CPSC 490 Fall 2016:
Extended Path Tracing Exploration

Wendy Chen
Advisor: Professor Holly Rushmeier

December 17, 2016

Abstract

Path tracing is an unbiased Monte Carlo rendering technique that can be used to create physically realistic images. Path tracing is one algorithm that solves Kajiya's "rendering equation" (1986), which describes radiance emitted towards a view direction from a particular point, given functions for incoming light and bidirectional scattering. This report gives an overview of the rendering equation and Monte Carlo methods for evaluating the equation. Extensive discussion is focused on how Monte Carlo numerical integration works, various methods for generating samples, and how Kajiya's equation can be rewritten to provide an intuitive understanding of how the path tracing algorithm came about. Next term, I would like to build upon the theory discussed to extend my implementation of a path tracer.

1 Kajiya's Rendering Equation

Introduced in 1986, Kajiya's rendering equation describes how light scatters within a scene. Kajiya's equation is an integral equation for rendering that can be solved by Monte Carlo methods and approximated with other methods.

As presented in the original paper, the equation is: [5, p. 143]

$$I(x, x') = g(x, x')\Big[\epsilon(x, x') + \int_S \rho(x, x', x'')I(x', x'')dx''\Big]$$

The terms represent:

$I(x, x')$	intensity of light passing from point x' to point x
$g(x, x')$	geometry term that indicates visibility
$\epsilon(x, x')$	intensity of emitted light from x' to x
$\rho(x, x', x'')$	intensity of incident light at point x' from point x''

Because the intensity term I that we are solving for appears again on the right hand side of the equation within the integral term, it becomes apparent that solving the rendering equation involves evaluating a recursive integral. The rendering equation can be thought of as a Neumann series describing light as a sum of scattered terms: a once scattered term, then twice scattered term, etc.

Approximations of the solution to the rendering equation exist, such as the Utah approximation, which is a two term approximation. In his paper, Kajiya describes the use of Markov chains for solving integral equations. This method is borrowed from radiative heat transfer, which is estimated by counting the number

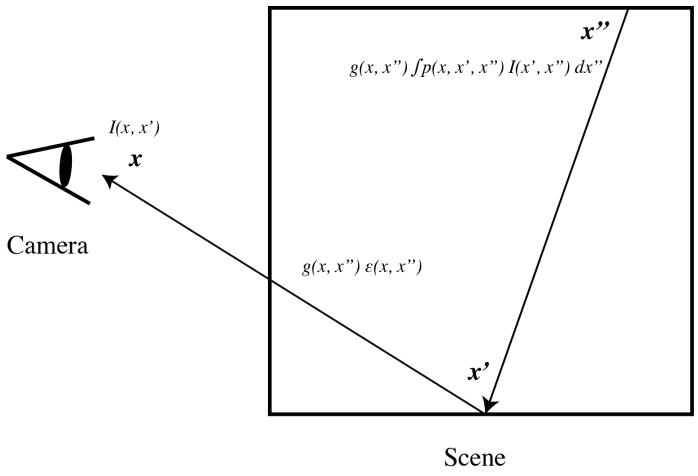

Figure 1: Intensity at x, described by x' and x''

of packets absorbed by each surface. Each path probability is the product of an initial state and transition probabilities. The estimate of the solution to the rendering equation is independent of the probability distribution of the paths. However, the rate of convergence is highly dependent on the manner of choosing the transition probabilities.

The path tracing algorithm is a Monte Carlo solution to Kajiya's rendering equation. In order to understand how path tracing works, we build up our understanding of stochastic rendering methods and Monte Carlo numerical integration over the next few sections.

2 Background on Stochastic Rendering

Path tracing is a stochastic rendering method. On the other hand, Whitted ray-tracing, the first recursive rendering method, is a deterministic method. Whereas deterministic rendering methods give the same result for each rendering, stochastic methods produce different results each time due to the nature of random processes. Over time, however, the stochastic result converges to the correct result.[6, p. 493]

Path tracing is able to simulate the behavior of real photons, which bounce around a scene before hitting our eye. Because path tracing mimics the behavior of light, this global illumination technique can simulate physical phenomena such as color bleeding between surfaces, caustics, etc. While path tracing is not a new algorithm, it has not been widely embraced until recently due to the large amount of computation required and noisy artifacts in path traced images.[1, p. 103] Path tracing algorithms require that the entire scene geometry be loaded into memory at once, whereas previously used algorithms like the Reyes scan-line algorithm are much more efficient.[1, p. 113] In addition, noise-free high quality images could not be produced with path tracing until optimizations in sampling and variance reduction were discovered.

Despite the lag in the wide adoption of path tracing, papers that were published around the same time as Kajiya's seminal paper showed what stochastic rendering processes were capabale of producing. In Rob Cook's 1984 paper on distributed ray tracing, he describes ways to sample various distributions to achieve effects such as blurred reflection and gloss, blurred transparency, penumbra, depth of field, and motion blur. [3, p. 137] For example, the soft shadows of a penumbra can be achieved by casting shadow rays to any spot on the light source. Rendering depth of field can be achieved by simulating the circle of confusion, or the point of light that falls on the focal plane. The circle of confusion can be simulated by shooting rays from a point on the focal plane over the area of the camera lens. Motion blur is an example of taking an integral over time. For non-spatial jittering, we can first divide the frame time into slices and randomly assign a slice

of time to each sample point. Each time within each slice is determined by jittering. In addition to sampling times, we also need to move scene objects and the camera to their correct locations in time.

