Metropolis Light Transport

Eric Veach *Leonidas J. Guibas*

Computer Science Department
Stanford University

Abstract

We present a new Monte Carlo method for solving the light transport problem, inspired by the Metropolis sampling method in computational physics. To render an image, we generate a sequence of light transport paths by randomly mutating a single current path (e.g. adding a new vertex to the path). Each mutation is accepted or rejected with a carefully chosen probability, to ensure that paths are sampled according to the contribution they make to the ideal image. We then estimate this image by sampling many paths, and recording their locations on the image plane.

Our algorithm is unbiased, handles general geometric and scattering models, uses little storage, and can be orders of magnitude more efficient than previous unbiased approaches. It performs especially well on problems that are usually considered difficult, e.g. those involving bright indirect light, small geometric holes, or glossy surfaces. Furthermore, it is competitive with previous unbiased algorithms even for relatively simple scenes.

The key advantage of the Metropolis approach is that the path space is explored locally, by favoring mutations that make small changes to the current path. This has several consequences. First, the average cost per sample is small (typically only one or two rays). Second, once an important path is found, the nearby paths are explored as well, thus amortizing the expense of finding such paths over many samples. Third, the mutation set is easily extended. By constructing mutations that preserve certain properties of the path (e.g. which light source is used) while changing others, we can exploit various kinds of coherence in the scene. It is often possible to handle difficult lighting problems efficiently by designing a specialized mutation in this way.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; I.3.3 [Computer Graphics]: Picture/Image Generation; G.1.9 [Numerical Analysis]: Integral Equations—Fredholm equations.

Keywords: global illumination, lighting simulation, radiative heat transfer, physically-based rendering, Monte Carlo integration, variance reduction, Metropolis-Hastings algorithm, Markov Chain Monte Carlo (MCMC) methods

E-mail: ericv@cs.stanford.edu, guibas@cs.stanford.edu

1 Introduction

There has been a great deal of work in graphics on solving the light transport problem efficiently. However, current methods are optimized for a fairly narrow class of input scenes. For example, many algorithms require a huge increase in computer resources when there is bright indirect lighting, or when most surfaces are non-diffuse reflectors. For light transport algorithms to be widely used, it is important to find techniques that are less fragile. Rendering algorithms must run within acceptable time bounds on real models, yielding images that are physically plausible and visually pleasing. They must support complex geometry, materials, and illumination — these are all essential components of real-life environments.

Monte Carlo methods are an attractive starting point in the search for such algorithms, because of their generality and simplicity. Especially appealing are unbiased algorithms, i.e. those that compute the correct answer on the average. For these algorithms, any error in the solution is guaranteed to show up as random variations among the samples (e.g., as image noise). This error can be estimated by simply computing the sample variance.

On the other hand, many methods used in graphics are biased. To make any claims about the correctness of the results of these algorithms, we must bound the amount of bias. In general this is very difficult to do; it cannot be estimated by simply drawing a few more samples. Biased algorithms may produce results that are not noisy, but are nevertheless incorrect. This error is often noticeable visually, in the form of discontinuities, excessive blurring, or objectionable surface shading.

In graphics, the first unbiased Monte Carlo light transport algorithm was proposed by Kajiya [10], building on earlier work by Cook et al. [4] and Whitted [26]. Since then many refinements have been suggested (e.g. see [1]). Often these improvements have been adapted from the neutron transport and radiative heat transfer literatures, which have a long history of solving similar problems [22].

However, it is surprisingly difficult to design light transport algorithms that are general, efficient, and artifact-free.[1] From a Monte Carlo viewpoint, such an algorithm must efficiently sample the transport paths from the light sources to the lens. The problem is that for some environments, most paths do not contribute significantly to the image, e.g. because they strike surfaces with low reflectivity, or go through solid objects. For example, imagine a brightly lit room next to a dark room containing the camera, with a door slightly ajar between them. Naive path tracing will be very inefficient, because it will have difficulty generating paths that

[1] In this regard, certain ray-tracing problems have been shown to be *undecidable*, i.e. they cannot be solved on a Turing machine [19]. We can expect that any light transport algorithm will perform very badly as the geometry and materials of the input scene approach a provably difficult configuration.

go through the doorway. Similar problems occur when there are glossy surfaces, caustics, strong indirect lighting, etc.

Several techniques have been proposed to sample these difficult paths more efficiently. One is *bidirectional path tracing*, developed independently by Lafortune and Willems [12, 13], and Veach and Guibas [24, 25]. These methods generate one subpath starting at a light source and another starting at the lens, then they consider all the paths obtained by joining every prefix of one subpath to every suffix of the other. This leads to a family of different importance sampling techniques for paths, which are then combined to minimize variance [25]. This can be an effective solution for certain kinds of indirect lighting problems.

Another idea is to build an approximate representation of the radiance in a scene, which is then used to modify the directional sampling of the basic path tracing algorithm. This can be done with a particle tracing prepass [9], or by adaptively recording the radiance information in a spatial subdivision [14]. Moderate variance reductions have been reported (50% to 70%), but there are several problems, including inadequate directional resolution to handle concentrated indirect lighting, and substantial space requirements. Similar ideas have been applied to particle tracing [17, 5].

We propose a new algorithm for importance sampling the space of paths, which we call *Metropolis light transport* (MLT). The algorithm samples paths according to the contribution they make to the ideal image, by means of a random walk through path space. In Section 2, we give a high-level overview of MLT, then we describe its components in detail. Section 3 summarizes the classical Metropolis sampling algorithm, as developed in computational physics. Section 4 describes the path integral formulation of light transport, upon which our methods are based. Section 5 shows how to combine these two ideas to yield an effective light transport algorithm. Results are presented in Section 6, followed by conclusions and suggested extensions in Section 7. To our knowledge, this is the first application of the Metropolis method to transport problems of any kind.

2 Overview of the MLT algorithm

To make an image, we sample the paths from the light sources to the lens. Each path $\bar{x}$ is a sequence $\mathbf{x}_0\mathbf{x}_1\ldots\mathbf{x}_k$ of points on the scene surfaces, where $k \geq 1$ is the length of the path (the number of edges). The numbering of the vertices along the path follows the direction of light flow.

We will show how to define a function f on paths, together with a measure μ, such that $\int_D f(\bar{x})\,d\mu(\bar{x})$ represents the power (flux) that flows from the light sources to the image plane along a set of paths D. We call f the *image contribution function*, since $f(\bar{x})$ is proportional to the contribution made to the image by light flowing along $\bar{x}$.

Our overall strategy is to sample paths with probability proportional to f, and record the distribution of paths over the image plane. To do this, we generate a sequence of paths $\bar{X}_0$, $\bar{X}_1$, …, $\bar{X}_N$, where each $\bar{X}_i$ is obtained by a random mutation to the path $\bar{X}_{i-1}$. The mutations can have almost any desired form, and typically involve adding, deleting, or replacing a small number of vertices on the current path.

However, each mutation has a chance of being rejected, depending on the relative contributions of the old and new paths. For example, if the new path passes through a wall, the mutation will be rejected (by setting $\bar{X}_i = \bar{X}_{i-1}$). The Metropolis framework gives a recipe for determining the acceptance probability for each mutation, such that in the limit the sampled paths $\bar{X}_i$ are distributed according to f (this is the *stationary distribution* of the random walk).

As each path is sampled, we update the current image (which is stored in memory as a two-dimensional array of pixel values). To do this, we find the image location (u, v) corresponding to each path sample $\bar{X}_i$, and update the values of those pixels whose filter support contains (u, v). All samples are weighted equally; the light and dark regions of the final image are caused by differences in the number of samples recorded there.

The MLT algorithm is summarized below. In the following sections, we will describe it in more detail.

$\bar{x} \leftarrow \textsc{InitialPath}()$
image $\leftarrow$ { array of zeros }
for $i \leftarrow 1$ **to** N
  $\bar{y} \leftarrow \textsc{Mutate}(\bar{x})$
  $a \leftarrow \textsc{AcceptProb}(\bar{y}|\bar{x})$
  if $\textsc{Random}() < a$
   then $\bar{x} \leftarrow \bar{y}$
  $\textsc{RecordSample}(\mathit{image}, \bar{x})$
return *image*

3 The Metropolis sampling algorithm

In 1953, Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller introduced an algorithm for handling difficult sampling problems in computational physics [15]. It was originally used to predict the material properties of liquids, but has since been applied to many areas of physics and chemistry.

The method works as follows (our discussion is based on [11]). We are given a state space Ω, and a non-negative function $f : \Omega \to \mathbb{R}^+$. We are also given some initial state $\bar{X}_0 \in \Omega$. The goal is to generate a random walk $\bar{X}_0$, $\bar{X}_1$, … such that $\bar{X}_i$ is eventually distributed proportionally to f, no matter which state $\bar{X}_0$ we start with. Unlike most sampling methods, the Metropolis algorithm does not require that f integrate to one.

Each sample $\bar{X}_i$ is obtained by making a random change to $\bar{X}_{i-1}$ (in our case, these are the path mutations). This type of random walk, where $\bar{X}_i$ depends only on $\bar{X}_{i-1}$, is called a *Markov chain*. We let $K(\bar{y} \mid \bar{x})$ denote the probability density of going to state $\bar{y}$, given that we are currently in state $\bar{x}$. This is called the *transition function*, and satisfies the condition $\int_\Omega K(\bar{y}\,|\,\bar{x})\,d\mu(\bar{y}) = 1$ for all $\bar{x} \in \Omega$.

