Chapter 11

Metropolis Light Transport

We propose a new Monte Carlo algorithm for solving the light transport problem, called *Metropolis light transport* (MLT). It is inspired by the Metropolis sampling method from computational physics, which is often used for difficult sampling problems in high-dimensional spaces. We show how the Metropolis method can be combined with the path integral framework of Chapter 8, in order to obtain an effective importance algorithm for the space of paths.

Paths are sampled according to the contribution they make to the ideal image, by means of a random walk through path space. Starting with a single seed path, we generate a sequence of light transport paths by applying random mutations (e.g. adding a new vertex to the current 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. This image is then estimated by sampling many paths, and recording their locations on the image plane.

The resulting 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 scenes with relatively simple illumination.

We start with a high-level overview of the MLT algorithm in Section 11.1, and then we describe its components in detail. Section 11.2 summarizes the classical Metropolis sampling algorithm, as developed in computational physics. Section 11.3 shows how to combine this idea with the path integral framework of Chapter 8, to yield an effective light transport algorithm. Section 11.4 discusses the properties that a good mutation strategy should have, and describes the strategies that we have implemented. In Section 11.5, we describe several refinements to the basic algorithm that can make it work more efficiently. Results are presented in Section 11.6, followed by conclusions and suggested extensions in Section 11.7. To our knowledge, this is the first application of the Metropolis method to transport problems of any kind.

11.1 Overview of the MLT algorithm

To make an image, we sample 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 flowing 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 $\bar{x}$. (It is closely related to the *measurement contribution function* f_j (described in Chapter 8), which specifies how much each path contributes to a given pixel value.)

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 $\bar{X}_0$, $\bar{X}_1, \ldots, \bar{X}_N$, 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

function METROPOLIS-LIGHT-TRANSPORT()
  $\bar{x} \leftarrow$ INITIALPATH()
  image $\leftarrow$ { array of zeros }
  for $i \leftarrow 1$ **to** N
    $\bar{y} \leftarrow$ MUTATE($\bar{x}$)
    $a \leftarrow$ ACCEPTPROB($\bar{x} \rightarrow \bar{y}$)
    if RANDOM() $< a$
      then $\bar{x} \leftarrow \bar{y}$
    RECORDSAMPLE(*image*, $\bar{x}$)
  return *image*

Figure 11.1: Pseudocode for the Metropolis light transport algorithm.

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.[1]

The basic structure of the MLT algorithm is summarized in Figure 11.1. We start with an image of zeros, and a single path $\bar{x}$ that contributes to the desired image. We then repeatedly propose a mutation to the current path, randomly decide whether or not to accept it (according to a carefully chosen probability), and update the image with a sample at the new path location.

The key advantage of the Metropolis approach is that the path space can be 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).

[1]At least, this is true of the basic algorithm; in Section 11.5, we describe optimizations that allow the samples to be weighted differently.

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.

In the remainder of this chapter, we will describe the MLT algorithm in more detail.

11.2 The Metropolis sampling algorithm

In 1953, Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller introduced an algorithm for handling difficult sampling problems in computational physics [Metropolis et al. 1953]. 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 Kalos & Whitlock [1986]). 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 $\bar{X}_0, \bar{X}_1, \ldots$ 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 must 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 $K(\bar{x} \to \bar{y})$ 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

$$\int_\Omega K(\bar{x} \to \bar{y})\, d\mu(\bar{y}) \;=\; 1 \quad \text{for all} \quad \bar{x} \in \Omega\,.$$

11.2.1 The stationary distribution

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

$$p_i(\bar{x}) \;=\; \int_\Omega K(\bar{y} \to \bar{x})\, p_{i-1}(\bar{y})\, d\mu(\bar{y})\,. \tag{11.1}$$

With mild conditions on K (discussed further in Section 11.4.1), the p_i will converge to a unique density function 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 $\bar{X}_0 = \bar{x}_0$, so that $p_0(\bar{x}) = 1$ for $\bar{x} = \bar{x}_0$, and $p_0(\bar{x}) = 0$ otherwise. Then after one transition, $\bar{X}_1$ is distributed with equal probability among $\bar{x}_0$ 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 density function p^*, which necessarily satisfies

$$p^*(\bar{x}) \;=\; \sum_{\bar{y}} K(\bar{y} \to \bar{x})\, p^*(\bar{y})\,.$$

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

11.2.2 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 $T(\bar{x} \to \bar{y})$ 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 $a(\bar{x} \to \bar{y})$ which will be defined below. That is, we let

$$\bar{X}_i \;=\; \begin{cases} \bar{X}_i' & \text{with probability } a(\bar{X}_{i-1} \to \bar{X}_i')\,, \\ \bar{X}_{i-1} & \text{otherwise}\,. \end{cases} \qquad (11.2)$$

To see how to set $a(\bar{x} \to \bar{y})$, suppose that we have already reached equilibrium, i.e. p_{i-1} is proportional to f. We must define $K(\bar{x} \to \bar{y})$ 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 $f(\bar{x})\, T(\bar{x} \to \bar{y})\, a(\bar{x} \to \bar{y})$, 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:

$$f(\bar{x})\, T(\bar{x} \to \bar{y})\, a(\bar{x} \to \bar{y}) \;=\; f(\bar{y})\, T(\bar{y} \to \bar{x})\, a(\bar{y} \to \bar{x})\,, \qquad (11.3)$$

a condition known as *detailed balance*. We can verify that if $p_{i-1} \propto f$ and condition (11.3) holds, then equilibrium is preserved:

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

Thus the unique equilibrium distribution must be proportional to f.

11.2.3 The acceptance probability

Recall that f is given, and T was chosen arbitrarily. Thus, equation (11.3) is a condition on the ratio $a(\bar{x} \to \bar{y}) / a(\bar{y} \to \bar{x})$. In order to reach equilibrium as quickly as possible, the best strategy is to make $a(\bar{x} \to \bar{y})$ and $a(\bar{y} \to \bar{x})$ as large as possible [Peskun 1973], which is achieved by letting

$$a(\bar{x} \to \bar{y}) \;=\; \min\left\{1, \frac{f(\bar{y})\, T(\bar{y} \to \bar{x})}{f(\bar{x})\, T(\bar{x} \to \bar{y})}\right\}\,. \qquad (11.4)$$

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.

11.2.4 Comparison with genetic algorithms

The Metropolis method differs from genetic algorithms [Goldberg 1989] in several ways. First, they have different purposes: genetic algorithms are intended for optimization problems, while the Metropolis method is intended for 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 & Lange [1994] 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.

11.3 Theoretical formulation of Metropolis light transport

To complete the MLT algorithm outlined in Section 11.1, 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 that affects many Metropolis applications. Most importantly, we must design a suitable set of mutations on paths, such that the Metropolis method will work efficiently. In this section we deal with the first two problems, by showing how the Metropolis method can be adapted to estimate all of the pixel values of an image simultaneously and without bias.

Recall that according to the path integral framework of Chapter 8, each measurement I_j can be expressed in the form

$$I_j \;=\; \int_{\Omega} f_j(\bar{x})\, d\mu(\bar{x})\,,$$

where Ω is the set of all transport paths, μ is the area-product measure, and f_j is the measurement contribution function. In our case, the measurements I_j are pixel values. This implies that each integrand f_j has the form

$$f_j(\bar{x}) \;=\; h_j(\bar{x})\, f(\bar{x})\,, \tag{11.5}$$

where h_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 region of the image plane along a set D of paths.[2] Note that h_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 density function p, and using the identity

$$I_j \;=\; E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{h_j(\bar{X}_i)\, f(\bar{X}_i)}{p(\bar{X}_i)}\right]\,. \tag{11.6}$$

Notice that if we could take samples according to the density function $p = (1/b)\, f$ (where b is the normalization constant $\int_\Omega f(\bar{x})\, d\mu(\bar{x})$), the estimate for each pixel would simply be

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

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

This approach requires the evaluation of b, and the ability to sample from a density function 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 the initial path $\bar{X}_0$, which will bias the results (this is called *start-up bias*).

[2] We define $f(\bar{x})$ to be zero for paths that do not contribute to any pixel value (so that we do not waste any samples there).

11.3.1 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 density function p_0 on paths (we use bidirectional path tracing for this purpose). To compensate for the fact that p_0 is not the desired equilibrium distribution $p^* = (1/b)\, f$, 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 $W_0\, h_j(\bar{X}_0)$ (see equation (11.6). All of these quantities are computable since $\bar{X}_0$ is known.

