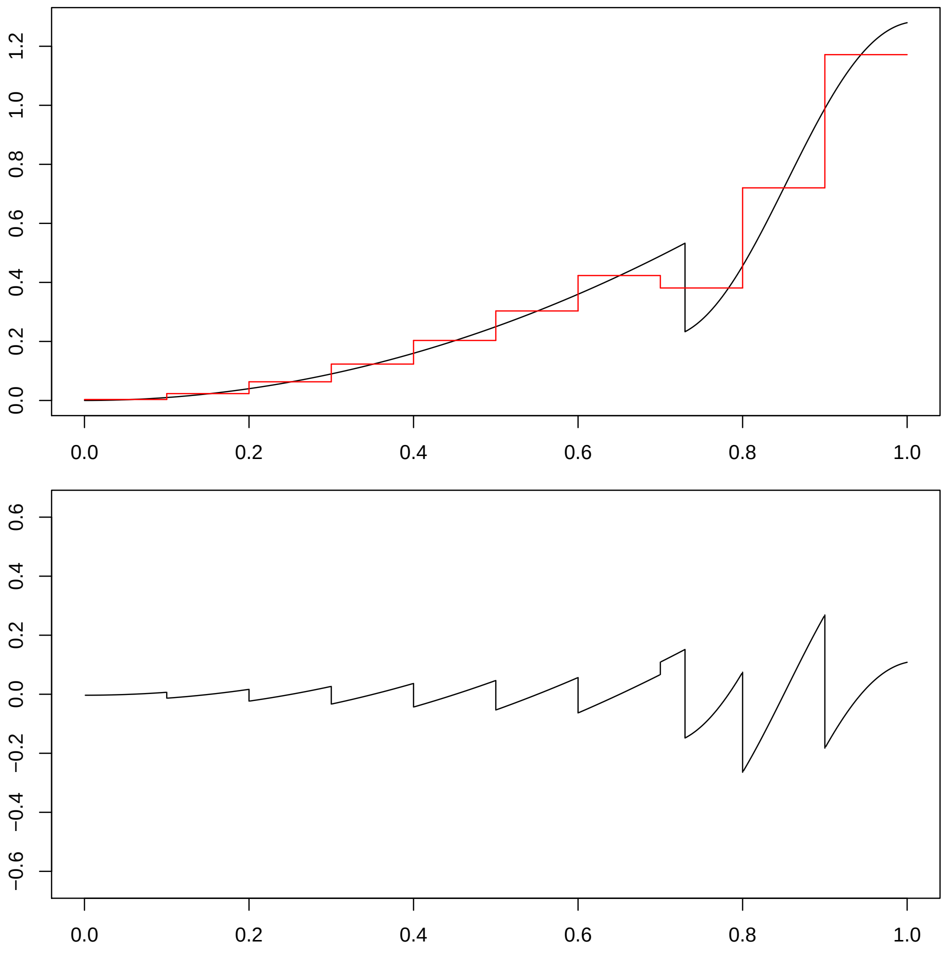


Figure 6.2: The upper plot shows a piece-wise smooth function f on $[0, 1)$. The step function is the best approximation $\bar{f}$ to f, in mean square error, among functions constant over intervals $[j/10, (j + 1)/10)$. The lower plot shows the difference $f - \bar{f}$ using a vertical scale similar to the upper plot.

sampling (Shirley [67]). Figure 6.3 shows a set of 16 points in the square, that are simultaneously stratified in each of 16 horizontal and vertical strata.

If the function f on $[0, 1)^2$ is dominated by either the horizontal coordinate or the vertical one, then we'll get an accurate answer, and we don't even need to know which is the dominant variable. Better yet, suppose that neither variable is

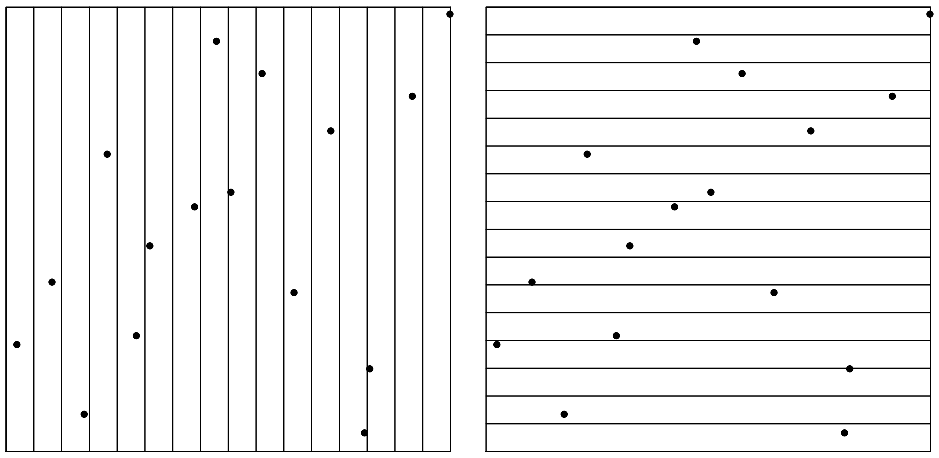


Figure 6.3: The left plot shows 16 points, one in each of 16 vertical strata. The right plot shows the same 16 points. There is one in each of 16 horizontal strata. These points form what is called a Latin hypercube sample, or an n-rooks pattern.

dominant but that

$$f(x) = f_H(x) + f_V(x) + f_{\text{RES}}(x) \tag{6.13}$$

where f_H depends only on the horizontal variable, f_V depends only on the vertical one, and the residual f_{RES} is defined by subtraction. Latin hypercube sampling will give an error that is largely unaffected by the additive part $f_H + f_V$. Stein [78] showed that the variance in Latin hypercube sampling is approximately σ^2_{RES}/n where σ^2_{RES} is the smallest variance of f_{RES} for any decomposition of the form (6.13). His result is for general d, not just $d = 2$.

Stratification with proportional allocation is never worse than crude MC. The same is almost true for Latin hypercube sampling. Owen [58] shows that for all $n \geq 2$, $d \geq 1$ and square integrable f, that

$$\text{Var}(\hat{I}_{\text{LHS}}) \leq \frac{\sigma^2}{n-1}.$$

For the worst f, Latin hypercube sampling is like using crude MC with one observation less.

The construction of a Latin hypercube sample requires uniform random permutations. A uniform random permutation of 0 through $n-1$ is one for which all $n!$ possible orderings have the same probability. Devroye [17] gives algorithms for such random permutations. One choice is to have an array $A_i = i$ for $i = 0, \dots, n-1$ and then for $j = n-1$ down to 1 swap A_j with A_k where k is uniformly and randomly chosen from 0 through j.

For $j = 1, \dots, d$, let π_j be independent uniform random permutations of $0, \dots, n-1$. Let $U_{ij} \sim U[0,1)^d$ independently for $i = 1, \dots, n$ and $j = 1, \dots, d$ and let X be a matrix with

$$X_{ij} = \frac{\pi_j(i-1) + U_{ij}}{n}.$$

Then the n rows of X form a Latin hypercube sample. That is we may take $x_i = (X_{i1}, \dots, X_{id})$. An integral estimate $\hat{I}$ is the same whatever order the $f(x_i)$ are summed. As a consequence we only need to permute $d-1$ of the d input variables. We can take $\pi_1(i-1) = i-1$ to save the cost of one random permutation.

Jittered sampling uses $n = k^2$ strata arranged in a k by k grid of squares while n-rooks provides simultaneous stratification in both an n by 1 grid and a 1 by n grid. It is natural to wonder which method is better. The answer depends on whether f is better approximated by a step function, constant within squares of size $1/k \times 1/k$ grid, or by an additive function with each term constant within narrower bins of width $1/n$. Amazingly, we don't have to choose. It is possible to arrange $n = k^2$ points in an n-rooks arrangement that simultaneously has one point in each square of a k by k grid. A construction for this was proposed independently by Chiu, Shirley and Wang [8] and by Tang [81]. The former handle more general grids of $n = k_1 \times k_2$ points. The latter reference arranges points in $[0,1)^d$ with $d \geq 2$ in a Latin hypercube such that every two dimensional projection of x_i puts one point into each of a grid of strata.

6.4 Uniformity and Discrepancy

The previous sections look at stratifications in which every cell in a rectangular grid or indeed in multiple rectangular grids gets the proper number of points. It is clear that a finite number of points in $[0,1)^d$ cannot be simultaneously stratified with respect to *every* hyper-rectangular subset of $[0,1)^d$, yet it is interesting to ask how far we might be able to go in that direction. This is a problem that has been studied since Weyl [96] originated his theory of uniform distribution. Kuipers and Niederreiter [44] summarize that theory.

Let a and c be points in $[0,1)^d$ for which $a < c$ holds componentwise, and then let $[a,c)$ denote the box of points x where $a \le x < c$ holds componentwise. We use $|[a,c)|$ to denote the d-dimensional volume of this box.

An infinite sequence of points $x_1, x_2, \cdots \in [0,1)^d$ is uniformly distributed if $\lim_{n\to\infty}(1/n)\sum_{i=1}^n 1_{a\le x_i<c} = |[a,c)|$ holds for all boxes. This means that $\hat{I}_n \to I$ for every function $f(x)$ of the form $1_{a\le x<c}$ and so for any finite linear combination of such indicators of boxes. Riemann integrable functions are well approximated by linear combinations of indicators of boxes; if the sequence (x_i) is uniformly distributed then $\lim_{n\to\infty}|\hat{I}_n - I| = 0$ for any function f that is Riemann integrable. Thus uniformly distributed sequences can be used to provide a deterministic law of large numbers.

To show that a sequence is uniformly distributed it is enough to show that $\hat{I}_n \to I$ when f is the indicator of a suitable subset of boxes. Anchored boxes take the form $[0,a)$ for some point $a \in [0,1)^d$. If $\hat{I}_n \to I$ for all indicators of anchored boxes, then the same holds for all boxes. For integers $b \ge 2$ a b-adic box is a Cartesian product of the form

$$\prod_{j=1}^d \Bigl[\frac{\ell_j}{b^{k_j}}, \frac{\ell_j+1}{b^{k_j}}\Bigr). \tag{6.14}$$

for integers $k_j \ge 0$ and $0 \le \ell_j < b^{k_j}$. When $b = 2$ the box is called dyadic. An arbitrary box can be approximated by b-ary boxes. If $\hat{I} \to I$ for all indicators of b-adic boxes then the sequence (x_i) is uniformly distributed. A mathematically more interesting result is the Weyl condition. The sequence (x_i) is uniformly distributed if and only if $\hat{I}_n \to I$ for all trigonometric polynomials $f(x) = e^{2\pi\sqrt{-1}k\cdot x}$ where $k \in \mathbb{Z}^d$.

If x_i are independent $U[0,1)^d$ variables, then (x_i) is uniformly distributed with probability one. Of course we hope to do better than random points. To that end, we need a numerical measure of how uniformly distributed a sequence of points is. These measures are called discrepancies, and there are a great many of them. One of the simplest is the star discrepancy

$$D_n^* = D_n^*(x_1,\dots,x_n) = \sup_{a\in[0,1)^d}\Bigl|\frac{1}{n}\sum_{i=1}^n 1_{0\le x_i<a} - \bigl|[0,a)\bigr|\Bigr| \tag{6.15}$$

Figure 6.4 illustrates this discrepancy. It shows an anchored box $[0,a) \in [0,1)^2$ and a list of $n = 20$ points. The anchored box has 5 of the 20 points so $(1/n)\sum_{i=1}^n 1_{0\le x_i<a} =$

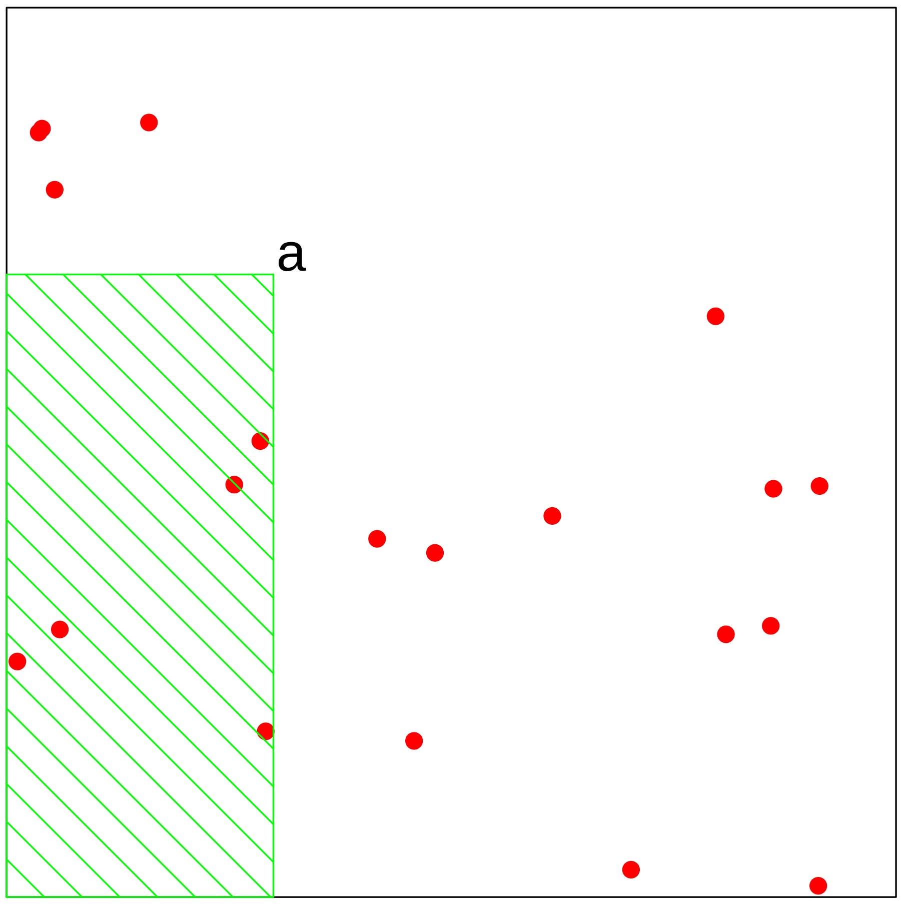


Figure 6.4: Shown are 20 points in the unit square and an anchored box (shaded) from $(0, 0)$ to $a = (.3, .7)$. The anchored box $[0, a)$ has volume 0.21 and contains a fraction $5/20 = 0.2$ of the points.

0.20. The volume of the anchored box is 0.21, so the difference is $|0.2 - 0.21| = 0.01$. The star discrepancy D_n^* is found by maximizing this difference over all anchored boxes $[0, a)$.

For $x_i \sim U[0, 1)^d$, Chung [9] showed that

$$\limsup_{n\to\infty} \frac{\sqrt{2n}D_n^*}{\sqrt{\log(\log(n))}} = 1 \tag{6.16}$$

so $D_n^* = O((\log\log(n)/n)^{1/2})$ with probability one. An iterated logarithm grows slowly with n, so D_n^* may be only slightly larger than $n^{-1/2}$ for large n.

It is known that a deterministic choice of (x_i) can yield D_n^* much smaller than (6.16). There are infinite sequences (x_i) in $[0,1)^d$ with $D_n^*(x_1,\dots,x_n) = O(\log(n)^d/n)$. Such sequences are called "low discrepancy" sequences, and some of them are described in chapter 6.5. It is suspected but not proven that infinite sequences *cannot* be constructed with $D_n^* = o(\log(n)^d/n)$; see Beck and Chen [5].

In an infinite sequence, the first m points of $x_1,\dots,x_n$ are the same for any $n \ge m$. If we knew in advance the value of n that we wanted then we might use a sequence customized for that value of n, such as $x_{n1},\dots,x_{nn} \in [0,1)^d$, without insisting that $x_{ni} = x_{n+1\,i}$. In this setting $D_n^*(x_{n1},\dots,x_{nn}) = O(\log(n)^{d-1}/n)$ is possible. The effect is like reducing d by one, but the practical cost is that such a sequence is not extensible to larger n.

There is a connection between better discrepancy and more accurate integration. Hlawka [29] proved the Koksma-Hlawka inequality

$$|\hat{I} - I| \le D_n^*(x_1,\dots,x_n)V_{\text{HK}}(f). \tag{6.17}$$

The factor $V_{\text{HK}}(f)$ is the total variation of f in the sense of Hardy and Krause. Niederreiter [56] gives the definition.

Equation (6.17) shows that a deterministic law of large numbers can be much better than the random one, for large enough n and a function f with finite variation $V_{\text{HK}}(f)$. One often does see QMC methods performing much better than MC, but equation (6.17) is not good for predicting when this will happen. The problem is that D_n^* is hard to compute, $V_{\text{HK}}(f)$ is harder still, and that the bound (6.17) can grossly overestimate the error. In some cases V_{HK} is infinite while QMC still beats MC. Schlier [66] reports that even for QMC the variance of f is more strongly related to the error than is the variation.

6.5 Digital Nets and Related Methods

Niedereitter [56] presents a comprehensive account of digital nets and sequences. We will define them below, but first we illustrate a construction for $d = 1$.

The simplest digital nets are the radical inverse sequences initiated by van der Corput [85, 86]. Let $b \ge 2$ be an integer base. The non-negative integer n can be written as $\sum_{k=1}^{\infty} n_k b^{k-1}$ where $n_k \in \{0,1,\dots,b-1\}$ and only finitely many n_k are not zero. The base b radical inverse function is $\phi_b(n) = \sum_{k=1}^{\infty} n_k b^{-k} \in$

ℓ	ℓ base 2	$\phi_2(\ell)$	
0	0.	0.000	0.000
1	1.	0.100	0.500
2	10.	0.010	0.250
3	11.	0.110	0.750
4	100.	0.001	0.125
5	101.	0.101	0.625
6	110.	0.011	0.375
7	111.	0.111	0.875

Table 6.1: The first column shows integers ℓ from 0 to 7. The second column shows ℓ in base 2. The third column reflects the digits of ℓ through the binary point to construct $\phi_2(\ell)$. The final column is the decimal version of $\phi_2(\ell)$.

$[0, 1)$. A radical inverse sequence consists of $\phi_b(i)$ for n consecutive values of i, conventionally 0 through $n - 1$.

Table 6.1 illustrates a radical inverse sequence, using $b = 2$ as van der Corput did. Because consecutive integers alternate between even and odd, the van der Corput sequence alternates between values in $[0, 1/2)$ and $[1/2, 1)$. Among any 4 consecutive van der Corput points there is exactly one in each interval $[k/4, (k + 1)/4)$ for $k = 0, 1, 2, 3$. Similarly any b^m consecutive points from the radical inverse sequence in base b are stratified with respect to b^m congruent intervals of length $1/b^m$.

If $d > 1$ then it would be a serious mistake to simply replace a stream of pseudo-random numbers by the van der Corput sequence. For example with $d = 2$ taking points $x_i = (\phi_2(2i - 2), \phi_2(2i - 1)) \in [0, 1)^2$ we would find that all x_i lie on a diagonal line with slope 1 inside $[0, 1/2) \times [1/2, 1)$.

For $d > 1$ we really need a stream of quasi-random d-vectors. There are several ways to generalize the van der Corput sequence to $d \geq 1$. The Halton [25] sequence in $[0, 1)^d$ works with d relatively prime bases $b_1, \dots, b_d$. Usually these are the first d prime numbers. Then for $i \geq 1$,

$$x_i = (\phi_2(i - 1), \phi_3(i - 1), \phi_5(i - 1), \dots, \phi_{b_d}(i - 1)) \in [0, 1)^d.$$

The Halton sequence has low discrepancy: $D_n^* = O((\log n)^d/n)$.

The Halton sequence is extensible in both n and d. For small d the points of the Halton sequence have a nearly uniform distribution. The left panel of Figure 6.5 shows a two dimensional portion of the Halton sequence using prime bases

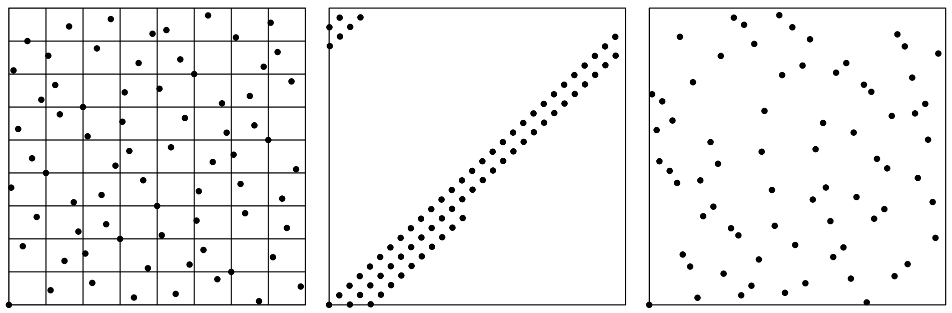


Figure 6.5: The left panel shows the first $2^3 \times 3^2 = 72$ points of the Halton sequence using bases 2 and 3. The middle panel shows the first 72 points for the 10'th and 11'th primes, 29 and 31 respectively. The right panel shows these 72 points after Faure's [20] permutation is applied.

2 and 3. The second panel shows the same points for bases 29 and 31 as would be needed with $d = 11$. While they are nearly uniform in both one dimensional projections, their two dimensional uniformity is seriously lacking. When it is possible to identify the more important components of x, these should be sampled using the smaller prime bases.

The poorer distribution for larger primes can be mitigated using a permutation of Faure [20]. Let π be a permutation of $\{0, \dots, b-1\}$. Then the radical inverse function can be generalized to $\phi_{b,\pi}(n) = \sum_{k=1}^{\infty} \pi(n_k) b^{-k}$. It still holds that any consecutive b^m values of $\phi_{b,\pi}(i)$ stratify into b^m boxes of length $1/b^m$. Faure's transformation π_b of $0, \dots, b-1$ is particularly simple. Let $\pi_2 = (0, 1)$. For even $b > 2$ take $\pi_b = (2\pi_{b/2}, 2\pi_{b/2} + 1)$, so $\pi_4 = (0, 2, 1, 3)$. For odd $b > 2$ put $k = (b-1)/2$ and $\eta = \phi_{b-1}$. Then add 1 to any member of η greater than or equal to k. Then $\pi_b = (\eta(0), \dots, \eta(k-1), k, \eta(k), \dots, \eta(b-2))$. For example with $b = 5$ we get $k = 2$, and after the larger elements are incremented, $\eta = (0, 3, 1, 4)$. Finally $\pi_5 = (0, 3, 2, 1, 4)$. The third plot in Figure 6.5 shows the effect of Faure's permutations on the Halton sequence.

Digital nets provide more satisfactory generalizations of radical inverse sequences to $d \geq 2$. Recall the b-ary boxes in (6.14). The box there has volume b^{-K} where $K = k_1 + \cdots + k_d$. Ideally we would like nb^{-K} points in every such box. Digital nets do this, at least for small enough K.

Let $b \geq 2$ be an integer base and let $m \geq t \geq 0$ be integers. A (t, m, d)*–net in base* b is a finite sequence $x_1, \dots, x_{b^m}$ for which every b-ary box of volume b^{t-m} contains exactly b^t points of the sequence.