The stationary distribution. Each $\bar{X}_i$ is a random variable with some distribution p_i, which is determined from p_{i-1} by

$$p_i(\bar{x}) = \int_\Omega K(\bar{x}\,|\,\bar{y})\; p_{i-1}(\bar{y})\; d\mu(\bar{y})\;. \tag{1}$$

With mild conditions on K (discussed further in Section 5.2), the p_i will converge to a unique distribution p, called the *stationary distribution*. Note that p does not depend on the initial state $\bar{X}_0$.

To give a simple example of this idea, consider a state space consisting of n^2 vertices arranged in an $n \times n$ grid. Each vertex is connected to its four neighbors by edges, where the edges "wrap" from left to right and top to bottom as necessary (i.e. with the topology of a torus). A transition consists of randomly moving from the current vertex $\bar{x}$ to one of the neighboring vertices $\bar{y}$ with a probability of 1/5 each, and otherwise staying at vertex $\bar{x}$.

Suppose that we start at an arbitrary vertex $\bar{X}_0 = \bar{x}^*$, so that $p_0(\bar{x}) = 1$ for $\bar{x} = \bar{x}^*$, and $p_0(\bar{x}) = 0$ otherwise. After one transition, $\bar{X}_1$ is distributed with equal probability among $\bar{x}^*$ and its four neighbors. Similarly, $\bar{X}_2$ is randomly distributed among 13 vertices (although not with equal probability). If this process is continued, eventually p_i converges to a fixed probability distribution p, which necessarily satisfies

$$p(\bar{x}) = \textstyle\sum_{\bar{y}} K(\bar{x} \,|\, \bar{y})\, p(\bar{y}) \;.$$

For this example, p is the uniform distribution $p(\bar{x}) = 1/n^2$.

Detailed balance. In a typical physical system, the transition function K is determined by the physical laws governing the system. Given some arbitrary initial state, the system then evolves towards equilibrium through transitions governed by K.

The Metropolis algorithm works in the opposite direction. The idea is to invent or construct a transition function K whose resulting stationary distribution will be proportional to the given f, and which will converge to f as quickly as possible. The technique is simple, and has an intuitive physical interpretation called *detailed balance*.

Given $\bar{X}_{i-1}$, we obtain $\bar{X}_i$ as follows. First, we choose a tentative sample $\bar{X}'_i$, which can be done in almost any way desired. This is represented by the *tentative transition function* T, where $T(\bar{y} \,|\, \bar{x})$ gives the probability density that $\bar{X}'_i = \bar{y}$ given that $\bar{X}_{i-1} = \bar{x}$.

The tentative sample is then either accepted or rejected, according to an acceptance probability $a(\bar{y} \,|\, \bar{x})$ which will be defined below. That is, we let

$$\bar{X}_i = \begin{cases} \bar{X}'_i & \text{with probability } a(\bar{X}'_i \,|\, \bar{X}_{i-1}) \;, \\ \bar{X}_{i-1} & \text{otherwise} \;. \end{cases} \tag{2}$$

To see how to set $a(\bar{y} \,|\, \bar{x})$, suppose that we have already reached equilibrium, i.e. p_{i-1} is proportional to f. We must define $K(\bar{y} \,|\, \bar{x})$ such that the equilibrium is maintained. To do this, consider the density of transitions between any two states $\bar{x}$ and $\bar{y}$. From $\bar{x}$ to $\bar{y}$, the transition density is proportional to $f(\bar{x})\; T(\bar{y} \,|\, \bar{x})\; a(\bar{y} \,|\, \bar{x})$, and a similar statement holds for the transition density from $\bar{y}$ to $\bar{x}$. To maintain equilibrium, it is sufficient that these densities be equal:

$$f(\bar{x})\; T(\bar{y} \,|\, \bar{x})\; a(\bar{y} \,|\, \bar{x}) = f(\bar{y})\; T(\bar{x} \,|\, \bar{y})\; a(\bar{x} \,|\, \bar{y}) \;, \tag{3}$$

a condition known as *detailed balance*. We can verify that if $p_{i-1} \propto f$ and condition (3) holds, then equilibrium is preserved:

$$\begin{aligned} p_i(\bar{x}) \;&=\; \left[1 - \textstyle\int_\Omega a(\bar{y} \,|\, \bar{x})\; T(\bar{y} \,|\, \bar{x})\; d\mu(\bar{y})\right]\; p_{i-1}(\bar{x}) \qquad (4) \\ &\qquad + \textstyle\int_\Omega a(\bar{x} \,|\, \bar{y})\; T(\bar{x} \,|\, \bar{y})\; p_{i-1}(\bar{y})\; d\mu(\bar{y}) \\ &=\; p_{i-1}(\bar{x}) \;. \end{aligned}$$

The acceptance probability. Recall that f is given, and T was chosen arbitrarily. Thus, equation (3) is a condition on the ratio $a(\bar{y} \,|\, \bar{x}) / a(\bar{x} \,|\, \bar{y})$. In order to reach equilibrium as quickly as possible, the best strategy is to make $a(\bar{y} \,|\, \bar{x})$ and $a(\bar{x} \,|\, \bar{y})$ as large as possible [18], which is achieved by letting

$$a(\bar{y} \,|\, \bar{x}) = \min\left\{1, \frac{f(\bar{y})\; T(\bar{x} \,|\, \bar{y})}{f(\bar{x})\; T(\bar{y} \,|\, \bar{x})}\right\} \;. \tag{5}$$

According to this rule, transitions in one direction are always accepted, while in the other direction they are sometimes rejected, such that the expected number of moves each way is the same.

Comparison with genetic algorithms. The Metropolis method differs from genetic algorithms [6] in several ways. First, they have different purposes: genetic algorithms solve optimization problems, while the Metropolis method solves sampling problems (there is no search for an optimum value). Genetic algorithms work with a population of individuals, while Metropolis stores only a single current state. Finally, genetic algorithms have much more freedom in choosing the allowable mutations, since they do not need to compute the conditional probability of their actions.

Beyer and Lange [2] have applied genetic algorithms to the problem of integrating radiance over a hemisphere. They start with a population of rays (actually directional samples), which are evolved to improve their distribution with respect to the incident radiance at a particular surface point. However, their methods do not seem to lead to a feasible light transport algorithm.

4 The path integral formulation of light transport

Often the light transport problem is written as an integral equation, where we must solve for the equilibrium radiance function L. However, it can also be written as a pure integration problem, over the domain of all transport paths. The MLT algorithm is based on this formulation.[2] We start by reviewing the light transport and measurement equations, and then show how to transform them into an integral over paths.

The light transport equation. We assume a geometric optics model where light is emitted, scattered, and absorbed only at surfaces, travels in straight lines between surfaces, and is perfectly incoherent. Under these conditions, the *light transport equation* is given by[3]

$$\begin{aligned} L(\mathbf{x}' \!\to\! \mathbf{x}'') &= L_{\mathrm{e}}(\mathbf{x}' \!\to\! \mathbf{x}'') \qquad (6) \\ &\quad + \int_{\mathcal{M}} L(\mathbf{x} \!\to\! \mathbf{x}')\; f_{\mathrm{s}}(\mathbf{x} \!\to\! \mathbf{x}' \!\to\! \mathbf{x}'')\; G(\mathbf{x} \!\leftrightarrow\! \mathbf{x}')\; dA(\mathbf{x}). \end{aligned}$$

Here $\mathcal{M}$ is the union of all scene surfaces, A is the area measure on $\mathcal{M}$, $L_{\mathrm{e}}(\mathbf{x}' \!\to\! \mathbf{x}'')$ is the emitted radiance leaving $\mathbf{x}'$ in the direction of $\mathbf{x}''$, L is the equilibrium radiance function, and f_{s} is the bidirectional scattering distribution function (BSDF). The notation $\mathbf{x} \to \mathbf{x}'$ symbolizes the direction of light flow between two points of $\mathcal{M}$, while $\mathbf{x} \leftrightarrow \mathbf{x}'$ denotes symmetry in the argument pair.[4] The function G represents the throughput of a differential beam between $dA(\mathbf{x})$ and $dA(\mathbf{x}')$, and is given by

$$G(\mathbf{x} \!\leftrightarrow\! \mathbf{x}') = V(\mathbf{x} \!\leftrightarrow\! \mathbf{x}')\; \frac{|\cos(\theta_{\mathrm{o}})\; \cos(\theta'_{\mathrm{i}})|}{\|\mathbf{x} - \mathbf{x}'\|^2} \;,$$

where θ_{o} and θ'_{i} represent the angles between the segment $\mathbf{x} \leftrightarrow \mathbf{x}'$ and the surface normals at $\mathbf{x}$ and $\mathbf{x}'$ respectively, while $V(\mathbf{x} \!\leftrightarrow\! \mathbf{x}') = 1$ if $\mathbf{x}$ and $\mathbf{x}'$ are mutually visible and is zero otherwise.

[2] Note that two different formulations of bidirectional path tracing have been proposed: one based on a measure over paths [24, 25], and the other based on the *global reflectance distribution function* (GRDF) [13]. However, only the path measure approach defines the notion of probabilities on paths, as required for combining multiple estimators [25] and the present work.

[3] Technically, the equations of this section deal with *spectral radiance*, and apply at each wavelength separately.

[4] There is redundancy in this representation; e.g. $L(\mathbf{x} \!\to\! \mathbf{x}') = L(\mathbf{x} \!\to\! \mathbf{x}'')$ whenever $\mathbf{x}'$ and $\mathbf{x}''$ lie in the same direction from $\mathbf{x}$.