Additional samples $\bar{X}_1$, $\bar{X}_2$, . . ., $\bar{X}_N$ 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 density function p_i, which only approaches the stationary distribution $p^* = (1/b)\, f$ 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, and use the following estimate for pixel j:

$$I_j = E\left[\frac{1}{N} \sum_{i=1}^{N} W_i\, h_j(\bar{X}_i) \right] . \tag{11.7}$$

We give a proof that this estimate is unbiased in Appendix 11.A. However, the following explanation may give some additional insight. Recall that the initial path is a random variable, so that the expected value in (11.7) is an average over all possible values of $\bar{X}_0$. Thus, consider a large group of initial paths $\bar{X}_{0,j}$ obtained by sampling p_0 many times. If p_0 is the stationary distribution $(1/b)\, f$, and all the paths are weighted equally, then this group of paths is in equilibrium: the distribution of paths does not change as mutations are applied. Now suppose that we again sample a large group of initial paths, this time from an arbitrary density function p_0, and that we assign each path the weight $f(\bar{X}_{0,j})/p_0(\bar{X}_{0,j})$. Even though this does not give the desired distribution of *paths*, the distribution of *weight* is proportional to the desired equilibrium f. The equilibrium is preserved as the paths are mutated (just as in the first case), which leads to an unbiased estimate of I_j.

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 is difficult to obtain in some cases. (Often the reason for using the Metropolis method in the first place is the lack of suitable alternatives.)

11.3.2 Initialization

In practice, initializing the MLT algorithm with a single seed path does not work well. If we generate only one path $\bar{X}_0$ (e.g. using bidirectional path tracing), it is likely that $W_0 = 0$ (for example, the path may go 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. This does not contradict the fact that the algorithm is unbiased, since bias refers only to the *expected* value on a particular run.

The obvious solution is to run n copies of the algorithm in parallel (with different random initial paths), and accumulate all the samples into one image. The strategy we have implemented has two phases. First we sample a moderately large number of paths $\bar{X}_{0,1}$, . . ., $\bar{X}_{0,n}$, and let $W_{0,1}$, . . ., $W_{0,n}$ be the corresponding weights. We then select a representative sample of n' of these paths (where n' is much smaller than n), and assign them equal weights. (The reasons for doing this are discussed below.) These paths are used as independent seeds for the Metropolis phase of the algorithm.

Specifically, each representative path $\bar{X}'_{0,i}$ is chosen from among the initial paths $\bar{X}_{0,j}$ according to discrete probabilities that are proportional to $W_{0,j}$. All of these paths $\bar{X}'_{0,i}$ are assigned the same weight:

$$W'_{0,i} \;=\; \frac{1}{n} \sum_{j=1}^{n} W_{0,j} \;.$$

It is straightforward to show that this resampling procedure is unbiased.[3]

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 in order to apply the resampling step; they can be regenerated by restarting the random number generator with the same seed.

[3] The resampling can be optimized slightly by choosing the new paths with equal spacing in the cumulative weight distribution of the $\bar{X}_{0,j}$; this ensures that the same path is not selected twice, unless its weight is at least a fraction $1/n'$ of the total.

It is often reasonable to choose $n' = 1$ (i.e. to initialize the Metropolis algorithm with a single representative seed path). In this case, the purpose of sampling n paths in the first phase is to estimate the mean value of W_0, which determines the absolute image brightness.[4] If the image is desired only up to a constant scale factor, then the first phase can be terminated as soon as a single path with $f(\bar{x}) > 0$ is found. The main reasons for retaining more than one seed path (i.e. for choosing $n' > 1$) are to implement convergence tests (see below) or lens subpath mutations (see Section 11.4.4).

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. (Observe that even if the algorithm is initialized using $100\,000$ bidirectional samples, this would represent less than one sample per pixel for an image of reasonable size.)

11.3.3 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 the samples generated by a single run of the Metropolis algorithm, since consecutive samples are highly correlated.)

To test for convergence, the Metropolis phase can be started 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. For each such pixel, we thus have available n' independent, unbiased samples of its true value. (Each sample value changes as the algorithm proceeds, since it depends on how many path mutations have contributed to the specified pixel of a particular test image.) The sample variance of these pixels can then be tested periodically, until the results are within prespecified bounds. Notice that unlike most graphics problems, the number of independent samples per pixel remains constant (at

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

n') as the algorithm proceeds — it is the *values* of the samples that change.

If the radiance values that contribute to a given pixel can be bounded in advance, more advanced convergence techniques could in theory be applied. In particular Dagum et al. [1995] have proposed an algorithm that can estimate the expected value of a random variable Z to within a factor of $(1 + \epsilon)$ with a guaranteed probability of at least $1 - \delta$. They assume only that Z is bounded within a known range $[0, M]$. Furthermore, the number of independent samples used by their algorithm is proven to optimal for every given ϵ, δ, and Z to within a constant factor. In the case of the Metropolis light transport, observe that an arbitrary number of independent samples can be generated by restarting the algorithm with new seed paths. However, once again it seems impractical to apply this technique to every pixel of an image.

These convergence testing procedures add a small amount of bias, but this is inevitable for any technique that makes guarantees about the quality of its results. Note that the first technique we described bounds the sample variance of the test pixels, while the second technique bounds the actual error. Also note that unbiased techniques such as two-stage adaptive sampling [Kirk & Arvo 1991] do not make any guarantees about the final image quality, due to the possibility of outlying samples during the second stage of sampling.

Finally, note that in all of our tests the number of mutations was specified manually, both to eliminate bias and so that we would have explicit control over the computation time.

11.3.4 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 frequencies. 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 c_i is sampled with an estimator of the form c_i/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.[5]

11.4 Good mutation strategies

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 paths are very small, or because too many mutations are rejected.

Correlation can be minimized by choosing a suitable set of path mutations. We first consider some of the properties that these mutations should have, in order to minimize the error in the final image. Then we describe three specific mutation strategies that we have implemented, namely *bidirectional mutations*, *perturbations*, and *lens subpath mutations*. These strategies are designed to satisfy different subsets of the goals mentioned below; our implementation uses a mixture of all three (as we will discuss in Section 11.4.5).

11.4.1 Desirable mutation properties

In this section, we describe the properties that a good mutation strategy should have. These are the main factors that need to be considered when a mutation strategy is designed.

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

[5]Another way to handle color is to have a separate run for each frequency. However, this is inefficient (we get less information from each path) and leads to unnecessary color noise. Note that it is *not* necessary to have a separate run at each wavelength in order to handle dispersion (i.e. a refractive index that varies with wavelength). It can be handled perfectly well in the model described above, by randomly sampling a spectral band only when a dispersive material is actually encountered (and using a weight of the usual form f/p).

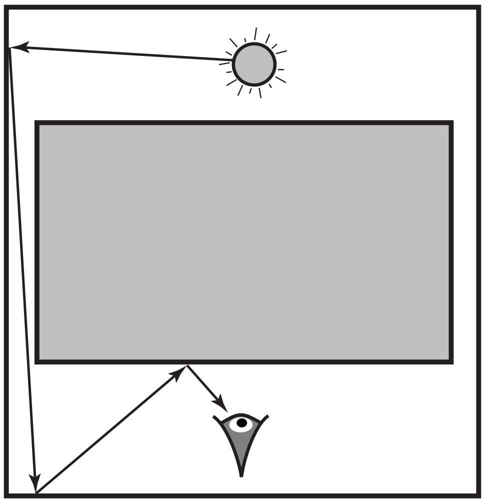


Figure 11.2: If only additions and deletions of a single vertex are allowed, then paths cannot mutate from one side of the barrier to the other.

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 11.2, 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 from one side of the barrier to the other, 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 $T(\bar{x} \to \bar{y}) > 0$ 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 11.4.2).

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.

We now consider some specific mutation strategies that address these goals. 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 $T(\bar{x} \to \bar{y})$ of each mutation: in other words, the mutation strategy is allowed to depend on the current path.

11.4.2 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 $p_{\mathrm{d}}[l, m]$ to the deletion of each subpath $\mathbf{x}_l \ldots \mathbf{x}_m$. The endpoints of this subpath are not included, so that $\mathbf{x}_l \ldots \mathbf{x}_m$ consists of $m - l$ edges and $m - l - 1$ vertices (with indices satisfying $-1 \leq l < m \leq k + 1$).

In our implementation, the deletion probability $p_{\mathrm{d}}[l, m]$ is a product two factors. The first factor $p_{\mathrm{d},1}$ depends only on the subpath length (i.e. the number of edges); 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 11.5.

The density function $p_{\mathrm{d}}[l, m]$ is normalized and sampled to determine the deleted subpath. At this point, $\bar{x}$ has been split into two (possibly empty) pieces $\mathbf{x}_0 \ldots \mathbf{x}_l$ and $\mathbf{x}_m \ldots \mathbf{x}_k$. To complete the mutation, we must generate a new subpath that connects these two pieces.