Clearly $t = 0$ corresponds to better stratification. For given values of b, m, and d, particularly for large d, there may not exist a net with $t = 0$, and so nets with $t > 0$ are widely used.

Faure [19] provides a construction of $(0, m, p)$–nets in base p where p is a prime number. The first component of these nets is the radical inverse function in base p applied to 0 through $b^m - 1$. Figure 6.6 shows 81 points of a $(0, 4, 2)$–net in base 3. There are 5 different shapes of 3-ary box with volume $1/81$. The aspect ratios are $1 \times 1/81$, $1/3 \times 1/27$, $1/9 \times 1/9$, $1/17 \times 1/3$, and $1/81 \times 1$. Latin hypercube samples of 81 points balance the first and last of these, jittered sampling balances the third, while multi-jittered sampling balances the first, third, and fifth. A $(0, 4, 2)$–net balances 81 different 3-ary boxes of each of these 5 aspect ratios. If f is well approximated by a sum of the corresponding 405 indicator functions, then $|\hat{I} - I|$ will be small.

The extensible version of a digital net is a digital sequence. A (t, s)*–sequence in base* b is an infinite sequence (x_i) for $i \geq 1$ such that for all integers $r \geq 0$ and $m \geq t$, the points $x_{rb^m+1}, \dots, x_{(r+1)b^m}$ form a (t, m, d)–net in base b. This sequence can be expressed as an infinite stream of (t, m, d)–nets, simultaneously for all $m \geq t$. Faure [19] provided a construction of $(0, p)$-sequences in base p. Niederreiter [55] showed that construction can be extended to $(0, q)$–sequences in base q where $q = p^r$ is a power of a prime p. The Faure net shown in Figure 6.6 is in fact the first 81 points of the first two variables in a $(0, 3)$-sequence in base 3.

For $m \geq t$ and $1 \leq \lambda < b$, the first λb^m points in a (t, d)–sequence are balanced with respect to all b-ary boxes of volume b^{t-m} or larger. If n is not of the form λb^m, then the points do not necessarily balance any non-trivial b-ary boxes.

The Faure sequence and Niederreiter's generalization of it, require $b \geq d$. When the dimension is large then it becomes necessary to use a large base b, and then either b^m is very large, or m is very small. Then the Sobol' [75] sequences become attractive. They are (t, d)–sequences in base $b = 2$. The quality parameter t depends on d. Niederreiter [55] combined the methods of Sobol' and Faure, generating new sequences. Any (t, s)–sequence is a low discrepancy sequence, as shown in Niederreiter [56]. The smallest values of t, for given b and d, among known (t, d)–sequence constructions, are those of Niedereitter and Xing [54].

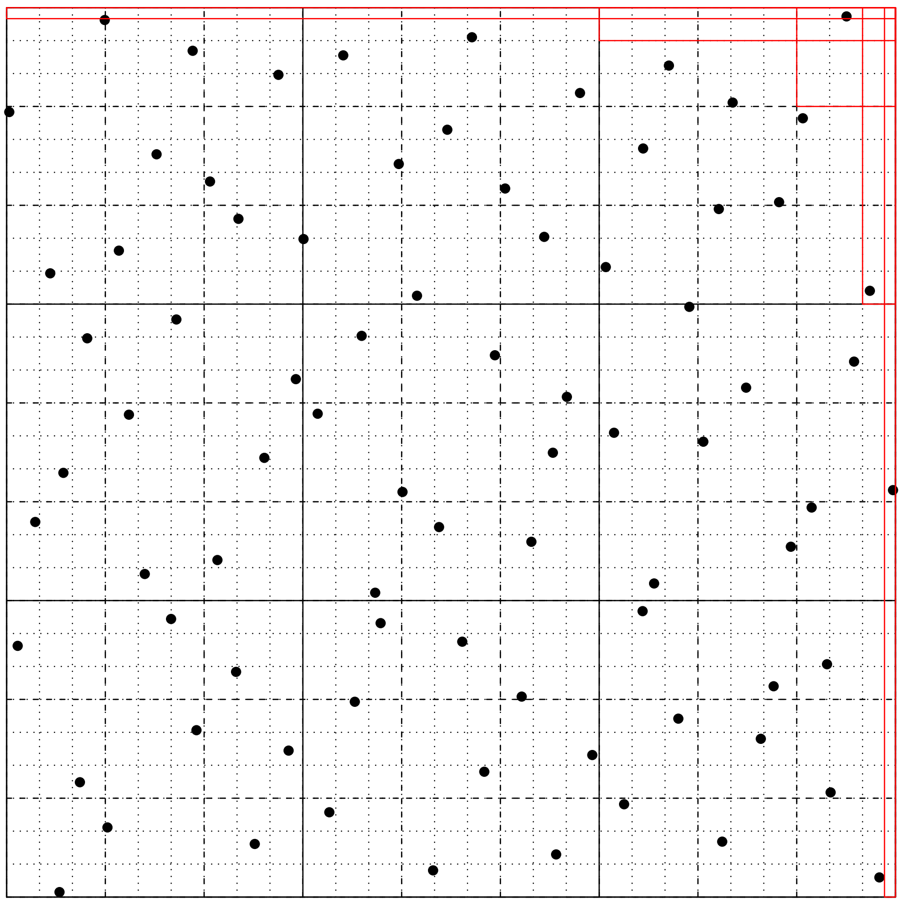


Figure 6.6: Shown are 81 points of a $(0, 4)$–net in base 3. Reference lines are included to make the 3-ary boxes more visible. There 5 different shapes of 3-ary box balanced by these points. One box of each shape is highlighted.

6.6 Integration Lattices

In addition to digital nets and sequences, there is a second major QMC technique, known as integration lattices. The simplest example of an integration lattice is a rank one lattice. These take the form

$$x_i = \frac{i-1}{n}(g_1, \dots, g_d) \mod n \tag{6.18}$$

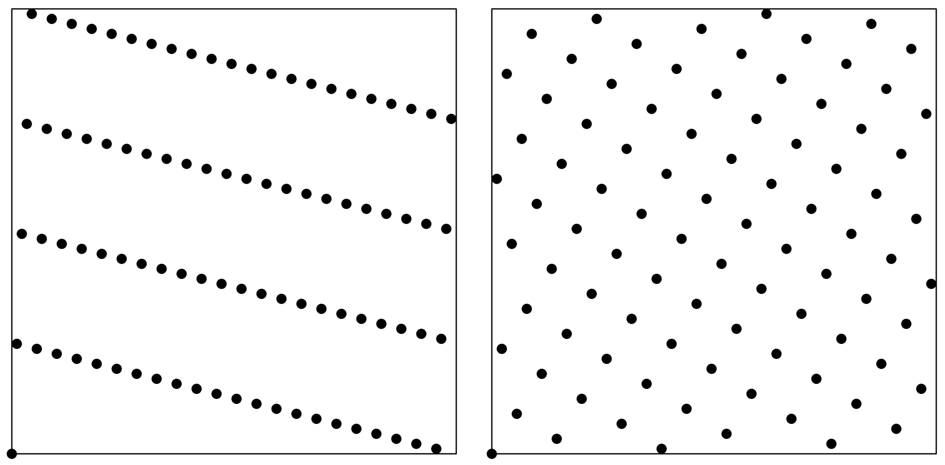


Figure 6.7: Shown are the points of two integration lattices in the unit square. The lattice on the right has much better uniformity, showing the importance of making a good choice of lattice generator.

for $i = 1, \dots, n$. Usually $g_1 = 1$. Figure 6.7 shows two integration lattices in $[0, 1)^2$ with $n = 89$. The first has $g_2 = 22$ and the second one has $g_2 = 55$.

It is clear that the second lattice in Figure 6.7 is more evenly distributed than the first one. The method of good lattice points is the application of the rule (6.18) with n and g carefully chosen to get good uniformity. Fang and Wang [18] and Hua and Wang [31] describe construction and use of good lattice points, including extensive tables of n and g.

Sloan and Joe [74] describe integration lattices in general, including lattices of rank higher than 1. A lattice of rank r for $1 \le r \le d$ requires r vectors like g to generate it. The origin of lattice methods is in Korobov [43]. Korobov's rules have $g = (1, h, h^2, \dots, h^{d-1})$ so that the search for a good rule requires only a careful choice of two numbers n and h.

Until recently, integration lattices were not extensible. Extensible integration lattices are a research topic of current interest, following the publication of Hickernell, Hong, L'Ecuyer and Lemieux [28].

Integration lattices are not as widely used in computer graphics as digital nets. Their periodic structure is likely to produce unwanted aliasing artifacts, at least in some applications. Compared to digital nets, integration lattices are very good at integrating smooth functions, especially smooth periodic functions.

6.7 Randomized Quasi-Monte Carlo

QMC methods may be thought of as derandomized MC. Randomized QMC (RQMC) methods re-randomize them. The original motivation is to get sample based error estimates.

In RQMC, one takes a QMC sequence (a_i) and transforms it into random points (x_i) such that x_i retain a QMC property and the expectation of $\hat{I}$ is I. The simplest way to achieve the latter property is to have each $x_i \sim U[0,1)^d$. With RQMC we can repeat a QMC integration R times independently getting $\hat{I}_1, \dots, \hat{I}_R$. The combined estimate $\hat{I} = (1/R)\sum_{r=1}^{R} \hat{I}_r$ has expected value I and an unbiased estimate of the RMSE of $\hat{I}$ is $[R(R-1)]^{-1}\sum_{r=1}^{r}(\hat{I}_r - \hat{I})^2$.

Cranley and Patterson [15] proposed a rotation modulo one

$$x_i = a_i + U \mod 1$$

where $U \sim U[0,1)^d$ and both addition and remainder modulo one are interpreted componentwise. It is easy to see that each $x_i \sim U[0,1)^d$. Cranley and Patterson proposed rotations of integration lattices. Tuffin [83] considered applying such rotations to digital nets. They don't remain nets, but they still look very uniform.

Owen [57] proposes a scrambling of the base b digits of a_i. Suppose that a_i is the i'th row of the matrix A with entries A_{ij} for $j = 1, \dots, d$, and either $i = 1, \dots, n$ for a finite sequence or $i \geq 1$ for an infinite one. Let $A_{ij} = \sum_{k=1}^{\infty} b^{-k} a_{ijk}$ where $a_{ijk} \in \{0, 1, \dots, b-1\}$. Now let $x_{ijk} = \pi_{j \cdot a_{ij1} \dots a_{ij\,k-1}}(a_{ijk})$ where $\pi_{j \cdot a_{ij1} \dots a_{ij\,k-1}}$ is a uniform random permutation of $0, \dots, b-1$. All the permutations required are independent, and the permutation applied to the k'th digits of A_{ij} depends on j and on the preceding $k-1$ digits.

Applying this scrambling to any point $a \in [0,1)^d$ produces a point $x \sim U[0,1)^d$. If (a_i) is a (t, m, d)–net in base b or a (t, d)–sequence in base b, then with probability 1, the same holds for the scrambled version (x_i). The scrambling described above requires a great many permutations. Random linear scrambling is a partial derandomization of scrambled nets, given by Matoušek [49] and also in Hong and Hickernell [30]. Random linear scrambling significantly reduces the number of permutations required from $O(db^m)$ to $O(dm^2)$.

For integration over a scrambled digital sequence we have $\text{Var}(\hat{I}) = o(1/n)$ for any f with $\sigma^2 < \infty$. Thus for large enough n a better than MC result will be obtained. For integration over a scrambled $(0, m, d)$-net Owen [58] shows that

$$\text{Var}(\hat{I}) \le \Bigl(\frac{b}{b-1}\Bigr)^{\min(d-1,m)} \frac{\sigma^2}{n} \le \frac{2.72\,\sigma^2}{n}.$$

That is scrambled $(0, m, d)$–nets cannot have more than $e = \exp(1) \doteq 2.72$ times the Monte Carlo variance for finite n. For nets in base $b = 2$ and $t \ge 0$, Owen [60] shows that

$$\text{Var}(\hat{I}) \le 2^t 3^d \,\frac{\sigma^2}{n}.$$

Compared to QMC, we expect RQMC to do no harm. After all, the resulting x_i still have a QMC structure, and so the RMSE should be $O(n^{-1}(\log n)^d)$. Some forms of RQMC reduce the RMSE to $O(n^{-3/2}(\log n)^{(d-1)/2})$ for smooth enough f. This can be understood as random errors cancelling where deterministic ones do not. Surveys of RQMC appear in Owen [61] and L'Ecuyer and Lemieux [46].

6.8 Padding and Latin Supercube Sampling

In some applications d is so large that it becomes problematic to construct a meaningful QMC sequence. For example the number of random vectors needed to follow a single light path in a scene with many reflective objects can be very large and may not have an a priori bound. As another example, if acceptance-rejection sampling (Devroye [17]) is used to generate a random variable then a large number of random variables may need to be generated in order to produce that variable.

Padding is a simple expedient solution to the problem. One uses a QMC or RQMC sequence in dimension s for what one expects are the s most important input variables. Then one pads out the input with $d - s$ independent $U[0, 1)$ random variables. This technique was used in Spanier [76] for particle transport simulations. It is also possible to pad with a $d - s$ dimensional Latin hypercube sample as described in Owen [59], even when d is conceptually infinite.

In Latin supercube sampling, the d input variables of x_i are partitioned into some number k of groups. The j'th group has dimension $d_j \ge 1$ and of course $\sum_{j=1}^k d_j = d$. A QMC or RQMC method is applied in each of the k groups.

Just as the van der Corput sequence cannot simply be substituted for a pseudo-random generator, care has to be taken in using multiple (R)QMC methods within the same problem. It would not work to take k independent randomizations of the same QMC sequence. The fix is to randomize the run order of the k groups relative to each other, just as Latin hypercube sampling randomizes the run order of d stratified samples.

To describe LSS, for $j = 1, \ldots, k$ and $i = 1, \ldots, n$ let $a_{ji} \in [0,1)^{d_j}$. Suppose that $a_{j1}, \ldots, a_{jn}$ are a (R)QMC point set. For $j = 1, \ldots, k$, let $\pi_j(i)$ be independent uniform permutations of $1, \ldots, n$. Then let $x_{ji} = a_{j\pi_j(i)}$. The LSS has rows x_i comprised of $x_{1i}, \ldots, x_{ki}$. Owen [59] shows that in Latin supercube sampling the function f can be written as a sum of two parts. One, from within groups of variables, is integrated with an (R)QMC error rate, while the other part, from between groups of variables, is integrated at the Monte Carlo rate. Thus a good grouping of variables is important as is a good choice of (R)QMC within groups.

Chapter 7

Monte Carlo Path Tracing

By Pat Hanrahan

This Chapter discusses *Monte Carlo Path Tracing*. Many of these ideas appeared in James Kajiya's original paper on the Rendering Equation. Other good original sources for this material is L. Carter and E. Cashwell's book *Particle-Transport Simulation with the Monte Carlo Methods* and J. Spanier and E. Gelbard's book *Monte Carlo Principles and Neutron Transport Problems*.

7.1 Solving the Rendering Equation

To derive the rendering equation, we begin with the *Reflection Equation*

$$L_r(\vec{x}, \vec{\omega}_r) = \int_{\Omega_i} f_r(\vec{x}, \vec{\omega}_i \to \vec{\omega}_r)\, L_i(\vec{x}, \vec{\omega}_i)\, \cos\theta_i d\omega_i.$$

The reflected radiance L_r is computed by integrating the incoming radiance over a hemisphere centered at a point of the surface and oriented such that its north pole is aligned with the surface normal. The BRDF f_r is a probability distribution function that describes the probability that an incoming ray of light is scattered in a random outgoing direction. By convention, the two direction vectors in the BRDF point outward from the surface.

One of the basic laws of geometric optics is that radiance does not change as light propagates (assuming there is no scattering or absorption). In free space, where light travels along straight lines, radiance does not change along a ray. Thus, assuming that a point $\vec{x}$ on a receiving surface sees a point $\vec{x}'$ on a source surface,

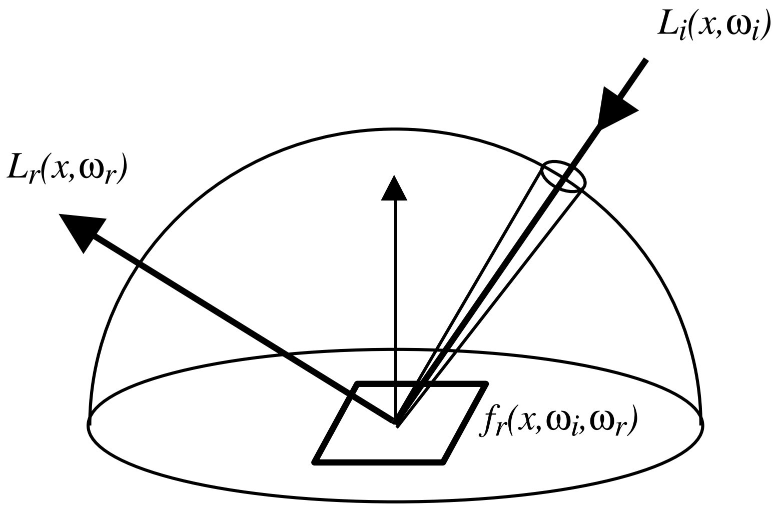


Figure 7.1: Integrating over the upper hemisphere.

the incoming radiance is equal to the outgoing radiance:

$$L_i(\vec{x},\vec{\omega}_i(\vec{x}',\vec{x})) = L_o(\vec{x}',\vec{\omega}_o'(\vec{x},\vec{x})).$$

Where we use the direction as a function of two points as

$$\vec{\omega}(\vec{x}_1,\vec{x}_2) = \vec{\omega}(\vec{x}_1 \to \vec{x}_2) = \frac{\vec{x}_2 - \vec{x}_1}{|\vec{x}_2 - \vec{x}_1|}.$$

The standard convention is to parameterize the incoming radiance L_i at $\vec{x}$ with the direction from the receiver $\vec{x}$ to the source $\vec{x}'$. When using this convention, the incoming radiance is defined on a ray pointing in the direction opposite to the direction of light propagation.

It is also useful to introduce notation for the two-point radiance

$$L(\vec{x},\vec{x}') = L(\vec{x} \to \vec{x}') = L_o(\vec{x},\vec{\omega}(\vec{x},\vec{x}')).$$

and the three-point BRDF

$$f_r(\vec{x},\vec{x}',\vec{x}'') = f_r(\vec{x} \to \vec{x}' \to \vec{x}'') = f_r(\vec{x}',\vec{\omega}(\vec{x}',\vec{x}),\vec{\omega}(\vec{x}'',\vec{x}'))$$

(Note: the two-point radiance function defined here is different than the two-point intensity function defined in Kajiya's original paper.)

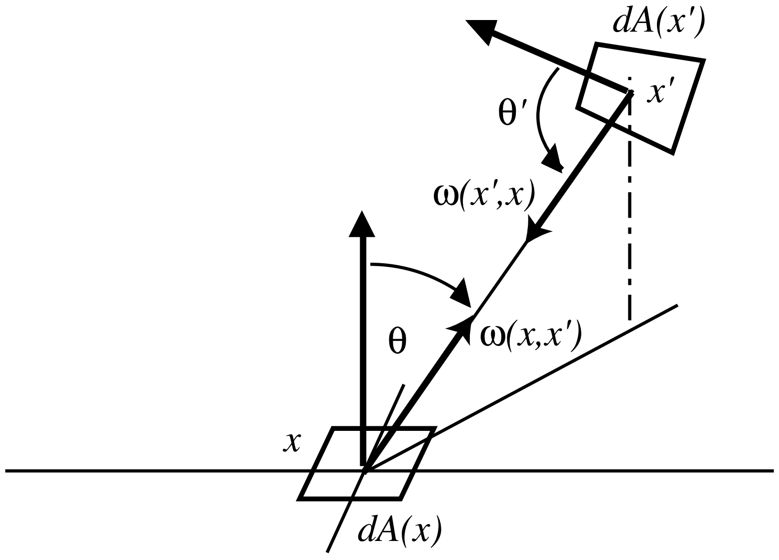


Figure 7.2: Two-point geometry.

Point to point functions are useful since they are intuitive and often clarify the geometry and physics. For example, if $\vec{x}$ sees $\vec{x}'$, then $\vec{x}'$ sees $\vec{x}$. This mutual visibility is represented as the two-point visibility function, $V(\vec{x}, \vec{x}')$, which is defined to be 1 if a line segment connecting $\vec{x}$ to $\vec{x}'$ does not intersect any opaque object, and 0 otherwise.

The reflection equation involves an integral over the upper hemisphere. This integral may be converted to an integral over other surfaces by changing of variables from solid angles to surface areas. This is easily done by relating the solid angle subtended by the source to its surface area.

$$d\omega_i = \frac{\cos\theta_o'}{|\vec{x} - \vec{x}'|^2} dA(\vec{x}')$$

The projected solid angle then becomes

$$\cos\theta_i \, d\omega_i = G(\vec{x}, \vec{x}') \, dA(\vec{x}')$$

where

$$G(\vec{x}, \vec{x}') = G(\vec{x}', \vec{x}) = \frac{\cos\theta_i \cos\theta_o'}{|\vec{x} - \vec{x}'|^2} V(\vec{x}, \vec{x}')$$

In these equations we are making a distinction between the parameters used to specify points on the surface (the $\vec{x}$'s) and the measure that we are using when performing the integral (the differential surface area $dA(\vec{x})$). Sometimes we will be less rigorous and just use dA or dA' when we mean $dA(\vec{x})$ and $dA(\vec{x}')$. The geometry factor G is related to the *differential form factor* by the following equation: $F(\vec{x}, \vec{x}')dA' = \frac{G(\vec{x},\vec{x}')}{\pi}dA'$.

Performing this change of variables in the reflection equation leads to the following integral

$$L_r(\vec{x}, \vec{\omega}) = \int_A f_r(\vec{x}, \vec{\omega}(\vec{x}, \vec{x}'), \vec{\omega})\, L_o(\vec{x}', \vec{\omega}(\vec{x}', \vec{x}))\, G(\vec{x}, \vec{x}')\, V(\vec{x}, \vec{x}')\, dA(\vec{x}')$$

In this equation, $\vec{x}$ and $\vec{\omega}$ are indendent variables and we are integrating over surface area which is parameterized by $\vec{x}'$; thus, the incoming direction $\vec{\omega}_i$ and the direction of L_o are functions of these positions and are indicated as such.

The final step in the derivation is to account for energy balance

$$L_o(\vec{x}, \vec{\omega}) = L_e(\vec{x}, \vec{\omega}) + L_r(\vec{x}, \vec{\omega}).$$

This states that the outgoing radiance is the sum of the emitted and reflected radiances. Placing an emission function on each surface allows us to create area light sources. Inserting the reflection equation into the energy balance equation results in the *Rendering Equation*.

$$L(\vec{x}, \vec{\omega}) = L_e(\vec{x}, \vec{\omega}) + \int_A f_r(\vec{x}, \vec{\omega}(\vec{x}, \vec{x}'), \vec{\omega})\, L(\vec{x}', \vec{\omega}(\vec{x}', \vec{x}))\, G(\vec{x}, \vec{x}')\, V(\vec{x}, \vec{x}')\, dA'$$

For notational simplicity, we will drop the subscript o on the outgoing radiance.