Figure 2: "1984" by Thomas Porter. The image was included in Rob Cook's 1986 paper on stochastic sampling.[2, p. 68]

Clearly, stochastic sampling allows us to simulate a wide range of phenomena and render astoundingly physically accurate images. We next discuss how Monte Carlo integration works, which will allow us to implement the path tracing algorithm.

3 Monte Carlo Foundations for Path tracing

3.1 Monte Carlo Integration

Kajiya's rendering equation is a continuous equation in which calculating the intensity of incident light at a point requires taking an integral over the hemisphere of possible directions. Because such integrals generally cannot be solved analytically, we must find ways to discretize the problem so that we can solve it numerically. Monte Carlo numerical integration allows us to evaluate such integrals through the use of random sampling, disregarding the integrand's dimensionality. The average result obtained by the Monte Carlo estimator, or its expected value, is equivalent to the true value of the function it estimates.

To explain Monte Carlo integration, we first review concepts from probability:

X represents a random variable, or the value chosen by a random process.

$P(x)$, the cumulative distribution function (CDF), of a random variable is the probability that a value from its distribution is less than or equal to some value x. We write this as, $P(x) = Pr\{X \leq x\}$.

$p(x)$, the probability density function (PDF), of a random variable X describes how likely it will take on a particular value. $p(x)$ is the derivative of a random variable's CDF, where $p(x) = \frac{dP(x)}{dx}$. From the relationship between a random variable's PDF and CDF, we can see that we can integrate over an interval of $p(x)$ in order to calculate the cumulative probability that a random variable takes a value in this interval: $P(x\epsilon[a,b]) = \int_a^b p(x)dx$.

The random variable that we will concern ourselves with is ξ, the canonical uniform random variable.[7, p. 639] ξ takes any value in its domain $[0,1]$ with equal probability. We use ξ because most software today can easily generate such a value with a pseudo-random number generator. As we will show later, we can

transform this uniform distribution into any arbitrary distribution, which will allow us to sample any kind of distribution.

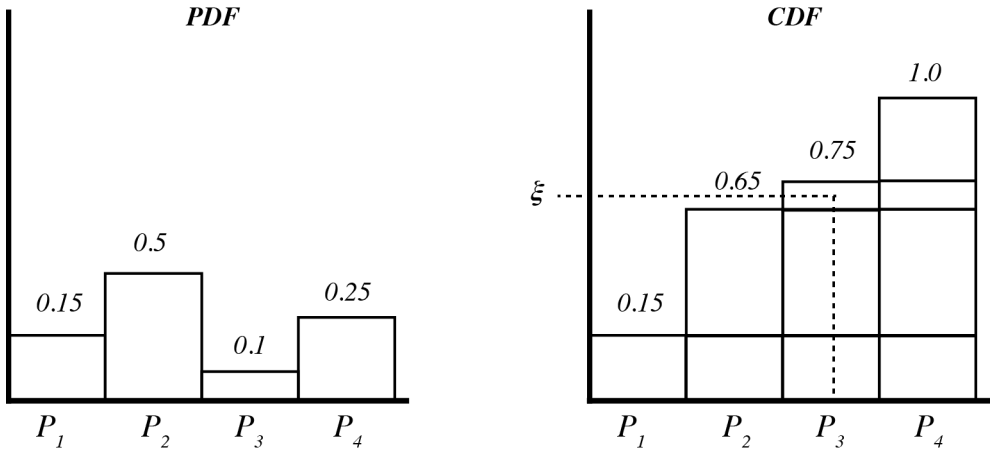

Figure 3: PDF and CDF for a discrete, non-continuous example. We show that we can map a uniformly sampled ξ onto any arbitrary distribution.

$E_p[f(x)]$ is the expected value of a function f, which is the average value of the function with the likelihood of each value given by $p(x)$. For a domain D, the expected value is defined as, $E_p[f(x)] = \int_D f(x)p(x)dx$.

The Monte Carlo estimator allows us to evaluate any arbitrary integral, for we will see that the expected value of the estimator is equal to the integral. Suppose we have the integral $\int_a^b f(x)dx$. Suppose we also have uniform random variables $X_i \epsilon [a, b]$. The Monte Carlo estimator F_N is defined as:

$$F_N = (b-a)\Big(\frac{\sum_{i=1}^{N} f(X_i)}{N}\Big)$$

We can show that the expected value $E[F_N]$ is equal to the integral $\int_a^b f(x)dx$. We know that the PDF $p(x)$ of uniform random variable X_i is $\frac{1}{b-a}$. With algebra, we see that: [7, p. 642]

$$\begin{aligned}
E[F_N] &= E\Big[\frac{b-a}{N}\sum_{i=1}^{N} f(X_i)\Big] \\
&= \frac{b-a}{N}\sum_{i=1}^{N} E\Big[f(X_i)\Big] \\
&= \frac{b-a}{N}\sum_{i=1}^{N} \int_a^b f(x)p(x)dx \\
&= \frac{b-a}{N}\sum_{i=1}^{N} \int_a^b f(x)\frac{1}{b-a}dx \\
&= \frac{1}{N}\sum_{i=1}^{N} \int_a^b f(x)dx \\
&= \int_a^b f(x)dx
\end{aligned}$$

We do not even need to require that the random variables be uniform. If we draw random variables X_i from an arbitrary PDF $p(x)$, then the estimator becomes:

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}$$

The estimator can be extended to multiple dimensions as well, where sampling remains independent of the integral's dimension. Thus, Monte Carlo integration provides us with a way of solving the rendering equation, which we will see becomes an infinite dimension integral when path tracing is introduced. [7, p. 643]

3.2 Uniform Random Sampling

From the previous discussion on Monte Carlo integration, it is apparent that the estimator relies on our ability to sample a desired distribution function. Given any pseudo-random number generator, we can already draw uniform random samples from the domain $[0, 1)$. We can transform this uniform distribution to any arbitrary PDF as long as there exists transformation r that is one-to-one and smooth. Transforming between distributions is the key idea behind the inversion method of generating samples.

