Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
12.4.2004
Tik-111.500 Seminar on computer graphics
Spring 2004: Image-based methods

Metropolis Light Transport

Petri Häkkinen
46561N

Metropolis Light Transport

Petri Häkkinen
HUT, Telecommunications Software and Multimedia Laboratory

phakkine@cc.hut.fi

Abstract

Metropolis Light Transport is an unbiased and robust Monte Carlo algorithm for solving global illumination problems. The algorithm works in two phases. In the first phase, a bidirectional path tracer is used to generate an initial population of light transport paths. In the second phase, each initial path is mutated in turn and each mutation is either accepted or rejected with a carefully chosen probability. The contributions of resulting paths are then recorded on the image plane.

1 INTRODUCTION

To synthesize a realistic image on a computer system is to solve the light transport problem. One way to solve the problem is to sample light transport paths originating from light sources and ending at the lens of a virtual camera. Path tracing algorithm (Kajiya, 1986) does just this and can produce realistic and unbiased images, that usually suffer from noise but converge to the correct result. There are certain situations where the path tracing algorithm is inefficient. Scenes containing strong indirect lighting, caustics and small geometric holes are especially hard to render with it. In contrast, the Metropolis Light Transport (MLT) algorithm (Veach & Guibas, 1997) excels in these kind of hard lighting situations.

The key idea of MLT is to reuse light transport paths that make contribution to the final image. These paths are generated by mutating existing paths. A mutation could move vertices on a path, delete a part of the path or add new vertices. Once an important path is found, the path space is explored locally around it. There are a number of different mutation strategies, each of which are designed to help in specific lighting situations. Probabilities play a key role in MLT. Each mutation has a carefully chosen acceptance probability, so that mutations are either accepted or rejected based on the relative contribution to the final image.

In section 2, we describe related work from classical ray tracing to bidirectional ray tracing and photon mapping. Section 3 gives an overview of the basic Metropolis

Light Transport algorithm and various important details of the algorithm can be found in the subsections. Details of the mutation strategies are described in section 4. Finally, results and conclusions are presented in sections 5 and 6.

2 RELATED WORK

Classical ray tracing (Whitted, 1980) follows light rays originating from the camera towards the scene. The intersection of rays with the nearest surface points are calculated and illumination at those points are evaluated. If a surface is a diffuse one, illumination is calculated by evaluating the incoming radiance from point light sources in the scene. For each light source a shadow ray is cast from the point of intersection towards the light. If the shadow ray is not blocked by any surface, direct illumination is calculated and accumulated. Otherwise the intersection point is considered to be in shadow for that particular light. Perfectly specular surfaces are handled by splitting the ray into reflected and transmitted parts and recursively calling the ray tracing procedure. Classical ray tracing does not take all types of light transport paths into account and so full global illumination cannot be calculated with it.

The basic ray tracing algorithm has been later extended to handle antialiasing, soft shadows, blurred reflections, motion blur and depth of field. An extended version of the ray tracing algorithm, *distributed ray tracing* (Cook *et al.* , 1984), uses Monte Carlo techniques for integration. Distributed ray tracing uses multiple rays per pixel and distributes them over the pixel area, camera lens and time. Because rays are distributed randomly over the integration domain, images produced by the algorithm are mostly free of aliasing but suffer from noise – a general feature of Monte Carlo integration. Distributed ray tracing supports area light sources by placing random sample points on light source surfaces. Blurred reflections are produced by jittering the reflected ray direction. While clearly better than classical ray tracing, distributed ray tracing does not calculate global illumination.

Path tracing (Kajiya, 1986) was the first Monte Carlo ray tracing method to support all kinds of light transport paths and that can produce unbiased renderings of scenes, i.e. images that are correct on the average. The path tracing algorithm solves the rendering equation in a special *path integral form*. The path integral form makes it possible to estimate the multidimensional integral over the space of light transport paths by randomly walking in the path space. The random walk, starting at the lens and hopping from surface to the next until a light source is reached, is a form of *Markov chain* (Peskun, 1973). The path tracing algorithm can render images that are photorealistic, but like images rendered using distributed ray tracing, they suffer from noise. The noise is due to sample variance. The amount of noise depends on the number of samples used, so that using infinitely many samples would result in totally noiseless image. A number of variance reduction schemes have been proposed for path tracing, including stratification, use of expected sample values, and importance sampling. In importance sampling, knowledge about the scene is used to direct samples to regions

of the path space where it matters most.