We start by choosing the number of vertices l' and m' to be added to each side. This is done in two steps: first, we choose the new subpath length, $k_{\mathrm{a}} = l' + m' + 1$. 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 k_{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 l' and m' (subject to the condition $l' + m' + 1 = k_{\mathrm{a}}$), according to another discrete distribution $p_{\mathrm{a},2}$ that assigns equal probability to each candidate value of l'. For convenience, we let $p_{\mathrm{a}}[l', m']$ 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 11.4.1), so that the algorithm can never get "stuck" forever in a small subregion of the path space. (However, if the mutations are poorly chosen then the algorithm might get stuck for a long finite time.)

Parameter values. The following values have provided reasonable results on our test cases. For the probability $p_{\mathrm{d},1}[k_{\mathrm{d}}]$ of deleting a subpath of length $k_{\mathrm{d}} = m - l$, we use

$p_{\mathrm{d},1}[1] = 0.25$, $p_{\mathrm{d},1}[2] = 0.5$, and $p_{\mathrm{d},1}[k_\mathrm{d}] = 2^{-k_\mathrm{d}}$ for $k_\mathrm{d} \geq 3$. For the probability $p_{\mathrm{a},1}[k_\mathrm{a}]$ of adding a subpath of length k_a, we use $p_{\mathrm{a},1}[k_\mathrm{d}] = 0.5$, $p_{\mathrm{a},1}[k_\mathrm{d} \pm 1] = 0.15$, and $p_{\mathrm{a},1}[k_\mathrm{d} \pm j] = 0.2(2^{-j})$ for $j \geq 2$.

11.4.2.1 Evaluation of the acceptance probability.

Observe that the acceptance probability $a(\bar{x} \to \bar{y})$ from (11.4) can be written as the ratio

$$a(\bar{x} \to \bar{y}) = \frac{R(\bar{x} \to \bar{y})}{R(\bar{y} \to \bar{x})}, \qquad \text{where} \qquad R(\bar{x} \to \bar{y}) = \frac{f(\bar{y})}{T(\bar{x} \to \bar{y})}. \tag{11.8}$$

The form of $R(\bar{x} \to \bar{y})$ 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 $T(\bar{x} \to \bar{y})$.

Specifically, $T(\bar{x} \to \bar{y})$ is the product of the discrete probability $p_\mathrm{d}[l, m]$ for deleting the subpath $\mathbf{x}_l \ldots \mathbf{x}_m$, and the probability density for generating the $l' + m'$ new vertices of $\bar{y}$. To calculate the latter, we must take into account all $l' + m' + 1$ ways that the new vertices can be split between subpaths generated from $\mathbf{x}_l$ and $\mathbf{x}_m$. (Although these vertices were generated by a particular choice of l', the probability $T(\bar{x} \to \bar{y})$ must take into account all of these 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 $T(\bar{x} \to \bar{y})$. 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 $\mathbf{x}_1\mathbf{x}_2$ (see Figure 11.3). It is replaced by new vertex $\mathbf{z}_1$ by casting a ray from $\mathbf{x}_1$, so that the new path is

$$\bar{y} = \mathbf{x}_0\, \mathbf{x}_1\, \mathbf{z}_1\, \mathbf{x}_2\, \mathbf{x}_3\,.$$

This corresponds to the random choices $l = 1$, $m = 2$, $l' = 1$, $m' = 0$.

Let $P_{\sigma^\perp}(\mathbf{x} \to \mathbf{x}')$ denote the probability density of sampling the direction from $\mathbf{x}$ to $\mathbf{x}'$, measured with respect to projected solid angle.[6] Then the probability density of sampling

[6]Recall that if $P_\sigma(\mathbf{x} \to \mathbf{x}')$ is the density with respect to ordinary solid angle, then $P_{\sigma^\perp} = P_\sigma \,/\, |\cos(\theta_\mathrm{o})|$, where θ_o is the angle between $\mathbf{x} \to \mathbf{x}'$ and the surface normal at $\mathbf{x}$.

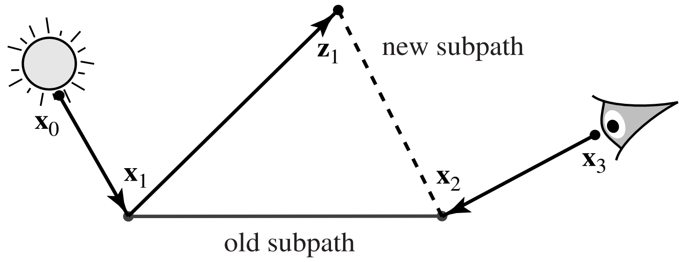


Figure 11.3: A simple example of a bidirectional mutation. The original path $\bar{x} = \mathbf{x}_0\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ is modified by deleting the edge $\mathbf{x}_1\mathbf{x}_2$ and replacing it with a new vertex $\mathbf{z}_1$. The new vertex is generated by sampling a direction at $\mathbf{x}_1$ (according to the BSDF) and casting a ray. This yields a mutated path $\bar{y} = \mathbf{x}_0\mathbf{x}_1\mathbf{z}_1\mathbf{x}_2\mathbf{x}_3$.

the vertex $\mathbf{x}'$ (measured with respect to surface area) is given by $P_{\sigma^\perp}(\mathbf{x} \to \mathbf{x}')\, G(\mathbf{x} \leftrightarrow \mathbf{x}')$.

We now have all of the information necessary to compute $R(\bar{x} \to \bar{y})$. From definition (8.7), the numerator is

$$\begin{aligned} f(\bar{y}) \;&=\; f_{\mathrm{s}}(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{z}_1)\, G(\mathbf{x}_1 \leftrightarrow \mathbf{z}_1)\, f_{\mathrm{s}}(\mathbf{x}_1 \to \mathbf{z}_1 \to \mathbf{x}_2) \\ &\quad \cdot G(\mathbf{z}_1 \leftrightarrow \mathbf{x}_2)\, f_{\mathrm{s}}(\mathbf{z}_1 \to \mathbf{x}_2 \to \mathbf{x}_3)\,, \end{aligned}$$

where the factors shared between $R(\bar{x} \to \bar{y})$ and $R(\bar{y} \to \bar{x})$ have been omitted. The denominator is

$$\begin{aligned} T(\bar{x} \to \bar{y}) \;&=\; p_{\mathrm{d}}[1,2] \Big\{ p_{\mathrm{a}}[1,0]\, P_{\sigma^\perp}(\mathbf{x}_1 \to \mathbf{z}_1)\, G(\mathbf{x}_1 \leftrightarrow \mathbf{z}_1) \\ &\qquad + p_{\mathrm{a}}[0,1]\, P_{\sigma^\perp}(\mathbf{x}_2 \to \mathbf{z}_1)\, G(\mathbf{x}_2 \leftrightarrow \mathbf{z}_1) \Big\}\,. \end{aligned}$$

In a similar way, we find that the factor $R(\bar{y} \to \bar{x})$ for the mutation in the reverse direction is given by

$$R(\bar{y} \to \bar{x}) \;=\; \frac{f_{\mathrm{s}}(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2)\, G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2)\, f_{\mathrm{s}}(\mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}_3)}{p_{\mathrm{d}}[1,3]\, p_{\mathrm{a}}[0,0]}\,,$$

where p_{d} and p_{a} now refer to the path $\bar{y}$.

Implementation. We now describe how to compute the acceptance probability for bidirectional mutations in general form, and we also discuss how to implement this calculation efficiently.

Let $\bar{x} = \mathbf{x}_0 \ldots \mathbf{x}_k$ be the old path, let $\mathbf{x}_l \ldots \mathbf{x}_m$ be the deleted subpath, and let $\mathbf{z}_1 \ldots \mathbf{z}_{k_\mathrm{a}-1}$ be the vertices of the new subpath. This yields a mutated path $\bar{y}$ of the form

$$\begin{aligned} \bar{y} &= \mathbf{y}_0 \ldots \mathbf{y}_{k'} \\ &= \mathbf{x}_0 \ldots \mathbf{x}_l \, \mathbf{z}_1 \ldots \mathbf{z}_{k_\mathrm{a}-1} \, \mathbf{x}_m \ldots \mathbf{x}_k \,, \end{aligned}$$

where $k' = k - k_\mathrm{d} + k_\mathrm{a}$ is the length of the new path $\bar{y}$. (Recall that $k_\mathrm{d} = m - l$ and $k_\mathrm{a} = l' + m' + 1$ represent the number of edges in the old and new subpaths respectively.