The rendering equation couples radiance at the receiving surfaces (the left-hand side) to the radiances of other surfaces (inside the integrand). This equation applies at all points on all surfaces in the environment. It is important to recognize the knowns and the unknowns. The emission function L_e and the BRDF f_r are knowns since they depends on the scene geometry, the material characteristics, and the light sources. The unknown is the radiance L on all surfaces. To compute the radiance we must solve this equation. This equation is an example of an integral equation, since the unknown L appears inside the integral. Solving this equation in the main goal of Monte Carlo Path Tracing.

The rendering equation is sometimes written more compactly in operator form. An operator is a method for mapping a function to another function. In our case, the function is the radiance.

$$L = L_e + K \circ L$$

Sometimes it is useful to break the operator K into two operators, T and S. T is the *transfer* operator and applied first; it takes outgoing light on one surface and transfers it to another surface.

$$L_i(\vec{x}, \vec{\omega}(\vec{x}', \vec{x})) = T \circ L(\vec{x}, \vec{\omega}(\vec{x}, \vec{x}'))$$

S is the scattering or reflection operator which takes the incoming light distribution and computes the outgoing light distribution.

$$L_r(\vec{x}, \vec{\omega}) = S \circ L_i(\vec{x}, \vec{\omega}')$$

Operator equations like the rendering equation may be solved using iteration.

$$\begin{aligned} L^0 &= L_e \\ L^1 &= L_e + K \circ L^0 = L_e + K \circ L_e \\ \ldots \\ L^n &= L_e + K \circ L^{n-1} = \sum_{i=0}^{n} K^n \circ L_e \end{aligned}$$

Noting that $K^0 = I$, where I is the identity operator. This infinite sum is called the Neumann Series and represents the formal solution (not the computed solution) of the operator equation.

Another way to interpret the Neumann Series is to draw the analogy between

$$\frac{1}{1-x} = (1-x)^{-1} = 1 + x + x^2 \ldots,$$

and

$$(I-K)^{-1} = I + K + K^2 \ldots.$$

The rendering equation

$$(I-K) \circ L = Le$$

then has the following solution

$$L = (I-K)^{-1} \circ L_e$$

Note that $(I-K)^{-1}$ is just an operator acting on the emission function. This operator spreads the emitted light over all the surfaces.

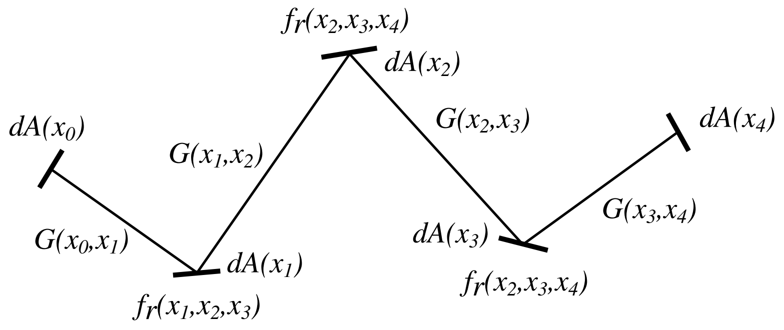


Figure 7.3: A path from point $\vec{x}_1$ to $\vec{x}_5$.

It is useful to write out the formal solution

$$L(\vec{x},\vec{\omega}) = \sum_{i=0}^{\infty} K^i \circ L_e(\vec{x}_0,\vec{\omega}_0))$$

in all its gory detail. Let's consider one term:

$$\begin{aligned} L^n(\vec{x},\vec{\omega}) &= L(\vec{x}_n,\vec{x}_{n+1}) = K^n \circ L_e \\ &= \int_A ... \int_A L_e(\vec{x}_0,\vec{x}_1) G(\vec{x}_0,\vec{x}_1) f_r(\vec{x}_0,\vec{x}_1,\vec{x}_2) G(\vec{x}_1,\vec{x}_2)... \\ &\quad G(\vec{x}_{n-1},\vec{x}_n) f_r(\vec{x}_{n-1},\vec{x}_n,\vec{x}_{n+1})\, dA_0\, dA_1 ...d\vec{A}(\vec{x})_n \end{aligned}$$

This integral has a very simple geometric and physical intuition. It represents a family of light *paths*. Each path is characterized by the number of bounces or length n. There are many possible paths of a given length. Paths themselves are specified by a set vertices. The first vertex is a point on the light source and subsequent vertices are points on reflecting surfaces. The total contribution due to all paths of a given length is formed by integrating over all possible light and surface positions. This involves doing n integrals over surface areas. However, we must weight paths properly when performing the integral. A integrand for a particular path consists of alternating sequence of geometry and reflection terms. Finally, the final solution to the equation is the sum of paths of all lengths; or more simply, all possible light paths.

Note that these are very high dimensional integrals. Specifically, for a path of length n the integral is over a $2n$ dimensional space. The integral also involves very complicated integrands that include visibility terms and complex reflection functions defined over arbitrary shapes. It turns out these complexities is what makes Monte Carlo the method of choice for solving the rendering equation.

There is one more useful theoretical and practical step, and that is to relate the solution of the rendering equation to the process of image formation in the camera. The equation that governs this process is the *Measurement Equation*

$$M = \int_A \int_\Omega \int_T R(\vec{x}, \vec{\omega}, t) L(\vec{x}, \vec{\omega}, t)\, dt\, d\omega\, dA.$$

The response function $R(\vec{x}, \vec{\omega}, t)$ depends on the the pixel filter (the $\vec{x}$ dependence), the aperture (the $\vec{\omega}$ dependence), and the shutter (the t dependence). Other factors such as transformations of rays by the lens system and spectral sensitivities may also be included, but we will ignore these factors to simplify the presentation.

As seen above, the pixels value in the image is a function that involves nested integrals. These integrals are very complicated, but we can easily evaluate the integrand which corresponds to sampling the function.

- Sampling a pixel over (x, y) prefilters the image and reduces aliasing.
- Sampling the camera aperture (u, v) produces depth of field.
- Sampling in time t (the shutter) produces motion blur.
- Sampling in wavelength λ simulates spectral effects such as dispersion
- Sampling the reflection function produces blurred reflection.
- Sampling the tranmission function produces blurred transmission.
- Sampling the solid angle of the light sources produces penumbras and soft shadows.
- Sampling paths accounts for interreflection.

Sampling in x, y, u, v and t has been discussed previously. Sampling light sources and performing hemispherical integration has also been discussed. What remains is to sample paths.

7.2 Monte Carlo Path Tracing

First, let's introduce some notation for paths. Each path is terminated by the eye and a light.

E - the eye.

L - the light.

Each bounce involves an interaction with a surface. We characterize the interaction as either reflection or tranmission. There are different types of reflection and transmission functions. At a high-level, we characterize them as

D - diffuse reflection or transmission

G - glossy reflection or tranmission

S - specular reflection or refraction

Diffuse implies that light is equally likely to be scattered in any direction. Specular implies that there is a single direction; that is, given an incoming direction there is a unique outgoing direction. Finally, glossy is somewhere in between.

Particular ray-tracing techniques may be characterized by the paths that they consider.

Appel Ray casting: $E(D|G)L$

Whitted Recursive ray tracing: $E[S^*](D|G)L$

Kajiya Path Tracing: $E[(D|G|S)^+(D|G)]L$

Goral Radiosity: ED^*L

The set of traced paths are specified using regular expressions, as was first proposed by Shirley. Since all paths must involve a light L, the eye E, and at least one surface, all paths have length at least equal to 3.

A nice thing about this notation is that it is clear when certain types of paths are not traced, and hence when certain types of light transport is not considered by the algorithm. For example, Appel's algorithm only traces paths of length 3, ignoring longer paths; thus, only direct lighting is considered. Whitted's algorithm traces paths of any length, but all paths begin with a sequence of 0 or more mirror reflection and refraction steps. Thus, Whitted's technique ignores paths such as the following $EDSDSL$ or $E(D|G)^*L$. Distributed ray tracing and path tracing includes multiple bounces involving non-specular scattering such as $E(D|G)^*L$. However, even these methods ignore paths of the form $E(D|G)S^*L$; that is, multiple specular bounces from the light source as in a caustic. Obviously, any technique that ignores whole classes of paths will not correctly compute the solution to the rendering equation.

Let's now describe the basic Monte Carlo Path Tracing Algorithm:

```
Step 1. Choose a ray given (x,y,u,v,t)
    weight = 1

Step 2. Trace ray to find point of intersection with the nearest surface.

Step 3. Randomly decide whether to compute emitted or reflected light.

    Step 3A. If emitted,
        return weight * Le

    Step 3B. If reflected,
        weight *= reflectance
        Randomly scatter the ray according to the BRDF pdf
        Go to Step 2.
```

This algorithm will terminate as long as a ray eventually hits a light source. For simplicity, we assume all light sources are described by emission terms attached to surfaces. Latter we will discuss how to handle light sources better.

A variation of this algorithm is to trace rays in the opposite direction, from light sources to the camera. We will assume that reflective surface never absorb light, and that the camera is a perfect absorber.

```
Step 1. Choose a light source according to the light source power distribution.
    Generate a ray from that light source according to its intensity distribution.
    weight = 1

Step 2. Trace ray to find point of intersection.

Step 3. Randomly decide whether to absorb or reflect the ray.
Step 3A. If scattered,
    weight *= reflectance
    Randomly scatter the ray according to the BRDF.
    Go to Step 2.
Step 3B. If the ray is absorbed by the camera film,
    Record weight at x, y
    Go to Step 1.
```

The first algorithm is an example of forward ray tracing; in forward ray tracing rays start from the eye and propagate towards the lights. Forward ray tracing is also called eye ray tracing. In contrast, in backward ray tracing rays start at the light and trace towards the eye. As we will discuss in a subsequent section, Both methods are equivalent because the physics of light transport does not change if paths are reversed. Both methods have there advantages and disadvantages, and in fact may be coupled.

The above simple algorithms form the basis of Monte Carlo Path Tracing. However, we must be more precise. In particular, there are two theoretical and practical challenges:

Challenge 1 : Sampling an infinite sum of paths in an unbiased way.

Challenge 2 : Finding good estimators with low variance.

7.3 Random Walks and Markov Chains

To understand more about why path tracing works, let's consider a simpler problem: a discrete random walk. Instead of a physical system with continuous variables, such as position and direction, consider a discrete physical system comprised of n states. Path tracing as described above is an example of a random walk where we move from sample to sample, or from point to point, where the samples are drawn from a continuous probability distribution. In a discrete random walk, we move from state to state, and the samples are drawn from a discrete probability distribution.

A random walk is characterized by three distributions:

1. Let p_i^0 be the probability of starting in state i.
2. Let $p_{i,j}$ is the probability of moving from state i to state j.

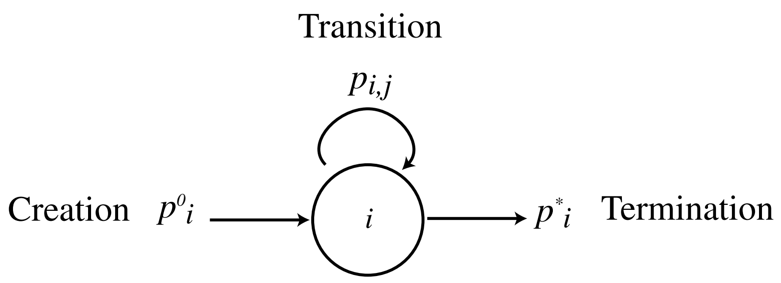


Figure 7.4: State transition diagram for a discrete random walk.

3. Let p_i^* is the probability of being terminated in state i.

Because the probability of transition and termination must sum to one, $p_i^* = 1 - \sum_{j=0} p_{i,j}$; that is, the probability of terminating in state i is equal to the probability of *not* moving from state i to j.

A discrete random walk consists of the following steps.

Step 1. Create a random particle in state i with probability p_i^0.

Step 2. With probability p_i^*, terminate in state i.
Score particle in state i by incrementing the counter for state i
Go to Step 1.

Step 3. Randomly select new state according to the transition probability distribution.
Set i to new j.
Go to Step 2.

Random walks are also called Markov Chains. A Markov Chain is a sequence of states generated by a random process. We will return to Markov Chains when we discuss the Metropolis Algorithm. Markov Chains also come up Bayesian reasoning and in learning theory (e.g. Hidden Markov Models). Keep your eyes open for Markov Chains; you will see these techniques used more and more in computer graphics.

Given a set of particles following random walks, the problem is to compute the final probability of a particle being terminated in state i. To solve this problem, we introduce another random variable P_i^n, which is the probability of being in state i after n transitions. Since each state transition is independent of the previous transitions, this probability may be computed using a simple recurrence

$$\begin{aligned}
P_j^0 &= p_j^0 \\
P_j^1 &= p_j^0 + \sum_i p_{i,j} P_i^0 \\
\ldots & \\
P_j^n &= p_j^0 + \sum_i p_{i,j} P_i^{n-1}.
\end{aligned}$$

Defining a matrix M whose entries are $M_{i,j} = p_{i,j}$, the above process can be viewed as the following iterative product of a matrix times a vector

$$\begin{aligned}
P^0 &= p^0 \\
P^1 &= p^0 + MP^0 \\
\ldots & \\
P^n &= p^0 + MP^{n-1}.
\end{aligned}$$

And this procedure may be recognized as the iterative solution of the following matrix equation

$$(I - M)P = p^0$$

since then

$$P = (I - M)^{-1} p^0 = p^0 + M(p^0 + M(p^0 \ldots = \sum_{i=0} M^i p^0.$$

This process will always converge assuming the matrices are probability distributions. The basic reason for this is that probabilities are always less than one, and so a product of probabilities quickly tends towards zero.. Thus, the random walk provides a means for solving linear systems of equations, assuming that the matrices are probability transition matrices. Note the similiarity of this discrete iteration of matrices to the iterative application of the continuous operator when we solve the rendering equation using Neumann Series.

This method for solving matrix equations using discrete random walks may be directly applied to the radiosity problem. In the radiosity formulation,

$$B_i = E_i + \rho_i \sum_j F_{i,j} B_j$$

where the form factor equals

$$F_{i,j} = \frac{1}{\pi A_i} \int_{A_i} \int_{A_j} G(\vec{x}, \vec{x}') V(\vec{x}, \vec{x}') \, dA(\vec{x}) \, dA(\vec{x}')$$

Recall that the form factors may be interpreted as a probabilities. The form factor $F_{i,j}$ is the percentage of outgoing light leaving surface element A_i that falls on surface element A_j. In fact, the form factor is the probability that a random ray leaving surface element A_i makes it to A_j (although one has to be careful about how one defines a random ray). Thus, form factor matrices may be interpested as transition matrices. In this equation, ρ is the diffuse reflectance and is equal to $\rho = B/E$; The reflectance must be positive and less than 1. The absorption or termination probability is thus equal to $1 - \rho$.

These observations lead to a method for solving the matrix radiosity equation using a discrete random walk. The major issue, which is always a case with radiosity solution techniques, is computing the form factor matrix. This process is expensive and error prone because of the complexity of the environment and the difficulty in doing exact visible surface determination. The form-factor matrix is also very large. For example, a scene consisting of a million surface elements would require a million squared matrix. Therefore, in practice, the form factor matrix is often calculated on-the-fly. Assuming a particle is on some surface element i, an outgoing particle may be sent off in a random direction where the random direction is chosen from a cosine-weighted distribution (here the cosine is with respect to the surface element normal). The particle is then ray-traced and the closent point of intersection on surface element j is found. This random process is roughly equivalent to one generated from a known form-factor matrix.

It is interesting to prove that random walks provide an unbiased estimate of the solution of the linear system of equations. Although this proof is a bit formal, it is worthwhile working it through to get a flavor of the mathematical techniques involved.

The first step is to define a random variable on the space of all paths. Let's signify a path of length k as $\alpha_k = (i_1, i_2, ..., i_k)$; this path involves a sequence of transitions from state i_1 to state i_2 and ending up finally after k transitions in state i_k. The random variable α without the subscript is the set of all paths of length one to infinity.

The next step is to pick an estimator $W(\alpha)$ for each path. Then we can compute the expected value of the estimator by weighting W by the probability that a given path is sampled, $p(\alpha)$.

$$\begin{aligned} E[W] &= \sum_{\alpha} p(\alpha)W(\alpha) \\ &= \sum_{k=1}^{\infty} \sum_{\alpha_k} p(\alpha_k)W(\alpha_k) \\ &= \sum_{k=1}^{\infty} \sum_{i_1} ... \sum_{i_k} p(i_1, ..., i_k)W(i_1, ..., i_k) \end{aligned}$$

In the last line we group all paths of the same length together. The sums on each index i go from 1 to n - the number of states in the system. Thus, there are n^k paths of length k, and of course paths can have infinite length. There are a lot of paths to consider!

What is the probability $p(\alpha_k)$ of path α_k ending in state i_k? Assuming the discrete random walk process described above, this is the probability that the particle is created in state i_1, times the probability of making the necessary transitions to arrive at state i_k, times the probability of being terminated in state i_k

$$p(\alpha_k) = p^0_{i_1} p_{i_1,i_2} ... p_{i_{k-1},i_k} p^*_k$$

With these careful definitions, the expected value may be computed

$$\begin{aligned} E[W_j] &= \sum_{k=1}^{\infty} \sum_{\alpha_k} p(\alpha_k)W_j(\alpha_k) \\ &= \sum_{k=1}^{\infty} \sum_{i_1} ... \sum_{i_k} p^0_{i_1} p_{i_1,i_2} ... p_{i_{k-1},i_k} p^*_{i_k} W_j(\alpha_k) \end{aligned}$$

Recall, that our estimator counts the number of counts the number of particles that terminate in state j. Mathematically, we can describe this counting process with a delta function, $W_j(\alpha_k) = \delta_{i_k,j}/p^*_j$. This delta function only scores particles terminating in $i_k = j$. The expected value is then

$$\begin{aligned} E[W_j] &= \sum_{k=1}^{\infty} \sum_{i_1} ... \sum_{i_{k-1}} \sum_{i_k} p^0_{i_1} p_{i_1,i_2} ... p_{i_{k-1},i_k} p^*_{i_k} \delta_{i_k,j}/p^*_j \\ &= \sum_{k=1}^{\infty} \sum_{i_1} ... \sum_{i_{k-1}} p^0_{i_1} p_{i_1,i_2} ... p_{i_{k-1},j}. \end{aligned}$$

This sum may be recognized as the j component of the matrix equation

$$E[W] = p^0 + Mp^0 + M^2p^0 + ...$$

which is the desired solution of the linear system of equations.

Note that we had to be careful about the estimator. If we hadn't divided the count of the particles by the probability of a termination event, the expected value would not have equaled the right answer. Picking the wrong estimator - that is, an estimator that results in the wrong expected value - for a complex sampling process is one of the most common errors when uding Monte Carlo Techniques. Until you have a lot of experience, it is worthwhile convincing yourself that your estimator is unbiased.

This technique was originally developed by von Neumann and Ulam, the originators of the Monte Carlo Method. The estimator they used is often called the absorption estimator, since only particles that are absorbed are counted. An interesting variation, developed by Wasow, is to count all the number of particles that pass through state j (including those terminate as well as those that make a transition). This is called the collision estimator, since it counts all particles colliding with a surface. It is an interesting exercise to show that the collision estimator also provides an unbiased estimate of the solution to the linear equation. It is more challenging, but more interesting, to also derive the conditions when and if the collision estimator works better than the absorption estimator.

This basic proof technique is easy to generalize to continuous distributions, but the notation is messy. The details are described in Chapter 3 of the Spanier and Gelbard book on neutron transport [77], the most authoritative source if you wish to understand the theory behind Monte Carlo Techniques for solving integral equations and tranport problems.

7.4 Adjoint Equations and Importance Sampling

Recall, that the pixel response is equal to the sum over paths of length n

$$\begin{aligned} M_n &= \int_0 ... \int_n S(\vec{x}_0, \vec{x}_1) G(\vec{x}_0, \vec{x}_1) f_r(\vec{x}_0, \vec{x}_1, \vec{x}_2) G(\vec{x}_1, \vec{x}_2) \\ & \quad ... f_r(\vec{x}_{n-2}, \vec{x}_{n-1}, \vec{x}_n) G(x_{n-1}, x_n) R(\vec{x}_{n-1}, \vec{x}_n) dA_0\, dA_1 ... d\vec{A}(\vec{x})_n. \end{aligned}$$

where we have switched notation and written the source term as $S(\vec{x}, \vec{x}') = L_e(\vec{x}, \vec{x}')$.

As noted above this equation is symmetric under the interchange of lights and sensors. Switching L_e with R, and noting that

$$\begin{aligned} M_n &= \int_A \dots \int_A R(\vec{x}_0,\vec{x}_1)G(\vec{x}_0,\vec{x}_1)f_r(\vec{x}_0,\vec{x}_1,\vec{x}_2)G(\vec{x}_1,\vec{x}_2) \\ &\quad \dots f_r(\vec{x}_{n-2},\vec{x}_{n-1},\vec{x}_n)G(x_{n-1},x_n)S(\vec{x}_n,\vec{x}_{n-1})\,dA_0\,dA_1 \dots d\vec{A}(\vec{x})_n \\ &= \int_A \dots \int_A S(\vec{x}_n,\vec{x}_{n-1})G(x_n,x_{n-1})f_r(\vec{x}_n,\vec{x}_{n-1},\vec{x}_{n-2})G(\vec{x}_{n-1},\vec{x}_{n-2}) \\ &\quad \dots f_r(\vec{x}_2,\vec{x}_1,\vec{x}_0)G(\vec{x}_1,\vec{x}_0)R(\vec{x}_1,\vec{x}_0)\,dA_0\,dA_1 \dots d\vec{A}(\vec{x})_n \end{aligned}$$

In the second step, we noted from the symmetry of the geometry that

$$G(\vec{x}_i,\vec{x}_j) = G(\vec{x}_j,\vec{x}_i)$$

and because of the reciprocity principle the BRDF is also symmetric

$$f_r(\vec{x}_i,\vec{x}_j,\vec{x}_k) = f_r(\vec{x}_k,\vec{x}_j,\vec{x}_i)$$

These symmetries implie that we may ray trace from either the light or the eye; both methods will lead to the same integral.