The inversion technique of transforming between a uniform distribution and an arbitrary distribution can be generalized to transforming between any two arbitrary distributions, and it can be generalized further to transforming between distributions of multiple dimensions. For the one-dimensional case of drawing samples from an arbitrary PDF $p(x)$ when given an initial uniform distribution, however, the basic steps are as follows: [7, p. 645] First, we compute the CDF $P(x) = \int_0^x p(x')dx'$. We then compute the inverse $P^{-1}(x)$. We then sample a uniformly distributed random number ξ, which we can finally use to compute $X_i = P^{-1}(\xi)$.

To provide an example of the inversion technique, we illustrate how to draw uniform samples from a hemisphere.[7, p. 663] In path tracing, whenever we evaluate the incident lighting integral of the rendering equation, we need to sample from this hemisphere of directions. We would like our samples to be in (x, y, z) coordinates, but we can only directly pick uniform samples with respect to solid angle. In the following example, we will show how the inversion technique allows us to use uniform samples from solid angle to sample the (x, y, z) coordinates of the direction.

Uniformly Sampling a Hemisphere:

1. Transformation between arbitrary density functions $p_x(x)$ and $p_y(y)$ is related by:

$$p_y(y) = p_y(T(x)) = \frac{p_x(x)}{|J_T(x)|}$$

where $|J_T(x)|$ is the absolute value of the determinant of T's Jacobian matrix.

2. For the transformation $T(x)$ from Cartesian coordinates to spherical coordinates, we know:

$$\begin{aligned} x &= r \sin\theta \cos\phi, \\ y &= r \sin\theta \sin\phi, \\ z &= r \cos\theta \end{aligned}$$

where $0 \leq \theta \leq \pi$ and $0 \leq \phi \leq 2\pi$, and $|J_T(x)| = r^2 \sin\theta$. Thus, the relationship between $p(r, \theta, \phi)$ and $p(x, y, z)$ is:

$$p(r, \theta, \phi) = r^2 \sin\theta p(x, y, z)$$

3. We know that the change in variables between spherical coordinates and solid angles is related as:

$$d\omega = \sin\theta d\theta d\phi$$

Thus, we can find the transformation between density functions $p(\theta, \phi)$ and $p(\omega)$:

$$\begin{aligned} p(\theta, \phi)d\theta d\phi &= p(\omega)d\omega \\ p(\theta, \phi)d\theta d\phi &= p(\omega)\sin\theta d\theta d\phi \\ p(\theta, \phi) &= p(\omega)\sin\theta \end{aligned}$$

4. Because we want to choose a direction uniformly with resepct to solid angle, we know that the density function is constant, $p(\omega) = c$. By integrating, we find that:

$$\begin{aligned} \int_{H^2} p(\omega)d\omega &= 1 \\ \int_{H^2} cd\omega &= 1 \\ 2\pi c &= 1 \\ c &= \frac{1}{2\pi} \end{aligned}$$

Thus, we know that $p(\omega) = \frac{1}{2\pi}$. Then $p(\theta, \phi) = \frac{\sin\theta}{2\pi}$.

5. Next, we sample θ's marginal density function $p(\theta)$:

$$\begin{aligned} p(\theta) &= \int_0^{2\pi} p(\theta, \phi)d\phi \\ &= \int_0^{2\pi} \frac{\sin\theta}{2\pi} d\phi \\ &= \frac{\sin\theta}{2\pi}(2\pi) - \frac{\sin\theta}{2\pi}(0) \\ &= \sin\theta \end{aligned}$$

6. Using Bayes' rule, we can compute the conditional density for ϕ:

$$\begin{aligned} p(\phi|\theta) &= \frac{p(\theta, \phi)}{p(\theta)} \\ &= (\frac{\sin\theta}{2\pi})(\frac{1}{\sin\theta}) \\ &= \frac{1}{2\pi} \end{aligned}$$

7. Now we follow the first step of the inversion technique. Integrate to find the CDFs $P(\theta)$ and $P(\phi|\theta)$:

$$\begin{aligned} P(\theta) &= \int_0^\theta p(\theta')d\theta' \\ &= \int_0^\theta \sin\theta' d\theta' \\ &= -\cos\theta' \Big]_0^\theta \\ &= -\cos\theta - (-\cos(0)) \\ &= 1 - \cos\theta \end{aligned}$$

$$\begin{aligned} P(\phi|\theta) &= \int_0^\phi p(\phi'|\theta')d\phi' \\ &= \int_0^\phi \frac{1}{2\pi} d\phi' \\ &= \frac{\phi}{2\pi} \end{aligned}$$

8. Invert the CDFs, and replace $P(\theta)$ and $P(\phi|\theta)$ with canonical uniform random variables ξ_1 and ξ_2:

$$\begin{aligned} P(\theta) &= 1 - \cos\theta \\ \cos\theta &= 1 - P(\theta) \\ \theta &= \cos^{-1}(1 - P(\theta)) \\ \theta &= \cos^{-1}(\xi_1) \end{aligned}$$

$$\begin{aligned} P(\phi|\theta) &= \frac{\phi}{2\pi} \\ \phi &= 2\pi \cdot P(\phi|\theta) \\ \phi &= 2\pi\xi_2 \end{aligned}$$