Rather than evaluating the ratio $R(\bar{x} \to \bar{y})$ as we did in the example above, it is more convenient to evaluate its reciprocal:[7]

$$Q(\bar{x} \to \bar{y}) = \frac{1}{R(\bar{x} \to \bar{y})} = \frac{T(\bar{x} \to \bar{y})}{f(\bar{y})} \,. \tag{11.9}$$

This quantity can be evaluated efficiently using the same techniques that were developed for bidirectional path tracing in Chapter 10. In particular, suppose that we split $\bar{y}$ into two pieces, using the i-th edge of the new subpath as the connecting edge. In other words, consider the light subpath

$$\mathbf{y}_0 \ldots \mathbf{y}_{l+i-1} = \mathbf{x}_0 \ldots \mathbf{x}_l \, \mathbf{z}_1 \ldots \mathbf{z}_{i-1} \,,$$

and the eye subpath

$$\mathbf{y}_{l+i} \ldots \mathbf{y}_{k'} = \mathbf{z}_i \ldots \mathbf{z}_{k_\mathrm{a}-1} \mathbf{x}_m \ldots \mathbf{x}_k \,,$$

where $1 \le i \le k_\mathrm{a}$. These subpaths have $s = l + i$ and $t = (k' + 1) - (l + i)$ vertices respectively. Now let C_i^{bd} be the unweighted contribution from bidirectional tracing that would be computed in this situation:

$$C_i^{\mathrm{bd}} = C_{s,t}^* \,,$$

[7] The quantity $Q(\bar{x} \to \bar{y})$ has an interesting interpretation: it is simply the probability density of sampling the path $\bar{y}$, measured with respect to the *image contribution measure* defined by $\mu^{\mathrm{i}}(D) = \int_D f(\bar{x}) \, \mu(\bar{x})$. This measure μ^{i} is closely related to the *measurement contribution measure* μ_j^{m} defined in Appendix 8.A, except that it corresponds to the contribution made by a set of paths D to the entire image rather than to an individual measurement I_j.

where $C^*_{s,t}$ has already been defined in equation (10.5) . The value of $Q(\bar{x} \to \bar{y})$ can then be expressed as

$$Q(\bar{x} \to \bar{y}) \;=\; p_{\mathrm{d}}[l, m] \sum_{i=1}^{k_{\mathrm{a}}} \frac{p_{\mathrm{a}}[i-1, k_{\mathrm{a}}-i]}{C^{\mathrm{bd}}_i} \,. \tag{11.10}$$

To evaluate this sum efficiently we first compute the unweighted bidirectional contribution $C^{\mathrm{bd}}_{l'+1}$ (corresponding to the way the path was actually generated, using l' new light vertices and m' new eye vertices). This is done using the weights α^L_s, α^E_t and the connecting factor $c_{s,t}$ defined in Chapter 10. If the contribution $C^{\mathrm{bd}}_{l'+1}$ evaluates to zero (for example if the visibility test fails), then the mutation is immediately rejected. Otherwise, we compute the reciprocal value $1/C^{\mathrm{bd}}_{l'+1}$, and find the values of the other factors $1/C^{\mathrm{bd}}_i$ by iteratively applying the relationship (10.9) given in Chapter 10. This calculation is just a simple loop and can be done very efficiently.

11.4.3 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, we can explore the other nearby paths that also have a high contribution.

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 11.4). These are described below.

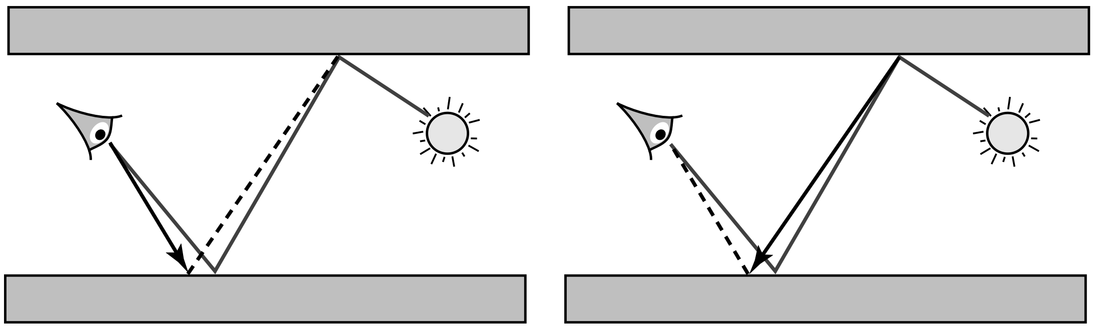


Figure 11.4: The lens edge can be perturbed by regenerating it from either side: we call these *lens perturbations* and *caustic perturbations*.

Lens perturbations. We delete a subpath $\mathbf{x}_m \ldots \mathbf{x}_k$ of the form $(L|D)DS^*E$ (where the symbols S, D, E, and L stand for specular, non-specular, lens, and light vertices respectively).[8] This is called the *lens subpath*, and consists of $k - m$ edges and $k - m - 1$ vertices (the vertex $\mathbf{x}_m$ is not included). Note that we require both $\mathbf{x}_m$ and $\mathbf{x}_{m+1}$ to be non-specular, since otherwise any perturbation would result in a path $\bar{y}$ for which $f(\bar{y}) = 0$.

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

$$R \;=\; r_2 \, \exp(-\ln(r_2/r_1)\, U)\,, \tag{11.11}$$

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. (If the perturbation moves a vertex from a specular to a non-specular material, then the mutation is immediately rejected.) This allows us to efficiently sample rare

[8] This is Heckbert's regular expression notation, as described in Section 8.3.1. We have not used the full-path notation of Section 8.3.2, although we assume that the light source has type $L(S|D)D$ and the lens has type $D(S|D)E$ with respect to the classifications introduced there.

combinations of events, e.g. specular reflection from a surface where 99% of the light is transmitted. This is important when only some of these combinations contribute to the image: for example, consider a scene model containing a glass window, where the environment beyond the window is dark. In this case, only reflections from the window will contribute significantly to the image.

The calculation of $a(\bar{x} \to \bar{y})$ is similar to the bidirectional case. The main difference is the method used to select a sample point on the image plane (i.e. equation (11.11) is used, rather than choosing a point uniformly at random within the image region).

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 not acceptable for lens perturbations.

Fortunately there is another way to perturb these paths, or in fact any path with a suffix $\mathbf{x}_m \ldots \mathbf{x}_k$ of the form $(D|L)S^*DE$ (see Figure 11.5). To do this, we generate a new subpath starting from the vertex $\mathbf{x}_m$. The direction of the segment $\mathbf{x}_m \to \mathbf{x}_{m+1}$ is perturbed by a random amount (θ, ϕ), where the $\theta = 0$ axis corresponds to the direction of the original ray. As before, the angle ϕ is chosen uniformly, while θ is exponentially distributed between two values θ_1 and θ_2:

$$\theta \;=\; \theta_2 \, \exp(-\ln(\theta_2/\theta_1)\, U)\,,$$

where U is uniformly distributed on $[0, 1]$. The technique is otherwise similar to lens perturbations, i.e. the new subpath is extended to the same length as the original, and the mode of scattering at each bounce is preserved.

Multi-chain perturbations. Neither of the above can handle paths with a suffix of the form $(D|L)DS^+DS^+E$, i.e. caustics seen through a specular surface. This can be handled by perturbing the path through more than one specular chain. A lens perturbation is used for the first chain DS^+E, and a new direction is chosen for the first edge of each subsequent chain DS^+D by perturbing the direction of the corresponding edge in the original subpath (using the same method described for caustic perturbations). Figure 11.6 shows an example of a situation where this technique is useful.

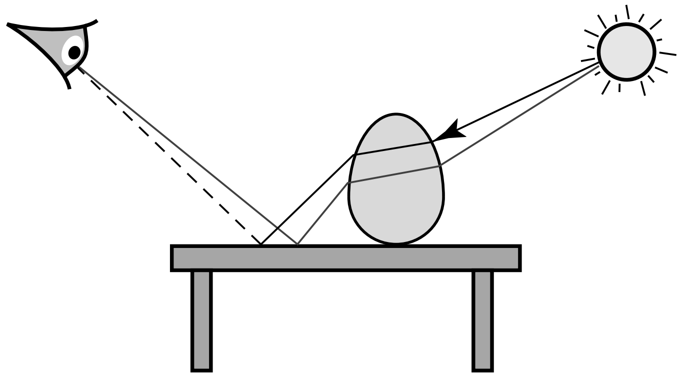


Figure 11.5: A caustic perturbation. A new path is generated by perturbing the direction of the ray from the light source by a small amount, and then tracing the perturbed ray through the same sequence of specular reflections and refractions as the original path.