Suppose now we break the path at some point k. The amount of light that makes to k is

$$\begin{aligned} L_S(\vec{x}_k,\vec{x}_{k+1}) &= \int_A \dots \int_A S(\vec{x}_0,\vec{x}_1)G(\vec{x}_0,\vec{x}_1)f_r(\vec{x}_0,\vec{x}_1,\vec{x}_2)G(\vec{x}_1,\vec{x}_2) \\ &\quad \dots G(\vec{x}_{k-1},\vec{x}_k)f_r(\vec{x}_{k-1},\vec{x}_k,\vec{x}_{k+1})\,dA_0 \dots d\vec{A}(\vec{x})_{k-1} \end{aligned}$$

In a similar way, treating the sensor as a virtual light source, we can compute the amount of light coming from the sensor makes it to k.

$$\begin{aligned} L_R(\vec{x}_k,\vec{x}_{k+1}) &= \int_{k+2} \dots \int_n f_r(\vec{x}_k,\vec{x}_{k+1},\vec{x}_{k+2})G(\vec{x}_{k+1},\vec{x}_{k+2}) \\ &\quad \dots f_r(\vec{x}_{n-2},\vec{x}_{n-1},\vec{x}_n)G(x_{n-1},x_n)R(\vec{x}_{n-1},\vec{x}_n)\,dA_{k+2} \dots d\vec{A}(\vec{x})_n \end{aligned}$$

The measured response is then

$$M = \int_A \int_A L_S(\vec{x}_k,\vec{x}_{k+1})G(\vec{x}_k,\vec{x}_{k+1})L_R(\vec{x}_k,\vec{x}_{k+1})\,dA_k\,dA_{k+1}$$

Note the use of the notation L_S and L_R to indicate radiance “cast” from the source vs. the receiver.

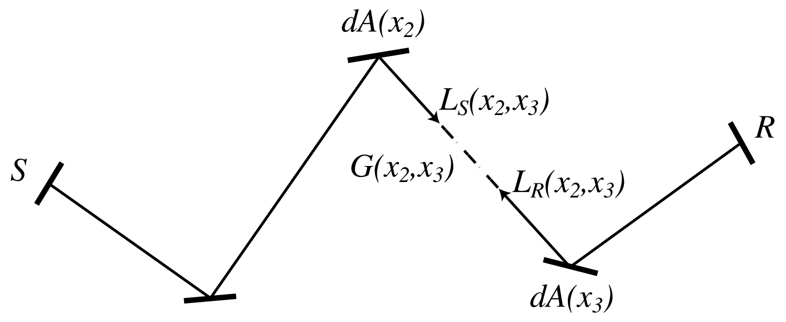


Figure 7.5: A path with both the forward and backward (adjoint) solution of the transport equation. The forward solution is generated from the source term S and the backward solution is generated from the received term R. For physical situations where the transport equation is invariant under path reversal, the forward and backward equations are the same.

We make two observations about this equation. First, this equation can be considered the *inner product* of two radiance functions. If we consider radiance to be a function on rays $r = (\vec{x}, \vec{\omega})$, then if we have functions $f(r)$ and $g(r)$, the inner product of f and g is

$$< f, g >= \int f(r)g(r)d\mu(r)$$

where $d\mu(r)$ is the appropriate measure on rays. The natural way to measure the rays between two surface elements A and A' is $d\mu(r) = G(x, x')\, dA\, dA'$. Equivalently, considering r to be parameterized by position $\vec{x}$ and direction $\vec{\omega}$, the $d\mu(r) = d\vec{\omega} \circ d\vec{A}(\vec{x})(\vec{x})$.

Second, this integral naturally leads to a method for importance sampling a path. Suppose we are tracing light and arrive at surface k. To compute the sensor response, we need to integrate L against R. In this sense, R may be considered an importance function for sampling the next directions, since we want a sampling technique that is proportional to R to achieve low variance. But R is the solution of the reversed transport equation that would be computed if we were to trace rays from the sensor. R tells us how much light from the sensor would make it to this point. Thus, the backward solution provides an importance function for the forward solution, and vice versa. This is the key idea between bidirectional ray tracing.

Manipulating about adjoint equations is easy using the operator notation. Using the operator notation, an integral equation is just

$$K \circ f = \int K(x,y) f(y)\, dy.$$

We want to estimate the integral given by the measurement equation, which is just the inner product of two functions

$$M = < f, K \circ g > = \int f(x) \left(\int K(x,y) g(y)\, dy \right)\, dx.$$

This of f as the response of the sensor and $K \circ g$ as the solution of the rendering equation. This equation may be rearranged

$$\begin{aligned} < f, K \circ g > &= \int f(x) \left(\int K(x,y) g(y)\, dy \right)\, dx \\ &= \left(\int f(x) K(x,y)\, dx \right) g(y)\, dy \\ &= < K^+ f, g > . \end{aligned}$$

Note the difference between

$$K \circ f = \int K(x,y) f(y)\, dy$$

and

$$K^+ \circ f = \int K(x,y) f(x)\, dx.$$

One integral is over the first variable, the other is over the second variable. Of course, if $K(x,y) = K(y,x)$ these two integrals are the same, in which case $K^+ = K$ and the operator is said to be *self-adjoint*.

This notation provides a succinct way of proving that the forward estimate is equal to the backward estimate of the rendering equation. Recall

$$K \circ L_S = S$$

We can also write a symmetric equation in the other direction

$$K \circ L_R = R$$

Then,

$$\begin{aligned} < R, L_S > &= < K \circ L_R, L_S > \\ &= < L_R, K^+ \circ L_S > \\ &= < L_R, K \circ L_S > \\ &= < L_R, S > \end{aligned}$$

This result holds even if the operator is not self-adjoint. We will leave the demonstration of that fact as an exercise.

This is a beautiful result, but what does it mean in practice. Adjoint equations have lots of applications in all areas of mathematical physics. What they allow you to do is create output sensitive algorithms. Normally, when you solve an equation you solve for the answer *everywhere*. Think of radiosity; when using the finite element method you solve for the radiosity on all the surfaces. The same applies to light ray tracing or the classic discrete random walk; you solve for the probability of a particle landing in any state. However, in many problems you only want to find the solution at a few points. In the case of image synthesis, we only need to compute the radiance that we see, or that falls on the film. Computing the radiance at other locations only needs to be done if its effects are observable.

We can model the selection of a subset of the solution as the inner product of the response function times the radiance. If we only want to observe a small subset of the solution, we make the response function zero in the locations we don't care about. Now consider the case when all the surfaces act as sources and only the film plane contributed a non-zero response. Running a particle tracing algorithm forward from the sources would be very inefficient, since only rarely is a particle terminated on the film plane. However, running the algorithm in the reverse direction is very efficient, since all particles will terminate on sources. Thus each particle provides useful information. Reversing the problem has led to a much more efficient algorithm.

The ability to solve for only a subset of the solution is a big advantage of the Monte Carlo Technique. In fact, in the early days of the development of the algorithm, Monte Carlo Techniques were used to solve linear systems of equations. It turns out they are very efficient if you want to solve for only one variable. But be wary: more conventional techniques like Gaussian elimination are much more effective if you want to solve for the complete solution.

Chapter 8

The Rendering Equation and Path Tracing

by Philip Dutre

This chapter gives various formulations of the rendering equation, and outlines several strategies for computing radiance values in a scene.

8.1 Formulations of the rendering equation

The global illumination problem is in essence a transport problem. Energy is emitted by light sources and transported through the scene by means of reflections (and refractions) at surfaces. One is interested in the energy equilibrium of the illumination in the environment.

The transport equation that describes global illumination transport is called the rendering equation. It is the integral equation formulation of the definition of the BRDF, and adds the self-emittance of surface points at light sources as an initialization function. The self-emitted energy of light sources is necessary to provide the environment with some starting energy. The radiance leaving some point x, in direction Θ, can be expressed as an integral over all hemispherical directions incident on the point x (figure 8.1):

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) cos(N_x, \Psi) d\omega_\Psi$$

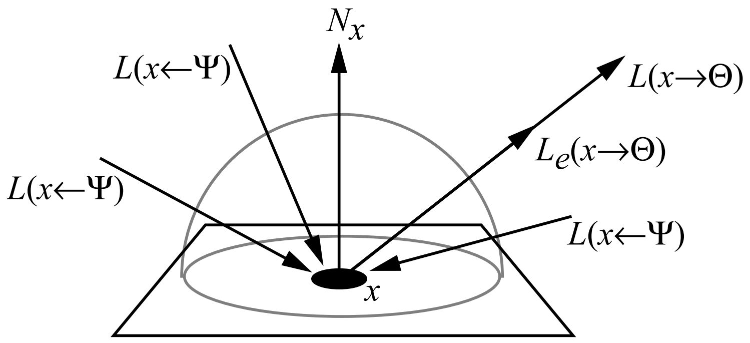


Figure 8.1: Rendering equation

One can transform the rendering equation from an integral over the hemisphere to an integral over all surfaces in the scene. Also, radiance remains unchanged along straight paths, so exitant radiance can be transformed to incident radiance and vice-versa, thus obtaining new versions of the rendering equation. By combining both options with a hemispheric or surface integration, four different formulations of the rendering equation are obtained. All these formulations are mathematically equivalent.

Exitant radiance, integration over the hemisphere

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) L(y \to -\Psi) \cos(N_x, \Psi) d\omega_\Psi$$

with

$$y = r(x, \Theta)$$

When designing an algorithm based on this formulation, integration over the hemisphere is needed, and as part of the function evaluation for each point in the integration domain, a ray has to be cast and the nearest intersection point located.

Exitant radiance, integration over surfaces

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_A f_r(x, \Psi \leftrightarrow \Theta) L(y \to \overrightarrow{yx}) V(x, y) G(x, y) dA_y$$

with

$$G(x, y) = \frac{cos(N_x, \Psi) cos(N_y, \Psi)}{r_{xy}^2}$$

Algorithms based on this formulation need to evaluate the visibility $V(x, y)$ between two points x and y, which is a different operation than casting a ray from x in a direction Θ.

Incident radiance, integration over the hemisphere

$$L(x \leftarrow \Theta) = L_e(x \leftarrow \Theta) + \int_{\Omega_y} f_r(y, \Psi \leftrightarrow -\Theta) L(y \leftarrow \Psi) \cos(N_y, \Psi) d\omega_\Psi$$

with

$$y = r(x, \Theta)$$

Incident radiance, integration over surfaces

$$L(x \leftarrow \Theta) = L_e(x \leftarrow \Theta) + \int_{A} f_r(y, \Psi \leftrightarrow \overrightarrow{yz}) L(y \leftarrow \overrightarrow{yz}) V(y, z) G(y, z) dA_z$$

with

$$y = r(x, \Theta)$$

8.2 Importance function

In order to compute the average radiance value over the area of a pixel, one needs to know the radiant flux over that pixel (and associated solid angle incident w.r.t. the aperture of the camera). Radiant flux is expressed by integrating the radiance distribution over all possible surface points and directions. Let $S = A_p \times \Omega_p$ denote all surface points A_p and directions Ω_p visible through the pixel. The flux $\Phi(S)$ is written as:

$$\Phi(S) = \int_{A_p} \int_{\Omega_p} L(x \rightarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x$$

When designing algorithms, it is often useful to express the flux as an integral over all possible points and directions in the scene. This can be achieved by introducing the initial importance function $W_e(x \leftarrow \Theta)$:

$$\Phi(S) = \int_{A} \int_{\Omega} L(x \rightarrow \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x$$

$W_e(x \leftarrow \Theta)$ is appropriately defined by:

$$W_e(x \leftarrow \Theta) = \begin{cases} 1 & \text{if } (x,\Theta) \in S \\ 0 & \text{if } (x,\Theta) \notin S \end{cases}$$

The average radiance value is then given by:

$$L_{average} = \frac{\int_A \int_\Omega L(x \to \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x}{\int_A \int_\Omega W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x}$$

We now want to develop the notion of importance further, by considering the possible influence of some energy value at each pair (x, Θ) on the value $\Phi(S)$. Or: if a single radiance value $L(x \to \Theta)$ is placed at (x, Θ), and if there are no other sources of illumination present, how large would the resulting value of $\Phi(S)$ be? This influence value attributed to $L(x \to \Theta)$ is called the importance of (x, Θ) w.r.t. S, is written as $W(x \leftarrow \Theta)$, and depends only on the geometry and reflective properties of the objects in the scene.

The equation expressing $W(x \leftarrow \Theta)$ can be derived by taking into account two mechanisms in which $L(x \to \Theta)$ can contribute to $\Phi(S)$:

Self-contribution If $(x, \Theta) \in S$, then $L(x \to \Theta)$ fully contributes to $\Phi(S)$. This is called the self-importance of the set S, and corresponds to the above definition of $W_e(x \leftarrow \Theta)$.

Indirect contributions It is possible that some part of $L(x \to \Theta)$ contributes to $\Phi(S)$ through one or more reflections at several surfaces. The radiance $L(x \to \Theta)$ travels along a straight path and reaches a surface point $r(x, \Theta)$. Energy is reflected at this surface point according to the BRDF. Thus, there is a hemisphere of directions at $r(x, \Theta)$, each emitting a differential radiance value as a result of the reflection of the radiance $L(r(x, \Theta) \leftarrow -\Theta)$. By integrating the importance values for all these new directions, we have a new term for $W(x \leftarrow \Theta)$.

Both terms combined produces the following equation:

$$W(x \leftarrow \Theta) = W_e(x \leftarrow \Theta) + \int_{\Omega_z} f_r(z, \Psi \leftrightarrow -\Theta) W(z \leftarrow \Psi) \cos(N_r(x, \Theta), \Psi) d\omega_\Psi$$

with

$$z = r(x, \Theta)$$

Mathematically, this equation is identical to the transport equation of incident radiance, and thus, the notion *incidence* can be attributed to importance. The source function $W_e = 1$ if x is visible through the pixel and Θ is a direction pointing through the pixel to the aperture of the virtual camera.

To enhance the analogy with radiance as a transport quantity, exitant importance can be defined as:

$$W(x \to \Theta) = W((r, \Theta) \leftarrow -\Theta)$$

and also:

$$W(x \to \Theta) = W_e(x \to \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) W(x \leftarrow \Psi) cos(N_x, \Psi) d\omega_\Psi$$

An expression for the flux of through every pixel, based on the importance function, can now be written. Only the importance of the light sources needs to be considered when computing the flux:

$$\Phi(S) = \int_A \int_{\Omega_x} L_e(x \to \Theta) W(x \leftarrow \Theta) cos(N_x, \Theta) d\omega_\Theta dA_x$$

It is also possible to write $\Phi(S)$ in the following form:

$$\Phi(S) = \int_A \int_{\Omega_x} L_e(x \leftarrow \Theta) W(x \to \Theta) cos(N_x, \Theta) d\omega_\Theta dA_x$$

and also:

$$\begin{aligned} \Phi(S) &= \int_A \int_{\Omega_x} L(x \to \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x \\ \Phi(S) &= \int_A \int_{\Omega_x} L(x \leftarrow \Theta) W_e(x \to \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x \end{aligned}$$

There are two approaches to solve the global illumination problem: The first approach starts from the pixel, and the radiance values are computed by solving one of the transport equations describing radiance. A second approach computes the flux starting from the light sources, and computes for each light source the corresponding importance value. If one looks at various algorithms in some more detail:

- Stochastic ray tracing propagates importance, the surface area visible through each pixel being the source of importance. In a typical implementation, the importance is never explicitly computes, but is implicitly done by tracing rays through the scene and picking up illumination values from the light sources.

- Light tracing is the dual algorithm of ray tracing. It propagates radiance from the light sources, and computes the flux values at the surfaces visible through each pixel.

- Bidirectional ray tracing propagates both transport quantities at the same time, and in an advanced form, computes a weighted average of all possible inner products at all possible interactions.

8.3 Path formulation

The above description of global illumination transport algorithms is based on the notion of radiance and importance. One can also express global transport by considering path-space, and computing a transport measure over each individual path. Path-space encompasses all possible paths of any length. Integrating a transport measure in path-space then involves generating the correct paths (e.g. random paths can be generated using an appropriate Monte Carlo sampling procedure), and evaluating the throughput of energy over each generated path. This view was developed by Spanier and Gelbard and introduced into rendering by Veach.

$$\Phi(S) = \int_{\Omega^*} f(\overline{x}) d\mu(\overline{x})$$

in which Ω^* is the path-space, $\overline{x}$ is a path of any length and $d\mu(\overline{x})$ is a measure in path space . $f(\overline{x})$ describes the throughput of energy and is a succession of $G(x, y)$, $V(x, y)$ and BRDF evaluations, together with a L_e and W_e evaluation at the beginning and end of the path.

An advantage of the path formulation is that paths are now considered to be the sample points for any integration procedure. Algorithms such as Metropolis light transport or bidirectional ray tracing are often better described using the path formulation.

8.4 Simple stochastic ray tracing

In any pixel-driven rendering algorithm we need to use the rendering equation to evaluate the appropriate radiance values. The most simple algorithm to compute this radiance value is to apply a basic and straightforward MC integration scheme to the standard form of the rendering equation:

$$\begin{aligned} L(x \to \Theta) &= L_e(x \to \Theta) + L_r(x \to \Theta) \\ &= L_e(x \to \Theta) + \int_{\Omega_x} L(x \leftarrow \Psi) f_r(x, \Theta \leftrightarrow \Psi) \cos(\Psi, N_x) d\omega_\Psi \end{aligned}$$

The integral is evaluated using MC integration, by generating N random directions Ψ_i over the hemisphere Ω_x, according to some pdf $p(\Psi)$. The estimator for $L_r(x \to \Theta)$ is given by:

$$\langle L_r(x \to \Theta) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{L(x \leftarrow \Psi_i) f_r(x, \Theta \leftrightarrow \Psi_i) \cos(\Psi_i, N_x)}{p(\Psi_i)}$$

$L(x \leftarrow \Psi_i)$, the incident radiance at x, is unknown. It is now necessary to trace the ray leaving x in direction Ψ_i through the scene to find the closest intersection point $r(x, \Psi)$. Here, another radiance evaluation is needed. The result is a recursive procedure to evaluate $L(x \leftarrow \Psi_i)$, and as a consequence, a path, or a tree of paths if $N > 1$, is generated in the scene.

These radiance evaluations will only yield a non-zero value, if the path hits a surface for which L_e has a value different from 0. In other words, in order to compute a contribution to the illumination of a pixel, the recursive path needs to reach at least one of the light sources in the scene. If the light sources are small, the resulting image will therefore mostly be black. This is expected, because the algorithm generates paths, starting at a point visible through a pixel, and slowly working towards the light sources in a very uncoordinated manner.

8.5 Russian Roulette

The recursive path generator described above needs a stopping condition to prevent the paths being of infinite length. We want to cut off the generation of paths, but at the same time, we have to be very careful about not introducing any bias into the image generations process. Russian Roulette addresses the problem of keeping the lengths of the paths manageable, but at the same time leaves room for exploring all possible paths of any length. Thus, an unbiased image can still be produced.

The idea of Russian Roulette can best be explained by a simple example: suppose one wants to compute a value V. The computation of V might be computationally very expensive, so we introduce a random variable r, which is uniformly distributed over the interval $[0, 1]$. If r is larger than some threshold value $\alpha \in [0, 1]$, we proceed with computing V. However, if $r \leq \alpha$, we do not compute V, and assume $V = 0$. Thus, we have a random experiment, with an expected value of $(1 - \alpha)V$. By dividing this expected value by $(1 - \alpha)$, an unbiased estimator for V is maintained.

If V requires recursive evaluations, one can use this mechanism to stop the recursion. α is called the absorption probability. If α is small, the recursion will continue many times, and the final computed value will be more accurate. If α is large, the recursion will stop sooner, and the estimator will have a higher variance. In the context of our path tracing algorithm, this means that either accurate paths of a long length are generated, or very short paths which provide a less accurate estimate.

In principle any value for α can be picked, thus controlling the recursive depth and execution time of the algorithm. $1 - \alpha$ is often set to be equal to the hemispherical reflectance of the material of the surface. Thus, dark surfaces will absorb the path more easily, while lighter surfaces have a higher chance of reflecting the path.

8.6 Indirect Illumination

In most path tracing algorithms, direct illumination is explicitly computed separately from all other forms of illumination (see previous chapter on direct illumination). This section outlines some strategies for computing the indirect illumination in a scene. Computing the indirect illumination is usually a harder problem, since one does not know where most important contributions are located. Indirect illumination consists of the light reaching a target point x after at least one reflection at an intermediate surface between the light sources and x.

8.6.1 Hemisphere sampling

The rendering equation can be split in a direct and indirect illumination term. The indirect illumination (i.e. not including any direct contributions from light sources to the point x) contribution to $L(x \to \Theta)$ is written as:

$$L_{indirect}(x \to \Theta) = \int_{\Omega_x} L_r(r(x, \Psi) \to -\Psi) f_r(x, \Theta \leftrightarrow \Psi) \cos(\Psi, N_x) d\omega_\Psi$$

The integrand contains the reflected terms L_r from other points in the scene, which are themselves composed of a direct and indirect illumination part. In a closed environment, $L_r(r(x, \Psi) \to -\Psi)$ usually has a non-zero value for all (x, Ψ) pairs. As a consequence, the entire hemisphere around x needs to be considered as the integration domain.

The most general MC procedure to evaluate indirect illumination, is to use any hemispherical pdf $p(\Psi)$, and generating N random directions Ψ_i. This produces the following estimator:

$$\langle L_{indirect}(x \to \Theta) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{L_r(r(x, \Psi_i) \to -\Psi_i) f_r(x, \Theta \leftrightarrow \Psi_i) \cos(\Psi_i, N_x)}{p(\Psi_i)}$$

In order to evaluate this estimator, for each generated direction Ψ_i, the BRDF and the cosine term are to be evaluated, a ray from x in the direction of Ψ_i needs to be traced, and the reflected radiance $L_r(r(x, \Psi_i) \to -\Psi_i)$ at the closest intersection point $r(x, \Psi_i)$ has to be evaluated. This last evaluation shows the recursive nature of indirect illumination, since this reflected radiance at $r(x, \Psi_i)$ can be split again in a direct and indirect contribution.

The simplest choice for $p(\Psi)$ is $p(\Psi) = 1/2\pi$, such that directions are sampled proportional to solid angle. Noise in the resulting picture will be caused by variations in the BRDF and cosine evaluations, and variations in the reflected radiance L_r at the distant points.

The recursive evaluation can again be stopped using Russian Roulette, in the same way as was done for simple stochastic ray tracing. Generally, the local hemispherical reflectance is used as an appropriate absorption probability. This choice can be explained intuitively: One only wants to spend work (i.e. tracing rays and evaluating $L_{indirect}(x)$) proportional to the amount of energy present in different parts of the scene.

8.6.2 Importance sampling

Uniform sampling over the hemisphere does not use any knowledge about the integrand in the indirect illumination integral. However, this is necessary to reduce noise in the final image, and thus, some form of importance sampling is needed. Hemispherical pdf's proportional (or approximately proportional) to any of the following factors can be constructed:

Cosine sampling

Sampling directions proportional to the cosine lobe around the normal N_x prevents directions to be sampled near the horizon of the hemisphere where $\cos(\Psi, N_x)$ yields a very low value, and thus possibly insignificant contributions to the computed radiance value.

BRDF sampling

BRDF sampling is a good noise-reducing technique when a glossy or highly specular BRDFs is present. It diminishes the probability that directions are sampled where the BRDF has a low value or zero value. Only for a few selected BRDF models, however, is it possible to sample exactly proportional to the BRDF. Even better would be trying to sample proportional to the product of the BRDF and the cosine term. Analytically, this is even more difficult to do, except in a few rare cases where the BRDF model has been chosen carefully.

Incident radiance field sampling

A last technique that can be used to reduce variance when computing the indirect illumination is to sample a direction Ψ according to the incident radiance values $L_r(x \leftarrow \Psi)$. Since this incident radiance is generally unknown, an adaptive technique needs to be used, where an approximation of $L_r(x \leftarrow \Psi)$ is constructed during the execution of the rendering algorithm.