The measurement equation. Light transport algorithms estimate a finite number of measurements of the equilibrium radiance L. We consider only algorithms that compute an image directly, so that the measurements consist of many pixel values $m_1, \ldots, m_M$, where M is the number of pixels in the image. Each measurement has the form

$$m_j = \int_{\mathcal{M}\times\mathcal{M}} W_{\mathrm{e}}^{(j)}(\mathbf{x}\to\mathbf{x}')\, L(\mathbf{x}\to\mathbf{x}')\, G(\mathbf{x}\leftrightarrow\mathbf{x}')\, dA(\mathbf{x})\, dA(\mathbf{x}'), \tag{7}$$

where $W_{\mathrm{e}}^{(j)}(\mathbf{x}\to\mathbf{x}')$ is a weight that indicates how much the light arriving at $\mathbf{x}'$ from the direction of $\mathbf{x}$ contributes to the value of the measurement.[5] For real sensors, $W_{\mathrm{e}}^{(j)}$ is called the *flux responsivity* (with units of $[\mathrm{W}^{-1}]$), but in graphics it is more often called an *importance function*.[6]

The path integral formulation. By recursively expanding the transport equation (6), we can write measurements in the form

$$\begin{aligned} m_j = &\int_{\mathcal{M}^2} L_{\mathrm{e}}(\mathbf{x}_0\to\mathbf{x}_1)\, G(\mathbf{x}_0\leftrightarrow\mathbf{x}_1)\, W_{\mathrm{e}}^{(j)}(\mathbf{x}_0\to\mathbf{x}_1)\, dA(\mathbf{x}_0)\, dA(\mathbf{x}_1) \\ &+ \int_{\mathcal{M}^3} L_{\mathrm{e}}(\mathbf{x}_0\to\mathbf{x}_1)\, G(\mathbf{x}_0\leftrightarrow\mathbf{x}_1)\, f_{\mathrm{s}}(\mathbf{x}_0\leftrightarrow\mathbf{x}_1\leftrightarrow\mathbf{x}_2) \\ &\qquad G(\mathbf{x}_1\leftrightarrow\mathbf{x}_2)\, W_{\mathrm{e}}^{(j)}(\mathbf{x}_1\to\mathbf{x}_2)\, dA(\mathbf{x}_0)\, dA(\mathbf{x}_1)\, dA(\mathbf{x}_2) \\ &+ \cdots . \end{aligned} \tag{8}$$

The goal is to write this expression in the form

$$m_j = \int_{\Omega} f_j(\bar{x})\, d\mu(\bar{x}) , \tag{9}$$

so that we can handle it as a pure integration problem.

To do this, let Ω_k be the set of all paths of the form $\bar{x} = \mathbf{x}_0\mathbf{x}_1\ldots\mathbf{x}_k$, where $k \geq 1$ and $\mathbf{x}_i \in \mathcal{M}$ for each i. We define a measure μ_k on the paths of each length k according to

$$d\mu_k(\mathbf{x}_0\ldots\mathbf{x}_k) = dA(\mathbf{x}_0)\cdots dA(\mathbf{x}_k) ,$$

i.e. μ_k is a product measure. Next, we let Ω be the union of all the Ω_k, and define a measure μ on Ω by[7]

$$\mu(D) = \sum_{k=1}^{\infty} \mu_k(D\cap\Omega_k) . \tag{10}$$

The integrand f_j is defined by extracting the appropriate term from the expansion (8) — see Figure 1. For example,

$$f_j(\mathbf{x}_0\mathbf{x}_1) = L_{\mathrm{e}}(\mathbf{x}_0\to\mathbf{x}_1)\; G(\mathbf{x}_0\leftrightarrow\mathbf{x}_1)\; W_{\mathrm{e}}^{(j)}(\mathbf{x}_0\to\mathbf{x}_1) .$$

We call f_j the *measurement contribution function*.

There is nothing tricky about this; we have just expanded and rearranged the transport equations. The most significant aspect is that we have removed the sum over different path lengths, and replaced it with a single integral over an abstract measure space of paths.

[5]The function $W_{\mathrm{e}}^{(j)}$ is zero almost everywhere. It is non-zero only if $\mathbf{x}'$ lies on the virtual camera lens, and the ray $\mathbf{x}\to\mathbf{x}'$ is mapped by the lens to the small region of the image plane corresponding to the filter support of pixel j.

[6]Further references and discussion may be found in [23].

[7]This measure on paths is similar to that of Spanier and Gelbard [22, p. 85]. However, in our case infinite-length paths are excluded. This makes it easy to verify that (10) is in fact a measure, directly from the axioms [7].

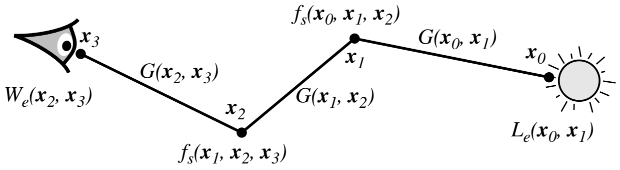


Figure 1: The measurement contribution function f_j is a product of many factors (shown for a path of length 3).

5 Metropolis light transport

To complete the MLT algorithm outlined in Section 2, there are several tasks. First, we must formulate the light transport problem so that it fits the Metropolis framework. Second, we must show how to avoid *start-up bias*, a problem which affects many Metropolis applications. Most importantly, we must design a suitable set of mutations on paths, such that the Metropolis method will work efficiently.

5.1 Reduction to the Metropolis framework

We show how the Metropolis method can be adapted to estimate all of the pixel values m_j simultaneously and without bias.

Observe that each integrand f_j has the form

$$f_j(\bar{x}) = w_j(\bar{x})\; f(\bar{x}) , \tag{11}$$

where w_j represents the filter function for pixel j, and f represents all the other factors of f_j (which are the same for all pixels). In physical terms, $\int_D f(\bar{x})\, d\mu(\bar{x})$ represents the radiant power received by the image area of the image plane along a set D of paths.[8] Note that w_j depends only on the last edge $\mathbf{x}_{k-1}\mathbf{x}_k$ of the path, which we call the *lens edge*.

An image can now be computed by sampling N paths $\bar{X}_i$ according to some distribution p, and using the identity

$$m_j = E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{w_j(\bar{X}_i)f(\bar{X}_i)}{p(\bar{X}_i)}\right] .$$

If we could let $p = (1/b)\, f$ (where b is the normalization constant $\int_{\Omega} f(\bar{x})\, d\mu(\bar{x})$), the estimate for each pixel would be

$$m_j = E\left[\frac{1}{N}\sum_{i=1}^{N} b\; w_j(\bar{X}_i)\right] .$$

This equation can be evaluated efficiently for all pixels at once, since each path contributes to only a few pixel values.

This idea requires the evaluation of b, and the ability to sample from a distribution proportional to f. Both of these are hard problems. For the second part, the Metropolis algorithm will help; however, the samples $\bar{X}_i$ will have the desired distribution only in the limit as $i\to\infty$. In typical Metropolis applications, this is handled by starting in some fixed initial state $\bar{X}_0$, and discarding the first k samples until the random walk has approximately converged to the equilibrium distribution. However, it is often difficult to know how large k should be. If it is too small, then the samples will be strongly influenced by the choice of $\bar{X}_0$, which will bias the results (this is called *start-up bias*).

[8]We assume that $f(\bar{x})=0$ for paths that do not contribute to any pixel value (so that we do not waste any samples there).

Eliminating start-up bias. We show how the MLT algorithm can be initialized to avoid start-up bias. The idea is to start the walk in a random initial state $\bar{X}_0$, which is sampled from some convenient path distribution p_0 (we use bidirectional path tracing for this purpose). To compensate for the fact that p_0 is not the desired distribution $(1/b)\, f$, the sample $\bar{X}_0$ is assigned a weight: $W_0 = f(\bar{X}_0)/p_0(\bar{X}_0)$. Thus after one sample, the estimate for pixel j is $W_0\, w_j(\bar{X}_0)$. All of these quantities are computable since $\bar{X}_0$ is known.

Additional samples $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_N$ are generated by mutating $\bar{X}_0$ according to the Metropolis algorithm (using f as the target density). Each of the $\bar{X}_i$ has a different distribution p_i, which only approaches $(1/b)\, f$ as $i \to \infty$. To avoid bias, however, it is sufficient to assign these samples the same weight $W_i = W_0$ as the original sample, where the following estimate is used for pixel j:

$$m_j = E\left[\frac{1}{N} \sum_{i=1}^{N} W_i\, w_j(\bar{X}_i)\right] . \qquad (12)$$

To show that this is unbiased, recall that the initial state was chosen randomly, and so we must average over all choices of $\bar{X}_0$ when computing expected values. Consider a group of starting paths obtained by sampling p_0 many times. If we had $p_0 = (1/b)\, f$ and $W_0 = b$, then obviously this starting group would be in equilibrium. For general p_0, the choice $W_0 = f/p_0$ leads to exactly the same distribution of weight among the starting paths, and so we should expect that these initial conditions are unbiased as well. (See Appendix A for a proof.)

This technique for removing start-up bias is not specific to light transport. However, it requires the existence of an alternative sampling method p_0, which for many Metropolis applications is not easy to obtain.

Initialization. In practice, initializing the MLT algorithm with a single sample does not work well. If we generate only one path $\bar{X}_0$ (e.g. using bidirectional path tracing), it is quite likely that $W_0 = 0$ (e.g. the path goes through a wall). Since all subsequent samples use the same weight $W_i = W_0$, this would lead to a completely black final image. Conversely, the initial weight W_0 on other runs may be much larger than expected. Although the algorithm is unbiased, this statement is only useful when applied to the average result over many runs. The obvious solution is to run n copies of the algorithm in parallel, and accumulate all the samples into one image.