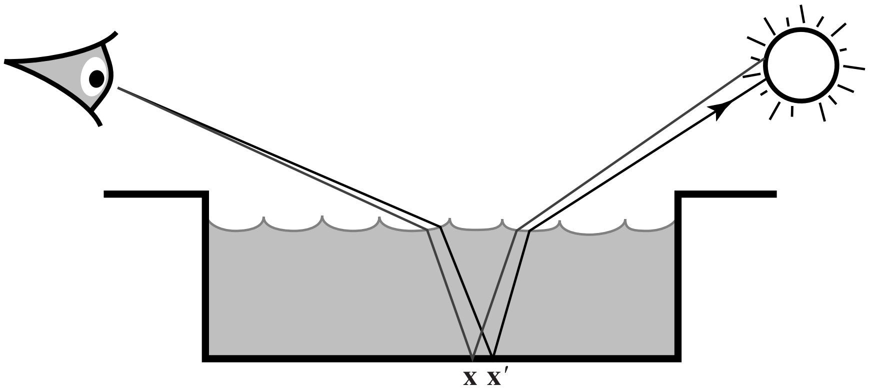


Figure 11.6: Using a two-chain perturbation to sample caustics in a pool of water. First, the lens edge is perturbed to generate a point $\mathbf{x}'$ on the pool bottom. Then, the direction from original point $\mathbf{x}$ toward the light source is perturbed, and a ray is cast from $\mathbf{x}'$ in this direction.

Parameter values. For lens perturbations, the image resolution is a guide to the useful range of values. We use a minimum perturbation size of $r_1 = 0.1$ pixels, while r_2 is chosen such that the perturbation region is 5% of the image area. For caustic perturbations, we also make use of the image resolution. Specifically, the maximum perturbation angle is defined as

$$\theta_2 \;=\; \theta(r_2)\,\frac{\|\mathbf{x}_k - \mathbf{x}_{k-1}\|}{\sum_{i=m+1}^{k-1}\|\mathbf{x}_i - \mathbf{x}_{i-1}\|}\,,$$

where $\mathbf{x}_m \ldots \mathbf{x}_k$ is the perturbed subpath, and $\theta(r)$ is the angle through which the ray $\mathbf{x}_k \to \mathbf{x}_{k-1}$ needs to be perturbed to change the image location by a distance of r pixels. A similar rule defines θ_1 in terms of r_1. The purpose of these formulas is to ensure that caustic perturbations change the image location by an amount that is similar to that used for lens perturbations.

Finally, for multi-chain perturbations, we use $\theta_1 = 0.0001$ radians and $\theta_2 = 0.1$ radians. The image resolution cannot be used as a guide here, so the range of useful perturbation values is larger. Note that in our experiments, we have not found the MLT algorithm to be particularly sensitive to any of these values.

11.4.4 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 one way to do this. We initialize the algorithm with n' independent seed paths (Section 11.3), which are mutated in a rotating sequence. At all times, we also store a current lens subpath $\bar{x}_{\rm e}$. A lens subpath mutation consists of deleting the lens subpath of the current path $\bar{x}$, and replacing it with $\bar{x}_{\rm e}$. This happens whenever a lens subpath mutation is selected for the current path (as opposed to a perturbation or bidirectional mutation). After the lens subpath $\bar{x}_{\rm e}$ has been re-used a fixed number of times $n_{\rm e}$, it is discarded and a new one is generated. We chose $n' \gg n_{\rm e}$, to prevent the same lens subpath from being used more than once on the same path.

Each lens subpath $\bar{x}_e$ is generated by casting a ray through a random point on the image plane, and following zero or more specular bounces until a non-specular vertex is found. (At a material with specular and non-specular components, we randomly choose between them.) 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 increment its tally. If that pixel already has its quota of lens subpaths, we search for a non-full pixel using the concept of a *rover* (named after a similar idea in certain memory management schemes). The rover is simply an index into a pseudo-random ordering of the image pixels, such that every pixel appears exactly once.[9] If the randomly chosen pixel from the first step is full, we check the pixel corresponding to the rover, and if necessary we visit additional pixels in pseudo-random order until a non-full one is found. Note that 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 acceptance probability $a(\bar{x} \to \bar{y})$ is computed in a similar way to the bidirectional case, except that the new subpath can be generated in only one way. (Subpath re-use does not influence the calculation.)

11.4.5 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 (Chapter 9). We would like a set of mutations that cover all the possibilities, even though we may not (and

[9]The low-order bits of a linear congruential generator can be used for this purpose.

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.

11.5 Refinements

This section describes a number of general techniques that improve the efficiency of MLT.

Direct lighting. We use standard techniques for direct lighting (e.g. see Shirley et al. [1996]), rather than the Metropolis algorithm. In most cases, these standard methods give better results at lower cost, since the Metropolis samples are not as well-stratified across the image plane (Section 11.4.1). By excluding direct lighting paths from the Metropolis calculation, we can apply more effort to the indirect lighting.

This optimization is easy to implement; it can be done as part of the lens subpath mutation strategy, which already generates a fixed number of subpaths at each pixel. To compute the direct lighting, we perform a standard ray tracing calculation as each lens subpath is generated (independent of the current MLT path). These contributions are accumulated in the same image as the Metropolis samples.[10] We also need to remove the direct lighting paths from the Metropolis portion of the algorithm, but this is easy: when a mutation generates a direct lighting path, we simply reject it. An even better approach is to modify the mutation strategies themselves, in order to avoid generating these paths in the first place.

Finally, note that if the lighting is especially difficult (e.g. due to visibility), then the direct lighting "optimization" may be a disadvantage. For example, imagine a large building with many rooms and lights, but where only one room is visible. Unless the direct lighting strategy does a good job of excluding all the unimportant lights, then MLT can be substantially more efficient.

[10] To do this, we must know in advance how many direct lighting samples there will be at each pixel; adaptive sampling of the image plane is not allowed.

Use of expected values. For each proposed mutation, there is a probability $a(\bar{x} \to \bar{y})$ of accumulating an image sample at $\bar{y}$, and a probability $1 - a(\bar{x} \to \bar{y})$ 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, this optimization replaces a random variable by its expected value (see [Kalos & Whitlock 1986, p. 105]). This is especially useful for sampling the dim regions of the image, which would otherwise receive very few samples. Note that this optimization does not affect the random walk itself; each transition is accepted or rejected in the same way as before.

Two-stage MLT. For images with large brightness variations, the MLT algorithm can spend most of its time sampling the brightest regions. This is undesirable, since it means that brighter pixels are estimated with a higher relative accuracy. Specifically, the variance of pixel j is proportional to I_j, the standard error is proportional to $\sqrt{I_j}$, and the relative error is proportional to $1/\sqrt{I_j}$. As a first approximation, it would be better for the relative errors at all the pixels to be the same (because the human eye is sensitive to contrast differences). To achieve this, we would like an algorithm that generates approximately the same number of samples at every pixel (with a sample value that varies according to the brightness of the ideal image).

The MLT algorithm can easily be modified to approach this goal, by precomputing a test image I_0 at a low sampling density. Then rather than sampling according to the image contribution function f, we sample according to

$$f'(\bar{x}) \;=\; f(\bar{x}) \,/\, I_0(\bar{x})\,, \tag{11.12}$$

where $I_0(\bar{x})$ depends only on the image location of $\bar{x}$. This function f' is used instead of f everywhere in the MLT algorithm, including the computation of the paths weights W_0 during initialization. To compensate for this, each MLT sample value is multiplied by $I_0(\bar{x})$ just before it is accumulated in the image.

The end result is that the MLT sample values are no longer constant across the image; instead, they vary according to the test image I_0. This does not introduce any bias; it simply means that the bright parts of the image are estimated using a smaller number of samples with larger values.[11]

This optimization is mainly useful for images where the range of intensities is very large. Note that the brightest regions of an image are often light sources or directly lit surfaces, in which case handling the direct lighting separately will solve most of the problem.

Importance sampling for mutation probabilities. We describe a technique that can increase the efficiency of MLT substantially, by increasing the average acceptance probability $a(\bar{x} \to \bar{y})$. The idea is to implement a form of importance sampling with respect to $a(\bar{x} \to \bar{y})$ when deciding which mutation to attempt, by weighting each possible mutation according to the probability with which the deleted subpath can be regenerated. (This is the factor $p_{\mathrm{d},2}$ mentioned in Section 11.4.2.)