8.6.3 Overview

It is now possible to build a full global illumination renderer using stochastic path tracing. The efficiency, accuracy and overall performance of the complete algorithm will be determined by the choice of all of the following parameters. As is usual in MC evaluations, the more samples or rays are generated, the less noisy the final image will be.

Number of viewing rays per pixel The amount of viewing rays through each pixel is responsible for effects such as aliasing at visible boundaries of objects or shadows.

Direct Illumination:

- The total number of shadow rays generated at each surface point x;
- The selection of a single light source for each shadow ray;
- The distribution of the shadow ray over the area of the selected light source.

Indirect Illumination (hemisphere sampling):

- Number of indirect illumination rays;
- Exact distribution of these rays over the hemisphere (uniform, cosine, ...);
- Absorption probabilities for Russian Roulette.

The better one makes use of importance sampling, the better the final image and the less noise there will be. An interesting question is, given a maximum amount of rays one can use per pixel, how should these rays best be distributed to reach the highest possible accuracy for the full global illumination solution? This is still an open problem. There are generally accepted ’default’ choices, but there are no hard and fast choices. It generally is accepted that branching out equally at all levels of the tree is less efficient. For indirect illumination, a branching factor of 1 is often used after the first level. Many implementations even limit the indirect rays to one per surface point, and compensate by generating more viewing rays.

Chapter 9

Metropolis Sampling

By Matt Pharr

A new approach to solving the light transport problem was recently developed by Veach and Guibas, who applied the Metropolis sampling algorithm [91, 87] (first introduced in Section 2.3.3 of these notes.)[1]. The Metropolis algorithm generates a series of samples from a non-negative function f that are distributed proportionally to f's value [52]. Remarkably, it does this without requiring anything more than the ability to evaluate f; it's not necessary to be able to integrate f, normalize it, and invert the resulting pdf. Metropolis sampling is thus applicable to a wider variety of sampling problems than many other techniques. Veach and Guibas recognized that Metropolis could be applied to the image synthesis problem after it was appropriately reformulated; they used it to develop a general and unbiased Monte Carlo rendering algorithm which they named Metropolis Light Transport (MLT).

MLT is notable for its *robustness*: while it is about as efficient as other unbiased techniques (e.g. bidirectional ray tracing) for relatively straightforward lighting problems, it distinguishes itself in more difficult settings where most of the light transport happens along a small fraction of all of the possible paths through the scene. Such settings were difficult for previous algorithms unless they had specialized advance-knowledge of the light transport paths in the scene (e.g. "a lot of light is coming through that doorway"); they thus suffered from noisy images due

[1]We will refer to the Monte Carlo sampling algorithm as "the Metropolis algorithm" here. Other commonly-used shorthands for it include M(RT)^2, for the initials of the authors of the original paper, and Metropolis-Hastings, which gives a nod to Hastings, who generalized the technique [22]. It is also commonly known as Markov Chain Monte Carlo.

to high variance because most of they generated would have a low contribution, but when they randomly sampled an important path, there would be a large spike in the contribution to the image. In contrast, the Metropolis method leads to algorithms that naturally and automatically adapt to the subtleties of the particular transport problem being solved.

The basic idea behind MLT is that a sequence of light-carrying paths through the scene is computed, with each path generated by mutating the previous path in some manner. These mutations are done in a way that ensures that the overall distribution of sampled paths in the scene is proportional to the contribution these paths make to the image being generated. This places relatively few restrictions on the types of mutations that can be applied; in general, it is possible to invent unusual sampling techniques that couldn't be applied to other MC algorithms without introducing bias.

MLT has some important advantages compared to previous unbiased approaches to image synthesis:

- *Path re-use*: because paths are often constructed using some of the segments of the previous one, the incremental cost (i.e. number of rays that need to be traced) for generating a new path is much less than the cost of generating a path from scratch.

- *Local exploration*: when paths that make large contributions to the final image are found, it's easy to sample other paths that are similar to that one by making small perturbations to the path.

The first advantage increases overall efficiency by a relatively fixed amount (and in fact, path re-use can be applied to some other light transport algorithms.) The second advantage is the more crucial one: once an important transport path has been found, paths that are similar to it (which are likely to be important) can be sampled. When a function has a small value over most of its domain and a large contribution in only a small subset of it, local exploration amortizes the expense (in samples) of the search for the important region by letting us stay in that area for a while.

In this chapter, we will introduce the Metropolis sampling algorithm and the key ideas that it is built on. We will then show how it can be used for some low-dimensional sampling problems; this setting allows us to introduce some of the important issues related to the full Metropolis Light Transport algorithm without getting into all of the tricky details. We first show its use in one-dimension. We then demonstrate how it can be used to compute images of motion-blurred objects; this pushes up the domain of the problem to three dimensions and also provides a simpler setting in which to lay more groundwork. Finally, we will build on this basis to make connections with and describe the complete MLT algorithm. We will not attempt to describe every detail of MLT here; however the full-blown presentation in MLT paper [91] and the MLT chapter in Veach's thesis [87] should be much more approachable with this groundwork.

9.1 Overview

The Metropolis algorithm generates a set of samples x_i from a function f, which is defined over a state space Ω, $f : \Omega \to \mathbb{R}$. (See Figure 9.1 for notation used in this chapter.) After the first sample x_0 is selected (more details on this later), each subsequent sample x_i is generated by proposing a random *mutation* to x_{i-1} to compute a proposed sample x'. The mutation may be accepted or rejected, and x_i is set to either x' or x_{i-1}, respectively. When these transitions from one state to another are chosen subject to a few limitations (described shortly), the distribution of x_i values that results eventually reaches equilibrium; this distribution is the *stationary distribution*.

The way in which mutations are accepted or rejected guarantees that in the limit, the distribution of the set of samples $x_i \in \Omega$ is proportional to $f(x)$'s density function. Thus, even if we are unable to integrate f analytically, normalize it, and invert the integral so that we can sample from it, the Metropolis algorithm still generates samples from f's normalized density function f_{pdf}.

9.1.1 Detailed Balance

In order to generate this particular distribution of samples, we need to accept or reject mutations in a particular way. Assume that we have a method of proposing mutations that makes a given state x into a proposed state x' (this might be done by perturbing x in some way, or even by generating a completely new value.) We also need to be able to compute a tentative transition function $T(x \to x')$ that gives the probability density of the mutation technique's proposing a transition to x', given that the current state is x. (In general, the need to be able to compute this density is the only limitation on how we might choose to mutate—there's a lot of

ξ	Uniform random number between 0 and 1
$f(x)$	Function being sampled
Ω	State space over which f is defined
x	A sample value, $x \in \Omega$
x'	A proposed new sample value, based on some mutation strategy
x_i	The ith sample in a Markov chain $x_0, x_1, \ldots, x_n$ generated by the Metropolis sampling algorithm
$p(x)$	A probability density function
$\mathbf{I}(f)$	The integrated value of $f(x)$ over all of Ω, $\mathbf{I}(f) = \int_\Omega f(x) \mathrm{d}\Omega$
f_{pdf}	Normalized probability density of f's distribution, $f_{\mathrm{pdf}} = f / \mathbf{I}(f)$
$\hat{f}(x)$	Reconstructed function that approximates f_{pdf}
$T(x \to x')$	Density of proposed transition from one state x to another x'
$a(x \to x')$	Acceptance probability of mutating from one state x to another x'
$h_j(u, v)$	Value of the jth pixel's reconstruction filter for an image sample at (u, v)
I_j	Image function value at pixel j

Figure 9.1: Notation used in this chapter

freedom left over!) We also define an *acceptance probability* $a(x \to x')$ that gives the probability of accepting a proposed mutation from x to x'.

The key to the Metropolis algorithm is the definition of $a(x \to x')$ such that the distribution of samples is proportional to $f(x)$. If the random walk is already in equlibrium, the transition density between any two states must be equal:[2]

$$f(x)\, T(x \to x')\, a(x \to x') = f(x')\, T(x' \to x)\, a(x' \to x). \tag{9.1}$$

This property is called *detailed balance*. Since f and T are set, Equation 9.1 tells us how a must be defined. In particular, a definition of a that maximizes the rate at which equilibrium is reached is

$$a(x \to x') = \min\left(1, \frac{f(x')\, T(x' \to x)}{f(x)\, T(x \to x')}\right) \tag{9.2}$$

One thing to notice from Equation 9.2 is that if the transition probability density is the same in both directions, the acceptance probability simplifies to

$$a(x \to x') = \min\left(1, \frac{f(x')}{f(x)}\right) \tag{9.3}$$

[2]See Kalos and Whitlock [38] or Veach's thesis [87] for a rigorous derivation.

```
x = x0
for i = 1 to n
    x' = mutate(x)
    a = accept(x, x')
    if (random() < a)
        x = x'
    record(x)
```

Figure 9.2: Pseudocode for the basic Metropolis sampling algorithm. We generate n samples by mutating the previous sample and computing acceptance probabilities as in Equation 9.2. Each sample x_i is then recorded in a data structure.

For some of the basic mutations that we'll use, this condition on T will be met, which simplifies the implementation.

Put together, this gives us the basic Metropolis sampling algorithm shown in pseudo-code in Figure 9.2. We can apply the algorithm to estimating integrals such as $\int f(x)g(x)\mathrm{d}\Omega$. The standard Monte Carlo estimator, Equation 2.6, says that

$$\int_\Omega f(x)g(x)\,\mathrm{d}\Omega \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)g(x_i)}{p(x_i)} \tag{9.4}$$

where x_i are sampled from a density function $p(x)$. Thus, if we apply Metropolis sampling and generate a set of samples $x_1, \ldots, x_N$, from a density function $f_{\mathrm{pdf}}(x)$ proportional to $f(x)$, we have

$$\int_\Omega f(x)g(x)\,\mathrm{d}\Omega \approx \left[\frac{1}{N}\sum_{i=1}^{N} g(x_i)\right]\cdot \mathbf{I}(f) \tag{9.5}$$

where $\mathbf{I}(f)$ is the value of $f(x)$ integrated over all of Ω.

9.1.2 Expected Values

Because the Metropolis algorithm naturally avoids parts of Ω where $f(x)$'s value is relatively low, few samples will be accumulated there. In order to get some information about $f(x)$'s behavior in such regions, the *expected values* technique can be used to enhance the basic Metropolis algorithm.

At each mutation step, we record a sample at both the current sample x and the proposed sample x', regardless of which one is selected by the acceptance criteria.

```
x = x0
for i = 1 to n
    x′ = mutate(x)
    a = accept(x, x′)
    record(x, (1-a) * weight)
    record(x′, a * weight)
    if (random() < a)
        x = x′
```

Figure 9.3: The basic Metropolis algorithm can be improved using *expected values*. We still decide which state to transition into as before, but we record a sample at each of x and x', proportional to the acceptance probability. This gives smoother results, particularly in areas where f's value is small, where otherwise few samples would be recorded.

Each of these recorded samples has a weight associated with it, where the weights are the probabilities $(1-a)$ for x and a for x', where a is the acceptance probability. Comparing the pseudocode in Figures 9.2 and 9.3, we can see that in the limit, the same weight distribution will be accumulated for x and x'. Expected values more quickly gives us more information about the areas where $f(x)$ is low, however.

Expected values doesn't change the way we decide which state, x or x' to use at the next step; that part of the computation remains the same.

9.2 One Dimensional Setting

Consider using Metropolis to sample the following one-dimensional function, defined over $\Omega = [0, 1]$ and zero everywhere else (see Figure 9.4).

$$f^1(x) = \begin{cases} (x - .5)^2 & : \quad 0 \le x \le 1 \\ 0 & : \quad \text{otherwise} \end{cases} \tag{9.6}$$

For this example, assume that we don't actually know the exact form of f^1—it's just a black box that we can evaluate at particular x values. (Clearly, if we knew that f^1 was just Equation 9.6, there'd be no need for Monte Carlo sampling!)

We'd like to generate a set of samples of f^1 using Metropolis sampling. These samples will be used to reconstruct a new function $\hat{f}^1$ that approximates f^1's pdf. A random choice between the strategies will be made each time a mutation is proposed, according to a desired distribution of how frequently each is to be used.

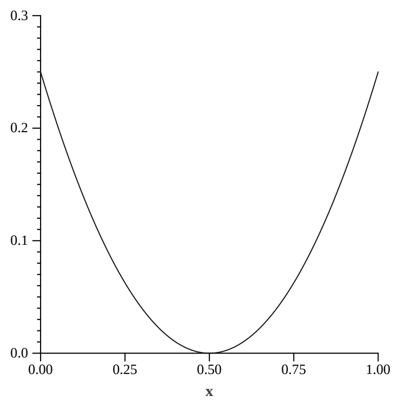


Figure 9.4: Graph of the function f^1, which is used to demonstrate Metropolis sampling in this section.

9.2.1 Mutations and Transition Functions

We will first describe two basic mutation strategies, each of which depends on a uniform random number ξ between zero and one. Our first mutation, mutate_1, discards the current sample x and uniformly samples a new one x' from the entire state space $[0, 1]$. Mutations like this one that sample from scratch are important to make sure that we don't get stuck in one part of state space and never sample the rest of it (an example of the problems that ensue when this happens will be shown below.) The transition function for this mutation is straightforward. For mutate_1, since we are uniformly sampling over $[0, 1]$, the probability density is uniform over the entire domain; in this case, the density is just one everywhere. We have

$$\begin{aligned}\mathrm{mutate}_1(x) &\rightarrow \xi \\ T_1(x \rightarrow x') &= 1\end{aligned}$$

The second mutation adds a random offset between $\pm .05$ to the current sample x in an effort to sample repeatedly in the parts of f that make a high contribution to the overall distribution. The transition probability density is zero if x and x' are far enough away that mutate_2 will never mutate from one to the other; otherwise the density is constant. Normalizing the density so that it integrates to one over its domain gives the value $\frac{1}{0.1}$.

$$\begin{aligned} \text{mutate}_2(x) &\rightarrow x + .1 * (\xi - .5) \\ T_2(x \rightarrow x') &= \begin{cases} \frac{1}{0.1} & : \ |x - x'| \leq .05 \\ 0 & : \ \text{otherwise} \end{cases} \end{aligned}$$

The second mutation is important for overall efficiency: when we find a sample x_i where $f^1(x_i)$ is larger than most other values of $f^1(x)$, we would like to examine samples close to x_i in the state space, since $f(x)$ is also likely to be large there. Furthermore, this mutation has the important property that it makes us more likely to accept proposed mutations: if we only used the first mutation strategy, we would find it difficult to mutate away from a sample x_i where $f^1(x_i)$ had a relatively large value; proposed transitions to other states where $f^1(x') \ll f(x_i)$ are rejected with high probability, so many samples in a row would be accumulated at x_i. (Recall the definition of the acceptance probability $a(x \rightarrow x')$ in Equation 9.3.) Staying in the same state for many samples in a row leads to increased variance—intuitively, it makes sense that the more we move around Ω, the better the overall results will be. Adding the second mutation to the mix increases the probability of being able to accept mutations around such samples where $f^1(x)$ is relatively high, giving a better distribution of samples in the end.

9.2.2 Start-up bias

Before we can go ahead and use the Metropolis algorithm, one other issue must be addressed: *start-up bias*. The transition and acceptance methods above tell us how to generate new samples x_{i+1}, but all presuppose that the current sample x_i has itself *already* been sampled with probability proportional to f. A commonly used solution to this problem is to run the Metropolis sampling algorithm for some number of iterations from an arbitrary starting state, discard the samples that are generated, and then start the process for real, assuming that that has brought us to an appropriately sampled x value. This is unsatisfying for two reasons: first, the expense of taking the samples that were then discarded may be high, and second, we can only guess at how many initial samples must be taken in order to remove start-up bias.

Veach proposes another approach which is unbiased and straightforward. If an alternative sampling method is available, we sample an initial value x_0 using any density function $x_0 \sim p(x)$. We start the Markov chain from the state x_0, but we weight the contributions of all of the samples that we generate by the weight

$$w = \frac{f(x_0)}{p(x_0)}.$$

This method eliminates start-up bias completely and does so in a predictable manner.

The only potential problem comes if $f(x_0) = 0$ for the x_0 we chose; in this case, all samples will have a weight of zero, leading to a rather boring result. This doesn't mean that the algorithm is biased, however; the expected value of the result still converges to the correct distribution (see [87] for further discussion and for a proof of the correctness of this technique.)

To reduce variance from this step, we can instead sample a set of N candidate sample values, $y_1, \ldots, y_N$, defining a weight for each by

$$w_i = \frac{f(y_i)}{p(y_i)}. \tag{9.7}$$

We then choose the starting x_0 sample for the Metropolis algorithm from the y_i with probability proportional to their relative weights and compute a sample weight w as the average of all of the w_i weights. All subsequent samples x_i that are generated by the Metropolis algorithm are then weighted by the sample weight w.

For our particular f^1 example above, we only need to take a single sample with a uniform pdf over Ω, since $f^1(x) > 0$ except for a single point in Ω which there is zero probability of sampling.

$$x_0 = \xi$$

The sample weight w is then just $f^1(x_0)$.

9.2.3 Initial Results

We can now run the Metropolis algorithm and generate samples x_i of f^1. At each transition, we have two weighted samples to record (recall Figure 9.3.) A simple approach for reconstructing the approximation to f^1's probability distribution $\hat{f}^1$ is just to store sums of the weights in a set of buckets of uniform width; each sample falls in a single bucket and contributes to it. Figure 9.5 shows some results. For both graphs, we followed a chain of 10,000 mutations, storing the sample weights in fifty buckets over $[0, 1]$. The weighting method for eliminating start-up bias was used.

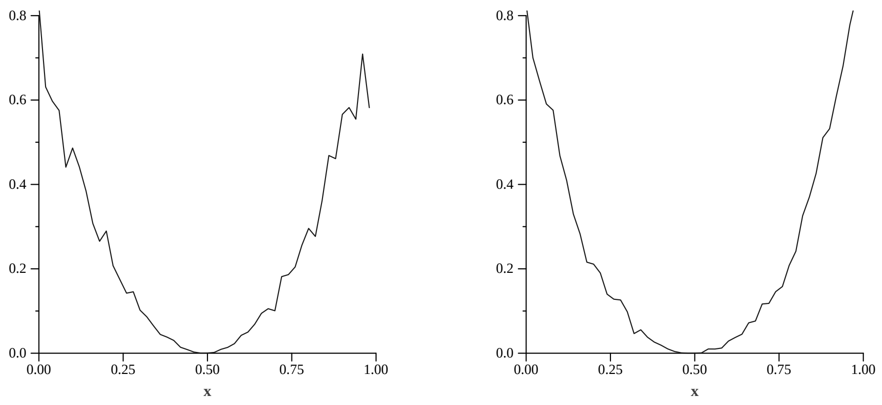


Figure 9.5: On the left, we always mutate by randomly selecting a completely new x value. Convergence is slow, but the algorithm is finding the right distribution. On the right, we perturb the current sample by $\pm .05$ 90% of the time, and pick a completely new x value the remaining 10%.

On the left graph, we used only mutate_1 when a new x' value is to be proposed. This alone isn't a very useful mutation, since it doesn't let us take advantage of the times when we find ourselves in a region of Ω where f has a relatively large value and generate many samples in that neighborhood. However, the graph does suggest that the algorithm is converging to the correct distribution.

On the right, we randomly chose between mutate_1 and mutate_2 with probabilities of 10% and 90%, respectively. We see that for the same number of samples taken, we converge to f's distribution with less variance. This is because we are more effectively able to concentrate our work in areas where f's value is large, and propose fewer mutations to parts of state space where f's value is low. For example, if $x = .8$ and the second mutation proposes $x' = .75$, this will be accepted $f(.75)/f(.8) \approx 69\%$ of the time, while mutations from $.75$ to $.8$ will be accepted $\min(1, 1.44) = 100\%$ of the time. Thus, we see how the algorithm naturally tends to try to avoid spending time sampling around dip in the middle of the curve.

One important thing to note about these graphs is that the y axis has units that are different than those in Figure 9.4, where f^1 is graphed. Recall that we just have a set of samples distributed according to the probability density f^1_{pdf}; as such (for example), we would get the same sample distribution for another function $g = 2f^1$. If we wish to reconstruct an approximation to f^1 directly, we must compute a normalization factor and use it to scale f^1_{pdf}. We explain this process in more detail in Section 9.3.2.

9.2.4 Ergodicity

Figure 9.6 shows the surprising result of what happens if we only use mutate_2 to suggest sample values. On the left, we have taken 10,000 samples using just that mutation. Clearly, things have gone awry—we didn't generate *any* samples $x_i > .5$ and the result doesn't bear much resemblance to f^1.

Thinking about the acceptance probability again, we can see that it would take a large number of mutations, each with low probability of acceptance, to move x_i down close enough to $.5$ such that mutate_2's short span would be enough to get us to the other side. Since the Metropolis algorithm tends to keep us away from the lower-valued regions of f (recall the comparison of probabilities for moving from $.8$ to $.75$, versus moving from $.75$ to $.8$), this happens quite rarely. The right side of Figure 9.6 shows what happens if we take 300,000 samples. This was enough to make us jump from one side of $.5$ to the other a few times, but not enough to get us close to the correct distribution.

This problem is an example of a more general issue that must be addressed with Metropolis sampling: it's necessary that it be possible to reach all states $x \in \Omega$ where $f(x) > 0$ with non-zero probability. In particular, it suffices that $T(x \to x') > 0$ for all x and x' where $f(x) > 0$ and $f(x') > 0$. Although the first condition is in fact met when we use only mutate_2, many samples would be necessary in practice to converge to an accurate distribution. Periodically using mutate_1 ensures sufficiently better coverage of Ω such that that this problem goes away.

9.2.5 Mutations via PDFs

If we have a pdf that is similar to some component of f, then we can use that to derive one of our mutation strategies as well. Note that if we had a pdf that was exactly proportional to f, all this Metropolis sampling wouldn't be necessary, but lacking that we often can still find pdfs that approximate some part of the function being sampled. Adding such an extra mutation strategy to the mix can improve overall robustness of the Metropolis algorithm, by ensuring good coverage of important parts of state space.