The strategy we have implemented is to sample a moderately large number of paths $\bar{X}_0^{(1)}, \ldots, \bar{X}_0^{(n)}$, with corresponding weights $W_0^{(1)}, \ldots, W_0^{(n)}$. We then resample the $\bar{X}_0^{(i)}$ to obtain a much smaller number n' of equally-weighted paths (chosen with equal spacing in the cumulative weight distribution of the $\bar{X}_0^{(i)}$). These are used as independent seeds for the Metropolis phase of the algorithm.

The value of n is determined indirectly, by generating a fixed number of eye and light subpaths (e.g. 10 000 pairs), and considering all the ways to link the vertices of each pair. Note that it is not necessary to save all of these paths for the resampling step; they can be regenerated by restarting the random number generator with the same seed.

It is often reasonable to choose $n' = 1$ (a single Metropolis seed). In this case, the purpose of the first phase is to estimate the mean value of W_0, which determines the absolute image brightness.[9] If the image is desired only up to a constant scale factor, then n can be chosen to be very small. The main reasons for retaining more than one seed (i.e. $n' > 1$) are to implement convergence tests (see below) or lens subpath mutations (Section 5.3.3).

Effectively, we have separated the image computation into two subproblems. The initialization phase estimates the overall image brightness, while the Metropolis phase determines the relative pixel intensities across the image. The effort spent on each phase can be decided independently. In practice, however, the initialization phase is a negligible part of the total computation time (e.g., even 100 000 bidirectional samples typically constitute less than one sample per pixel).

Convergence tests. Another reason to run several copies of the algorithm in parallel is that it facilitates convergence testing. (We cannot apply the usual variance tests to samples generated by a single run of the Metropolis algorithm, since consecutive samples are highly correlated.)

To test for convergence, the Metropolis phase runs with n' independent seed paths, whose contributions to the image are recorded separately (in the form of n' separate images). This is done only for a small representative fraction of the pixels, since it would be too expensive to maintain many copies of a large image. The sample variance of these test pixels is then computed periodically, until the results are within prespecified bounds.[10]

Spectral sampling. Our discussion so far has been limited to monochrome images, but the modifications for color are straightforward.

We represent BSDF's and light sources as point-sampled spectra (although it would be easy to use some other representation). Given a path, we compute the energy delivered to the lens at each of the sampled wavelengths. The resulting spectrum is then converted to a tristimulus color value (we use RGB) before it is accumulated in the current image.

The image contribution function f is redefined to compute the luminance of the corresponding path spectrum. This implies that path samples will be distributed according to the luminance of the ideal image, and that the luminance of every filtered image sample will be the same (irrespective of its color). Effectively, each color component h is sampled with an estimator of the form h/p, where p is proportional to the luminance.

Since the human eye is substantially more sensitive to luminance differences than other color variations, this choice helps to minimize the apparent noise.[11]

5.2 Designing a mutation strategy

The main disadvantage of the Metropolis method is that consecutive samples are correlated, which leads to higher variance than we would get with independent samples. This can happen either because the proposed mutations to the path are very small, or because too many mutations are rejected.

This problem can be minimized by choosing a suitable set of path mutations. We consider some of the properties that these mutations should have, to minimize the error in the final image.

[9] More precisely, $E[W_0] = \int f = b$, which represents the total power falling on the image region of the film plane.

[10] Note that in all of our tests, the number of mutations for each image was specified manually, so that we would have explicit control over the computation time.

[11] Another way to handle color is to have a separate run for each wavelength. However, this is inefficient (we get less information from each path) and leads to unnecessary color noise.

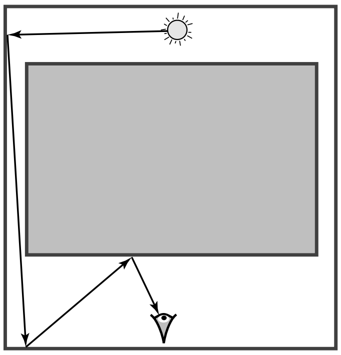


Figure 2: If only additions and deletions of a single vertex are allowed, then paths cannot mutate from one side of the barrier to the other.

High acceptance probability. If the acceptance probability $a(\bar{y}\,|\,\bar{x})$ is very small on the average, there will be long path sequences of the form $\bar{x}$, $\bar{x}$, ..., $\bar{x}$ due to rejections. This leads to many samples at the same point on the image plane, and appears as noise.

Large changes to the path. Even if the acceptance probability for most mutations is high, samples will still be highly correlated if the proposed path mutations are too small. It is important to propose mutations that make substantial changes to the current path, such as increasing the path length, or replacing a specular bounce with a diffuse one.

Ergodicity. If the allowable mutations are too restricted, it is possible for the random walk to get "stuck" in some subregion of the path space (i.e. one where the integral of f is less than b). To see how this can happen, consider Figure 2, and suppose that we only allow mutations that add or delete a single vertex. In this case, there is no way for the path to mutate to the other side of the barrier, and we will miss part of the path space.

Technically, we want to ensure that the random walk converges to an *ergodic* state. This means that no matter how $\bar{X}_0$ is chosen, it converges to the same stationary distribution p. To do this, it is sufficient to ensure that $T(\bar{y}\,|\,\bar{x}) > 0$ for every pair of states $\bar{x}$, $\bar{y}$ with $f(\bar{x}) > 0$ and $f(\bar{y}) > 0$. In our implementation, this is always true (see Section 5.3.1).

Changes to the image location. To minimize correlation between the sample locations on the image plane, it is desirable for mutations to change the lens edge $\mathbf{x}_{k-1}\mathbf{x}_k$. Mutations to other portions of the path do not provide information about the path distribution over the image plane, which is what we are most interested in.

Stratification. Another potential weakness of the Metropolis approach is the random distribution of samples across the image plane. This is commonly known as the "balls in bins" effect: if we randomly throw n balls into n bins, we cannot expect one ball per bin. (Many bins may be empty, while the fullest bin is likely to contain $\Theta(\log n)$ balls.) In an image, this unevenness in the distribution produces noise.

For some kinds of mutations, this effect is difficult to avoid. However, it is worthwhile to consider mutations for which some form of stratification is possible.

Low cost. It is also desirable that mutations be inexpensive. Generally, this is measured by the number of rays cast, since the other costs are relatively small.

5.3 Good mutation strategies

We now consider three specific mutation strategies, namely *bidirectional mutations*, *perturbations*, and *lens subpath mutations*. These strategies are designed to satisfy different subsets of the goals mentioned above; our implementation uses a mixture of all three (as we discuss in Section 5.3.4).

Note that the Metropolis framework allows us greater freedom than standard Monte Carlo algorithms in designing sampling strategies. This is because we only need to compute the *conditional* probability $T(\bar{y}\,|\,\bar{x})$ of each mutation: in other words, the mutation strategy is allowed to depend on the current path.

5.3.1 Bidirectional mutations

Bidirectional mutations are the foundation of the MLT algorithm. They are responsible for making large changes to the path, such as modifying its length. The basic idea is simple: we choose a subpath of the current path $\bar{x}$, and replace it with a different subpath. We divide this into several steps.

First, the subpath to delete is chosen. Given the current path $\bar{x} = \mathbf{x}_0 \ldots \mathbf{x}_k$, we assign a probability $p_{\mathrm{d}}[s,t]$ to the deletion of each subpath $\mathbf{x}_s \ldots \mathbf{x}_t$. The subpath endpoints are not included, i.e. $\mathbf{x}_s \ldots \mathbf{x}_t$ consists of $t - s$ edges and $t-s-1$ vertices, with indices satisfying $-1 \le s < t \le k+1$.

In our implementation, $p_{\mathrm{d}}[s,t]$ is a product two factors. The first factor $p_{\mathrm{d},1}$ depends only on the subpath length; its purpose is to favor the deletion of short subpaths. (These are less expensive to replace, and yield mutations that are more likely to be accepted, since they make a smaller change to the current path). The purpose of the second factor $p_{\mathrm{d},2}$ is to avoid mutations with low acceptance probabilities; it will be described in Section 5.4.

To determine the deleted subpath, the distribution $p_{\mathrm{d}}[s,t]$ is normalized and sampled. At this point, $\bar{x}$ has been split into two (possibly empty) pieces $\mathbf{x}_0 \ldots \mathbf{x}_s$ and $\mathbf{x}_t \ldots \mathbf{x}_k$.

To complete the mutation, we first choose the number of vertices s' and t' to be added to each side. We do this in two steps: first, we choose the new subpath length, $l_{\mathrm{a}} = s' + t' + 1$. It is desirable that the old and new subpath lengths be similar, since this will tend to increase the acceptance probability (i.e. it represents a smaller change to the path). Thus, we choose l_{a} according to a discrete distribution $p_{\mathrm{a},1}$ which assigns a high probability to keeping the total path length the same. Then, we choose specific values for s' and t' (subject to $s' + t' + 1 = l_{\mathrm{a}}$), according to another discrete distribution $p_{\mathrm{a},2}$ that assigns equal probability to each candidate value of s'. For convenience, we let $p_{\mathrm{a}}[s',t']$ denote the product of $p_{\mathrm{a},1}$ and $p_{\mathrm{a},2}$.

To sample the new vertices, we add them one at a time to the appropriate subpath. This involves first sampling a direction according to the BSDF at the current subpath endpoint (or a convenient approximation, if sampling from the exact BSDF is difficult), followed by casting a ray to find the first surface intersected. An initially empty subpath is handled by choosing a random point on a light source or the lens as appropriate.

Finally, we join the new subpaths together, by testing the visibility between their endpoints. If the path is obstructed, the mutation is immediately rejected. This also happens if any of the ray casting operations failed to intersect a surface.

Notice that there is a non-zero probability of throwing away the entire path, and generating a new one from scratch. This automatically ensures the ergodicity condition (Section 5.2), so that the algorithm can never get "stuck" in a small subregion of the path space.