Let $\bar{x} = \mathbf{x}_0 \ldots \mathbf{x}_k$ be the current path, and consider a mutation that deletes the subpath $\mathbf{x}_l \ldots \mathbf{x}_m$. The insight is that given only the deleted subpath, it is already possible to compute some of the factors in the acceptance probability $a(\bar{x} \to \bar{y})$. In particular, from equation (11.8) we see that $a(\bar{x} \to \bar{y})$ is proportional to

$$Q(\bar{y} \to \bar{x}) \;=\; 1 \,/\, R(\bar{y} \to \bar{x})\,,$$

and from equation (11.10) we see that given only the path $\bar{x}$, it is possible to compute all the components of $Q(\bar{y} \to \bar{x})$ except for the discrete probabilities p_{d} and p_{a}. (These probabilities depend on the path $\bar{y}$, which has not been generated yet). If we simply set these unknown quantities to one, we obtain

$$p_{\mathrm{d},2} \;=\; \sum_{i=1}^{k_{\mathrm{a}}} (1/C_i^{\mathrm{bd}})\,, \tag{11.13}$$

where i refers to the i-th edge of the deleted subpath $\mathbf{x}_l \ldots \mathbf{x}_m$, and C_i^{bd} is the unweighted contribution defined below equation (11.10).

This quantity is proportional to a subset of the factors in the acceptance probability $a(\bar{x} \to \bar{y})$. Thus by weighting the discrete probabilities for each mutation type by this factor, we can avoid mutations that are unlikely to be accepted. With bidirectional mutations,

[11] Note that if not enough samples are used to create the test image, then some pixels will be zero (which is not allowed by the estimate (11.12)). This problem can be solved by filtering the test image before it is used. The simplest approach is to extract the brightest parts of the test image, and weight the other pixels uniformly.

for example, this factor is applied to each of the $O(k^2)$ possibilities for the deleted subpath $\mathbf{x}_l \ldots \mathbf{x}_m$. The computation can be made more efficient by approximating $p_{\mathrm{d},2}$ even further. For example, equation (11.13) can be evaluated for many mutations in parallel by replacing the sum of the $1/C_i^{\mathrm{bd}}$ by their maximum.

11.6 Results

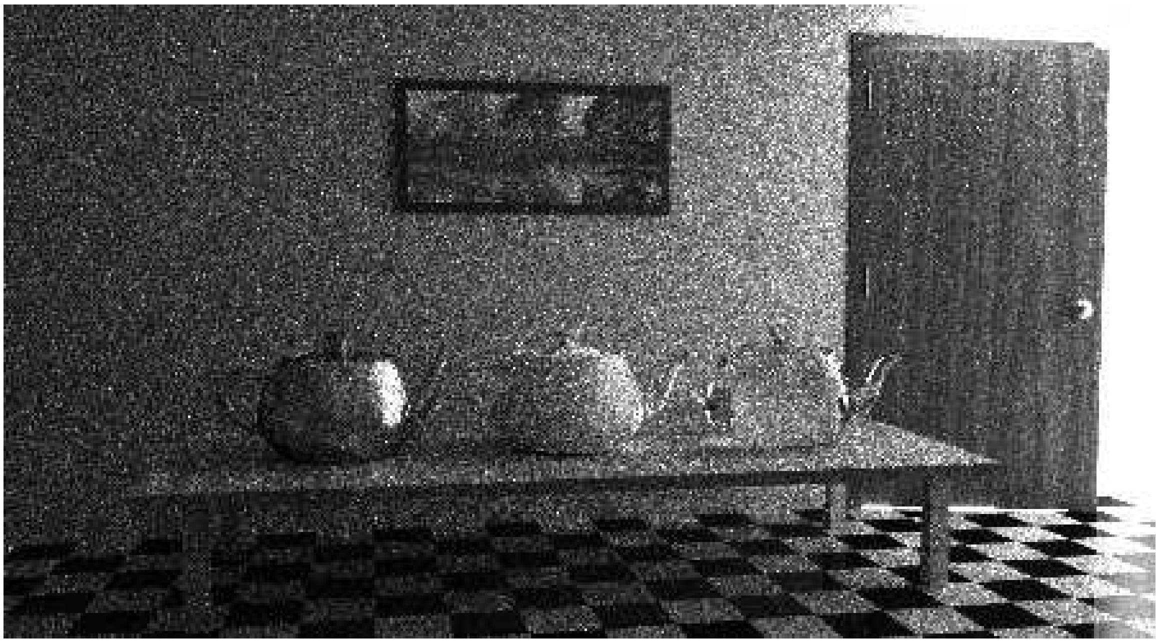
We have rendered test images that compare Metropolis light transport with classical and bidirectional path tracing. Our path tracing implementations support efficient direct lighting calculations, importance-sampled BSDF's, Russian roulette on shadow rays, and several other optimizations.

Figure 11.7 shows a test scene with difficult indirect lighting. All of the light in this scene comes through a slightly open doorway, which lets through about 0.1% of the light in the adjacent room. The light source is a diffuse ceiling panel at the far end of that room (which is quite large), so that most of the light coming through the doorway has already bounced several times.

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 11.8 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



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

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

Figure 11.7: All of the light in this scene comes through a slightly open doorway, which lets through about 0.1% of the light in the adjacent room. The MLT algorithm is able to generate paths efficiently by always preserving a path segment that goes through the small opening between the rooms. The images are 900 by 500 pixels, and include paths up to length 10.

existing paths. The image quality degrades with magnification (for the same computation time), but only slowly. This is due to the fact that the average mutation cost goes up as we zoom into the caustic (since each successful perturbation requires at least four ray-casting operations). Once the caustic fills the entire image, the image quality remains virtually constant.[12]

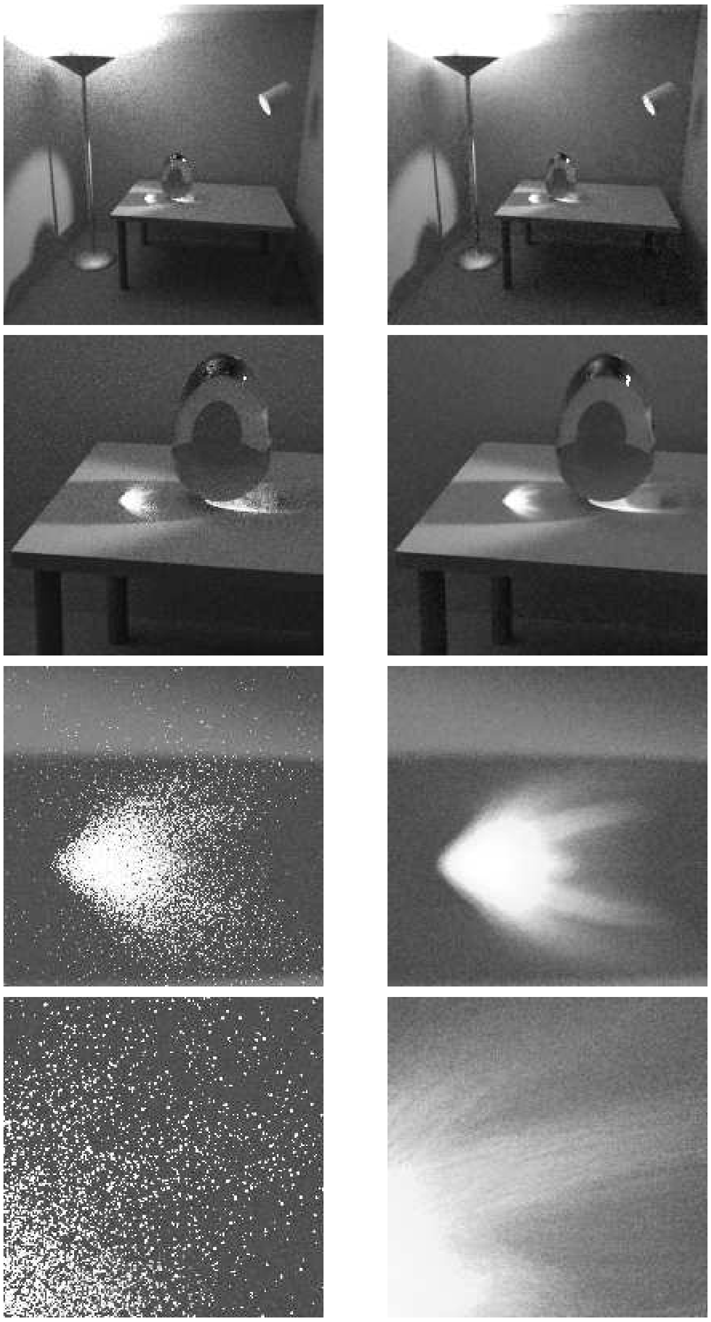
Notice the streaky appearance of the noise at the highest magnification. This is due to caustic perturbations: each ray from the spotlight is perturbed within a narrow cone; however, the lens maps this cone of directions into an elongated shape. The streaks are due to long strings of caustic mutations 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 11.9 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 in this case, because when a path strikes the bottom of the pool, a reflected direction is sampled according to the BRDF. Only a very small number of these paths contribute to the image, because the light source occupies about 1% of the hemisphere of directions above the pool.[13] (Bidirectional path tracing does not help for these paths, because they can be generated only starting from the eye.) As in the previous example, perturbations are the key to sampling these caustics efficiently. However, for this scene it is multi-chain rather than caustic perturbations that are important (recall Figure 11.6). One interesting feature of MLT is that it obtains these results without special handling of the light sources or specular surfaces — see Mitchell & Hanrahan [1992] or Collins [1995] for good examples of what