9. Convert spherical coordinates to Cartesian coordinates to obtain uniform random (x, y, z) sample, remembering that $\sin(\arccos(x)) = \sqrt{1 - x^2}$:

$$\begin{aligned} x &= \sin\theta\cos\phi = \cos(2\pi\xi_2)\sqrt{1 - \xi_1^2} \\ y &= \sin\theta\sin\phi = \sin(2\pi\xi_2)\sqrt{1 - \xi_1^2} \\ z &= \cos\theta = \xi_1 \end{aligned}$$

3.3 Importance Sampling

We have not yet discussed where noise from Monte Carlo based rendering techniques comes from. Reviewing probability, we define the variance of a function to be its expected deviation from its expected value:

$V[f(x)] = E\Big[(f(x))^2\Big] - E[f(x)]^2$. Intuitively, we can imagine that the the average result of the Monte Carlo estimator computed with fewer samples will be more likely to deviate from the estimator's expected value. To decrease noise, the simplest solution is to take more samples. Because error in the Monte Carlo estimator decreases at rate $O(\sqrt{N})$ with respect to number of samples taken, we would need to take four times as many samples to reduce error in half. [7, p. 643, 679]

A more efficient variance reduction technique is importance sampling. The idea behind importance sampling is that we can reduce noise and reduce the number of samples we need to take if we sample from a distribution that is similar to the shape of the integrand. In other words, we try to sample areas of the function's domain that contribute the highest value. Because we are no longer uniformly sampling the function, we also need to weight each sample with inverse proportion to its contribution to the estimator. Weighting the samples lets us account for the fact that we are now taking fewer samples in areas of the function's domain that contribute very little.

To illustrate, we will show how to sample a hemisphere when the distribution we want is not uniform. [7, p. 668]

Concentric Cosine-Weighted Sampling for a Hemisphere

Sampling a hemisphere with a cosine-weighted distribution is useful because it mimics how light scattering is weighted in a standard bidirectional scattering distribution function (BSDF). Incident radiance is most likely to come from directions close to the top of the hemisphere, rather than directions along the rim of the hemisphere.

Thus, we would like to sample directions ω from a PDF such that:

$$\begin{aligned} p(\omega) &\propto \cos\theta, \\ \text{thus,}\quad p(\omega) &= c\cos\theta. \end{aligned}$$

Normalizing $\int_{H^2} c \cdot p(\omega) d\omega = 1$, we find that $c = \frac{1}{\pi}$.

Because we know $p(\theta, \phi) = \sin\theta p(\omega)$, we see that:

$$\begin{aligned} p(\theta,\phi) &= \sin\theta \cdot p(\omega) \\ &= \sin\theta \cdot c\cos\theta \\ &= \frac{1}{\pi}\cos\theta\sin\theta \end{aligned}$$

From here, we can follow the inversion technique once more and compute the marginal and conditional densities again. We do not illustrate the remaining steps, as a detailed example was provided in the previous section on uniform sampling.

Besides sampling from a cosine-weighted PDF, Malley's method is another technique used to generate cosine-weighted samples. Malley's method relies on uniformly sampling a unit disk, and then vertically projecting the points upward onto the hemisphere, resulting in a cosine-weighted distribution. We mention Malley's method here simply to show that it is equivalent to picking a cosine-weighted PDF and sampling from it.

First, the transformation between polar coordinates and Cartesian coordinates on a disk gives us:

$$p(r,\phi) = r \cdot p(x,y), \text{ where } p(x,y) = \frac{1}{\pi} \text{ after normalization,}$$
$$\text{thus, } \; p(r,\phi) = \frac{r}{\pi}$$

Projecting onto a hemisphere, we can rewrite r as $r = \sin\theta$. Finally, to transform between polar coordinates on the disk and spherical coordinates on the hemisphere, we have $p(\theta,\phi) = |J_T| \cdot p(r,\phi)$, where the determinant of the Jacobian is:

$$|J_T| = \begin{vmatrix} \cos\theta & 0 \\ 0 & 1 \end{vmatrix} = \cos\theta.$$

Thus, we see that Malley's method gives us the same cosine-weighted distribution we wanted for $p(\theta,\phi)$:

$$\begin{aligned} p(\theta,\phi) &= |J_T| \cdot p(r,\phi) \\ &= \cos\theta \cdot \frac{r}{\pi} \\ &= \frac{1}{\pi}\cos\theta\sin\theta. \end{aligned}$$

3.4 Multiple Importance Sampling

So far, we have only discussed using the Monte Carlo estimator to compute integrals containing a single function $\int f(x)dx$. We often would like to compute integrals that are the product of multiple functions $\int f(x)g(x)dx$. An example where integrals with multiple functions come up is when evaluating direct lighting integrals that contain distributions describing both the placement of lights in the scene and the object's BRDF. For an intuitive understanding, imagine a scene where a mirror-like object is angled away from a small light source. If we draw samples based on the object's BRDF, it is unlikely that we will generate a direction that hits the light source. Alternatively, if we draw samples based on the light source's direction, it is unlikely that we will generate a sample where the BRDF's value is nonzero. [7, p. 690]

Multiple importance sampling (MIS) allows us to get around the problem when no single distribution can effectively describe an integral composed of multiple distributions. The idea is that we draw samples from multiple distributions and weight the samples accordingly. By sampling multiple distributions at once, we will more likely generate a sample that matches the shape of our integrand.