If we can generate random samples from a probability density function p, $x \sim$

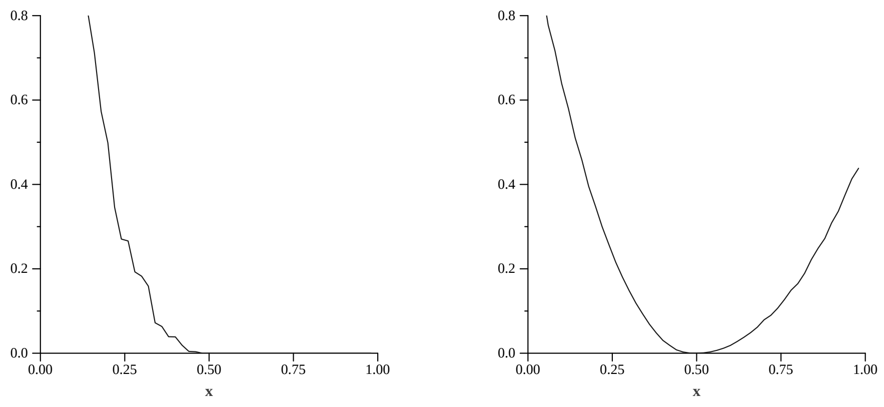


Figure 9.6: Two examples that show why it is important to periodically pick a completely new sample value. On the left, we ran 10,000 iterations using only mutate_2, and on the right, 300,000 iterations. It is very unlikely that a series of mutations will be able to move from one side of the curve, across 0.5, to the other side, since mutations to areas where f^1's value is low will usually be rejected. As such, the results are inaccurate for these numbers of iterations. (It's small solace that they would be correct in the limit.)

p, the transition function is straightforward:

$$T(x \to x') = p(x').$$

i.e. the current state x doesn't matter for computing the transition density: we propose a transition into a state x' with a density that depends only on the newly proposed state x' and not at all on the current state.

To apply this to the one-dimensional f^1 example, we add a mutation based on a linear probability density function p_1 that is somewhat similar to the shape of f^1. (see Figure 9.7).

$$p_1(x) = \begin{cases} 1.2 & : \quad x \le 1/3 \\ .6 & : \quad 1/3 < x \le 2/3 \\ 1.2 & : \quad 2/3 < x \end{cases}$$

Note that $p_1(x)$ is a valid probability density function; it integrates to one over the domain $[0, 1]$. We can apply the function inversion technique described in Section 2.3.1 to derive a sampling method. Inverting the integral $\xi = \int_0^x p_1(x)\mathrm{d}x$

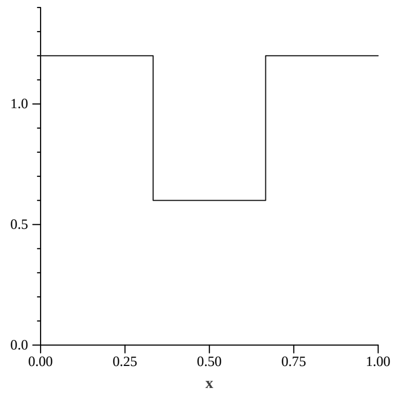


Figure 9.7: Linear pdf used to sample f^1. We can also develop mutation strategies based on pdfs that approximate some component of the function that we're sampling. Here, we're using a simple linear function that is roughly similar to the shape of f^1.

gives us:

$$x = \begin{cases} \frac{1}{3}\frac{\xi}{.4} & : \quad \xi \le .4 \\ \frac{1}{3} + \frac{1}{3}\frac{(\xi - .4)}{.2} & : \quad .4 < \xi \le .6 \\ \frac{2}{3} + \frac{1}{3}\frac{(\xi - .6)}{.4} & : \quad \xi > .6 \end{cases} \tag{9.8}$$

Our third mutation strategy, mutate_3, just generates a uniform random number ξ and proposes a mutation to a new state x' according to Equation 9.8. Results of using this mutation alone are shown in Figure 9.8; the graph is not particularly better than the previous results, but in any case, it is helpful to have a variety of methods with which to develop mutations.

9.3 Motion Blur

We will now show how Metropolis sampling can be used to solve a tricky problem with rendering motion-blurred objects. For objects that are moving at a high speed across the image plane, the standard distribution ray tracing approach can be inefficient; if a pixel is covered by an object for only a short fraction of the overall time, most of the samples taken in that pixel will be black, so that a large number are needed to accurately compute the pixel's color. We will show in this section how Metropolis can be applied to solve this problem more efficiently. In addition to being an interesting application of Metropolis, this also gives us an opportunity

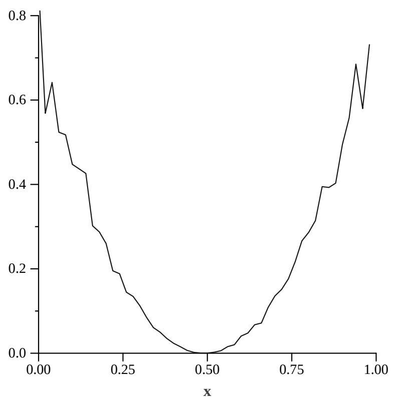


Figure 9.8: Result of reconstructing f^1's pdf using the new mutate_3.

to lay further groundwork before moving onto the complete MLT algorithm.

The basic setting of the motion blur problem is that we have a two-dimensional image plane, with some number of pixels, p_u and p_v, in the u and v directions. The time range is from zero to one. We define the scene's radiance function $L(u, v, t)$, which gives the radiance visible along the ray through the scene from the position (u, v) on the image plane, at time t. (L can be computed with standard ray tracing methods.) The state space Ω for this problem is thus the triples $(u, v, t) \in \mathbb{R}^3$, where L's value is zero for (u, v) outside the image and where $t < 0$ or $t > 1$. The measure is just the product measure $\mathrm{d}u\,\mathrm{d}v\,\mathrm{d}t$.

The key step to being able to apply Metropolis sampling to image synthesis problems is the definition of the *image contribution function* [87, Chapter 11, Section 3]. For an image with j pixels, each pixel I_j has a value that is the product of the pixel's image reconstruction filter h_j and the radiance L that contributes to the image:

$$I_j = \int_\Omega h_j(u, v)\, L(u, v, t)\,\mathrm{d}u\,\mathrm{d}v\,\mathrm{d}t$$

The standard Monte Carlo estimate of I_j is[3]

$$I_j \approx \frac{1}{N} \sum_{i=1}^{N} \frac{h_j(x_i)\, L(x_i)}{p(x_i)}, \tag{9.9}$$

where p is a probability density function used for sampling $x_i \in \Omega$ and where we

[3]Because the filter support of h_j is usually only a few pixels wide, a small number of samples x_i will contribute to each pixel.

have written h_j and L as functions of samples $x_i \in \mathbb{R}^3$ (even though h_j typically only depends on the u and v sample location of x_i.)

In order to be able to apply Metropolis sampling to the problem of estimating pixel values I_j, we can apply Equation 9.5 to rewrite this as

$$I_j \approx \frac{1}{N} \sum_{i=1}^{N} h_j(x_i) \cdot \left(\int_{\Omega} L(x) \, \mathrm{d}\Omega \right), \tag{9.10}$$

since the Metropolis sampling algorithm generates a set of samples x_i according to the distribution that exactly matches L's probability density function.

In the remainder of this section, we will describe an application of Metropolis sampling to the motion blur problem and compare the results to standard approaches. Our example scene is a series of motion-blurred balls, moving across the screen at increasing velocities from top-to-bottom (see Figure 9.9).

9.3.1 Basic Mutation Strategies

We will start with two basic mutation strategies for this problem. First, to ensure ergodicity, we generate a completely new sample 10% of the time. This is just like the one dimensional case of sampling f^1. Here, we choose three random numbers from the range of valid image and time sample values.

Our second mutation is a pixel and time perturbation that helps with local exploration of the space. Each time it is selected, we randomly move the pixel sample location up to ± 8 pixels in each direction (for a 512 by 512 image), and up to $\pm .01$ in time. If we propose a mutation that takes us off of the image or out of the valid time range, the transition is immediately rejected. The performance of the algorithm isn't too sensitive to these values, though see below for the results of some experiments where they were pushed to extremes.

The transition probabilities for each of these mutations are straightforward, analogous to the one dimensional examples.

Figure 9.9 shows some results. The top image was rendered with distribution ray tracing (with a stratified sampling pattern), and the bottom the Metropolis sampling approach with these two mutations was used. The sample total number of samples was taken for each. Note that Metropolis does equally well regardless of the velocity of the balls, while fast moving objects are difficult for distribution ray tracing to handle well. Because Metropolis sampling can locally explore the path space after it has found a sample that hits one of the balls, it is likely to find other samples that hit them as well, thus being more efficient—the small time perturbation is particularly effective for this. Note, however, that Metropolis doesn't do as well with the ball that is barely moving at all, while this is a relatively easy case for stratified sampling to handle well.

It's interesting to see the effect of varying the parameters to mutate_2. First, we tried greatly increasing the amount we can move in Ω, up to ± 80 pixels in each direction and $\pm .5$ in time. Figure 9.10 (top) shows the result. Because we are no longer doing a good job of local exploration of the space, the image is quite noisy. (One would expect it to degenerate to something like distribution ray tracing, but without the advantages of stratified sampling patterns that are easily applied in that setting.)

We then dialed down the parameters, to $\pm .5$ pixels of motion and $\pm .001$ in time, for a single mutation; see Figure 9.10 (bottom). Here the artifacts in the image are more clumpy—this happens because we find a region of state space with a large contribution but then have trouble leaving it and finding other important regions. As such, we don't do a good job of sampling the entire image.

As a final experiment, we replaced mutate_2 with a mutation that sampled the pixel and time deltas from an exponential distribution, rather than a uniform distribution. Given minimum and maximum pixel offsets $r_{\max}$ and $r_{\min}$ and time offsets $t_{\max}$ and $t_{\min}$, we computed

$$\begin{aligned} r &= r_{\max} \mathrm{e}^{-\log(r_{\max}/r_{\min})\xi} \\ dt &= t_{\max} \mathrm{e}^{-\log(t_{\max}/t_{\min})\xi} \end{aligned}$$

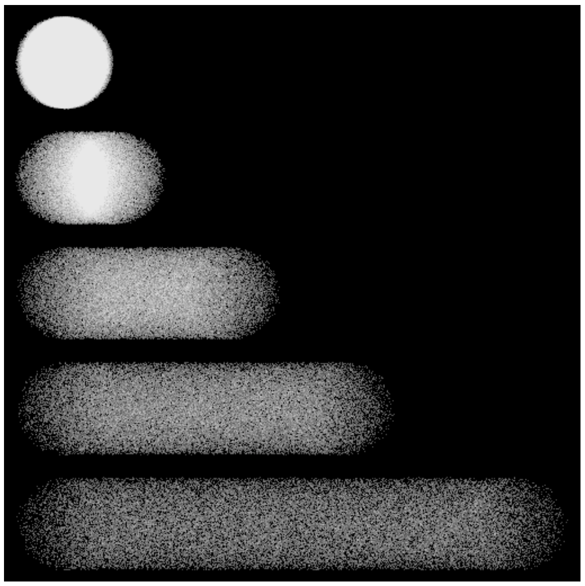
Given these offsets, a new pixel location was computed by uniformly sampling an angle $\theta = 2\pi\xi$, and the new image (u, v) coordinates were computed by

$$(u, v) = (r \sin\theta, r \cos\theta)$$

The new time value was computed by offsetting the old one by $\pm dt$, where addition and subtraction were chosen with equal probability.

We rendered an image using this mutation; the range of pixel offsets was $(.5, 40)$, and the range of time deltas was $(.0001, .03)$. Figure 9.11 shows the result. The improvement is not major, but is noticeable. In particular, see how noise is reduced along the edges of the fast-moving ball.

The advantage of sampling with an exponential distribution like this is that it naturally tries a variety of mutation sizes. It preferentially makes small mutations,



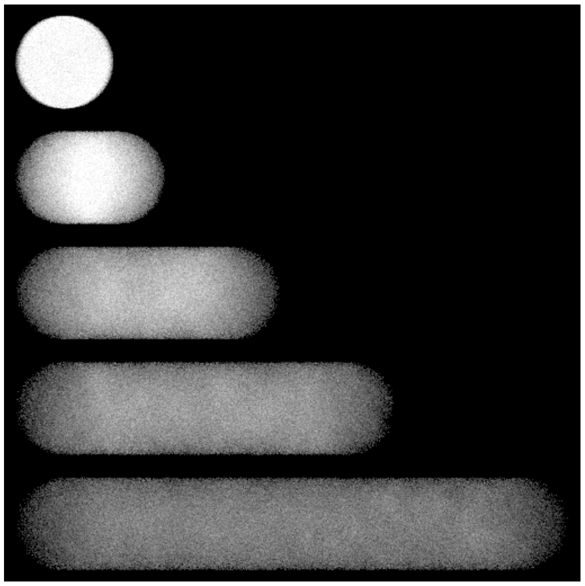
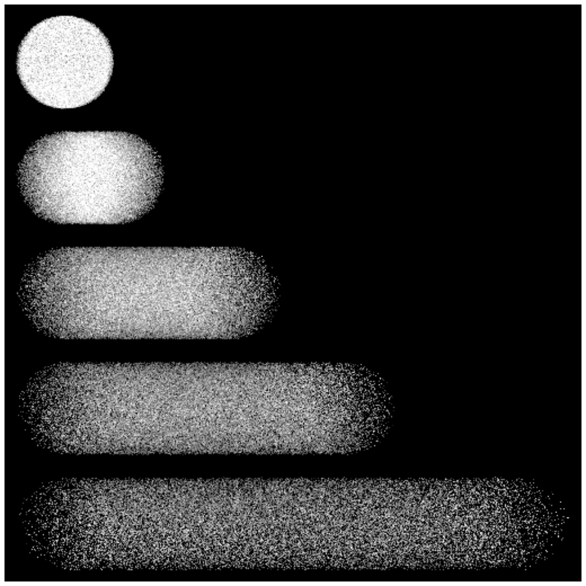


Figure 9.9: Basic motion blur results. On the top, we have applied distribution ray tracing with a stratified sampling pattern, and on the bottom, we have applied Metropolis sampling. The images are 512x512 pixels, with an average of 9 samples per pixel.



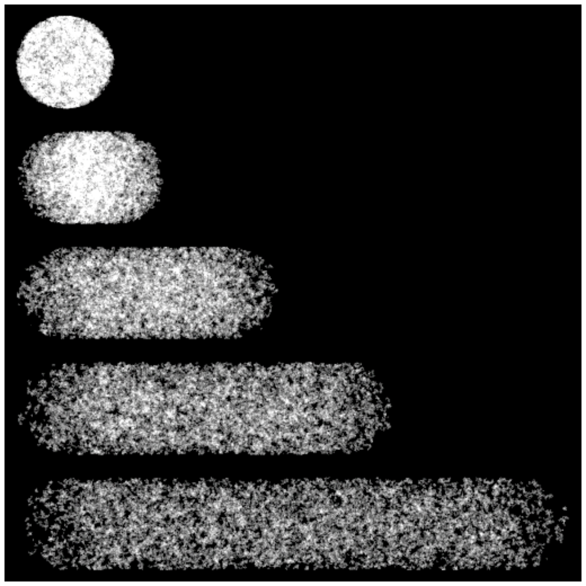


Figure 9.10: The effect of changing the parameters to the second motion blur mutation strategy. Top: if the mutations are too large, we get a noisy image. Bottom: too small a mutation leads to clumpy artifacts, since we're not doing a good job of exploring the entire state space.

close to the minimum magnitudes specified, which help locally explore the path space in small areas of high contribution where large mutations would tend to be rejected. On the other hand, because it also can make larger mutations, it also avoids spending too much time in a small part of path space, in cases where larger mutations have a good likelihood of acceptance.

9.3.2 Re-normalization

Recall that the set of samples generated by the Metropolis algorithm is from the normalized distribution f_{pdf} of the function that we apply it to. When we are computing an image, as in the motion blur example, this means that the image's pixel values need to be rescaled in order to be correct.

This problem can be solved with an additional short pre-processing step. We take a small number of random samples (e.g. 10,000) of the function f and estimate its integral over Ω. After applying Metropolis sampling, we have an image of sample densities. We then scale each pixel by the precomputed total image integral divided by the total number of Metropolis samples taken. It can be shown that the expected value is the original function f:

$$f(x) \approx \frac{1}{N} \sum_{i=1}^{N} \hat{f}(x_i) \, \mathbf{I}(f)$$

9.3.3 Adapting for large variation in f

When the image function I has regions with very large values, Metropolis sampling may not quite do what we want. For example, if the image has a very bright light source directly visible in some pixels, most of the Metropolis samples will naturally be clustered around those pixels. As a result, the other regions of the image will have high variance due to under-sampling. Figure 9.12 (top) shows an example of this problem. The bottom ball has been made a very bright red, 1,000 times brighter than the others; most of the samples concentrate on it, so the other balls are under-sampled.

Veach introduced *two-stage Metropolis* to deal with this problem [87]. Two-stage Metropolis attempts to keep the relative error at all pixels the same, rather than trying to just minimize absolute error. We do this by renormalizing the function L to get a new function L':

$$L'(x) = \frac{L(x)}{n(x)}$$

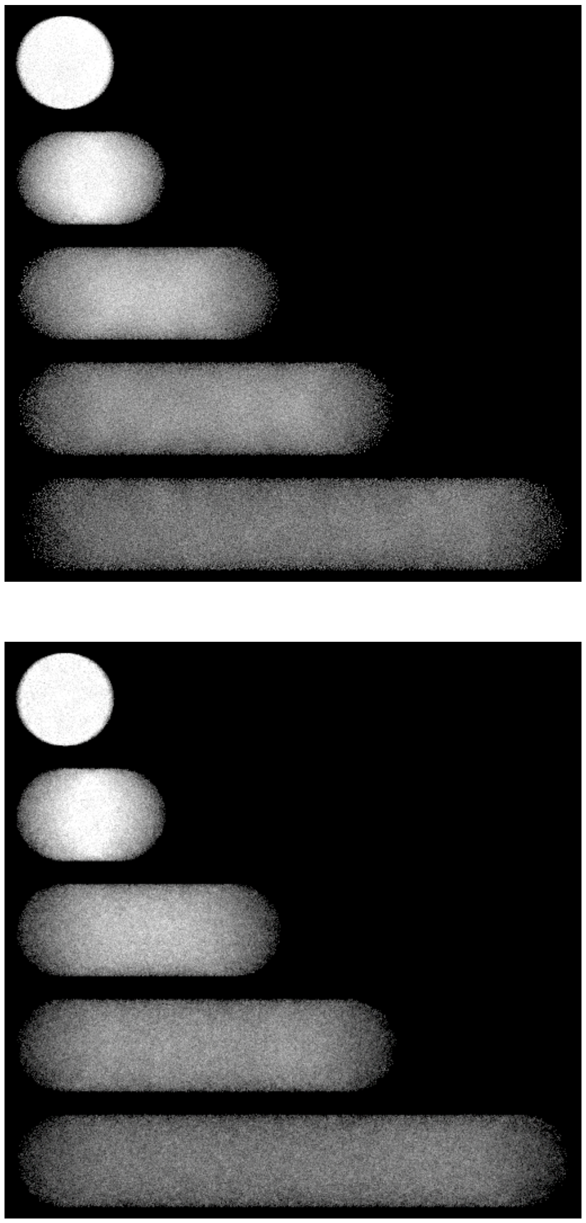


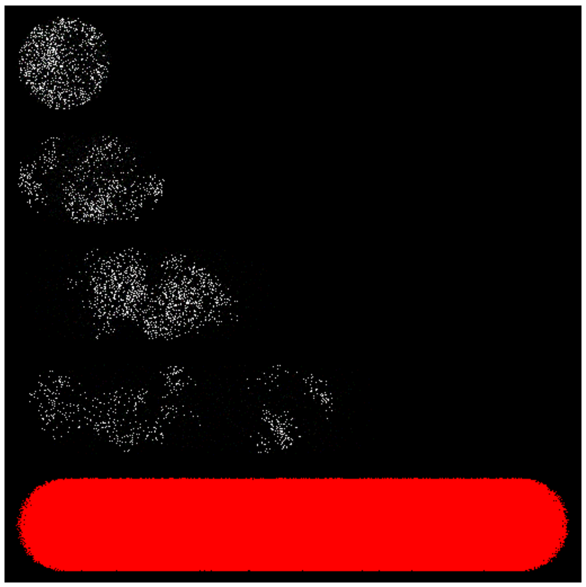
Figure 9.11: Comparison of Metropolis sampling with the first two mutation strategies (top) versus Metropolis sampling where the second strategy is replaced a mutation based on sampling pixel and time offsets from an exponential distribution (bottom). Note how noise is reduced along the edges of the fast-moving ball.

where n is a normalization function that roughly approximates $L(x)$'s magnitude, such that the range of values taken on by $L'(x)$ is much more limited. The Metropolis algorithm progresses as usual, just evaluating $L'(x)$ where it otherwise would have evaluated $L(x)$. The result is that samples are distributed more uniformly, resulting in a better image. We correct for the normalization when accumulating weights in pixels; by multiplying each weight by $n(x)$, the final image pixels have the correct magnitude.

For our example, we computed a normalization function by computing a low-resolution image (32 by 32 pixels) with distribution ray tracing and then blurring it. We then made sure that all pixels of this image had a non-zero value (we don't want to spend all of our sampling budget in areas where we inadvertently under-estimated $n(x)$, such that $L'(x) = L(x)/n(x)$ is large) and so we also set pixels in the normalization image with very low values to a fixed minimum. Applying Metropolis as before, we computed the image on the bottom of Figure 9.12. Here all of the balls have been sampled well, resulting in a visually more appealing result (even though absolute error is higher, due to the red ball being sampled with fewer samples.)

9.3.4 Color

For scenes that aren't just black-and-white, the radiance function $L(u, v, t)$ returns a spectral distribution in some form. This distribution must be converted to a single real value in order to compute acceptance probabilities (Equation 9.2). One option is to compute computing the luminance of the spectrum and use that; the resulting image (which is still generated by storing spectral values) is still correct, and there is the added advantage that the image is sampled according to its perceived visual importance.



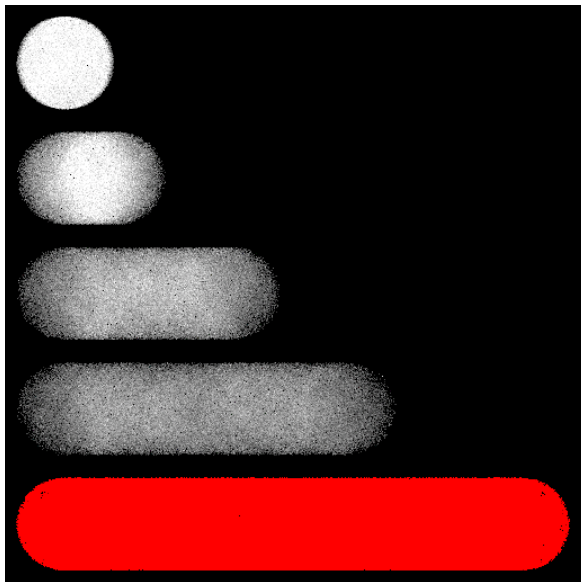


Figure 9.12: Two stage sampling helps when the image has a large variation in pixel brightness. Top: the red ball is much brighter than the others, resulting in too few samples being taken over the rest of the image. Bottom: by renormalizing the image function I, we distribute samples more evenly and generate a less noisy image.

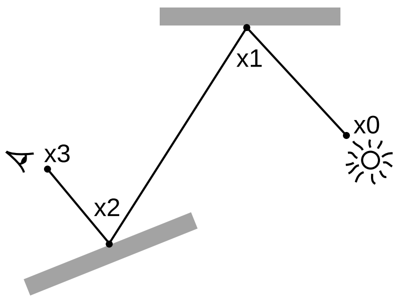


Figure 9.13: A path with three edges through a simple scene. The first vertex v_0 is on a light source, and the last, v_3 is at the eye.