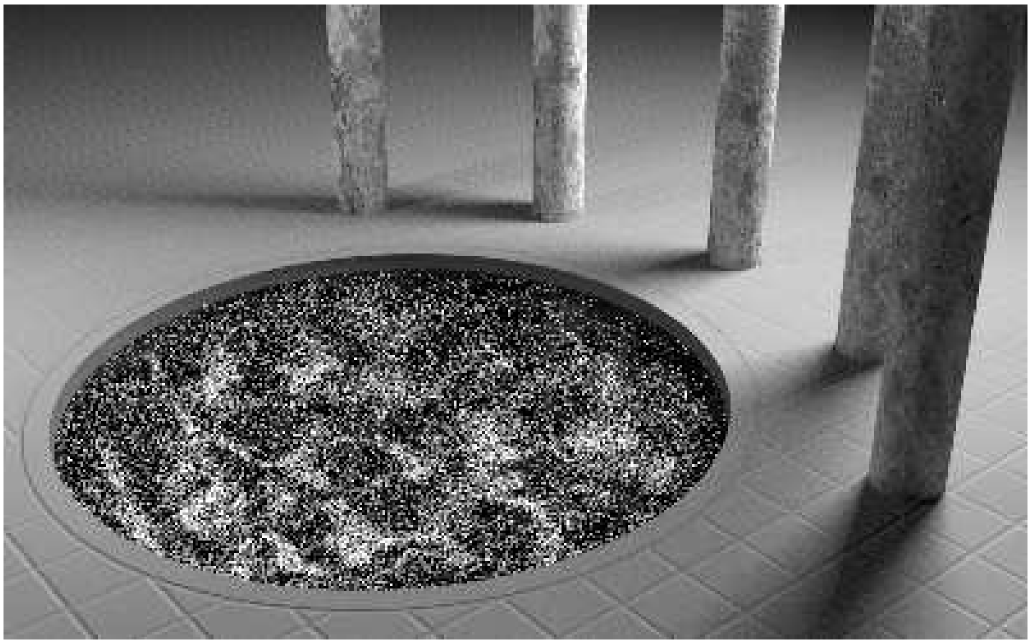
[12]Note that the according to the rules for caustic perturbations described in Section 11.4.3, the average perturbation angle decreases with linearly with the magnification. This implies that the average perturbation size is constant when measured in image pixels.

[13]Note that the brightness of the caustic is proportional to the solid angle occupied by the light source, as seen from the bottom of the pool. Thus in regions where the caustics are dim, the chance of a ray hitting the light source is actually much less than one percent.

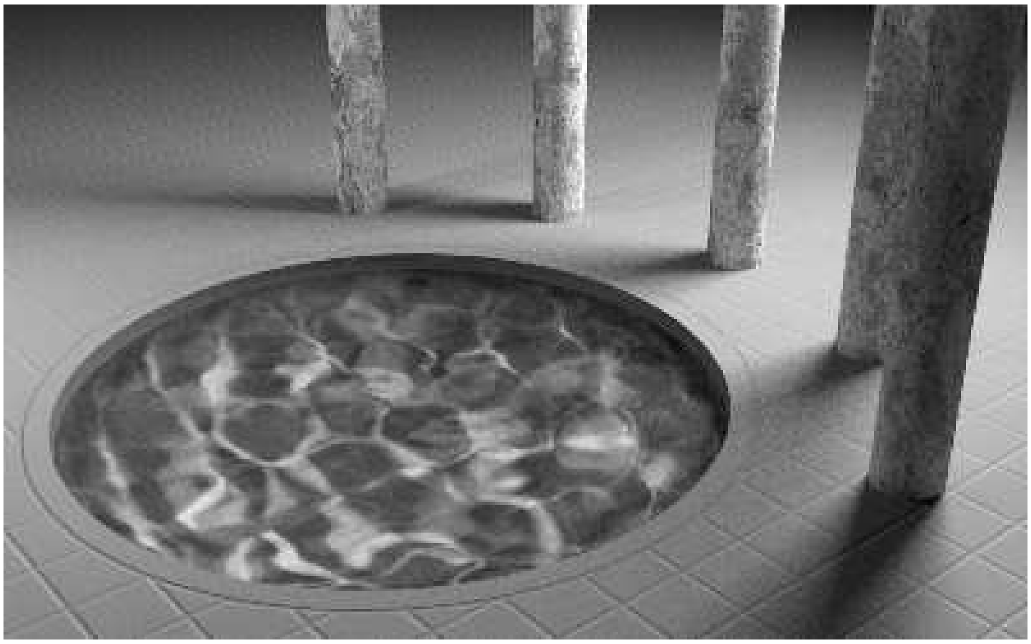


(a) (b)

Figure 11.8: These images show caustics formed by a spotlight shining on a glass egg. Column (a) was computed using bidirectional path tracing with 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 paths up to length 7, and the images are 200 by 200 pixels.



(a) Path tracing with 210 samples per pixel.



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

Figure 11.9: 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 samples these paths efficiently by means of perturbations. The images are 800 by 500 pixels.

Test Case	PT vs. MLT			BPT vs. MLT		
	l_1	l_2	l_∞	l_1	l_2	l_∞
Figure 11.7 (door)	7.7	11.7	40.0	5.2	4.9	13.2
Figure 11.8 (egg, top image)	2.4	4.8	21.4	0.9	2.1	13.7
Figure 11.9 (pool)	3.2	4.7	5.0	4.2	6.5	6.1

Table 11.1: This table shows numerical error measurements for path tracing (PT) and bidirectional path tracing (BPT) relative to Metropolis light transport (MLT), for the same computation time. The entries in the table were determined as follows. For each test image, we computed the relative error $e_j = (\tilde{I}_j - I_j)/I_j$ at each pixel, where $\tilde{I}_j$ corresponds to the algorithm being measured, and I_j is the value from a reference solution. Next, we computed the l_1, l_2, and l_∞ norms of the resulting array of errors e_j. Finally, we divided the error norms for path tracing and bidirectional path tracing by the corresponding error norm for MLT, to obtain the normalized results shown in the table above. Note that the gain in efficiency of MLT over the other algorithms is proportional to the square of the table entries.

can be achieved if this restriction is lifted.

We have also made numerical measurements in order to compare the performance of the various algorithms on each test scene. To do this, we first computed images using path tracing (PT), bidirectional path tracing (BPT), and Metropolis light transport (MLT), with the same computation time in each case. Next, we computed the relative error $e_j = (\tilde{I}_j - I_j)/I_j$ at each pixel, where $\tilde{I}_j$ corresponds to the algorithm being measured, and I_j is the value from a reference solution (created using bidirectional path tracing with a large number of samples, at a lower image resolution). We then computed the l_1, l_2, and l_∞ norms of the resulting array of errors e_j, and divided the error norms for PT and BPT by the corresponding error norm for MLT. This yielded the results shown in Table 11.1.

Note that the efficiency gain of MLT over the other methods is proportional to the *square* of the table entries, since the error obtained using path tracing and bidirectional path tracing decreases according to the square root of the number of samples. For example, the RMS relative error in the three-teapots image of Figure 11.7(a) is 4.9 times higher than in Figure 11.7(b), which implies that approximately 25 times more bidirectional path tracing samples would be required to achieve the same error levels as MLT. Even in the topmost images of Figure 11.8 (for which bidirectional path tracing is well-suited), notice that the results of

MLT are competitive.

