

Path Tracing part 2

Steve Rotenberg

CSE168: Rendering Algorithms

UCSD, Spring 2017

Monte Carlo Integration

Monte Carlo Integration

- The rendering (& radiance) equation is an infinitely recursive integral that we can't hope to solve analytically
- We need a numerical approach
- There are many numerical approaches to solving integrals, but they generally require taking advantage of some predictable properties of the function
- As our function is very complex (due to the complex geometry, materials, and lights in the scene), it is difficult to design a numerical scheme custom suited to the problem
- When all else fails, one often resorts to Monte Carlo integration, making use of brute force random sampling to approximate an integral
- This is widely used technique in mathematics and computer science, and is often the last resort for very complex problems that can't be solved any other way
- It is extremely powerful, as it can adapt to just about any type of problem
- It's biggest disadvantage however, is that it tends to be very slow

Monte Carlo Integration

- The radiance equation is:

$$L_r(\omega_r) = \int_{\Omega} f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \theta_i d\omega_i$$

- We talked about using Monte-Carlo integration to estimate the area of a circle
- We also applied the same approach to estimating the shadowing from an area light source, and estimating the color of a rectangular pixel
- We can use it to estimate the radiance reflected off a surface
- Instead of integrating over an infinite number of directions, we can estimate by summing over a finite number of randomly selected directions

$$L_r(\omega_r) \approx \frac{2\pi}{N} \sum^N f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \theta_i$$

Monte-Carlo Integration

$$L_r(\omega_r) \approx \frac{2\pi}{N} \sum^N f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \theta_i$$

- We need to generate N random directions in a hemisphere
- For each sample, we can then evaluate the incoming light $L_i(\omega_k)$ from the environment, multiply it by $\cos \theta_i$ and then add it to a running total, which finally gets scaled by $\frac{2\pi}{N}$
- We accept that $L_i(\omega_k)$ will vary unpredictably, so some samples will end up contributing a lot and some will contribute less
- The $\cos \theta_i$ term however, is not unpredictable, yet it will still cause some samples to contribute a lot and some to contribute almost nothing
- We would like to eliminate this by generating a cosine-weighted distribution (instead of using a uniform distribution that gets scaled by the cosine)

Cosine-Weighted Hemisphere

- We can accomplish this by simply changing from the hemispherical mapping formula to the cosine weighted hemisphere formula
- This gives us a better way to estimate the irradiance:
- If we have random directions ω_i distributed uniformly over a hemisphere:

$$L_r(\omega_r) \approx \frac{2\pi}{N} \sum^N f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \theta_i$$

- If we switch to a cosine weighted distribution for ω_i , we can use:

$$L_r(\omega_r) \approx \frac{\pi}{N} \sum^N f_r(\omega_i, \omega_r) L_i(\omega_i)$$

- Note that we adjusted the total scale factor because the area of a hemisphere is 2π , but the $\cos\theta$ scaled area of a hemisphere is π

Monte Carlo Strategies

- Adaptive Sampling
- Integral Partitioning
- Russian Roulette
- Importance Sampling

Adaptive Sampling

- Adaptive sampling can be used to determine when enough paths have been traced to reduce the error below some desired threshold
- This allows the computation time to be spent in the areas that need it
- We looked at this in detail in a previous lecture



Integral Partitioning

- The integral portion of the rendering equation contains the product of the hemisphere of incoming light with the BRDF
- If we think about the incoming hemisphere of light, we can separate it into the direct light coming straight from the light sources and the indirect light coming from one or more bounces
- We expect that the direct light is bright and has strong boundaries, and we also have the benefit of knowing exactly where the light sources are
- We expect the indirect light to be smoother, but we do not have any knowledge ahead of time about how it is distributed
- This is a good reason to treat those two parts of the integral differently, and use different techniques to evaluate them
- *Partitioning* is the process of breaking an integral into multiple parts, evaluating each part with a different technique, and then summing up the results

Integral Partitioning

- This is what we are doing at each bounce, when we shoot a shadow ray to the light source and shoot a reflection ray
- The shadow ray is evaluating the direct lighting part and the reflection ray is evaluating the indirect light
- With point and directional lights, it would be impossible for a bounced ray to hit the light, so we are enforcing this partitioning
- With area lights, we have to make sure that if a bounced ray hits an area light, then we ignore the emission contribution from the light, as it would have already been considered in the direct lighting partition

Path Length

- Some lighting situations do not involve a lot of bounces, such as outdoor lighting, and may look fine with only two or three bounces
- Indoor scenes with bright lights and bright walls will have a lot of indirect light and will require more bounces (maybe 5 or more)
- Scenes with shiny objects and dielectrics may require many bounces to render correctly (10+)
- It is always possible to construct a situation where we need even more bounces (such as a hall of mirrors)
- How do we determine how long our paths should be without resorting to just picking some number?