Generally it is difficult to design a single sampling technique that works well in all cases. This was the motivation for *multiple importance sampling* (Veach, 1997) where several sampling techniques are combined by weighting them carefully so that the strengths of each are preserved. Multiple importance sampling can be used to create robust and generally low-variance estimators for Monte Carlo integration.

In *bidirectional path tracing*, developed independently by Veach and Guibas (Veach & Guibas, 1996) and Lafortune and Willems (Lafortune & Willems, 1993), pairs of paths are generated starting from the light and from the lens. Each pair of paths are joined together in the middle by considering all possible ways that light can flow between the path vertices. Essentially, by varying the number of vertices on both ends bidirectional path tracing obtains different sampling techniques. These sampling techniques are weighted and combined using multiple importance sampling.

Photon mapping (Jensen, 2001) is a two pass algorithm for speeding up hard indirect lighting situations such as caustics. In the first pass many individual photons are traced from the light sources towards the surfaces. Photon hits are stored in a data structure called the photon map that is optimized for photon density queries. The second pass of the algorithm casts rays from the camera towards the surfaces and estimates the irradiance at those locations using the photon map. Photon mapping is typically used in combination with a path tracer where the path tracer is responsible for calculating the direct lighting and the photon map is used for caustics and indirect lighting.

3 OVERVIEW OF MLT

The basic MLT algorithm is sketched below.

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

Given an initial path $\bar{x}$, we choose a random mutation strategy and use it to mutate the path. The mutated path is accepted or rejected based on an acceptance probability that is carefully calculated from the current path and the new proposed path. If the mutation is accepted, the new path replaces the current path. If the mutation is rejected, the current path is not changed. In any case, path's contribution to the final image is recorded by updating the pixels whose filter kernel lie on the region of the path. In the

basic algorithm each path sample has the same weight so that intensity differences in the final image are due to different number of samples received by pixels.

Mutations can be almost arbitrary – and this is the power of MLT – because acceptance probabilities ensure that mutated paths are distributed according the contribution they make to the final image. The only requirement for a mutation strategy is that all possible light transport paths in the scene must have a non-zero probability of being explored. If this was not so, the resulting image would be biased because some transport paths would be ignored. This feature of MLT makes it possible to easily extend the palette of mutation strategies and even allows designing strategies that are good in some rare special cases. In this sense MLT is a form of multiple important sampling (Veach, 1997).

A simple way to enhance the basic algorithm is to use expected sample values, which is a standard Monte Carlo variance reduction technique. The idea is to make the rejected mutations help in improving the quality of the integral estimate. If we weight the mutated path $\bar{y}$ by its acceptance probability $a(\bar{y}|\bar{x})$ and the original path $\bar{x}$ by $1 - a(\bar{y}|\bar{x})$, we can record the contributions of both paths to the image without introducing bias. The modified version of the algorithm is listed below.

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

In the subsequent sections we will describe the details of the algorithm.

3.1 Acceptance probabilities

The acceptance probability for a mutation from path $\bar{x}$ to path $\bar{y}$ is defined as follows:

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

Where f is the image contribution function and T is the tentative transition function.

The image contribution function $f(\bar{x})$ is proportional to the relative contribution of path $\bar{x}$ and is therefore non-zero only for paths that start at a light source, intersect the image plane and end at the lens. Notice that if $f(\bar{y}) = 0$ then $a(\bar{y}|\bar{x}) = 0$, so mutated paths with zero contribution are always rejected.

The value of $f(\bar{x})$ is the product of various geometric and scattering terms along the path as illustrated in figure 1. The image contribution function for the path in the figure is calculated as follows:

$$f(\bar{x}) = L_e(x_0 \rightarrow x_1)\, G(x_0 \leftrightarrow x_1)\, f_r(x_0 \leftrightarrow x_1 \leftrightarrow x_2)$$
$$G(x_1 \leftrightarrow x_2)\, f_r(x_1 \leftrightarrow x_2 \leftrightarrow x_3)\, G(x_2 \leftrightarrow x_3)$$