For example, if we have two distributions p_f and p_g for the integral $\int f(x)g(x)dx$, then the MIS Monte Carlo estimator is: [7, p. 691]

$$\frac{1}{n_f}\sum_{i=1}^{n_f}\frac{f(X_i)g(X_i)w_f(X_i)}{p_f(X_i)} + \frac{1}{n_g}\sum_{i=1}^{n_g}\frac{f(Y_j)g(Y_j)w_g(Y_j)}{p_g(Y_j)},$$

where n_f is the number of samples from p_f, n_g is the number of samples from p_g, and w_f and w_g are used to weight the samples.

An example of a weighting function would be the balance heuristic:

$$w_s(x) = \frac{n_s p_s(x)}{\sum_i n_i p_i(x)}$$

4 Different Forms of Kajiya's Equation

Today, Kajiya's rendering equation frequently does not appear in its original form as presented at the beginning of this report. The equation can be understood in many different forms. In this section, we will illustrate different forms of the rendering equation until we reach a form from which we can readily observe how the path tracing algorithm came to be.

4.1 Light Transport Equation (LTE) in canonical form

Because the rendering equation tries to describe the transport of light radiance within a scene, it is commonly referred to as the light transport equation, which is written as: [7, p. 752]

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i$$

where the terms are:

$L_o(p, \omega_o)$	total outgoing radiance at point p along direction ω_o
$L_e(p, \omega_o)$	emitted radiance from point p along direction ω_o
$\int_{S^2} ...d\omega_i$	incoming radiance at point p, taken as a integral over the sphere of directions at p
$f(p, \omega_o, \omega_i)$	BSDF measuring fraction of radiance scattered from incident direction to outgoing direction
$L_i(p, \omega_i)$	incident radiance at point p from direction ω_i
$\|\cos \theta_i\|$	orientation of the surface that p lies on compared to the direction of incident light

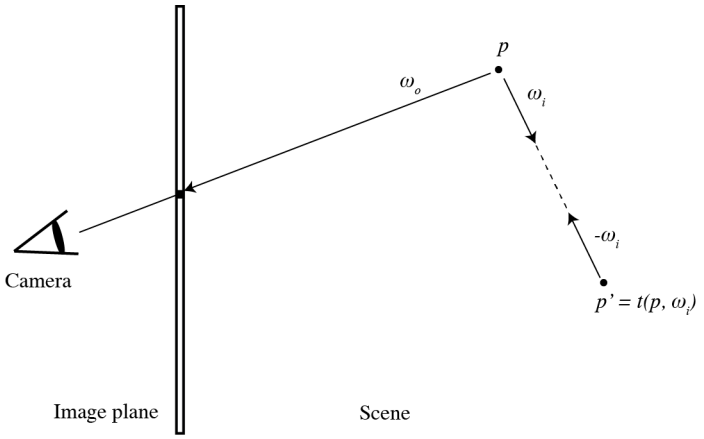

Figure 4: Relationship between incident radiance at p and outgoing radiance at p'.

As shown in Figure 4, the light transport equation computes the exitant radiance from the camera ray's intersection point p in direction ω_o from p to the film sample. Due to energy balance, we can rewrite incident radiance at p as outgoing radiance from p'. We use the trace operator $t(p, \omega)$ to describe the point on the first surface that ray ω intersects. An equivalent form of the light transport equation is:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i) L_i(t(p, \omega_i), -\omega_i) |\cos \theta_i| d\omega_i$$

In the form given above, we have rewritten all radiance as exitant radiance from points on surfaces. To find the numerical solution to this form of the equation, we use Monte Carlo methods to sample a number of directions from a distribution of directions on the sphere and cast rays to evaluate the integrand.

4.2 Surface form of the LTE

In the canonical form of the light transport equation, the geometry term was implicitly hidden in the trace operator $t(p, \omega)$. To get the surface form of the equation, we want to make the geometry term $G(p'' \leftrightarrow p')$ explicit.

To do so, in addition to rewriting directions ω as light paths between points p, we also need to multiply by the Jacobian $\frac{|\cos\theta'|}{r^2}$ to transform between solid angle and surface area. We keep the original $|\cos\theta|$ term from the light transport equation, and we add a visibility function $V(p \leftrightarrow p')$ that takes value 1 if visible, 0 otherwise. Our new geometry term is defined as: [7, p. 754]

$$G(p'' \leftrightarrow p') = V(p \leftrightarrow p')\frac{|\cos\theta||\cos\theta'|}{||p - p'||^2}$$

Our last change is to rewrite the integral to be taken over area instead of over directions on a sphere. The surface form of the light transport equation is: [7, p. 755]

$$L(p' \to p) = L_e(p' \to p) + \int_A f(p'' \to p' \to p)L(p'' \to p')G(p'' \leftrightarrow p')dA(p'')$$

The direction of the arrow indicates the direction along which the light travels. A describes the area of all surfaces in the scene. To find the numerical solution with Monte Carlo, we choose a number of points on surfaces according to a distribution over surface area and compute radiance transport between these points to evaluate the integrand. We trace rays in order to evaluate the visibility term $V(p \leftrightarrow p')$.