Russian Roulette

- If we limit the number of bounces to a fixed value, then we are introducing bias (consistent error) to the image as we will underestimate the total brightness
- Russian roulette is a strategy that randomly determines whether or not to terminate a path at each bounce, based on the total contribution of the path
- Above a certain brightness threshold, all bounces are accepted. Below that threshold, we assign a probability of acceptance
- If, for example, we determine that there is a $1/N$ chance of accepting a particular bounce, then we scale its contribution by N if it is accepted

Importance Sampling

Mean Estimate

- We saw that when we are trying to estimate a function using Monte Carlo integration, our estimate after N samples is computed as:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

Weighted Average

- A slightly more sophisticated approach is to take a *weighted average*

$$\bar{x} = \frac{\sum w_i x_i}{\sum w_i}$$

- Is there some set of weights w_i that will give us a better estimate than just taking the mean?

Importance Sampling

- The idea behind *importance sampling* is to attempt to select a weighted distribution of random numbers that will lead to a more accurate estimate of the function
- In order to do this however, we need to have some sort of idea of what the function might look like so we can select our weights

Monte Carlo Integration

- Let's say that we are trying to estimate an integral over some domain
- We'll choose a 1D function $f(t)$ over the domain $[0...1]$ as an example, but this can be extended to n-D functions over any domain

$$z = \int_{t=0}^1 f(t) dt$$

Monte Carlo Integration

$$z = \int_{t=0}^1 f(t) dt$$

- We want to estimate z , so we choose a set of N random numbers ξ_i on the $[0...1]$ interval and evaluate $f_i=f(\xi_i)$ for each one
- The estimate for the integral is then:

$$z \approx \frac{1}{N} \sum_{i=1}^N f(\xi_i)$$

Monte Carlo Integration

- Without knowing anything at all about $f()$, we can't really do much better than that
- However, if we have some sort of idea of the shape of $f()$, then we may be able to choose a set of weighted samples that do a better job
- For example, let's say that we know that $f()$ is actually the product of two other functions $g()$ and $h()$:

$$f(t)=g(t)h(t)$$

- Let's also say that we know $g(t)$ exactly, but have no idea what $h(t)$ is

Monte Carlo Integration

$$z = \int_{t=0}^1 f(t) dt = \int_{t=0}^1 g(t)h(t) dt$$

$$z \approx \frac{1}{N} \sum_{i=1}^N f(\xi_i) = \frac{1}{N} \sum_{i=1}^N g(\xi_i)h(\xi_i)$$

Importance Sampling

- If we choose a set of random numbers that match the distribution of $g()$, then we can estimate the function with:

$$Z \approx \frac{\sum w_i h(\xi_i)}{\sum w_i}$$

- This will usually lead to a much better estimate, and allow us to take fewer samples
- This process is known as *importance sampling*, since we placing more samples where we think they are more important (i.e., where we expect the function to have a higher value)
- The question that remains is how do we generate a set of random numbers that matches the distribution of $g()$?

Rejection Sampling

- We want to generate a bunch of samples that match the distribution of $g(t)$ over the $[0...1]$ interval
- One way to do this is with *rejection sampling*
- Let's say that the maximum value of $g(t)$ over the interval is g_{max}
- We start with a uniformly distributed random number ξ_i , but then choose a second random number δ_i
- If $g(\xi_i)$ is less than $g_{max} \delta_i$, then we keep ξ_i , otherwise, we reject it and try again
- We keep this up until we have a ξ_i that passes
- This is then used to evaluate $h(\xi_i)$ and contributes to our estimate
- This process may lead to a lot of rejections, which ultimately waste time, but this can be improved by using a tighter fitting bound than just g_{max}

Probability Density Function

- A more sophisticated way of generating these distributions uses the concept of a *probability density function* or *PDF*
- We start by normalizing $g(t)$ so that it represents 100% of the distribution

$$G(t) = \frac{g(t)}{\bar{g}}, \quad \text{where} \quad \bar{g} = \int_{-\infty}^{\infty} g(t) dt$$

- $G(t)$ will match the shape of $g(t)$, but will be scaled so that:

$$\int_{-\infty}^{\infty} G(t) dt = 1$$

- Note that $g(t)$ does not need to be smooth or continuous, but it does need to be positive for all values of t

Cumulative Distribution Function

- We can then compute the *cumulative distribution function* or *CDF*:

$$C(t) = \int_{-\infty}^t G(t)dt$$

- The graph of $C(t)$ will go from 0 to 1 in a continuous and non-decreasing way as t increases
- Note that:

$$G(t) = \frac{d}{dt} C(t)$$

Random Distributions

- To generate a random number with the desired distribution, we choose a uniformly distributed random number ξ_i and find out where $C(\alpha_i) = \xi_i$
- α_i will then be a random number matching the original distribution of $g()$
- We can then evaluate $\bar{g}h(\alpha_i)$ and use it in our estimate

BRDF Sampling

Importance Sampling

- *Importance sampling* refers to the technique of concentrating the sampling the integral in areas of high importance
- Rather than take a bunch of samples that contribute highly varying amounts to the final estimate, we would prefer to take samples that are all roughly equally important, so that we can make the most of each sample
- For rendering, this means that we want to concentrate our reflection rays in directions that contribute a lot to the reflected light
- In other words, we want to generate samples that correspond to the reflectivity described by the material BRDF