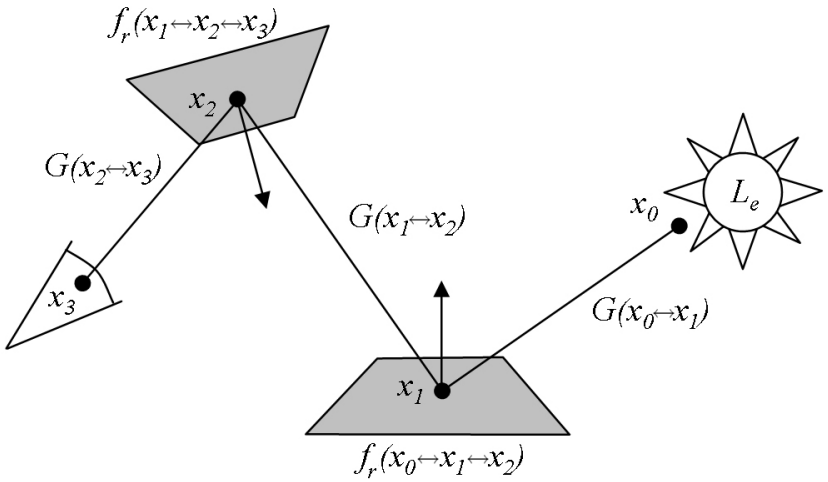


Figure 1: Calculating the contribution of a path of length 3.

$L_e(x_0 \rightarrow x_1)$ is the radiance emitted by the light source towards x_1 and the geometry term G is calculated exactly like in path tracing (Kajiya, 1986):

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

Where V, the visibility function, is one if $\mathbf{x}'$ is visible from $\mathbf{x}$ (and vice versa) and zero otherwise. θ_o and θ_i are the angles between surface normals and the vector $\mathbf{x} - \mathbf{x}'$ (figure 2).

Scattering terms f_r in figure 1 are calculated from the bidirectional reflectance distribution function (BRDF) and are dependent on the incident and outgoing directions, wavelength and material properties. For an ideal diffuse surface the BRDF is a constant:

$$f_r(\mathbf{x} \leftrightarrow \mathbf{x}' \leftrightarrow \mathbf{x}'') = \frac{\rho_d}{\pi}$$

Where ρ_d represents the diffuse reflectance and varies from 0 to 1.

The tentative transition function $T(\bar{y}|\bar{x})$ is a probability density function for mutations from path $\bar{x}$ to path $\bar{y}$. Because the probability for different mutations are

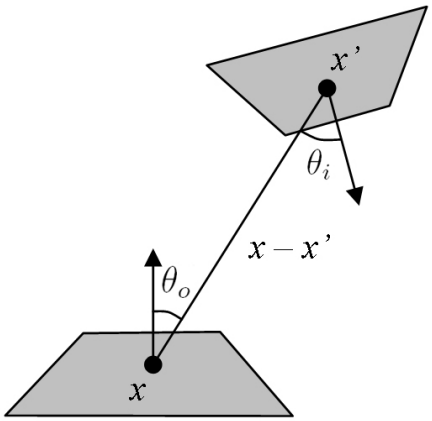


Figure 2: The geometry function G.

explicitly controlled by an implementation of MLT algorithm, T is known, and so the acceptance probability function can be calculated for any pair of paths $\bar{x}$ and $\bar{y}$.

For example, suppose that we have mutated a path by deleting one edge (x_a, x_b) in the current path at random and replaced it with a new vertex x_t and two edges (x_a, x_t) and (x_t, x_b). The new vertex x_t is generated by casting a ray in random direction from x_a and finding the intersection point with the nearest surface. To calculate $T(\bar{y}|\bar{x})$ we have to consider all possible ways that the edge to be deleted can be chosen. This gives us probability p_d. Then we have to consider all possible ways that the new vertex can be generated. We get p_a, the probability of sampling the direction from x_a to x_t. The tentative transition function $T(\bar{y}|\bar{x})$ is then the product of p_d and p_a.

3.2 Initialization

The choice of initial path influences the whole Metropolis sampling process. For example, if the initial path has a zero contribution to the image, mutation strategies are unable to find a more influential path (notice that $a(\bar{y}|\bar{x})$ is not defined in this case).

Recall that the basic MLT algorithm from section 3 records equally weighted samples to the image. This works because the paths are supposed to be distributed according to the image contribution function $f(\bar{x})$. But paths are guaranteed to have this desired distribution, the *equilibrium distribution*, only in the limit, after infinitely many Metropolis iterations. Clearly, to get unbiased results something must be done to correct the error. This problem is known as *start-up bias*. We discuss two solutions to it.

