Chapter 1

Monte Carlo Path Tracing

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

1.1 Solving the Rendering Equation

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

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

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

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

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

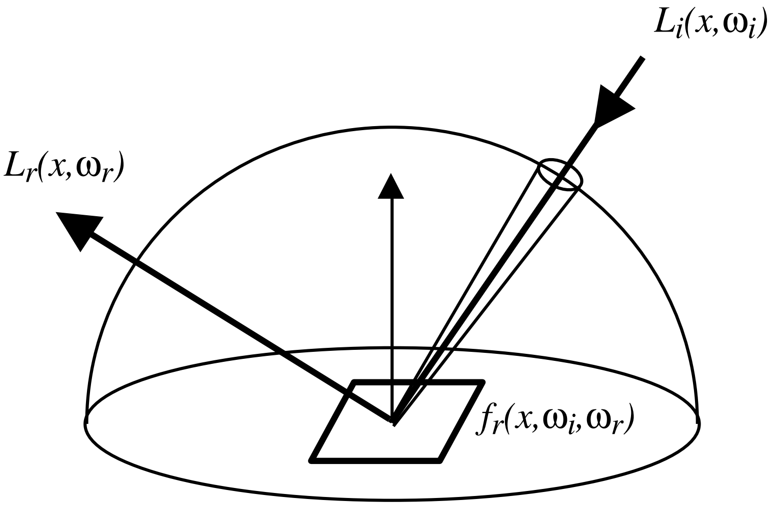

Figure 1.1: Integrating over the upper hemisphere.

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

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

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

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

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

and the three-point BRDF

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

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

Point to point functions are useful since they are intuitive and often clarify the geometry and physics. For example, if $\vec{x}$ sees $\vec{x}'$, then $\vec{x}'$ sees $\vec{x}$. This mutual visibility is represented as the two-point visibility function, $V(\vec{x},\vec{x}')$, which is defined

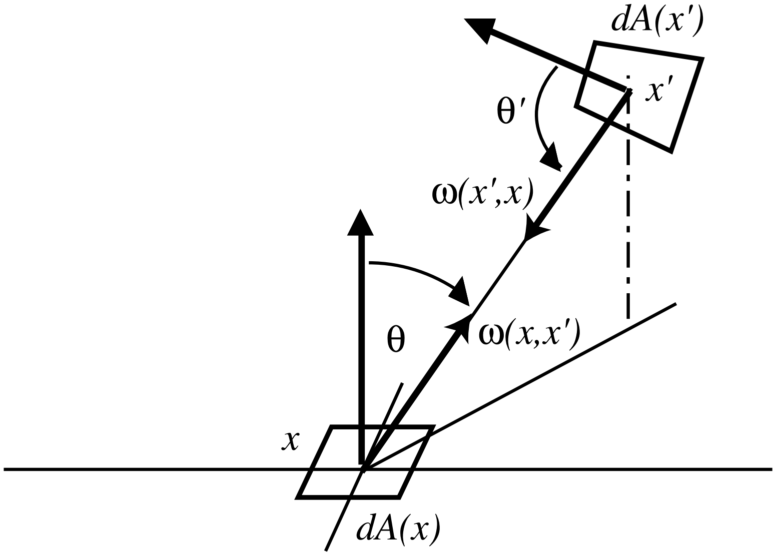

Figure 1.2: Two-point geometry.

to be 1 if a line segment connecting $\vec{x}$ to $\vec{x}'$ does not intersect any opaque object, and 0 otherwise.