9.4 Metropolis Light Transport

Using the groundwork of the last few sections, the Metropolis Light Transport algorithm can now be described. We will not explain all of its subtleties or all of the details of how to implement it efficiently; see Chapter 11 of Veach's thesis for both of these in great detail [87]. Rather, we will try to give a flavor for how it all works and will make connections between MLT and the sampling problems we have described in the previous few sections.

In MLT, the samples x from Ω are now sequences $v_0 v_1 \dots v_k$, $k \geq 1$, of vertices on scene surfaces. The first vertex, v_0, is on a light source, and the last, v_k is at the eye (see Figure 9.13). This marks a big change from our previous examples: the state space is now an infinite-dimensional space (paths with two vertices, paths with three vertices, ...). As long as there is non-zero probability of sampling any particular path length, however, this doesn't cause any problems theoretically, but it's is another idea that needs to be juggled when studying the MLT algorithm.

As before, the basic strategy is to propose mutations x' to paths x, accepting or rejecting mutations according to the detailed balance condition. The function $f(x)$ represents the differential radiance contribution carried along the path x, and the set of paths sampled will be distributed according to the image contribution function 9.10. (See [87] for a more precise definition of $f(x)$.) Expected values are also used as described previously to accumulate contributions at both the current path x's image location, as well as the image location for the proposed path x'.

A set of n starting paths $\bar{x}_i$ are generated with bidirectional path tracing, in a manner that eliminates startup bias. N candidate paths are sampled (recall Section 9.2.2), where $n \ll N$ and we select n of them along with appropriate weights.

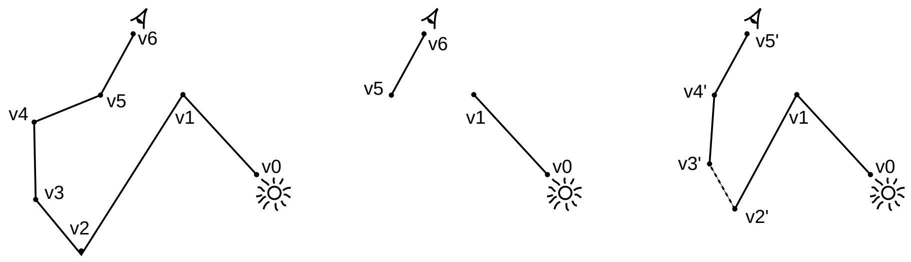


Figure 9.14: Example of a bidirectional mutation to a path. Left: we have a path of length 6 from the eye to the light source. Each vertex of the path represents a scattering event at a surface in the scene. Middle: the bidirectional mutation has decided to remove a sub-path from the middle of the current path; we are left with two short paths from the eye and from the light. Right: we have decided to add one vertex to each of the paths; their locations are computed by sampling the BRDF at the preceding vertices. We then trace a shadow ray between the new vertices v_2' and v_3' to complete the path.

We start with n separate paths, rather than just one, primarily to be able to improve stratification of samples within pixels (see Section 9.4.2 below as well as [87, Section 11.3.2].)

9.4.1 Path Mutations

A small set of mutations is applied to the paths in an effort to efficiently explore the space of all possible paths through the scene. The most important of the mutations is the *bidirectional mutation*. The bidirectional mutation is conceptually quite straightforward; a subpath from the middle of the current path is removed, and the two remaining ends are extended with zero or more new vertices (see Figure 9.14). The resulting two paths are then connected with a shadow ray; if it is occluded, the mutation is rejected, while otherwise the standard acceptance computation is performed.

Computation of the acceptance probability for bidirectional mutations requires that we figure out the values of the path contribution function $f(x)$ and $f(x')$ and the pair of $T(x \to x')$ densities. Recall from the introduction of the path integral that that $f(x)$ is a product of emitted importance from the eye, BRDFs values along the path, geometry terms $G(p, p') = \cos\theta_i \cos\theta_o / r^2$, and the differential irradiance from the light source; as such, computation of two the $f(x)$ values can be simplified by ignoring the terms of each of them that are shared between the two paths, since they just cancel out when the acceptance probability is computed.

Computation of the proposed transition densities is more difficult; see [87, Section 11.4.2.1] for a full discussion. The basic issue is that it is necessary to consider all of the possible ways that one *could* have sampled the path you ended up with, given the starting path. (For example, for the path in Figure 9.14, we might have generated the same path by sampling no new vertices from the light source, but two new vertices along the eye path.) This computation is quite similar in spirit to how importance sampling is applied to bidirectional path tracing.

The bidirectional mutation by itself is enough to ensure ergodicity; because there is some probability of throwing away the entire current path, we are guaranteed to not get stuck in some subset of path space.

Bidirectional mutations can be ineffective when a very small part of path space is where the most important light transport is happening—almost all of the proposed mutations will cause it to leave the interesting set of paths (e.g. those causing a caustic to be cast from a small specular object.) This problem can be ameliorated by adding *perturbations* to the mix; these perturbations try to offset some of the vertices of the current path from their current location, while still leaving the path mostly the same (e.g. preserving the mode of scattering—specular or non-specular, reflection or transmission, at each scattering event.)

One such perturbation is the *caustic perturbation* (see Figure 9.15). If the current path hits one or more specular surfaces before hitting a single diffuse surface and then the eye, then it's a caustic path. For such paths, we can make a slight shift to the outgoing direction from the light source and then trace the resulting path through the scene. If it hits all specular surfaces again and if the final diffuse surface hit is visible to the eye, we have a new caustic sample at a different image location. The caustic perturbation thus amortizes the possibly high expense of finding caustic paths.

Lens perturbations are based on a similar idea to caustic perturbations, where the direction of outgoing ray from the camera is shifted slightly, and then followed through the same set of types of scattering at scene surfaces. This perturbation is particularly nice since it keeps us moving over the image plane, and the more differently-located image samples we have, the better the quality of the final image.

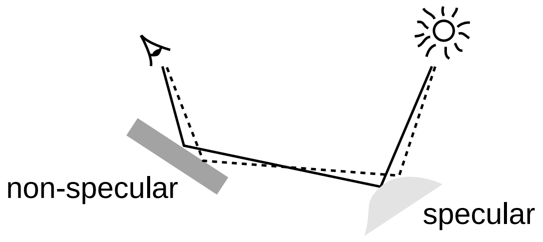


Figure 9.15: The caustic perturbation in action. Given an old path that leaves the light source, hits a specular surface, and then hits a non-specular surface before reaching the eye, we perturb the direction leaving the light source. We then trace rays to generate a new path through the scene and to the eye. (The key here is that because the last surface is non-specular, we aren't required to pick a particular outgoing direction to the eye—we can pick whatever direction is needed.)

9.4.2 Pixel Stratification

Another problem with using Metropolis to generate images is that random mutations won't do a good job of ensuring that pixel samples are well-stratified over the image plane. In particular, it doesn't even ensure that all of the pixels have *any* samples taken within them. While the resulting image is still unbiased, it may be perceptually less pleasing than an image computed with alternative techniques.

Veach has suggested a *lens subpath mutation* to address this problem. A set of pixel samples that must be taken is generated (e.g. via a Poisson-disk process, a stratified sampling pattern, etc.) As such, each pixel has some number of required sample locations associated with it. The new mutation type first sees if the pixel corresponding to the current sample path has any precomputed sample positions that haven't been used. If so, it mutates to that sample and traces a new path into the scene, following as many specular bounces as are found until a non-specular surface is found. This *lens subpath* is then connected with a path to a light source. If the randomly selected pixel does have its quota of lens subpath mutations already, the other pixels are examined in a pseudo-random ordering until one is found with remaining samples.

By ensuring a minimum set of well-distributed samples that are always taken, overall image quality improves and we do a better job of (for example) anti-aliasing geometric edges than we would to otherwise. Remember that this process just sets a minimum number of samples that are taken per pixel—if the number of samples allocated to ensuring pixel stratification is 10% of the total number of samples (for example), then most of our samples will still be taken in regions with high contribution to the final image.

9.4.3 Direct Lighting

Veach notes that MLT (as described so far) often doesn't do as well with direct lighting as standard methods, such as those described in Chapter 3 of these notes, or in Shirley et al's TOG paper [70]. The root of the problem is that the Metropolis samples over the light sources aren't as well stratified as they can be with standard methods.

A relatively straightforward solution can be applied: when a lens subpath is generated (recall that lens subpaths are paths from the eye that follow zero or more specular bounces before hitting a diffuse surface), standard direct lighting techniques are used at the diffuse surface to compute a contribution to the image for the lens subpath. Then, whenever a MLT mutation is proposed that includes direct lighting, it is immediately rejected since direct lighting was already included in the solution.

Veach notes, however, that this optimization may not always be more effective. For example, if the direct lighting cannot be sampled efficiently by standard techniques (e.g. due to complex visibility, most of the light sources being completely occluded, etc.), then MLT would probably be more effective.

9.4.4 Participating Media

Pauly et al have described an extension of MLT to the case of participating media [62]. The state space and path measure are extended to include points in the volume in addition to points on the surfaces. Each path vertex may be on a surface or at a point in the scattering volume. The algorithm proceeds largely the same way as standard MLT, but places some path vertices on scene surfaces and others at points in the volume. As such, it robustly samples the space of all contributing transport paths through the medium.

They also describe a new type of mutation, tailored toward sampling scattering in participating media—the *propagation perturbation*. This perturbation randomly offsets a path vertex along one of the two incident edges (see Figure 9.16). Like other perturbations, this mutation helps concentrate work in important regions of the path space.

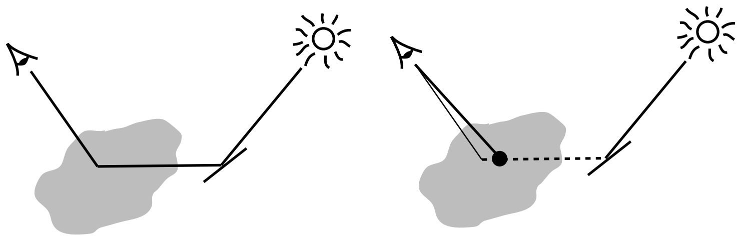


Figure 9.16: For path vertices that are in the scattering volume, rather than at a surface, the scattering propagation perturbation moves a vertex (shown here as a black circle) a random distance along the two path edges that are incident the vertex. Here we have chosen the dashed edge. If the connecting ray to the eye is occluded, the mutation is immediately rejected; otherwise the usual acceptance probability is computed.

Acknowledgements

Eric Veach was kind enough to read these notes through a series of drafts and offer extensive comments and many helpful suggestions for clearer presentation.

Chapter 10

Biased Techniques
By Henrik Wann Jensen

In the previous chapters we have seen examples of several *unbiased* Monte Carlo ray tracing (MCRT) techniques. These techniques use pure Monte Carlo sampling to compute the various estimates of radiance, irradiance etc. The only problem with pure MCRT is variance — seen as noise in the rendered images. The only way to eliminate this noise (and still have an unbiased algorithm) is to sample more efficiently and/or to use more samples.

In this chapter we will discuss several approaches for removing noise by introducing bias. We will discuss techniques that uses interpolation of irradiance to exploit the smoothness of the irradiance field. This approach makes it possible to use more samples at selected locations in the model. We will also discuss photon mapping, which stores information about the flux in a scene and performs a local evaluation of the statistics of the stored flux in order to speedup the simulation of global illumination.

10.1 Biased vs. Unbiased

An *unbiased* Monte Carlo technique does not have any systematic error. It can be stopped after any number of samples and the expected value of the estimator will be the correct value. This does not mean that all biased methods give the wrong result. A method can converge to the correct result as more samples are used and still be biased, such methods are *consistent*.

In chapter 2 it was shown how the integral of a function $g(x)$ can be expressed as the expected value of an estimator Ψ where Ψ is:

$$\Psi = \frac{1}{N} \sum_{i=1}^{N} \frac{g(x)}{p(x)} \ . \tag{10.1}$$

where $p(x)$ is a p.d.f. distributed according to x such that $p(x) > 0$ when $g(x) > 0$. The expected value of Ψ is the value of the integral:

$$E\{\Psi\} = I = \int_{x \in S} g(x)\, d\mu \ . \tag{10.2}$$

Since the expected value of the Ψ is our integral Ψ is an unbiased estimate of I.

In contrast a biased estimator Ψ^* have some other source of error such that:

$$E\{\Psi^*\} = I + \epsilon \ , \tag{10.3}$$

where ϵ is some error term. For a consistent estimator the error term will diminish as the number of samples is increased:

$$\lim_{N \to \infty} \epsilon = 0 \ . \tag{10.4}$$

Why should we be interested in anything, but unbiased methods? To answer this problem recall the slow convergence of pure Monte Carlo methods. To render images without noise it can be necessary to use a very high number of samples. In addition the eye is very sensitive to the high frequency noise that is typical with unbiased MCRT methods.

Going into the domain of biased methods we give up the ability to classify the error on the estimate by looking only at the variance. However the variance only gives us a probability the error is within a certain range and as such it is not a precise way of controlling the error. In addition the requirements for unbiased algorithms are quite restrictive; we cannot easily use adaptive sampling or stop the sampling once the estimate looks good enough — such simple decisions lead to biased methods [42].

With biased methods other sources of error are introduced to reduce the variance. This can lead to artifacts if the error is uncontrollable, so naturally we want a consistent method that will converge to the correct result as we use more samples. It is important to have the right balance between bias and variance. We want to eliminate the noise, but not introduce other strange artifacts. As we will see in the following the most common way of reducing noise is to blur the estimates; the eye

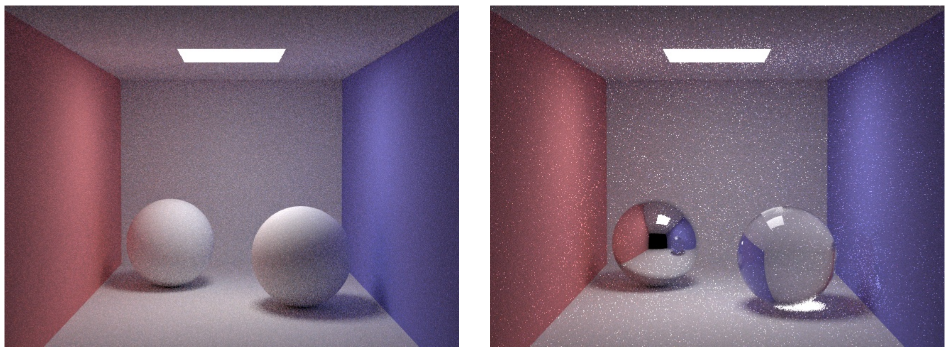


Figure 10.1: A path traced box scene with 10 paths/pixel. On the left the box has two diffuse spheres, and on the right box box has a mirror and a glass sphere. Note how the simple change of the sphere material causes a significant increase in the noise in the right image.

is fairly insensitive to slowly changing illumination. The trick is to avoid blurring edges and sharp features in the image and to have control over the amount of blur.

Consider the simple box scene in Figure 10.1. The figure contains two images of the box scene: one in which the box contains two diffuse spheres, and one in which the box contains a glass and a mirror sphere. Both images have been rendered with 10 paths per pixel. Even though the illumination of the walls is almost the same in the two images the introduction of the specular spheres is enough to make the path tracing image very noisy. In the following sections we will look at a number of techniques for eliminating this noise without using more samples.

10.2 Filtering Techniques

An obvious idea for reducing the noise in a rendered image is to postprocess the image and try to filter out the high frequency noise. This can be done to some degree with a low pass filter or with a median filter [23, 47]. The problem with these filters is that they remove other features than just the noise. Low pass filtering in particular will blur sharp edges of objects shadows etc. and in general low-pass filtered images look too blurry. Median filtering is much better at removing noise, but it still introduces artifacts along edges and other places since it cannot distinguish between noisy pixels and features of the illumination (such as small highlights). In addition median filtering is not *energy preserving*. By simply removing outliers in the image plane it is very likely that we take away energy or add energy to the

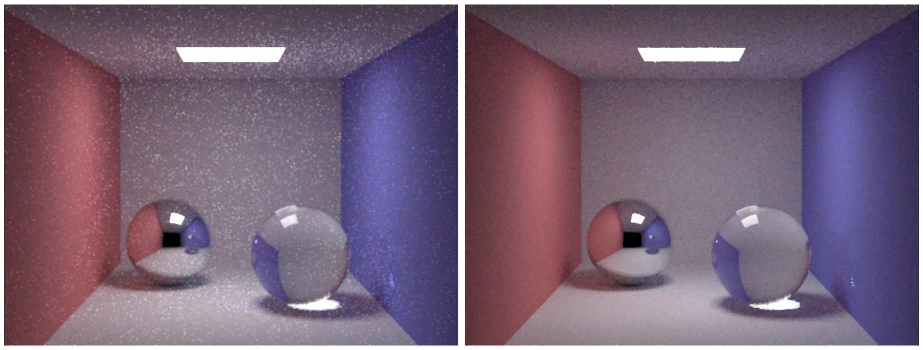


Figure 10.2: Filtered versions of the image of the box with specular spheres. On the left is the result of a 3x3 low-pass filter, and on the right the result of a 3x3 median filter.

rendered image. The effect of low-pass and median filtering on the box scene is shown in Figure 10.2

Several other more sophisticated filtering techniques have been used.

Jensen and Christensen [34] applied a median filter to the indirect illumination on diffuse surfaces based on the assumption that most of the noise in path tracing is found in this irradiance estimate. By filtering only this estimate before using it they removed a large fraction of the noise without blurring the edges, and features such as highlights and noisy textures. The problem with this approach is that it softens the indirect illumination and therefore blurs features due to indirect lighting. Also the technique is not energy-preserving.

Rushmeier and Ward [65] used a energy-preserving non-linear filter to remove noisy pixels by distributing the extra energy in the pixel over several neighboring pixels.

McCool [50] used an anisotropic diffusion filter which also preserves energy and allows for additional input about edges and textures in order to preserve these features.

Suykens and Willems [79] used information gathered during rendering (they used bidirectional path tracing) about the probabilities of various events. This enabled them to predict the occurrence of spikes in the illumination. By distributing the power of the spikes over a larger region in the image plane their method removes noise and preserves energy. The nice thing about this approach is that it allows for progressive rendering with initially blurry results that converges to the correct result as more samples are used.

10.3 Adaptive Sampling

Another way of eliminating noise in Monte Carlo ray traced images is to use adaptive sampling. Here the idea is to use more samples for the problematic (i.e. noisy) pixels.

One techniques for doing this is to compute the variance of the estimate based on the samples used for the pixel [48, 63]. Instead of using a fixed number of samples per pixel each pixel is sampled until the variance is below a given threshold. An estimate, s^2, of the variance for a given set of samples can be found using standard techniques from statistics:

$$s^2 = \frac{1}{N-1}\left[\frac{1}{N}\sum_{i=1}^{N} L_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} L_i\right)^2\right] \tag{10.5}$$

This estimate is based on the assumption that the samples L_i are distributed according to a normal distribution. This assumption is often reasonable, but it fails when for example the distribution within a pixel is bimodal (this happens when a light source edge passing through a pixel).

Using the variance as a stopping criteria can be effective in reducing noise, but it introduces bias as shown by Kirk and Arvo [42]. They suggested using a pilot sample to estimate the number of samples necessary. To eliminate bias the pilot sample must be thrown away.

The amount of bias introduced by adaptive sampling is, however, usually very small as shown by Tamstorf and Jensen [80]. They used bootstrapping to estimate the bias due to adaptive sampling and found that it is insignificant for most pixels in the rendered image (a notable exception is the edges of light sources).

10.4 Irradiance Caching

Irradiance caching is a technique that exploits the fact that the irradiance field often is smooth [95]. Instead of just filtering the estimate the idea is to cache and interpolate the irradiance estimates. This is done by examining the samples of the estimate more carefully to, loosely speaking, compute the expected smoothness of the irradiance around a given sample location. If the irradiance is determined to be sufficiently smooth then the estimate is re-used for this region.

To understand in more detail how this works let us first consider the evaluation of the irradiance, E, at a given location, x, using Monte Carlo ray tracing.

$$E(x) = \frac{\pi}{MN} \sum_{j=1}^{M} \sum_{i=1}^{N} L_{i,j}(\theta_j, \phi_i) \ , \tag{10.6}$$

where

$$\theta_j = \sin^{-1}\left(\sqrt{\frac{j - \xi_1}{M}}\right) \quad \text{and} \quad \phi_i = 2\pi \frac{i - \xi_2}{N} \ . \tag{10.7}$$

Here (θ_j, ϕ_i) specify a direction on the hemisphere above x in spherical coordinates. $\xi_1 \in [0,1]$ and $\xi_2 \in [0,1]$ are uniformly distributed random numbers, and M and N specify the subdivision of the hemisphere. $L_{i,j}(\theta_j, \phi_i)$ is evaluated by tracing a ray in the (θ_j, ϕ_i) direction. Note that the formula uses stratification of the hemisphere to obtain a better estimate than pure random sampling.

To estimate the smoothness of the local irradiance on the surface around the sample location Ward et al. [95] looked at the distances to the surfaces intersected by the rays as well as the local changes in the surface normal. This resulted in an estimate of the local relative change, ϵ_i, in irradiance as the surface location is changed away from sample i:

$$\epsilon_i(x, \vec{n}) = \frac{||x_i - x||}{R_0} + \sqrt{1 - 1\vec{n} \cdot \vec{n}_i} \ . \tag{10.8}$$

Here x_i is the original sample location and x is the new sample location (for which we want to compute the change), R_0 is the harmonic distance to the intersected surfaces, $\vec{n}_i$ is the sample normal, and $\vec{n}$ is the new normal.

Given this estimate of the local variation in irradiance Ward et al. developed a caching method where previously stored samples re-used whenever possible. All samples are stored in an octree-tree — this structure makes it possible to quickly locate previous samples. When a new sample is requested the octree is queried first for previous samples near the new location. For these nearby samples the change in irradiance, ϵ, is computed. If samples with a sufficiently low ϵ is found then these samples are blended using weights inversely proportional to ϵ:

$$E(x, \vec{n}) \approx \frac{\sum\limits_{i, w_i > 1/a} w_i(x, \vec{n}) E_i(x_i)}{\sum\limits_{i, w_i > 1/a} w_i(x, \vec{n})} \ . \tag{10.9}$$

Here $w_i = \frac{1}{\epsilon_i}$, a is the desired accuracy and E_i is the irradiance of sample i.

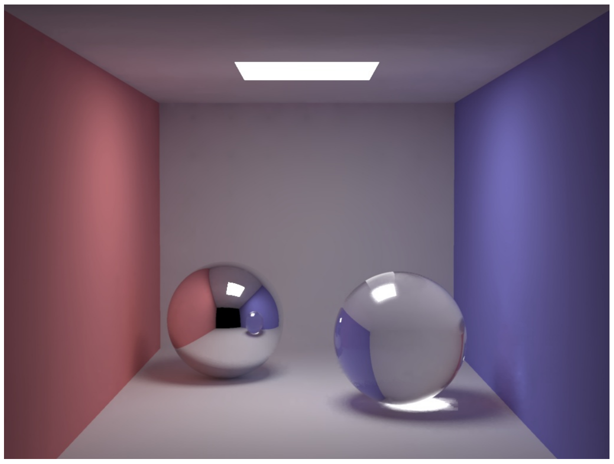


Figure 10.3: The box scene rendered using irradiance caching.