The first solution, similar to typical Metropolis applications, is to choose any initial path with a non-zero contribution and run the Metropolis algorithm on it without recording any samples. After some number k iterations the path is approximately distributed according to $f(\bar{x})$. The algorithm is then restarted and this time samples are

recorded. There are several problems to this solution. Computation time is wasted on calculating things that are essentially discarded. Also, it is generally hard to determine when to stop the initial iteration and restart. A major problem is that the result is unavoidably biased because the ramdom walk has only reached the equilibrium distribution approximately.

The second solution is more suitable for MLT. We start with an initial path picked from a distribution p_0. Because the initial path is not in equilibrium distribution, we compensate this by weighting the path with a factor $W = f(\bar{x}) \,/\, p_0(\bar{x})$. The weight is applied to samples that are recorded to the image and remains the same when the path is mutated.

However, there is a problem to this approach too. Consider that the weight for the initial path is essentially a random variable. Using only one initial path with a possibly low weight needs a lot of iterations to get a good image. A large weight would increase variance. The problem is that the choise of the initial path influences the Metropolis phase very much. This can be fixed by running several copies of the algorithm at the same time, each one with a different initial path but recording samples to a shared image. But varying weights can unnecessarily increase variance too. A better solution is to generate a larger set of paths and resample them to get a smaller number of *equally weighted* paths as seeds for Metropolis iterations. The resampling process picks paths according to the probability proportional to path weights W_i and assigns the same weight W' for all of them. The shared weight W' is calculated as follows:

$$W' = \frac{1}{n} \sum_{i=1}^{n} W_i$$

It is often reasonable to select just one representative sample from a larger set of paths (Veach, 1997). In this approach, the initial path generation phase estimates the total power falling on the image.

With start-up bias eliminated, any algorithm that can produce unobstructed paths from light sources towards the eye could be used to build the initial path population. In practice, MLT implementations use bidirectional path tracing.

3.3 Choosing a mutation to use

Given a set of mutation strategies, how do we choose a mutation to use for each path? There is no definite answer to this question because mutation strategies and path configurations can be almost anything imaginable. Veach and Guibas (1997) propose the use of importance sampling for mutation probabilities. The idea is to estimate the acceptance probability for each mutation before they are actually tried. The probabilities for each mutation strategy is then weighted by a factor based on the estimated acceptance probability. Veach and Guibas (1997) report that this kind of sampling substantially improves the efficiency of the basic algorithm.

Because knowing the exact acceptance probabilities a priori is not possible, a simple solution of just giving the different mutation strategies nearly equal probabilities is valid in many cases. This may increase the number of rejections but is applicable to a wide range of lighting situations.

3.4 Convergence testing

How do we know when to terminate the algorithm, i.e. how large N should be in the basic algorithm listed in section 3? A simple solution would be to select a fixed N. This way the computation time of the algorithm can be controlled. A better alternative would be to interactively display the image as it is being rendered and let the user terminate the algorithm when the quality is good enough.

If user interaction is not desired, there is a way to automate this process. Running several copies of the algorithm in parallel enables us to calculate sample variances. The sample variance σ^2 for a set of n samples is calculated as follows:

$$\mu = \sum_{i=1}^{n} \frac{x_i}{n}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n-1}$$

If the sample variance for any pixel in the image (or perhaps the average sample variation for the entire image) is lower than a user set limit, the algorithm is terminated. Unfortunately this approach may not be entirely intuitive to the user. It is not clear how the sample variance correlates to the actual perceived error.

4 MUTATION STRATEGIES

Designing a good set of mutations for the MLT algorithm is crucial for generating noise-free images. The main problem is to avoid clumping of samples in small regions of the path space and at the same time exploit the coherence by reusing a large part of the existing path. In other words, mutations must be of sufficient size, but not too large either, because acceptance probabilities of large mutations are likely to be low.

4.1 Properties of a good mutation strategy

A set of mutation strategies should contain mutations that have many of the following properties (Veach & Guibas, 1997):

- **High acceptance probability.** Many rejected mutations cause clumping of paths, which lead to higher variance. Computation time is also wasted by repeatedly evaluating and mutating the same path.

- **Large changes to the path.** Like low acceptance probabilities, small changes can cause clumping. Therefore, it is occasionally desirable to make large changes to the path. A large change could be an addition or deletion of one or more path vertices.