For comparison, we consider the techniques proposed by Jensen [1995] and Lafortune & Willems [1995a] for sampling difficult paths more efficiently. Basically, their idea is to build an approximate representation of the radiance in a scene, and use it to modify the directional sampling of the basic path tracing algorithm. The radiance information can be collected either with a particle tracing prepass [Jensen 1995], or by adaptively recording it in a spatial subdivision as the algorithm proceeds [Lafortune & Willems 1995a]. However, these techniques have several problems, including insufficient directional resolution to be able to sample concentrated indirect lighting efficiently, and substantial space and time requirements. In any case, the best variance reductions that have been reported are in the range of 50% to 70% (relative to standard path tracing), as opposed to the reductions of 96% to 99% reported in Table 11.1. (Similar ideas have also been applied to particle tracing algorithms [Pattanaik & Mudur 1995, Dutre & Willems 1995], with similar results.)

In our tests, the computation times were approximately 4 hours for the each image in Figure 11.7 (the door ajar), 15 minutes for the images in Figure 11.8 (the glass egg), and 2.5 hours for the images in Figure 11.9 (the pool), where all times were 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 11.4.4 is used). For high-resolution images, memory usage could be reduced further by collecting the samples in batches, sorting them in scanline order, and applying them to an image on disk.

11.7 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 needs little memory, and always computes an unbiased result.

The MLT algorithm offers interesting new possibilities for adaptive sampling without bias, since the mutation strategy is allowed to depend on the current path. For example, consider the strategy of replacing the light source vertex $\mathbf{x}_0$ with a new randomly sampled position on the same light source. This is potentially a simple, effective strategy for handling scenes with many lights: once an important light source is found, the MLT algorithm can efficiently generate many samples from it. (More generally, mutations could be proposed to nearby light sources by constructing a spatial subdivision.) This is clearly a form of adaptive sampling, since more samples are taken in regions nearby existing good samples. Unlike with standard Monte Carlo algorithms, however, no bias is introduced.

This also raises interesting possibilities for handling specular surfaces. For example, we could try a strategy similar to that above: when mutating a subpath containing a specular vertex, generate a new vertex on the same specular object. If only a small fraction of the specular surfaces in the scene made a large contribution to the image, this would provide a means of sampling them efficiently. Note that this technique is more powerful than simply flagging specular surfaces for extra sampling, since we do not need to assign an *a priori* probability to the sampling of each surface. This is important when a large number of specular surfaces are present, since in the MLT case the sampling efficiency is not affected once an important surface has been found.

The MLT framework could also be an advantage for techniques that generate specular vertices deterministically. In particular, recall the idea of generating a chain of specular vertices connecting two given points (as mentioned in Section 8.3.4). A simple example is that given two points $\mathbf{x}_1$ and $\mathbf{x}_3$ and a planar mirror, we might calculate the point $\mathbf{x}_2$ on the mirror that reflects light between them. (Note that it is also possible to handle non-planar surfaces, or sequences of such surfaces, using techniques described by Mitchell & Hanrahan [1992].) However, these analytic techniques have problems when there are many specular surfaces, since each possible surface and sequence of surfaces must be checked separately for a solution.

The MLT framework helps to solve the combinatorial aspect of this problem. Once an important specular chain is found, a new chain could be generated by simply perturbing one of its endpoints, and then regenerating the intermediate vertices using the same sequence of specular surfaces. For example, this could be used to efficiently sample caustics seen indirectly through specular reflectors, even a point light source is used, and when there are possibly many specular surfaces in the scene. On the other hand, recall that we cannot hope to solve all such problems efficiently, since provably difficult configurations of mirrors do exist [Reif et al. 1994].

The MLT algorithm can also be extended in other ways. For example, with modest changes we could use it to compute view-independent radiance solutions, by letting the I_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 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. Genetic algorithms may be useful in this regard, to optimize the parameter settings on a suite of test images. We hope to address some of these refinements and extensions in the future.

Appendix 11.A Proof of Unbiased Initialization

In this appendix, we show that the estimate

$$I_j \;=\; E\left[\frac{1}{N}\sum_{i=1}^{N} W_i\, h_j(\bar{X}_i)\right]$$

is unbiased (see Section 11.3.1). To do this, we show that the following *weighted equilibrium condition* is satisfied at each step of the random walk:

$$\int_{\mathbb{R}} w\, p_i(w, \bar{x})\, dw \;=\; f(\bar{x})\,, \tag{11.14}$$

where p_i is the joint density function of the i-th weighted sample $(W_i, \bar{X}_i)$. This is a sufficient condition for the above estimate to be unbiased, since

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

To show that the weighted equilibrium condition holds for all samples $(W_i, \bar{X}_i)$, we proceed by induction. For $i = 0$, we have

$$p_0(w, \bar{x}) \;=\; \delta(w - \frac{f(\bar{x})}{p_0(\bar{x})})\, p_0(\bar{x})\,,$$

where $\delta(w - w_0)$ is a Dirac distribution, corresponding to the fact that W_0 is chosen as a deterministic function of $\bar{X}_0$ rather than by random sampling. It is easy to verify that p_0 satisfies condition (11.14).

Next we verify that the Metropolis algorithm preserves the weighted equilibrium condition from one sample to the next. Since the mutations set $W_i = W_{i-1}$, the first part of equation (11.4) is still true when $p_j(\bar{x})$ is replaced by $p_j(w, \bar{x})$:

$$\begin{aligned} p_i(w, \bar{x}) \;&=\; p_{i-1}(w, \bar{x}) + \int_\Omega \Big\{ p_{i-1}(w, \bar{x})\, T(\bar{x} \to \bar{y})\, a(\bar{x} \to \bar{y}) \\ &\qquad - p_{i-1}(w, \bar{y})\, T(\bar{y} \to \bar{x})\, a(\bar{y} \to \bar{x}) \Big\}\, d\mu(\bar{y})\,. \end{aligned}$$

Multiplying both sides by w and integrating, we obtain

$$\begin{aligned} \textstyle\int_{\mathbb{R}} w\, p_i(w, \bar{x})\, dw \;&=\; \textstyle\int_{\mathbb{R}} w\, p_{i-1}(w, \bar{x})\, dw + \displaystyle\int_\Omega \Big\{ [\textstyle\int_{\mathbb{R}} w\, p_{i-1}(w, \bar{x})\, dw]\; T(\bar{x} \to \bar{y})\, a(\bar{x} \to \bar{y}) \\ &\qquad - [\textstyle\int_{\mathbb{R}} w\, p_{i-1}(w, \bar{y})\, dw]\; T(\bar{y} \to \bar{x})\, a(\bar{y} \to \bar{x}) \Big\}\, d\mu(\bar{y}) \end{aligned}$$

$$\begin{aligned}
&= \int_{\mathbb{R}} w \, p_{i-1}(w, \bar{x}) \, dw \;+\; \int_{\Omega} \Big\{ f(\bar{x}) \, T(\bar{x} \to \bar{y}) \, a(\bar{x} \to \bar{y}) \\
&\qquad\qquad - \, f(\bar{y}) \, T(\bar{y} \to \bar{x}) \, a(\bar{y} \to \bar{x}) \Big\} \, d\mu(\bar{y}) \\
&= \int_{\mathbb{R}} w \, p_{i-1}(w, \bar{x}) \, dw \\
&= f(\bar{x}) \,,
\end{aligned}$$

where we have used the detailed balance condition (11.3). Thus every mutation step preserves the weighted equilibrium condition (11.14). ■

It is interesting to note that even though the random walk is always in weighted equilibrium, the distributions of paths and weights change at each step. In particular, the path distribution is initially given by some arbitrary density function $p_0(\bar{x})$, and converges toward the stationary distribution $p^*(\bar{x})$. Similarly, the weight distribution $p_i(w \mid \bar{x})$ at a given point $\bar{x}$ starts out as a Dirac distribution

$$p_0(w \,|\, \bar{x}) \;=\; \delta(w - \frac{f(\bar{x})}{p_0(\bar{x})}) \,,$$

and gradually evolves toward an equilibrium $p^*(w \mid \bar{x})$. Furthermore this equilibrium does not depend on $\bar{x}$, since

$$p^*(w \,|\, \bar{x}) \, p^*(\bar{x}) \;\equiv\; p^*(\bar{x} \,|\, w) \, p^*(w) \,,$$

and the density functions $p^*(\bar{x} \,|\, w)$ and $p^*(\bar{x})$ are equal (i.e. the paths at each weight evolve toward the same equilibrium, since the transition rules do not depend on weight). Thus we have

$$p^*(w \,|\, \bar{x}) \;=\; p^*(w) \;=\; p_0(w) \,,$$

observing that the marginal weight density $p_0(w)$ does not change with time (recall that $W_i = W_{i-1}$).

The net effect is that the path and weight distributions may start far from equilibrium, and gradually converge toward it. However, this is done in such a way that the weighted equilibrium condition (11.14) is initially satisfied, and preserved at every step. Thus we can obtain unbiased results immediately, rather than waiting for the path and weight distributions to converge separately.