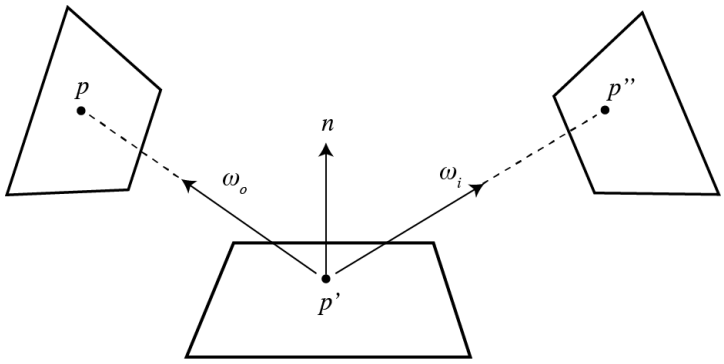

Figure 5: The three-point surface form of the rendering equation

4.3 Integrating over Paths (Generalizing three-point transport)

We would like to define incoming radiance at p_0 on the film plane as the integral of radiance over all paths between p_0 and an origin point on a light source.

In other words, the radiance along the path between the camera-ray intersection p_1 and the film plane p_0 is an infinite sum over all paths from p_1 that hit light sources. The infinite sum can be written as: [7, p. 756]

$$L(p_1 \to p_0) = \sum_{n=1}^{\infty} P(\bar{p}_n)$$

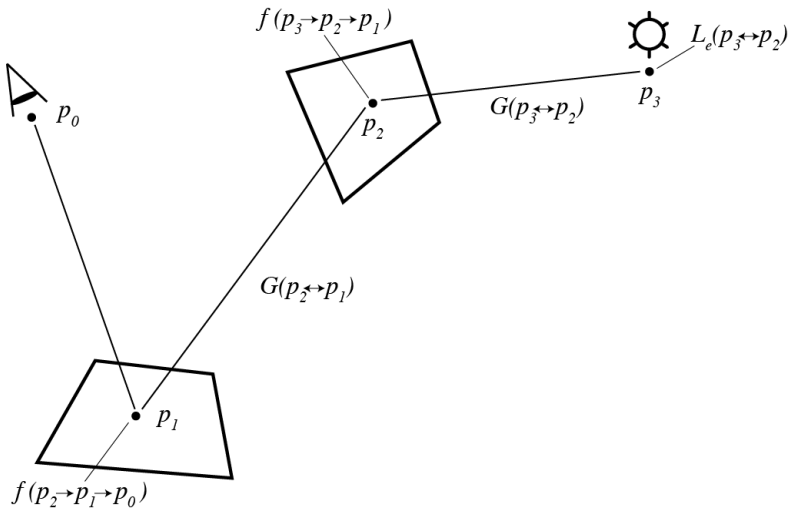

Figure 6: A sample path composed of four vertices

where the terms in the infinite sum are:

$\bar{p}_n$	is a path with $n+1$ vertices: p_0, p_1, ..., p_n
p_0	lies on the film plane
p_n	lies on a light source
$P(\bar{p}_n)$	is the radiance along the path

The radiance $P(\bar{p}_n)$ is defined as the initial emitted radiance times the throughput of the path—the fraction of the radiance from the light source that arrives at the camera after scattering along the vertices between them.

The throughput $T(\bar{p}_n)$, which is a fraction, is defined as: [7, p. 757]

$$T(\bar{p}_n) = \prod_{i=1}^{n-1} f(p_{i+1} \to p_i \to p_{i-1}) G(p_{i+1} \leftrightarrow p_i)$$

Thus, the radiance along a path is defined as: [7, p. 757]

$$P(\bar{p}_n) = \underbrace{\int_A \int_A \dots \int_A}_{\text{n - 1}} L_e(p_n \to p_{n-1}) T(\bar{p}_n) dA(p2) \dots dA(p_n)$$

To solve the path integral, we compute the Monte Carlo estimate of radiance arriving at p_0 along paths of length n by sampling a set of vertices with appropriate sampling density in the scene to generate a path, and we evaluate $P(\bar{p}_n)$ using our sampled vertices.

5 Unidirectional Path tracing

The path integral form of the rendering equation is what the path tracing algorithm seeks to evaluate. Thus, when we implement path tracing, we must solve two main problems.[7, p. 762] The first problem is how can

we estimate the sum of an infinite number of $P(\bar{p}_i)$ terms. The second problem is how do we pick vertices to generate paths $\bar{p}$.

Our previous discussion of sampling techniques helps us solve the second problem of path generation. At each point on our path, we can generate a sample using any of the techniques we have discussed to find the next vertex on our path. For example, if we would like to sample according to the bidirectional scattering distribution function (BSDF) at our current point, we could use importance sampling to generate a sample from a cosine-weighted distribution. The fractional value of the BSDF, given incoming and outgoing directions, is the throughput at that vertex.

To solve the first problem of evaluating an infinite sum, we note that the more vertices a path has, the less light the path scatters. Thus, we can estimate the sum of radiance along a path by always evaluating the first couple of terms, and then apply Russian roulette to determine when to terminate the sum.

6 Code for simple, unidirectional path tracer

I have currently implemented a simple, unidirectional path tracer in C. The code is based off my past independent study, where I added an acceleration structure to a ray-tracer written in JavaScript. Before this term, I had refactored the ray-tracer into C code. During this term, I have extended the implementation to include calculations for indirect lighting, thus, obtaining global illumination.

The direct lighting calculation remains simple and needs to be modified in the future to make use of the sampling techniques that I have studied this term. To calculate direct lighting at a point in the scene, a camera-ray is cast into the scene. From the intersection point of the camera-ray into the scene, shadow rays are cast to each of the lights present in the scene. If the light is visible, the light's contribution is counted. Currently, because only diffuse Lambertian surfaces are supported, the contribution is always calculated as a dot product between the surface normal and the ray cast to the light.

The direct lighting calculation follows a deterministic algorithm, so any images produced with only direct lighting will always look the same. Besides the lack of variance introduced by shooting a ray to every light, the point lights themselves cannot introduce variance, as they are always sampled at their origin points.