- **Ergodicity.** To produce unbiased renderings, it is important that all possible paths have some probability $p > 0$ of being explored, even if their contribution to the image is relatively small. The MLT algorithm must eventually give up on some local part of the path space and move on to explore other regions. If this property is not fulfilled, it is possible that some important paths are missed because some other paths look more promising at first glance.

- **Changes to the image location.** In order to render a complete image, we need to process all pixels of the image. Therefore mutations should change the location of the path on the image plane so that each pixel is processed and preferable do it so that each pixel get a number of samples proportional to the complexity of the light transport integral at that point. Furthermore, proper antialiasing requires that samples on each pixel are distributed in a manner that sample points are neither clumped together nor there are big gaps between samples.

- **Stratification.** Stratification is an important and simple variance reduction method, where the integration domain is split to smaller subdomain, each subdomain receiving a number of samples relative to the size of the subdomain.

- **Low cost.** Most Monte Carlo ray tracers spend their time largely in ray intersection calculation, so a preferable property of a mutation strategy is to use as few ray tracing operations as possible.

4.2 Bidirectional mutations

Bidirectional mutations choose a subpath of the current path and replace it with a new subpath, thus making possibly large scale changes to the path.

First, the mutation strategy must decide which subpath to delete (figure 3). Each possible subpath in the current path is considered and a probability of deletion for the subpath is calculated. Careful choice of probabilities is irrelevant for getting an unbiased result, because the Metropolis framework ensures that the process converges properly. The efficiency of bidirectional mutations can be greatly enhanced by importance sampling with respect to the acceptance probability. For the non-importance sampled version, the following probabilities of deletion p_d are reasonably good (Veach & Guibas, 1997):

$$p_d[1] = 0.25,\ p_d[2] = 0.5 \text{ and } p_d[l] = 2^{-l} \text{ for } l >= 3$$

Where $p_d[l]$ is the probability of deleting a subpath of length l. The probabilities do not sum to 1 so they have to be normalized before they can be used for sampling. These

values have been selected so that deletion of a small number of edges is preferred over a larger change. Small changes ensure that the acceptance probability of the mutated path will be high. Notice that it is possible that the whole path will be deleted, so that the ergodicity property is fulfilled and MLT can never get stuck in a local region of the path space.

After deletion the number of vertices to add is chosen. Like in the previous step the probability distribution does not affect the correctness of the image. However, to get a high acceptance probability for the mutation it is generally better to favor keeping the new path length close to the original length. The following probabilities provide reasonable results (Veach & Guibas, 1997): $p = 0.5$ for keeping the old path length, $p = 0.15$ for increasing it by 1 and $p = 0.15$ for decreasing it by 1. The remaining probability (0.2) is assigned to other suitable lengths as desired.

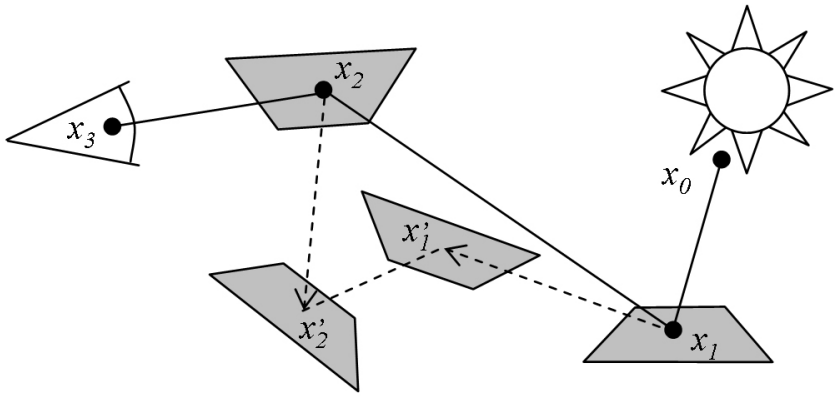


Figure 3: A path of length 3 is mutated to a path of length 5. Edge (x_1, x_2) is selected for deletion. Vertices x'_1 and x'_2 are obtained by casting a random ray at x_1 and x_2. The new path is completed by casting a ray from x'_1 to x'_2 to verify that light flow is not obstructed.