Parameter values. The following values have provided reasonable results on our test cases. For the probability $p_{\mathrm{d},1}[l_\mathrm{d}]$ of deleting a subpath of length $l_\mathrm{d} = t-s$, we use $p_{\mathrm{d},1}[1] = 0.25$, $p_{\mathrm{d},1}[2] = 0.5$, and $p_{\mathrm{d},1}[l_\mathrm{d}] = 2^{-l}$ for $l_\mathrm{d} \geq 3$. For the probability $p_{\mathrm{a},1}[l_\mathrm{a}]$ of adding a subpath of length l_a, we use $p_{\mathrm{a},1}[l_\mathrm{d}] = 0.5$, $p_{\mathrm{a},1}[l_\mathrm{d}-1] = 0.15$, and $p_{\mathrm{a},1}[l_\mathrm{d}+1] = 0.15$, with the remaining probability assigned to the other allowable subpath lengths.

Evaluation of the acceptance probability. Observe that $a(\bar{y}\,|\,\bar{x})$ can be written as the ratio

$$a(\bar{y}\,|\,\bar{x}) = \frac{R(\bar{y}\,|\,\bar{x})}{R(\bar{x}\,|\,\bar{y})}\,, \qquad \text{where} \quad R(\bar{y}\,|\,\bar{x}) = \frac{f(\bar{y})}{T(\bar{y}\,|\,\bar{x})}\,. \quad (13)$$

The form of $R(\bar{y} \mid \bar{x})$ is very similar to the sample value $f(\bar{y})/p(\bar{y})$ that is computed by standard Monte Carlo algorithms; we have simply replaced an absolute probability $p(\bar{y})$ by a conditional probability $T(\bar{y}\,|\,\bar{x})$.

Specifically, $T(\bar{y}\,|\,\bar{x})$ is the product of the discrete probability $p_\mathrm{d}[s,t]$ for deleting the subpath $\mathbf{x}_s \ldots \mathbf{x}_t$, and the probability density for generating the $s'+t'$ new vertices of $\bar{y}$. To calculate the latter, we must take into account all $s'+t'+1$ ways that the new vertices can be split between subpaths generated from $\mathbf{x}_s$ and $\mathbf{x}_t$. (Although these vertices were generated by a particular choice of s', the probability $T(\bar{y}\,|\,\bar{x})$ must take into account all ways of going from state $\bar{x}$ to $\bar{y}$.)

Note that the unchanged portions of $\bar{x}$ do not contribute to the calculation of $T(\bar{y}\,|\,\bar{x})$. It is also convenient to ignore the factors of $f(\bar{x})$ and $f(\bar{y})$ that are shared between the paths, since this does not change the result.

An example. Let $\bar{x}$ be a path $\mathbf{x}_0\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$, and suppose that the random mutation step has deleted the edge $\mathbf{x}_1\mathbf{x}_2$. It is replaced by new vertex $\mathbf{z}_0$ by casting a ray from $\mathbf{x}_1$, so that the new path is $\bar{y} = \mathbf{x}_0\mathbf{x}_1\mathbf{z}_0\mathbf{x}_2\mathbf{x}_3$. (This corresponds to the random choices $s=1$, $t=2$, $s'=1$, $t'=0$.)

Let $p_\mathrm{s}(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{z}_0)$ be the probability density of sampling the direction from $\mathbf{x}_1$ to $\mathbf{z}_0$, measured with respect to projected solid angle.[12] Then the probability density of sampling $\mathbf{z}_0$ (measured with respect to surface area) is given by $p_\mathrm{s}(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{z}_0)\, G(\mathbf{x}_1 \leftrightarrow \mathbf{z}_0)$.

We now have all of the information necessary to compute $R(\bar{y}\,|\,\bar{x})$. From (8), the numerator $f(\bar{y})$ is

$$f_\mathrm{s}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{z}_0)\, G(\mathbf{x}_1,\mathbf{z}_0)\, f_\mathrm{s}(\mathbf{x}_1,\mathbf{z}_0,\mathbf{x}_2)\, G(\mathbf{z}_0,\mathbf{x}_2)\, f_\mathrm{s}(\mathbf{z}_0,\mathbf{x}_2,\mathbf{x}_3)\,,$$

where the factors shared between $R(\bar{y}\,|\,\bar{x})$ and $R(\bar{x}\,|\,\bar{y})$ have been omitted (and we have dropped the arrow notation for brevity). The denominator $T(\bar{y}\,|\,\bar{x})$ is

$$p_\mathrm{d}[1,2] \left[\begin{array}{l} p_\mathrm{a}[1,0]\, p_\mathrm{s}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{z}_0)\, G(\mathbf{x}_1,\mathbf{z}_0) \\ +\, p_\mathrm{a}[0,1]\, p_\mathrm{s}(\mathbf{x}_3,\mathbf{x}_2,\mathbf{z}_0)\, G(\mathbf{x}_2,\mathbf{z}_0) \end{array} \right].$$

In a similar way, we find that $R(\bar{x}\,|\,\bar{y})$ is given by

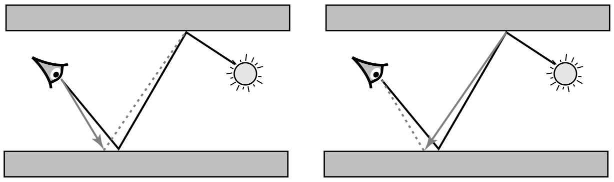
$$\{f_\mathrm{s}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)\, G(\mathbf{x}_1,\mathbf{x}_2)\, f_\mathrm{s}(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3)\} \;/\; \{p_\mathrm{d}[1,3]\; p_\mathrm{a}[0,0]\}\,,$$

where p_d and p_a now refer to $\bar{y}$. To implement this calculation in general, it is convenient to define functions

$$\begin{aligned} C(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3) &= f_\mathrm{s}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)\, G(\mathbf{x}_1,\mathbf{x}_2)\, f_\mathrm{s}(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3) \\ S(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2) &= f_\mathrm{s}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)/p_\mathrm{s}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)\,, \end{aligned} \quad (14)$$

and then express $1/R(\bar{y}\,|\,\bar{x})$ in terms of these functions. This formulation extends easily to subpaths of arbitrary length, and can be evaluated efficiently by precomputing C and S for each edge. In this form, it is also easy to handle specular BSDF's, since the ratio S is always well-defined.

[12]If $p'_\mathrm{s}(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{z}_0)$ is the density with respect to ordinary solid angle, then $p_\mathrm{s} = p'_\mathrm{s}/|\cos(\theta_\mathrm{o})|$, where θ_o is the angle between $\mathbf{x}_1 \to \mathbf{z}_0$ and the surface normal at $\mathbf{x}_1$.



Lens perturbation   Caustic perturbation

Figure 3: The lens edge can be perturbed by regenerating it from either side: we call these *lens perturbations* and *caustic perturbations*.

5.3.2 Perturbations

There are some lighting situations where bidirectional mutations will almost always be rejected. This happens when there are small regions of the path space in which paths contribute much more than average. This can be caused by caustics, difficult visibility (e.g. a small hole), or by concave corners where two surfaces meet (a form of singularity in the integrand). The problem is that bidirectional mutations are relatively large, and so they usually attempt to mutate the path outside the high-contribution region.

One way to increase the acceptance probability is to use smaller mutations. The principle is that nearby paths will make similar contributions to the image, and so the acceptance probability will be high. Thus, rather than having many rejections on the same path, we can explore the other nearby paths of the high-contribution region.

Our solution is to choose a subpath of the current path, and move the vertices slightly. We call this type of mutation a *perturbation*. While the idea can be applied to arbitrary subpaths, our main interest is in perturbations that include the lens edge $\mathbf{x}_{k-1}\mathbf{x}_k$ (since other changes do not help to prevent long sample sequences at the same image point). We have implemented two specific kinds of perturbations that change the lens edge, termed *lens perturbations* and *caustic perturbations* (see Figure 3). These are described below.

Lens perturbations. We delete a subpath $\mathbf{x}_t \ldots \mathbf{x}_k$ of the form $(L|D)DS^*E$ (where we have used Heckbert's regular expression notation [8]; S, D, E, and L stand for specular, non-specular, lens, and light vertices respectively). This is called the *lens subpath*, and consists of $k-t$ edges and $k-t-1$ vertices. (Note that if $\mathbf{x}_t$ were specular, then any perturbation of $\mathbf{x}_{t-1}$ would result in a path $\bar{y}$ for which $f(\bar{y}) = 0$.)

To replace the lens subpath, we perturb the old image location by moving it a random distance R in a random direction ϕ. The angle ϕ is chosen uniformly, while R is exponentially distributed between two values $r_1 < r_2$:

$$R = r_2 \exp(-\ln(r_2/r_1)\, U)\,, \quad (15)$$

where U is uniformly distributed on $[0,1]$.

We then cast a ray at the new image location, and extend the subpath through additional specular bounces to be the same length as the original. The mode of scattering at each specular bounce is preserved (i.e. specular reflection or transmission), rather than making new random choices.[13] This allows us to efficiently sample rare combinations of events, e.g. specular reflection from a surface where 99% of the light is transmitted.

The calculation of $a(\bar{y} \mid \bar{x})$ is similar to the bidirectional case. The main difference is the method used for directional sampling (i.e. distribution (15) instead of the BSDF).

[13]If the perturbation moves a vertex from a specular to a non-specular material, then the mutation is immediately rejected.