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

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

The projected solid angle then becomes

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

where

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

and

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

The rendering equation

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

then has the following solution

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

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

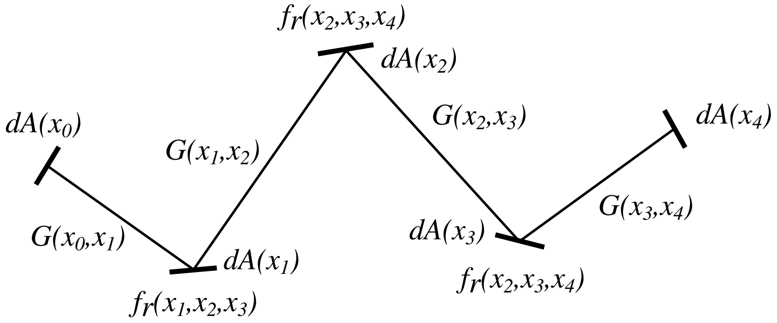

Figure 1.3: A path from point $\vec{x}_1$ to $\vec{x}_5$.

It is useful to write out the formal solution

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

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

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

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

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

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

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

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

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

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

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

1.2 Monte Carlo Path Tracing

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

E - the eye.

L - the light.

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

D - diffuse reflection or transmission

G - glossy reflection or tranmission

S - specular reflection or refraction

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

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

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

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

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

Goral Radiosity: ED^*L

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

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

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

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

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

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

    Step 3A. If emitted,
        return weight * Le

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

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

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

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

Step 2. Trace ray to find point of intersection.

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

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

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

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

Challenge 2 : Finding good estimators with low variance.

1.3 Random Walks and Markov Chains

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

A random walk is characterized by three distributions:

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

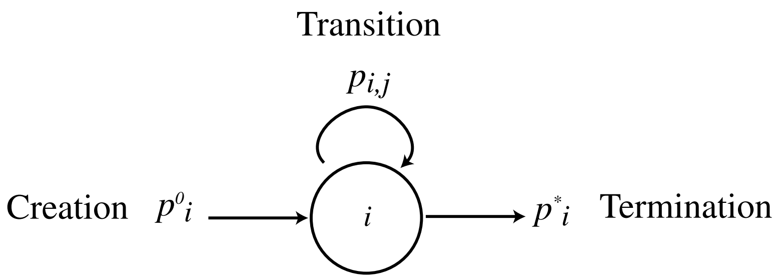

Figure 1.4: State transition diagram for a discrete random walk.

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

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

A discrete random walk consists of the following steps.

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

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

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

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

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

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

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

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

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

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

since then

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

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

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

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

where the form factor equals

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

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

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

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

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

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

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

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

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

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

With these careful definitions, the expected value may be computed

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

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

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

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

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

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

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

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

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

1.4 Adjoint Equations and Importance Sampling

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

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

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

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

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

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

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

and because of the reciprocity principle the BRDF is also symmetric

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

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

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

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

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

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

The measured response is then

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

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

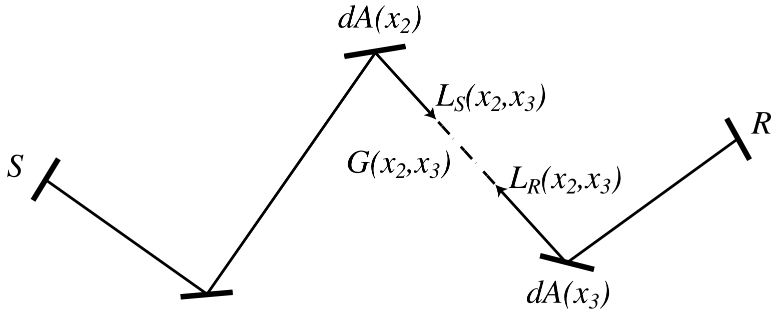

Figure 1.5: A path with both the forward and backward (adjoint) solution of the transport equation. The forward solution is generated from the source term S and the backward solution is generated from the received term R. For physical situations where the transport equation is invariant under path reversal, the forward and backward equations are the same.

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

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

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

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

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

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

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

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

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

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

Note the difference between

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

and

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

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

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

$$K \circ L_S = S$$

We can also write a symmetric equation in the other direction

$$K \circ L_R = R$$

Then,

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

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

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

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

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