All of the direct lighting computations can be performed without continuing to build paths after casting the first camera-ray shot into the scene. Extending my code to support path tracing only affects the indirect lighting computation, which is a stochastic process and produces slightly different results on each render. Due to the stochastic nature of the indirect lighting computation, the noisiness of the image is also affected by the number of samples taken per pixel.

For path generation, the first two bounces after the initial camera-ray is cast into the scene are always computed. The direction is chosen by implementing uniform sampling within a hemisphere. As explained in section 3.2, I used two psuedo-random numbers to generate a new direction with uniform probability. After the first two bounces, Russian roulette is used to determine when the path terminates. If the random number generated is less than the reflectance traveling along the last ray, then a new ray is cast. Otherwise, the path terminates. Intuitively, this makes sense because dark surfaces will have a low reflectance, and it will be more likely for the path to terminate.

For the first two bounces after the initial shot into the scene, the indirect lighting contribution is computed as the fractional contribution left after the initial contribution from direct lighting is multiplied by the throughput along the path. Throughput is compounded as the path's length increases.

After the first two bounces, a running contribution is kept until Russian roulette terminates the path. The lighting contribution found by Russian roulette is averaged over the number of samples taken for the roulette part of the path.

7 Images

Below are some images rendered with the current state of my unidirectional path tracer. Each image is 512 x 512 pixels, with twenty-five samples taken per pixel.

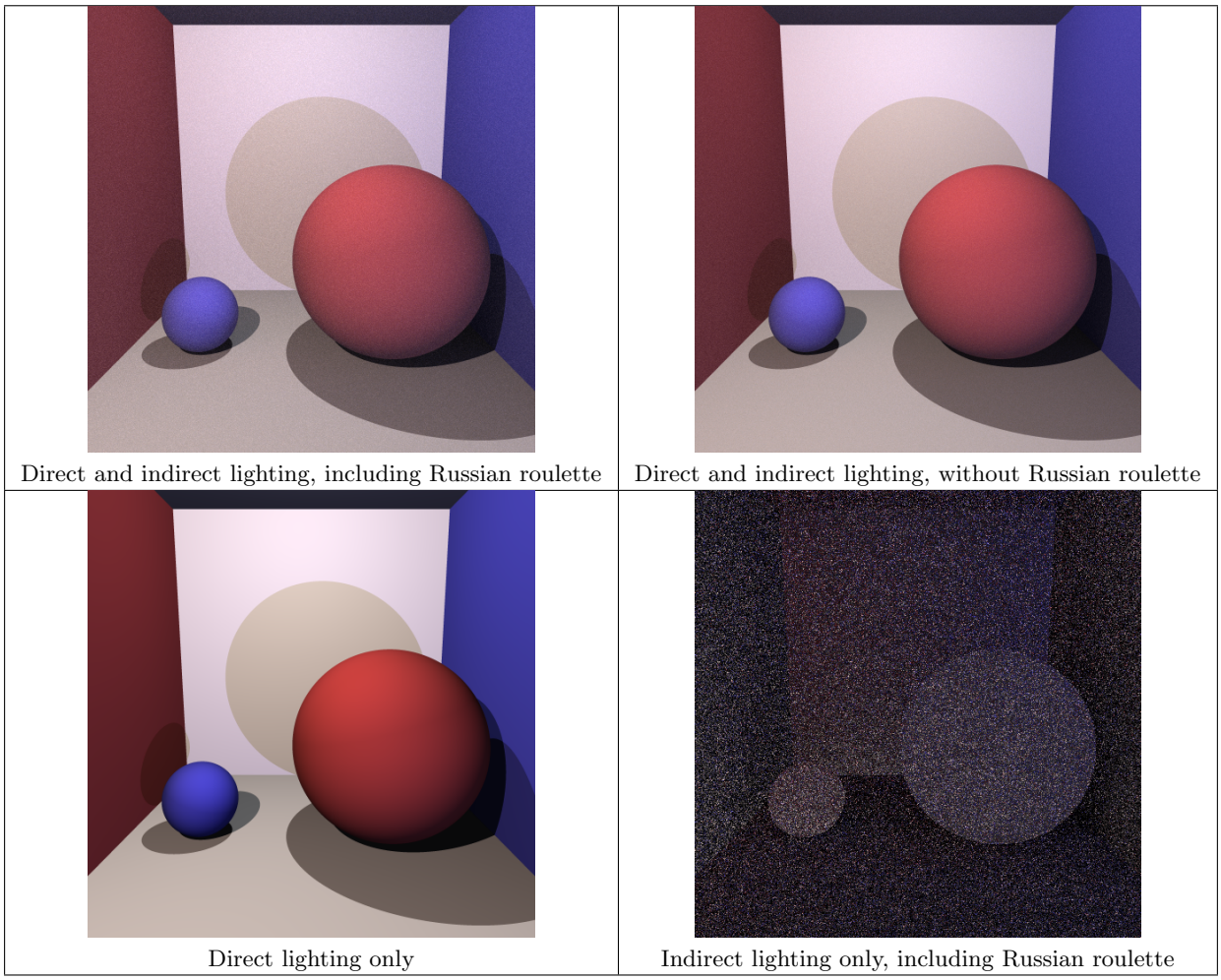

Figure 7: Comparison of lighting contributions for a sample scene.