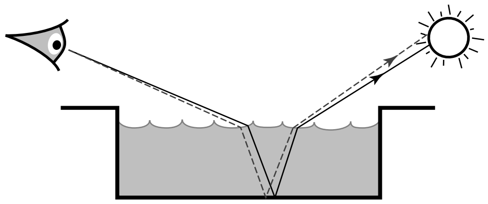


Figure 4: Using a two-chain perturbation to sample caustics in a pool of water. First, the lens edge is perturbed to generate a point $\mathbf{x}'$ on the pool bottom. Then, the direction from original point $\mathbf{x}$ toward the light source is perturbed, and a ray is cast from $\mathbf{x}'$ in this direction.

Caustic perturbations. Lens perturbations are not possible in some situations; the most notable example occurs when computing caustics. These paths have the form LS^+DE, which is unacceptable for lens perturbations.

However, there is another way to perturb paths with a suffix $\mathbf{x}_t \ldots \mathbf{x}_k$ of the form $(D|L)S^*DE$. To do this, we generate a new subpath starting from $\mathbf{x}_t$. The direction of the segment $\mathbf{x}_t \rightarrow \mathbf{x}_{t+1}$ is perturbed by an amount (θ, ϕ), where θ is exponentially distributed and ϕ is uniform. The technique is otherwise similar to lens perturbations.

Multi-chain perturbations. Neither of the above can handle paths with a suffix of the form $(D|L)DS^*DS^*DE$, i.e. caustics seen through a specular surface (see Figure 4). This can be handled by perturbing the path through more than one specular chain. At each non-specular vertex, we choose a new direction by perturbing the corresponding direction of the original subpath.

Parameter values. For lens perturbations, the image resolution is a guide to the useful range of values. We use $r_1 = 0.1$ pixels, while r_2 is chosen such that the perturbation region is 5% of the image area. For caustic and multi-chain perturbations, we use $\theta_1 = 0.0001$ radians and $\theta_2 = 0.1$ radians. The algorithm is not particularly sensitive to these values.

5.3.3 Lens subpath mutations

We now describe *lens subpath mutations*, whose goal is to stratify the samples over the image plane, and also to reduce the cost of sampling by re-using subpaths. Each mutation consists of deleting the lens subpath of the current path, and replacing it with a new one. (As before, the lens subpath has the form $(L|D)S^*E$.) The lens subpaths are stratified across the image plane, such that every pixel receives the same number of proposed lens subpath mutations.

We briefly describe how to do this. We initialize the algorithm with n' independent seed paths (Section 5.1), which are mutated in a rotating sequence. At all times, we also store a current lens subpath $\bar{x}_\mathrm{e}$. An eye subpath mutation consists of deleting the lens subpath of the current path $\bar{x}$, and replacing it with $\bar{x}_\mathrm{e}$.

The current subpath $\bar{x}_\mathrm{e}$ is re-used a fixed number of times n_e, and then a new one is generated. We chose $n' \gg n_\mathrm{e}$, to prevent the same lens subpath from being used multiple times on the same path.

To generate $\bar{x}_\mathrm{e}$, we cast a ray through a point on the image plane, and follow zero or more specular bounces until we obtain a non-specular vertex.[14] To stratify the samples on the image plane, we maintain a tally of the number of lens subpaths that have been generated at each pixel. When generating a new subpath, we choose a random pixel and, if it already has its quota of lens subpaths, we choose another one. We use a rover to make the search for a non-full pixel efficient. We also control the distribution of samples within each pixel, by computing a Poisson minimum-disc pattern and tiling it over the image plane.

The probability $a(\bar{y}\,|\,\bar{x})$ is similar to the bidirectional case, except that there is only one way of generating the new subpath. (Subpath re-use does not influence the calculation.)

5.3.4 Selecting between mutation types

At each step, we assign a probability to each of the three mutation types. This discrete distribution is sampled to determine which kind of mutation is applied to the current path.

We have found that it is important to make the probabilities relatively balanced. This is because the mutation types are designed to satisfy different goals, and it is difficult to predict in advance which types will be the most successful. The overall goal is to make mutations that are as large as possible, while still having a reasonable chance of acceptance. This can be achieved by randomly choosing between mutations of different sizes, so that there is a good chance of trying an appropriate mutation for any given path.

These observation are similar to those of *multiple importance sampling* (an alternative name for the technique in [25]). We would like a set of mutations that cover all the possibilities, even though we may not (and need not) know the optimum way to choose among them for a given path. It is perfectly fine to include mutations that are designed for special situations, and that result in rejections most of the time. This increases the cost of sampling by only a small amount, and yet it can increase robustness considerably.

5.4 Refinements

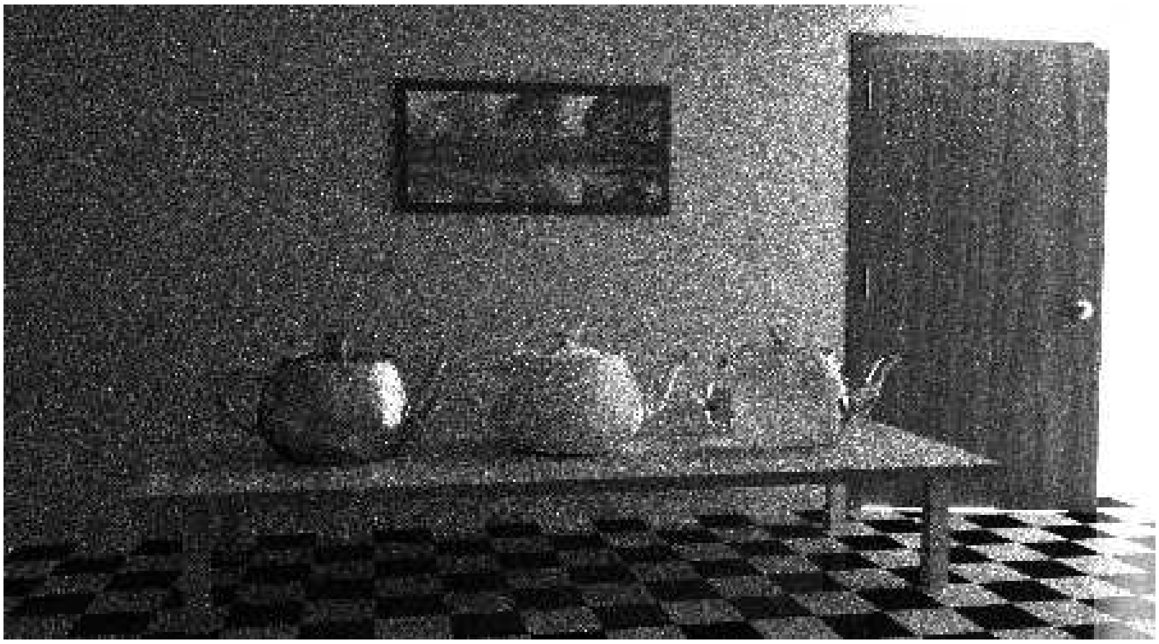
We describe several ideas that improve the efficiency of MLT.

Direct lighting. We use standard techniques for direct lighting (e.g. see [21]), rather than the Metropolis algorithm. In most cases, these standard methods give better results at lower cost, due to the fact that the Metropolis samples are not as well-stratified across the image plane (Section 5.2). By excluding direct lighting paths from the Metropolis calculation, we can apply more effort to the indirect lighting.

Using the expected sample value. For each proposed mutation, there is a probability $a(\bar{y}\,|\,\bar{x})$ of accumulating an image sample at $\bar{y}$, and a probability $1 - a(\bar{y}\,|\,\bar{x})$ of accumulating a sample at $\bar{x}$. We can make this more efficient by always accumulating a sample at both locations, weighted by the corresponding probability. Effectively, we have replaced a random variable by its expected value (a common variance reduction technique [11]). This is especially useful for sampling the dim regions of the image, which would otherwise receive very few samples.

Importance sampling for mutation probabilities. We describe a technique that can increase the efficiency of MLT substantially, by increasing the average acceptance probability. The idea is to implement a form of importance sampling which with respect to $a(\bar{y} \mid \bar{x})$, when deciding which mutation to attempt. This is done by weighting each possible mutation according to the probability with which the

[14] At a material with specular and non-specular components, we randomly choose between them.



(a) Bidirectional path tracing with 40 samples per pixel.

(b) Metropolis light transport with an average of 250 mutations per pixel [the same computation time as (a)].

Figure 5: All of the light in the visible portion of this scene comes through a door that is slightly ajar, such that about 0.1% of the light in the adjacent room comes through the doorway. The light source is a diffuse ceiling panel at the far end of a large adjacent room, so that virtually all of the light coming through the doorway has bounced several times. The MLT algorithm efficiently generates paths that go through the small opening between the rooms, by always preserving a path segment that goes through the doorway. The images are 900 by 500 pixels, and include the effects of all paths up to length 10.

deleted subpath can be regenerated. (This is the factor $p_{\mathrm{d},2}$ mentioned in Section 5.3.1.)

Let $\bar{x}$ be the current path, and consider a mutation that deletes the subpath $\mathbf{x}_s \ldots \mathbf{x}_t$. The insight is that given only the deleted subpath, it is already possible to compute some of the factors in the acceptance probability $a(\bar{y}\,|\,\bar{x})$. In particular, from (13) we see that $a(\bar{y} \mid \bar{x})$ is proportional to $1\,/\,R(\bar{x}\,|\,\bar{y})$, and that it is possible to compute all the components of $R(\bar{x} \mid \bar{y})$ except for the discrete probabilities p_d and p_a (these apply to $\bar{y}$, which has not been generated yet). The computable factors of $1\,/\,R(\bar{x}\,|\,\bar{y})$ are denoted $p_{\mathrm{d},2}$. In the example of Section 5.3.1, for instance, we have

$$p_{\mathrm{d},2}[1,2] = 1/C(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)\ .$$

The discrete probabilities for each mutation type are weighted by this factor, before a mutation is selected.