At this point the current path has been split to two halves and the new path length is known. Two end points are taken and a subpath for each of them is built by adding a proper amount of vertices one by one until the desired total path length is reached. The vertices are distributed using uniform random distribution to both halves. For each new vertex a ray is cast from the current subpath end point. The direction of the ray is distributed according to the bidirectional reflectance distribution function (BRDF). If the ray hits a surface then the hit location becomes the position of the new vertex. If the ray does not hit anything, the mutated path is rejected because it does not contribute to the image. It is possible that the deletion step removed the first or last vertex of the path, so that one or both halves may not contain any vertices. In this case the missing vertex is simply regenerated by selecting a random point on the lens or light source.

Finally, the two halves need to be connected. A single ray is cast between the end points of the two halves and if the ray is not blocked by any surface, the mutation is

complete. If the ray hits a surface, then no light is transported over the path and the mutation is rejected.

4.3 Perturbations

While the bidirectional mutation strategy provides large scale changes to the path, it has problems with caustics, small geometric details and generally high frequency features of the scene. The problem is that coarse changes in these cases leads to higher rejection rates. Therefore, the MLT algorithm has perturbation mutations that have been designed to handle these special cases. The idea of perturbations is to make very small changes the path by moving the path vertices a little. Here we concentrate on two types of perturbations: lens perturbations and caustics perturbations.

4.3.1 Lens perturbations

The motivation of lens perturbations is to efficiently sample small geometric details and phenomena that are concentrated on a small region that the camera can directly see. Lens perturbation directly affects the lens edge by displacing it with a random vector $\mathbf{R}$ (figure 4). The direction of $\mathbf{R}$ is distributed uniformly on the image plane. The magnitude of $\mathbf{R}$ is distributed exponentially between minimum displacement amount r_1 and maximum displacement amount r_2 using the following equation:

$$||\mathbf{R}|| = r_2 \, \exp(-\ln(r_2 \,/\, r_1) * U)$$

Where U is a uniform random number in the range $[0, 1]$. Sufficient values for the parameters are minimum displacement distance of 10% pixel size and maximum displacement region of 5% of the total image area (Veach & Guibas, 1997).

Once the lens edge has been displaced, a ray is cast from the lens towards the scene. If the ray does not hit anything or misses the image plane, the mutation is rejected. Otherwise, the first surface hit replaces a vertex on the path (x_2 in figure 4). One more ray cast operation is required to verify that the new path contributes to the image. If the path segment from x_2' to x_1 in figure 4 is obstructed, no light can flow between the vertices and the mutation is rejected. This technique works for all paths with the suffix $(L|D)D^*E$ in Heckbert's regular expression notation (Heckbert, 1990) where L,D,S and E stand for light, diffuse, specular and eye vertex, respectively.

To handle specular surfaces and paths with the suffix $(L|D)DS^*E$, we start at the lens and cast rays by bouncing from specular surface to the next until we eventually hit a diffuse surface. The number of specular bounces and scattering modes (transmission or reflection) must match at each vertex to the original path. If a non-specular surface is hit when a specular surface is expected, the mutation is rejected.

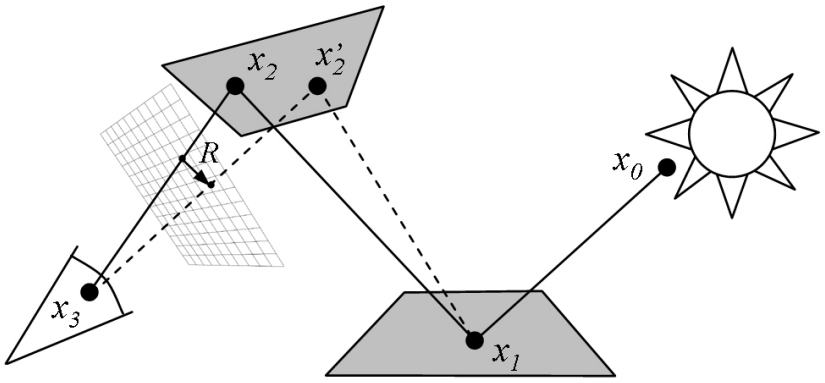


Figure 4: A path of the form $LDDE$ goes through a lens perturbation mutation that moves vertex x_2 to new position x_2'. The dashed line shows the mutated path.

4.3.2 Caustics perturbations

Caustics perturbations work on paths with the suffix $(D|L)S^+DE$. The strategy is similar to lens perturbations but here we cast rays starting from the light source or diffuse surface that transmits radiance towards the first specular bounce in the chain. The direction of the ray towards the specular surface is perturbed by turning it by an exponentially distributed random amount θ and rotating uniformly around the old direction by ϕ (figure 5). The calculation of θ is similar to the calculation of $||\mathbf{R}||$ for lens perturbation:

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

Good parameter values are $\theta_1 = 0.0001$ and $\theta_2 = 0.1$ radians (Veach & Guibas, 1997).