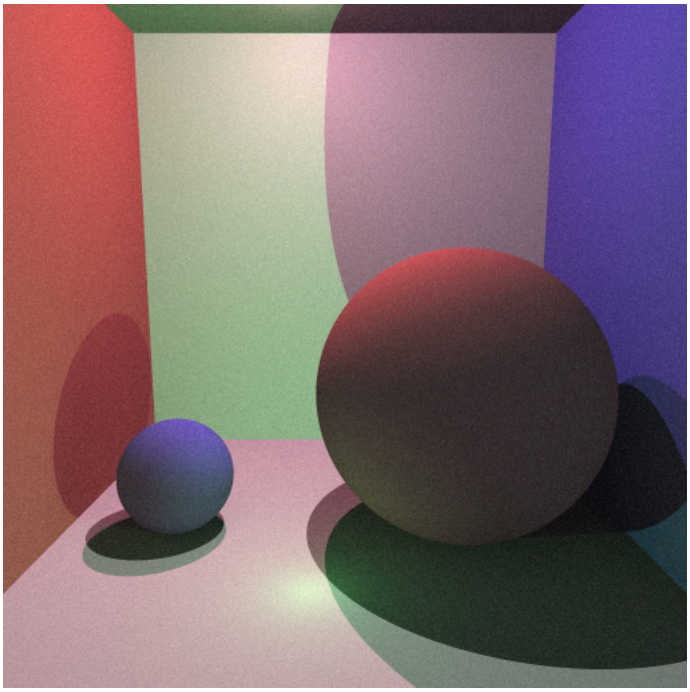

Figure 8: Same sample scene from Figure 7, but with pink and green lights.

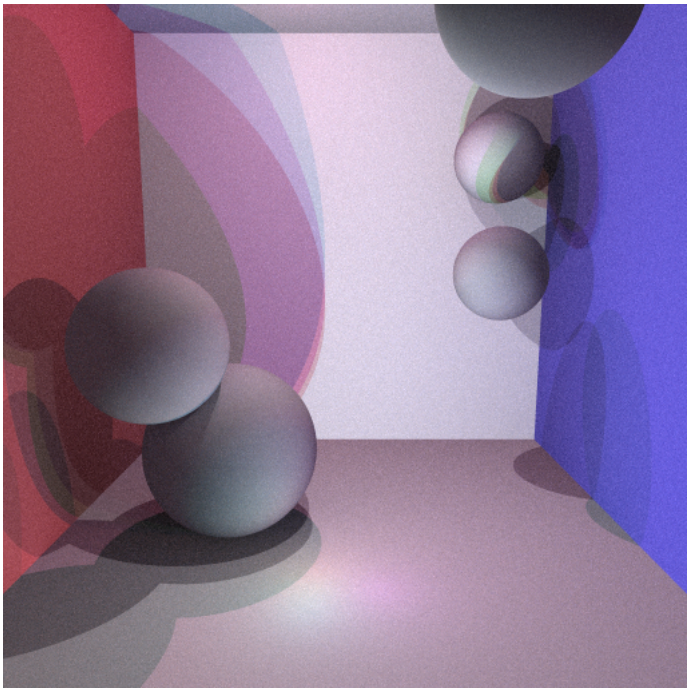

Figure 9: Floating spheres with multiple colored lights.

8 Future Work

Having studied more of the sampling theory behind path tracing this semester, I would like to continue to refine the sampling techniques used in the unidirectional path tracer over the next semester.

For path generation, I would like to implement cosine-weighted sampling instead of uniform sampling. By implementing importance sampling, I can expect faster convergence and less noise in each image with the same number of samples. Additionally, I would like to check that paths are being constructed correctly. I have not accounted for rotating the hemisphere of directions according to the normal of the surface.

For direct lighting, I would like to apply multiple-importance sampling techniques. To do so, I would need to extend the amount of information stored with each light to include its size and power. With this additional information, I would be able to sample lights based on how much relative radiance they emit when compared with one another.

I would also like to explore optimizations of unidirectional path tracing, such as bidirectional path tracing and Metropolis Light Transport. Beyond optimizations to the Monte Carlo based path tracing algorithm, I would like to finish implementing the acceleration structure. In order to create more complex scenes, I would like to add a scene parser to enable importing arbitrary geometry in obj format.

Before this term ended, I had begun reading about general material models. I would like to extend the number of materials handled by the path tracer, as it currently only renders Lambertian surfaces. Beyond Phong and Blinn-Phong reflectance, which I have had brief exposure to, I would like to explore more material models that I read about in Chapter 5 of *Digital Modeling of Material Appearance.*[4] Having read about various sampling techniques for BSDFs in *Physically Based Rendering Theory*, I am curious to understand how these material models were actually created.

Lastly, if I extend the unidirectional path tracer, I would like to carefully redesign the code and separate it into the appropriate abstraction layers. Currently, the implementation lives in one file since the scope of the path tracer is limited.

9 Acknowledgments

I would like to thank Professor Holly Rushmeier for guiding me in independent studies both last year and this year. Thank you to my former classmates Brian Li and Jakub Kowalik for sharing their enthusiasm and knowledge of graphics with me. Thank you to my friend Karl Li, whose deep understanding and excitement for path tracing inspired me to learn more.

References

[1] Per H. Christensen and Wojciech Jarosz. The path to path-traced movies. *Foundations and Trends® in Computer Graphics and Vision*, 10(2):103–175, October 2016.

[2] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, January 1986.

[3] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, January 1984.

[4] Julie Dorsey, Holly Rushmeier, and Franois Sillion. *Digital Modeling of Material Appearance*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[5] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[6] Greg Ward Larson and Rob Shakespeare. *Rendering with Radiance : the art and science of lighting visualization*. The Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, San Francisco (Calif.), 1998.

[7] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.