6 Results

We have rendered test images that compare Metropolis light transport with classical and bidirectional path tracing. Our path tracing implementations support efficient direct lighting calculations, importance-sampled BSDF's, Russian roulette on shadow rays, and several other optimizations.

Figure 5 shows a test scene with difficult indirect lighting. For equal computation times, Metropolis light transport gives far better results than bidirectional path tracing. Notice the details that would be difficult to obtain with many light transport algorithms: contact shadows, caustics under the glass teapot, light reflected by the white tiles under the door, and the brighter strip along the back of the floor (due to the narrow gap between the table and the wall). This scene contains diffuse, glossy, and specular surfaces, and the wall is untextured to clearly reveal the noise levels.

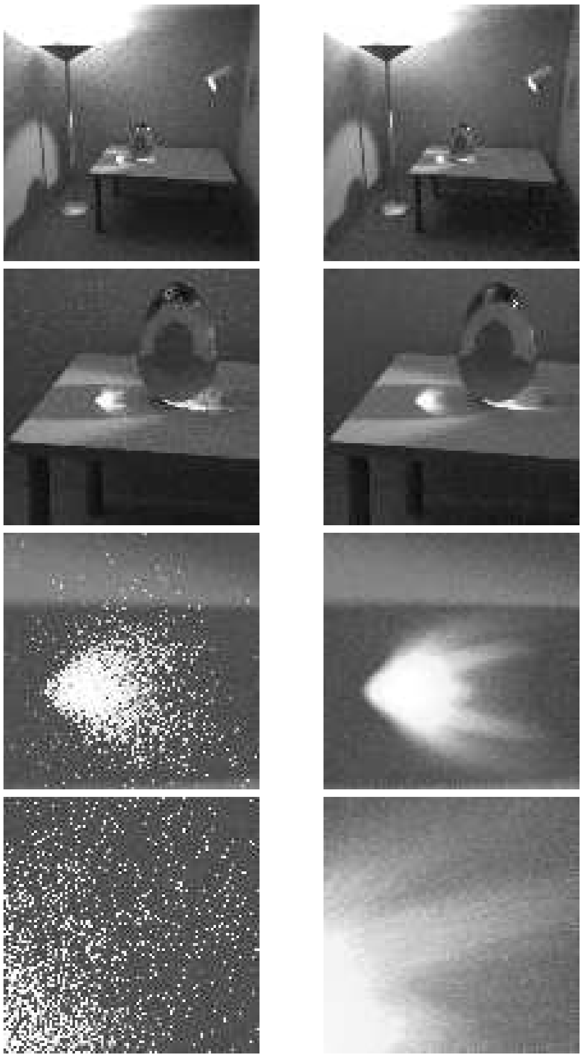
For this scene, MLT gains efficiency from its ability to change only part of the current path. The portion of the path through the doorway can be preserved and re-used for many mutations, until it is successfully mutated into a different path through the doorway. Note that perturbations are not essential to make this process efficient, since the path through the doorway needs to change only infrequently.

Figure 6 compares MLT against bidirectional path tracing for a scene with strong indirect illumination and caustics. Both methods give similar results in the top row of images (where indirect lighting from the floor lamp dominates). However, MLT performs much better as we zoom into the caustic, due to its ability to generate new paths by perturbing existing paths. The image quality degrades with magnification (for the same computation time), but only slowly. Notice the streaky appearance of the noise at the highest magnification. This is due to caustic perturbations: each ray from the spotlight is perturbed within a narrow cone; however, the lens maps this cone of directions into an elongated shape. The streaks are due to long strings of caustic mutations that were not broken by successful mutations of some other kind.

Even in the top row of images, there are slight differences between the two methods. The MLT algorithm leads to lower noise in the bright regions of the image, while the bidirectional algorithm gives lower noise in the dim regions. This is what we would expect, since the number of Metropolis samples varies according to the pixel brightness, while the number of bidirectional samples per pixel is constant.

Figure 7 shows another difficult lighting situation: caustics on the bottom of a small pool, seen indirectly through the ripples on the water surface. Path tracing does not work well, because when a path strikes the bottom of the pool, a reflected direction is sampled according to the BRDF. In this case, only a very small number of those paths will contribute, because the light source occupies about 1% of the visible hemisphere above the pool. (Bidirectional path tracing does not help for these paths, because they can be generated only starting from the eye.) As with Figure 6, perturbations are the key to sampling these caustics efficiently (recall Figure 4). One interesting feature of MLT is that it obtains these results without special handling of the light sources or specular surfaces — see [16] or [3] for good examples of what can be achieved if this restriction is lifted.

We have measured the performance of MLT relative to path tracing (PT) and bidirectional path tracing (BPT), for the same computation time. To do this, we computed the relative error $e_j = (\tilde{m}_j - m_j)/m_j$ at each pixel, where $\tilde{m}_j$ is the value computed by MLT, PT, or BPT, and m_j is the value from a reference image (computed using BPT with a large number of samples). Next, we computed the l_1, l_2, and l_∞ norms of the resulting error image (i.e. the array of e_j). Finally, we normalized the results, by dividing the error norms for PT and BPT by the corresponding error norm for MLT. This gave the following table:

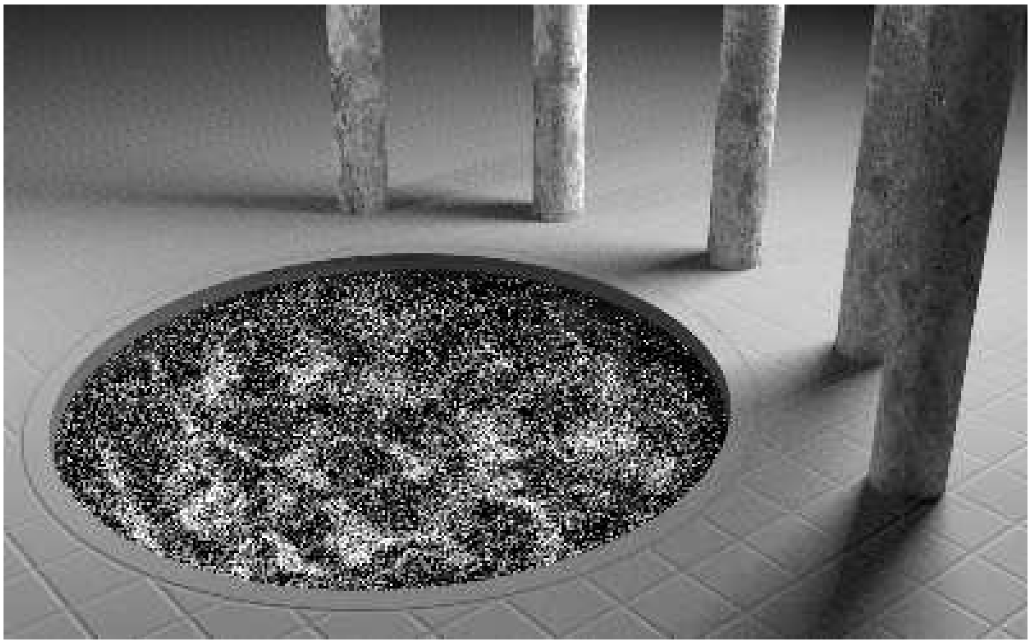


(a) (b)

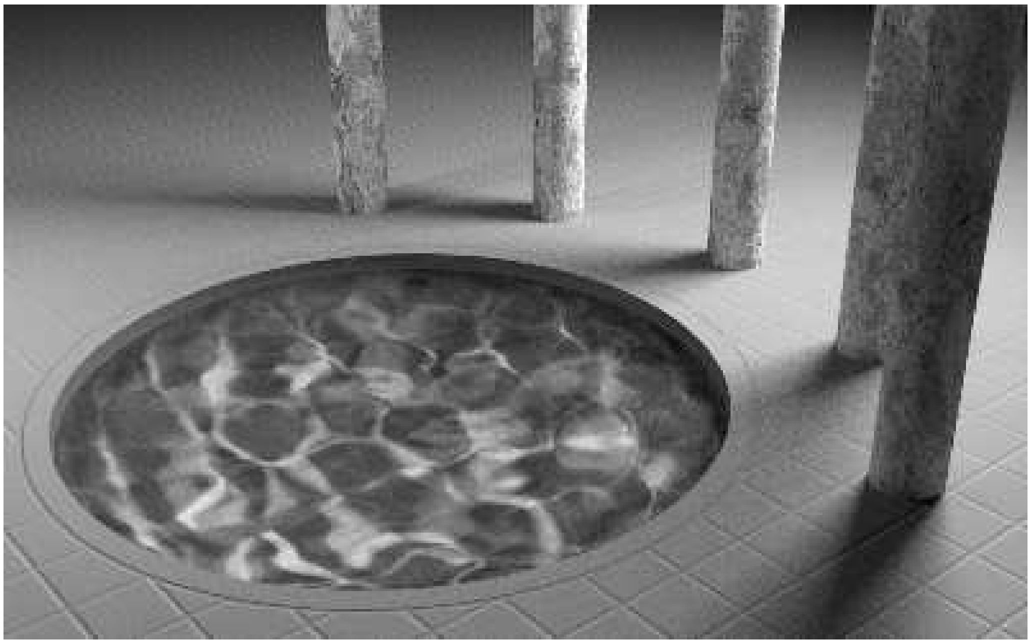
Figure 6: These images show caustics formed by a spotlight shining on a glass egg. Column (a) was computed with bidirectional path tracing using 25 samples per pixel, while (b) uses Metropolis light transport with the same number of ray queries (varying between 120 and 200 mutations per pixel). The solutions include all paths of up to length 7, and the images are 200 by 200 pixels.