4.4 Lens subpath mutations

The idea of the lens subpath mutation strategy is to distribute the samples on the pixels of the image and inside individual pixels. The mutation strategy keeps a current lens subpath $\bar{x}_e$ in memory and when a mutation is requested it replaces the lens edge of the path to be mutated with $\bar{x}_e$. Each $\bar{x}_e$ is used a fixed number of times and when this number is exceeded, $\bar{x}_e$ is reconstructed at some other pixel location. When the mutation strategy replaces a part of the current path, the validity of the mutation is verified by casting a ray as in other mutation strategies. If the mutated path does not contribute to the image, the path is rejected. Otherwise the acceptance probability is calculated as normally.

Lens subpath mutations have two important functions. First, they ensure that the entire image is processed so that all pixels receive a share of samples. Second, lens

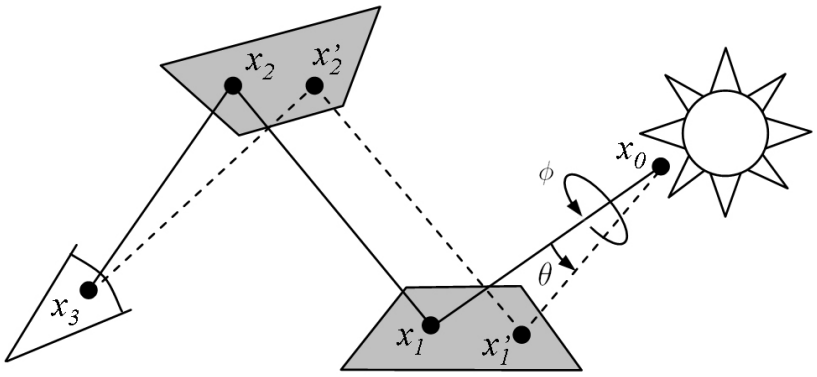


Figure 5: A path of the form $LSDE$ goes through a caustics perturbation. The dashed line shows the mutated path. The surface at x_1 is a specular one casting caustics on a diffuse surface at x_2.

subpath mutations allow the sharing of computation time by reusing subpaths for several pixels.

In comparison to other mutation strategies, lens subpath mutation is the only one that fully supports stratified sampling. Stratification is a variance reduction technique for Monte Carlo integration. The general idea of stratified sampling is to subdivide the integration domain to smaller subdomains and distribute the samples based on the relative sizes of subdomains. Stratification in lens subpath mutation strategy is used to distribute the lens subpath mutations uniformly on all pixels. Also, sample locations on individual pixels can be stratified or otherwise distributed evenly, e.g. using Poisson minimum-disc patterns. This helps in antialiasing edges and other geometric and texture details in image space.

To stratify the lens subpath mutations, the number of lens subpath mutations received by each pixel is stored. Additionally, a list of all pixels in random order are stored and a pointer to the current pixel is kept in memory. When a new lens subpath needs to be constructed, the algorithm moves forward in the random list of pixels until a pixel that has not yet received a full share of mutations is found.

5 RESULTS

Figure 6 shows a scene rendered with a bidirectional path tracer and a MLT implementation. Both algorithms were given the same amount of computation time. The image produced by the MLT algorithm contains much less noise. This is because MLT can utilize coherence and reuse segments of paths that pass through the slightly ajar doorway.

Overall, it seems that MLT is a robust and efficient algorithm for realistic image synthesis. However, it is not an automatic solution to all global illumination problems.

In particular, it is unclear how hard geometric situations like a very small hole in a wall should be handled. How does the Metropolis phase actually find this small detail or should it be the path initialization phase that discovers it?

6 CONCLUSIONS

We have discussed an algorithm called Metropolis Light Transport that can calculate unbiased solutions to global illumination problems. It starts by generating an initial population of light transport paths using a birectional path tracer. These seed light transport paths then go through a number of mutations. In the limit as the process is continued, the paths are distributed according to the contribution they make to the final image. The algorithm is robust and can efficiently calculate solutions to hard lighting situations such as strong indirect lighting and caustics. It is easy to extend the basic algorithm to handle a new type of lighting situation by designing a new mutation strategy for it.