If no previously computed sample has a sufficiently high weight then a new sample is computed.

To further improve this estimate Ward and Heckbert added estimates of the gradients of the irradiance [94]. Their approach looks not only at the distances to the nearest surfaces, but also in the relative change of the incoming radiance from different directions in order to more accurately estimate the gradient due to a change in position as well as orientation. The great thing about this approach is that it does not require any further samples, but simply uses a more sophisticated analysis of the samples in the irradiance estimate.

With a few minor modifications this interpolation scheme works quite well as can be seen in Figure 10.3. The areas where it fails are in the case where the overall smoothness assumption for the irradiance field is no longer true. This is particularly the case for caustics (e.g. the light focused through the glass sphere; the caustic on the wall is missed completely).

Another issue with irradiance caching is that the method can be quite costly in scenes where multiple diffuse light reflections are important, since the costly irradiance estimates are computed recursively for each sample ray. The recursive evaluation also results in a global error (a global bias). Each light bounce has some approximation and multiple light bounces distributes this error all over the scene. This global aspect can make the error hard to control.

10.5 Photon Mapping

Photon mapping [32] is a method that uses biasing to reduce variance in many places. This is in part done by aggressively storing and re-using information whenever possible. This section contains a short overview of the photon mapping technique (for all the details consult [33]).

From a light transport point of view photon mapping exploits that:

- The irradiance field is mostly smooth, but has important focusing effects such as caustics

- There are two important sources of light paths in a scene: the light sources and the observer

In addition photon mapping uses a point sampling data-structure that is independent of the scene geometry, and it therefore works with complex geometry as well as non-Lambertian reflection models.

Photon mapping is a two-pass method in which the first pass is building the *photon map*. This is done using *photon tracing* in which photons are emitted from the light sources and traced through the scene. When a photon intersects a diffuse surface it is stored in the photon map. The second pass is rendering in which the photon map is a static structure with information about the illumination in the model. The renderer computes various statistics from the photon map to make the rendering faster.

10.5.1 Pass 1: Photon Tracing

The first step is building the photon map. This is done by emitting photons from the light sources and tracing them through the scene using photon tracing. The photons are emitted according to the power distribution of the light source. As an example, a diffuse point light emits photons with equal probability in all directions. When a photon intersects a surface it is stored in the photon map (if the surface material has a diffuse component). In addition the photon is either scattered or absorbed based on the albedo of the material. For this purpose Russian roulette [4] is used to decide if a photon path is terminated (i.e. the photon is absorbed), or if the photon is scattered (reflected or transmitted). This is shown in Figure 10.4.

The photon tracing algorithm is unbiased. No approximations or sources of systematic error is introduced in the photon tracing step. In contrast many other

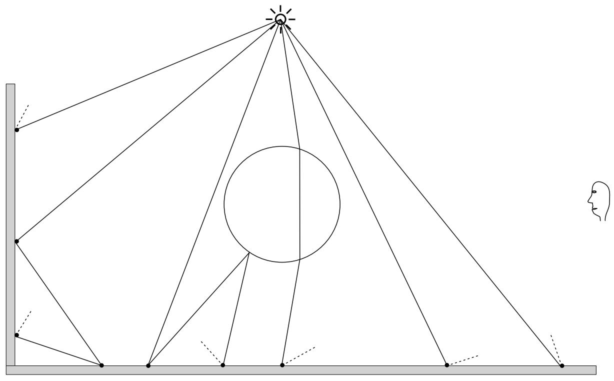


Figure 10.4: The photon map is build using photon tracing. From [33].

light propagation algorithms, such as progressive refinement radiosity [11], introduces a systematic error at each light bounce (due to the approximate representation of the illumination).

For efficiency reasons several photon maps are constructed in the first pass. A *caustics photon map* that stores only the photons that correspond to a caustic light path, a *global photon map* that stores all photon hits at diffuse surfaces (including the caustics), and a *volume photon map* that stores multiple scattered photons in participating media. In the following we will ignore the case of participating media (see [35, 33] for details).

10.5.2 The Radiance Estimate

The photon map represents incoming flux in the scene. For rendering purposes we want radiance. Using the expression for reflected radiance we find that:

$$L_r(x,\vec{\omega}) = \int_{\Omega} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}')(\vec{n}_x \cdot \vec{\omega}')\, d\vec{\omega}' \,, \tag{10.10}$$

where L_r is the reflected radiance at x in direction $\vec{\omega}$. Ω_x is the hemisphere of incoming directions, f_r is the BRDF and L_i is the incoming radiance. To use the

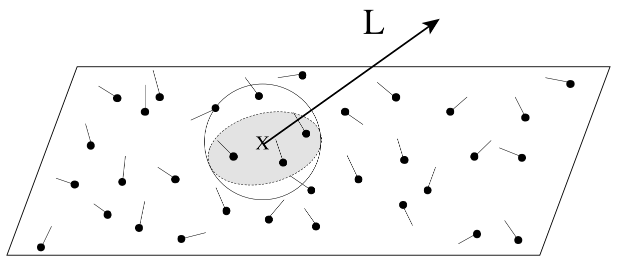


Figure 10.5: The radiance estimate is computed from the nearest photons in the photon map. From [33].

information in the photon map we can rewrite this integral as follows:

$$\begin{aligned} L_r(x,\vec{\omega}) &= \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d^2\Phi_i(x,\vec{\omega}')}{(\vec{n}_x\cdot\vec{\omega}')\, d\vec{\omega}_i'\, dA_i} (\vec{n}_x\cdot\vec{\omega}')\, d\vec{\omega}_i' \\ &= \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d^2\Phi_i(x,\vec{\omega}')}{dA_i}\,. \end{aligned} \tag{10.11}$$

Here we have used the relationship between radiance and flux to rewrite the incoming radiance as incoming flux instead. By using the nearest n photons around x from the photon map to estimate the incoming flux, we get:

$$L_r(x,\vec{\omega}) \approx \sum_{p=1}^{n} f_r(x,\vec{\omega}_p,\vec{\omega}) \frac{\Delta\Phi_p(x,\vec{\omega}_p)}{\Delta A}\,. \tag{10.12}$$

This procedure can be seen as expanding a sphere around x until it contains enough photons. The last unknown piece of information is ΔA. This is the area covered by the photons, and it is used to computed the photon density (the flux density). A simple approximation for ΔA is the projected area of the sphere used to locate the photons. The radius of this sphere is r (where r is the distance to the n'th nearest photon), and we get $\Delta A = \pi r^2$. This is equivalent to a nearest neighbor density estimate [73]. The radiance estimate is illustrated in Figure 10.5.

The radiance estimate is biased. There are two approximations in the estimate. It assumes that the nearest photons represents the illumination at x, and it uses a nearest neighbor density estimate. Both of these approximations can introduce artifacts in the rendered image. In particular the nearest neighbor density estimate is often somewhat blurry.

However, the radiance estimate is also consistent. As more photons are used in the photon map as well as the estimate it will converge to the correct value.

A useful property of the radiance estimate is that the bias is purely local (the error is a function of the local geometry and the local photon density).

10.5.3 Pass 2: Rendering

For rendering the radiance through each pixel is computed by averaging the result of several sample rays. The radiance for each ray is computed using distribution ray tracing. This ray tracer is using the photon map both to guide the sampling (importance sampling) as well as limit the recursion.

There are several strategies by which the photon map can be used for rendering. One can visualize the radiance estimate directly at every diffuse surface intersected by a ray. This approach will work (a very similar strategy is used by [69]), but it requires a large number of photons in both the photon map as well as the radiance estimate. To reduce the number of photons the two-pass photon mapping approach uses a mix of several techniques to compute the various components of the reflected radiance at a given surface location.

We distinguish between specular and diffuse reflection. Here specular means perfect specular or highly glossy, and diffuse reflection is the remaining part of the reflection (not only Lambertian).

For all specular surface components the two-pass method uses recursive ray tracing to evaluate the incoming radiance from the reflected direction. Ray tracing is pretty efficient at handling specular and highly glossy reflections.

Two different techniques are used for the diffuse surface component. The first diffuse surface seen either directly through a pixel or via a few specular reflections is evaluated accurately using Monte Carlo ray tracing. The direct illumination is computed using standard ray tracing techniques and similarly the irradiance is evaluated using Monte Carlo ray tracing or gathering (this sampling is improved by using the information in the photon map to importance sample in the directions where the local photons originated). Whenever a sample ray from the gathering step reaches another diffuse surface the estimate from the global photon map is used). The use of a gathering step means that the radiance estimate from the global photon map can be fairly coarse without affecting the quality of the final rendered image. The final component of the two-pass method is caustics, to reduce noise in the gathering step the caustics component is extracted and caustics are instead

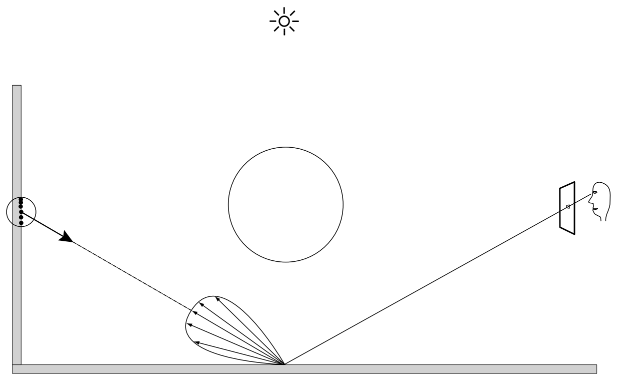


Figure 10.6: The rendering step uses a gathering step to compute the first diffuse bounce more accurately. From [33].

rendered by directly visualizing a caustics photon map — naturally this photon map should have a higher density than the global photon map used in the gathering step. Fortunately, most noticeable caustics are often caused by focusing of light, so this happens automatically. The gathering approach is illustrated in Figure 10.6

To make the gathering step faster it pays to use the irradiance caching method for Lambertian surfaces. Photon mapping works very well with irradiance caching since photon mapping handles the caustics which irradiance caching cannot handle very well. In addition photon mapping does not need the expensive recursive evaluation of irradiance values.

Figure 10.7 shows the rendering of the box scene using photon mapping. Here the global photon map has 200,000 photons and the caustics photon map has 50,000 photons. Notice the smooth overall appearance as well as the caustic on the right wall due to light reflected of the mirror sphere and focused through the glass sphere.

The bias in the photon mapping method is difficult to quantify. It is a mix of a local bias (the radiance estimate) and global bias (irradiance caching), and it is a function of the number of photons in the photon map, the number of photons in the radiance estimate and the number of sample rays. In general the bias from the photon map tends to appear as blurry illumination. Features gets washed out if too few photons are used. However, since the photon map is not visualized directly in the two-pass method it is somewhat harder to predict what the potential errors of using too few photons may be.

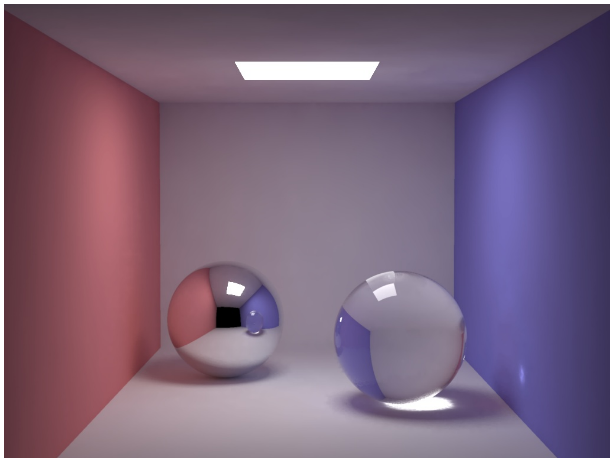


Figure 10.7: The box scene rendered using photon mapping.

Bibliography

[1] James Arvo. *Analytic Methods for Simulated Light Transport*. PhD thesis, Yale University, December 1995.

[2] James Arvo. Applications of irradiance tensors to the simulation of non-Lambertian phenomena. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 335–342, August 1995.

[3] James Arvo. Stratified sampling of spherical triangles. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 437–438, August 1995.

[4] James Arvo and David B. Kirk. Particle transport and image synthesis. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 63–66, August 1990.

[5] J. Beck and W. W. L. Chen. *Irregularities of Distribution*. Cambridge University Press, New York, 1987.

[6] Marcel Berger. *Geometry*, volume II. Springer-Verlag, New York, 1987. Translated by M. Cole and S. Levy.

[7] Kenneth Chiu, Peter Shirley, and Changyaw Wang. Multi-jittered sampling. In Paul Heckbert, editor, *Graphics Gems IV*, pages 370–374. Academic Press, Boston, 1994.

[8] Kenneth Chiu, Peter Shirley, and Changyaw Wang. Multi-jittered sampling. In Paul Heckbert, editor, *Graphics Gems IV*, pages 370–374. Academic Press, Boston, 1994.

[9] K.-L. Chung. An estimate concerning the Kolmogoroff limit distribution. *Transactions of the American Mathematical Society*, 67:36–50, 1949.

[10] William G. Cochran. *Sampling Techniques (3rd Ed)*. John Wiley & Sons, 1977.

[11] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 75–84, August 1988.

[12] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, San Diego, CA, 1993. Excellent book on radiosity algorithms.

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

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

[15] R. Cranley and T.N.L. Patterson. Randomization of number theoretic methods for multiple integration. *SIAM Journal of Numerical Analysis*, 13:904–914, 1976.

[16] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration (2nd Ed.)*. Academic Press, San Diego, 1984.

[17] Luc Devroye. *Non-uniform Random Variate Generation*. Springer, 1986.

[18] Kai-Tai Fang and Yuan Wang. *Number Theoretic Methods in Statistics*. Chapman and Hall, London, 1994.

[19] Henri Faure. Discrépance de suites associées à un système de numération (en dimension s). *Acta Arithmetica*, 41:337–351, 1982.

[20] Henri Faure. Good permutations for extreme discrepancy. *Journal of Number Theory*, 42:47–56, 1992.

[21] G. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer-Verlag, 1995.

[22] George S. Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer Verlag, New York, NY, 1996.

[23] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing (2nd Ed.)*. Addison-Wesley, Reading, MA, 1987.

[24] S. Haber. A modified Monte Carlo quadrature. *Mathematics of Computation*, 20:361–368, 1966.

[25] J.H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

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

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

[28] F. J. Hickernell, H. S. Hong, P. L'Ecuyer, and C. Lemieux. Extensible lattice sequences for quasi-Monte Carlo quadrature. *SIAM Journal on Scientific Computing*, 22(3):1117–1138, 2000.

[29] E. Hlawka. Funktionen von beschränkter Variation in der Theorie der Gleichverteilung. *Annali di Matematica Pura ed Applicata*, 54:325–333, 1961.

[30] H. S. Hong and F. J. Hickernell. Implementing scrambled digital sequences. *AMS Transactions on Mathematical Software*, 2003. To appear.

[31] L.K. Hua and Y. Wang. *Applications of number theory to numerical analysis*. Springer, Berlin, 1981.

[32] Henrik Wann Jensen. Global illumination using photon maps. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 21–30, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[33] Henrik Wann Jensen. *Realistic Image Synthesis using Photon Mapping*. AK Peters, 2001.

[34] Henrik Wann Jensen and Niels J. Christensen. Optimizing path tracing using noise reduction filters. In *Winter School of Computer Graphics 1995*, February 1995. held at University of West Bohemia, Plzen, Czech Republic, 14-18 February 1995.

[35] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 311–320. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

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

[37] M. H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*, volume I, *Basics*. John Wiley & Sons, New York, 1986.

[38] Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo Methods: Volume I: Basics*. John Wiley & Sons, New York, 1986.

[39] Alexander Keller. A quasi-Monte Carlo algorithm for the global illumination problem in a radiosity setting. In Harald Niederreiter and Peter Jau-Shyong Shiue, editors, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 239–251, New York, 1995. Springer-Verlag.

[40] David Kirk and James Arvo. Unbiased sampling techniques for image synthesis. *Computer Graphics*, 25(4):153–156, July 1991.

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

[42] David B. Kirk and James Arvo. Unbiased sampling techniques for image synthesis. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 153–156, July 1991.

[43] N. M. Korobov. The approximate computation of multiple integrals. *Dokl. Akad. Nauk SSSR*, 124:1207–1210, 1959.

[44] L. Kuipers and H. Niederreiter. *Uniform Distribution of Sequences*. John Wiley and Son, New York, 1976.

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

[46] P. L'Ecuyer and C. Lemieux. A survey of randomized quasi-Monte Carlo methods. In M. Dror, P. L'Ecuyer, and F. Szidarovszki, editors, *Modeling Uncertainty: An Examination of Stochastic Theory, Methods, and Applications*, pages 419–474. Kluwer Academic Publishers, 2002.

[47] Mark E. Lee and Richard A. Redner. A note on the use of nonlinear filtering in computer graphics. *IEEE Computer Graphics and Applications*, 10(3):23–29, May 1990.

[48] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically optimized sampling for distributed ray tracing. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 61–67, July 1985.

[49] J. Matoušek. On the L^2–discrepancy for anchored boxes. *Journal of Complexity*, 14:527–556, 1998.

[50] Michael McCool. Anisotropic diffusion for monte carlo noise reduction. *ACM Transactions on Graphics*, pages 171–194, April 1999.

[51] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–45, 1979.

[52] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[53] Don P. Mitchell. Spectrally optimal sampling for distributed ray tracing. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 157–164, July 1991.

[54] H. Niederreiter and C. Xing. Low-discrepancy sequences and global function fields with many rational places. *Finite Fields and Their Applications*, 2:241–273, 1996.

[55] Harald Niederreiter. Point sets and sequences with small discrepancy. *Monatshefte fur mathematik*, 104:273–337, 1987.

[56] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. S.I.A.M., Philadelphia, PA, 1992.

[57] A. B. Owen. Randomly permuted (t, m, s)-nets and (t, s)-sequences. In Harald Niederreiter and Peter Jau-Shyong Shiue, editors, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 299–317, New York, 1995. Springer-Verlag.

[58] A. B. Owen. Monte Carlo variance of scrambled equidistribution quadrature. *SIAM Journal of Numerical Analysis*, 34(5):1884–1910, 1997.

[59] A. B. Owen. Latin supercube sampling for very high dimensional simulations. *ACM Transactions on Modeling and Computer Simulation*, 8(2):71–102, 1998.

[60] Art B. Owen. Scrambling Sobol' and Niederreiter-Xing points. *Journal of Complexity*, 14(4):466–489, December 1998.

[61] Art B. Owen. Monte Carlo quasi-Monte Carlo and randomized quasi-Monte Carlo. In H. Niederreiter and J. Spanier, editors, *Monte Carlo and quasi-Monte Carlo Methods 1998*, pages 86–97, 1999.

[62] Mark Pauly, Tomas Kollig, and Alexander Keller. Metropolis light transport for participating media. In *Rendering Techniques 2000 (Proceedings of the Eurographics Rendering Workshop)*, pages 11–22, New York, June 2000. Eurographics, Springer Wien.

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

[64] R. Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, New York, 1981.

[65] Holly E. Rushmeier and Gregory J. Ward. Energy preserving non–linear filters. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 131–138. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[66] Ch. Schlier. A practitioner's view on qmc integration. *Mathematics and Computers in Simulation*, 2002.

[67] P. Shirley. Discrepancy as a quality measure for sample distributions. In Werner Purgathofer, editor, *Eurographics '91*, pages 183–194. North-Holland, September 1991.

[68] Peter Shirley. A ray tracing method for illumination calculation in diffuse-specular scenes. In *Proceedings of Graphics Interface '90*, pages 205–212, May 1990.

[69] Peter Shirley, Bretton Wade, Philip Hubbard, David Zareski, Bruce Walter, and Donald P. Greenberg. Global illumination via density estimation. In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

[70] Peter Shirley, Chang Yaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, January 1996. ISSN 0730-0301.

[71] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte Carlo methods for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, January 1996.

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

[73] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.

[74] Ian H. Sloan and S. Joe. *Lattice Methods for Multiple Integration*. Oxford Science Publications, Oxford, 1994.

[75] I. M. Sobol'. The distribution of points in a cube and the accurate evaluation of integrals (in Russian). *Zh. Vychisl. Mat. i Mat. Phys.*, 7:784–802, 1967.

[76] J. Spanier. Quasi-Monte Carlo Methods for Particle Transport Problems. In Harald Niederreiter and Peter Jau-Shyong Shiue, editors, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 121–148, New York, 1995. Springer-Verlag.

[77] Jerome Spanier and Ely M. Gelbard. *Monte Carlo Principles and Neutron Transport Problems*. Addison-Wesley, Reading, Massachusetts, 1969.

[78] Michael Stein. Large sample properties of simulations using Latin hypercube sampling. *Technometrics*, 29(2):143–51, 1987.

[79] Frank Suykens and Yves Willems. Adaptive filtering for progressive monte carlo image rendering. 2000.

[80] Rasmus Tamstorf and Henrik Wann Jensen. Adaptive sampling and bias estimation in path @acing. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 285–296, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.

[81] Boxin Tang. Orthogonal array-based Latin hypercubes. *Journal of the American Statistical Association*, 88:1392–1397, 1993.

[82] D. M. Titterington, A. F. M. Smith, and U. E. Makov. *The Statistical Analysis of Finite Mixture Distributions*. John Wiley & Sons, New York, NY, 1985.

[83] Bruno Tuffin. On the use of low discrepancy sequences in Monte Carlo methods. Technical Report 1060, I.R.I.S.A., Rennes, France, 1996.

[84] Greg Turk. Generating random points in triangles. In Andrew S. Glassner, editor, *Graphics Gems*, pages 24–28. Academic Press, New York, 1990.

[85] J. G. van der Corput. Verteilungsfunktionen I. *Nederl. Akad. Wetensch. Proc.*, 38:813–821, 1935.

[86] J. G. van der Corput. Verteilungsfunktionen II. *Nederl. Akad. Wetensch. Proc.*, 38:1058–1066, 1935.

[87] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, December 1997.

[88] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *5th Annual Eurographics Workshop on Rendering*, pages 147–162, June 13–15 1994.

[89] Eric Veach and Leonidas Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *SIGGRAPH '95 Conference Proceedings*, pages 419–428. Addison-Wesley, August 1995.

[90] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 419–428, August 1995.

[91] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 65–76. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

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

[93] Greg Ward. Adaptive shadow testing for ray tracing. In *Proceedings of the Second Eurographics Workshop on Rendering (Barcelona, May 1991)*, 1991.

[94] Gregory J. Ward and Paul Heckbert. Irradiance gradients. *Third Eurographics Workshop on Rendering*, pages 85–98, May 1992.

[95] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 85–92, August 1988.

[96] H. Weyl. Über die gleichverteilung von zahlen mod. eins. *Mathematische Annalen*, 77:313–352, 1916.

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

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

[99] S. K. Zaremba. The mathematical basis of Monte Carlo and quasi-Monte Carlo methods. *SIAM Review*, 10(3):303–314, July 1968.