Test Case	PT vs. MLT			BPT vs. MLT		
	l_1	l_2	l_∞	l_1	l_2	l_∞
Fig. 5	7.7	11.7	40.0	5.2	4.9	13.2
Fig. 6 (top)	2.4	4.8	21.4	0.9	2.1	13.7
Fig. 7	3.2	4.7	5.0	4.2	6.5	6.1

Note that the efficiency gain of MLT over the other methods is proportional to the *square* of the table entries, since the



(a) Path tracing with 210 samples per pixel.



(b) Metropolis light transport with an average of 100 mutations per pixel [the same computation time as (a)].

Figure 7: Caustics in a pool of water, viewed indirectly through the ripples on the surface. It is difficult for unbiased Monte Carlo algorithms to find the important transport paths, since they must be generated starting from the lens, and the light source only occupies about 1% of the hemisphere as seen from the pool bottom (which is curved). The MLT algorithm is able to sample these paths efficiently by means of perturbations. The images are 800 by 500 pixels.

error for PT and BPT decreases according to the square root of the number of samples. For example, the RMS relative error in Figure 5(a) is 4.9 times higher than in Figure 5(b), and so approximately 25 times more BPT samples would be required to achieve the same error levels. Even in the topmost images of Figure 6 (for which BPT is well-suited), notice that MLT and BPT are competitive.

The computation times were approximately 15 minutes for each image in Figure 6, 2.5 hours for the images in Figure 7, and 4 hours for the images in Figure 5 (all times measured on a 190 MHz MIPS R10000 processor). The memory requirements are modest: we only store the scene model, the current image, and a single path (or a small number of paths, if the mutation technique in Section 5.3.3 is used). For high-resolution images, the memory requirements could be reduced further by collecting the samples in batches, sorting them in scanline order, and applying them to an image on disk.

7 Conclusions

We have presented a novel approach to global illumination problems, by showing how to adapt the Metropolis sampling method to light transport. Our algorithm starts from a few seed light transport paths and applies a sequence of random mutations to them. In the steady state, the resulting Markov chain visits each path with a probability proportional to that path's contribution to the image. The MLT algorithm is notable for its generality and simplicity. A single control structure can be used with different mutation strategies to handle a variety of difficult lighting situations. In addition, the MLT algorithm has low memory requirements and always computes an unbiased result.

Many refinements of this basic idea are possible. For example, with modest changes we could use MLT to compute view-independent radiance solutions, by letting the m_j be the basis function coefficients, and defining $f(\bar{x}) = \sum_j f_j(\bar{x})$. We could also use MLT to render a sequences of images (as in animation), by sampling the the entire space-time of paths at once (thus, a mutation might try to perturb a path forward or backward in time). Another interesting problem is to determine the optimal settings for the various parameters used by the algorithm. The values we use have not been extensively tuned, so that further efficiency improvements may be possible. We hope to address some of these refinements and extensions in the future.

8 Acknowledgements

We would especially like to thank the anonymous reviewers for their detailed comments. In particular, review #4 led to significant improvements in the formulation and exposition of the paper. Thanks also to Matt Pharr for his comments and artwork.

This research was supported by NSF contract number CCR-9623851, and MURI contract DAAH04-96-1-0007.

References

[1] Arvo, J., and Kirk, D. Particle transport and image synthesis. *Computer Graphics (SIGGRAPH 90 Proceedings) 24*, 4 (Aug. 1990), 63–66.

[2] Beyer, M., and Lange, B. Rayvolution: An evolutionary ray tracing algorithm. In *Eurographics Rendering Workshop 1994 Proceedings* (June 1994), pp. 137–146. Also in *Photorealistic Rendering Techniques*, Springer-Verlag, New York, 1995.

[3] Collins, S. Reconstruction of indirect illumination from area luminaires. In *Rendering Techniques '95* (1995), pp. 274–283. Also in *Eurographics Rendering Workshop 1996 Proceedings* (June 1996).

[4] Cook, R. L., Porter, T., and Carpenter, L. Distributed ray tracing. *Computer Graphics (SIGGRAPH 84 Proceedings) 18*, 3 (July 1984), 137–145.

[5] Dutre, P., and Willems, Y. D. Potential-driven Monte Carlo particle tracing for diffuse environments with adaptive probability density functions. In *Rendering Techniques '95* (1995), pp. 306–315. Also in *Eurographics Rendering Workshop 1996 Proceedings* (June 1996).

[6] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[7] Halmos, P. R. *Measure Theory*. Van Nostrand, New York, 1950.

[8] Heckbert, P. S. Adaptive radiosity textures for bidirectional ray tracing. In *Computer Graphics (SIGGRAPH 90 Proceedings)* (Aug. 1990), vol. 24, pp. 145–154.

[9] Jensen, H. W. Importance driven path tracing using the photon map. In *Eurographics Rendering Workshop 1995* (June 1995), Eurographics.

[10] Kajiya, J. T. The rendering equation. In *Computer Graphics (SIGGRAPH 86 Proceedings)* (Aug. 1986), vol. 20, pp. 143–150.

[11] Kalos, M. H., and Whitlock, P. A. *Monte Carlo Methods, Volume I: Basics*. John Wiley & Sons, New York, 1986.

[12] Lafortune, E. P., and Willems, Y. D. Bi-directional path tracing. In *CompuGraphics Proceedings* (Alvor, Portugal, Dec. 1993), pp. 145–153.

[13] Lafortune, E. P., and Willems, Y. D. A theoretical framework for physically based rendering. *Computer Graphics Forum 13*, 2 (June 1994), 97–107.

[14] Lafortune, E. P., and Willems, Y. D. A 5D tree to reduce the variance of Monte Carlo ray tracing. In *Rendering Techniques '95* (1995), pp. 11–20. Also in *Eurographics Rendering Workshop 1996 Proceedings* (June 1996).

[15] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equations of state calculations by fast computing machines. *Journal of Chemical Physics 21* (1953), 1087–1091.

[16] Mitchell, D. P., and Hanrahan, P. Illumination from curved reflectors. In *Computer Graphics (SIGGRAPH 92 Proceedings)* (July 1992), vol. 26, pp. 283–291.

[17] Pattanaik, S. N., and Mudur, S. P. Adjoint equations and random walks for illumination computation. *ACM Transactions on Graphics 14* (Jan. 1995), 77–102.

[18] Peskun, P. H. Optimum monte-carlo sampling using markov chains. *Biometrika 60*, 3 (1973), 607–612.

[19] Reif, J. H., Tygar, J. D., and Yoshida, A. Computability and complexity of ray tracing. *Discrete and Computational Geometry 11* (1994), 265–287.

[20] Shirley, P., Wade, B., Hubbard, P. M., Zareski, D., Walter, B., and Greenberg, D. P. Global illumination via density-estimation. In *Eurographics Rendering Workshop 1995 Proceedings* (June 1995), pp. 219–230. Also in *Rendering Techniques '95*, Springer-Verlag, New York, 1995.

[21] Shirley, P., Wang, C., and Zimmerman, K. Monte Carlo methods for direct lighting calculations. *ACM Transactions on Graphics 15*, 1 (Jan. 1996), 1–36.

[22] Spanier, J., and Gelbard, E. M. *Monte Carlo Principles and Neutron Transport Problems*. Addison-Wesley, Reading, Massachusetts, 1969.

[23] Veach, E. Non–symmetric scattering in light transport algorithms. In *Eurographics Rendering Workshop 1996 Proceedings* (June 1996). Also in *Rendering Techniques '96*, Springer-Verlag, New York, 1996.

[24] Veach, E., and Guibas, L. Bidirectional estimators for light transport. In *Eurographics Rendering Workshop 1994 Proceedings* (June 1994), pp. 147–162. Also in *Photorealistic Rendering Techniques*, Springer-Verlag, New York, 1995.

[25] Veach, E., and Guibas, L. J. Optimally combining sampling techniques for Monte Carlo rendering. In *SIGGRAPH 95 Proceedings* (Aug. 1995), Addison-Wesley, pp. 419–428.

[26] Whitted, T. An improved illumination model for shaded display. *Communications of the ACM 32*, 6 (June 1980), 343–349.

Appendix A

To prove that (12) is unbiased, we show that the following identity is satisfied at each step of the random walk:

$$\int_{\mathbf{R}} w \; q_i(w, \bar{x}) \; dw = f(\bar{x}) \; , \qquad (16)$$

where q_i is the joint probability distribution of the i-th weighted sample $(W_i, \bar{X}_i)$. Clearly this condition is satisfied by q_0, noting that $q_0(w, \bar{x}) = \delta(w - f(\bar{x})/p_0(\bar{x}))\, p_0(\bar{x})$ (where δ denotes the Dirac delta distribution).

Next, observe that (4) is still true with p_j replaced by $q_j(w, \bar{x})$ (since the mutations set $W_i = W_{i-1}$). Multiplying both sides of (4) by w and integrating, we obtain

$$\int_{\mathbf{R}} w \; q_i(w, \bar{x}) \; dw = \int_{\mathbf{R}} w \; q_{i-1}(w, \bar{x}) \; ,$$

so that (16) is preserved by each mutation step.

Now given (16), the desired estimate (12) is unbiased since

$$\begin{aligned} E[W_i \; w_j(\bar{X}_i)] &= \int_{\Omega} \int_{\mathbf{R}} w \; w_j(\bar{x}) \; p_i(w, \bar{x}) \; dw \; d\mu(\bar{x}) \\ &= \int_{\Omega} w_j(\bar{x}) \; f(\bar{x}) \; d\mu(\bar{x}) \; = \; m_j \; . \end{aligned}$$