7 ACKNOWLEDGEMENTS

We would like to thank Janne Kontkanen for reviewing the paper and giving many improvement ideas, and Eetu Martola and Mikko Kallinen for proofreading the paper. Thanks also to Jaakko Lehtinen for helping with the layout of the paper.

REFERENCES

Cook, Porter & Carpenter. 1984. Distributed ray tracing. *Pages 137–145 of: Proceedings of SIGGRAPH'84*. ACM SIGGRAPH, Addison-Wesley.

Heckbert. 1990. Adaptive radiosity textures for bidirectional ray tracing. *Pages 145–154 of: Proceedings of SIGGRAPH'90*. ACM SIGGRAPH, Addison-Wesley.

Jensen. 2001. *Realistic Image Synthesis Using Photon Mapping*. A K Peters, Ltd.

Kajiya. 1986. The rendering equation. *Pages 143–150 of: Proceedings of SIGGRAPH'86*. ACM SIGGRAPH, Addison-Wesley.

Lafortune & Willems. 1993. Bi-directional path tracing. *Pages 145–153 of: Compu-Graphics Proceedings (Alvor, Portugal, Dec 1993)*.

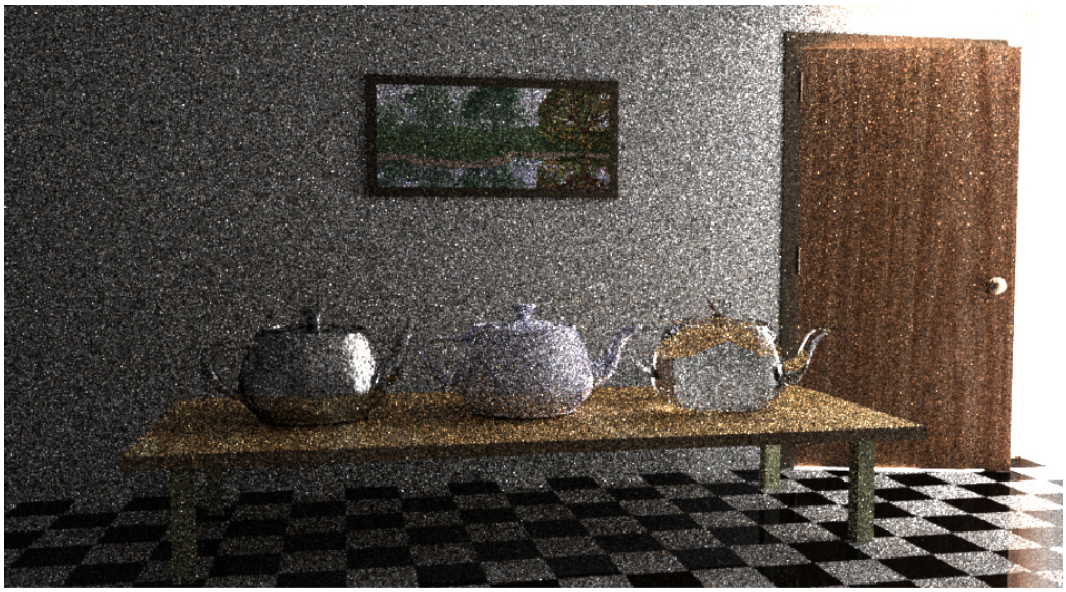
Peskun. 1973. Optimum monte-carlo sampling using markov chains. *Pages 607–612 of: Biometrika 60*.

Veach. 1997. *Robust Monte Carlo methods for light transport simulation*. Ph.D. thesis, Stanford University.

Veach & Guibas. 1996. Bidirectional estimators for light transport. *In: Proceedings of Eurographics Rendering Workshop 1996*.

Veach & Guibas. 1997. Metropolis Light Transport. *Pages 65–76 of: Proceedings of SIGGRAPH'97*. ACM SIGGRAPH, Addison-Wesley.

Whitted. 1980. An improved illumination model for shaded display. *Pages 343–349 of: Communications of the ACM*.



a) Bidirectional path tracing.

b) Metropolis light transport.

Figure 6: This scene is illuminated indirectly by a large area light source in an adjacent room behind the door. The door is slightly ajar to let through about 0.1% of the light. The computation time in both images is the same. The MLT algorithm produces a much better image because it can reuse segments of light paths that go through the doorway. Image courtesy of Veach & Guibas (1997).