BRDF Evaluation

- In classical ray tracing and rendering, BRDFs are evaluated directly- that is, we have some known view direction and light direction and we want to directly evaluate the reflected intensity
- In these cases, we are just plugging in values of ω_i and ω_r to the BRDF and computing the result
- This is called *forward evaluation* of the BRDF

Radiance Estimation

- Recall the radiance equation:

$$L_r(\omega_r) = \int_{\Omega} f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \theta_i d\omega_i$$

- To determine a pixel color, we have to evaluate (or at least estimate) this integral
- However, it's generally too complex to integrate analytically over all possible directions of incoming light, so we have to approximate it by summing over a subset of the possible directions

Radiance Estimation

$$L_r(\omega_r) = \int_{\Omega} f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \theta_i d\omega_i$$

- We need to choose a direction (or several directions) in the hemisphere to sample this integral. We can use any of the various number generators we've previously discussed (uniform, random, jittered, quasi-random), combined with the cosine weighted hemispherical mapping
- This will give us a set of ω_i values to plug into a summation to approximate the integral

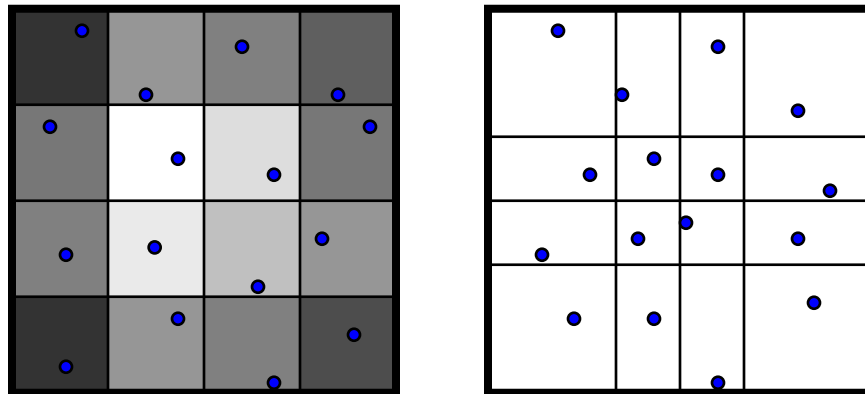
$$L_r(\omega_r) \approx \frac{\pi}{N} \sum^N f_r(\omega_i, \omega_r) L_i(\omega_i)$$

Radiance Estimation

- The cosine weighted hemisphere might work fine for diffuse BRDFs that are non-zero over a wide range of angles
- However, for specular (shiny) BRDFs, picking a random direction and then scaling by the BRDF for that direction will usually lead to 0
- Consider the Fresnel metal BRDF for example. Every random direction chosen would be scaled to 0 except for the impossible chance that it happens to pick the reflected direction
- Therefore, we need a smarter way to choose our directions to estimate the integral

BRDF Sampling

- A good solution is to *sample the BRDF*
- Instead of generating a bunch of evenly distributed samples and then scaling by the BRDF value for the chosen direction, we randomly generate a non-uniform distribution of samples that match the distribution of the BRDF function
- These samples can then be equally weighted
- This is parallel to the discussion we had about pixel sampling for antialiasing



BRDF Sampling

- Instead of passing the BRDF evaluation function a known input and output direction, we wish to *sample* the BRDF by providing an input direction and have it generate a random output direction that is statistically distributed according to the reflected intensity
- In other words, it will generate a random reflected vector, but the randomness will tend to favor the more likely reflected directions
- We can think of it as generating random incident directions that contribute equally to the reflected light towards the viewer
- Or, due to the reciprocity of BRDFs, we can think of it as shining an incident ray and randomly generating a reflected (scattered) ray

Fresnel BRDF Sampling

- The simplest BRDF to sample is the Fresnel metal BRDF: an input vector will always generate the same reflected vector
- A Fresnel dielectric might either reflect or refract the ray. The intensity of each possibility is determined by the Fresnel equations. Instead of generating two rays scaled by the intensity, we generate only one ray at full intensity, but randomly chosen according to the two possible intensities

Lambert BRDF Sampling

- As the Lambert BRDF is constant across the hemisphere, it makes sense to just use the hemispherical mapping to generate the direction uniformly
- However, we are really interested in evaluating the radiance equation, we can compensate for the cosine weighting term in the equation as well by using a cosine weighted hemisphere
- This generates samples that are of equal value

Microfacet BRDF Sampling

- The Cook-Torrance model and its related microfacet models (Oren-Nayar and Walter) were not designed with inverse BRDF sampling in mind
- As a result, there are no perfect ways to inverse sample them
- However, the cosine weighted hemisphere is always a decent fallback option
- It should work well for Oren-Nayar, which is similar to a Lambert surface, but will definitely not be well suited to shiny specular materials such as low-roughness Cook-Torrance materials

Multiple Importance Sampling

- Depending on the surface properties, sampling the reflection (BRDF) partition or sampling the light partition may generate excessive noise
- *Multiple importance sampling* is a technique for optimally combining the two by estimating the noise of each and weighting the result towards the less noisy, allowing for a smooth transition between techniques

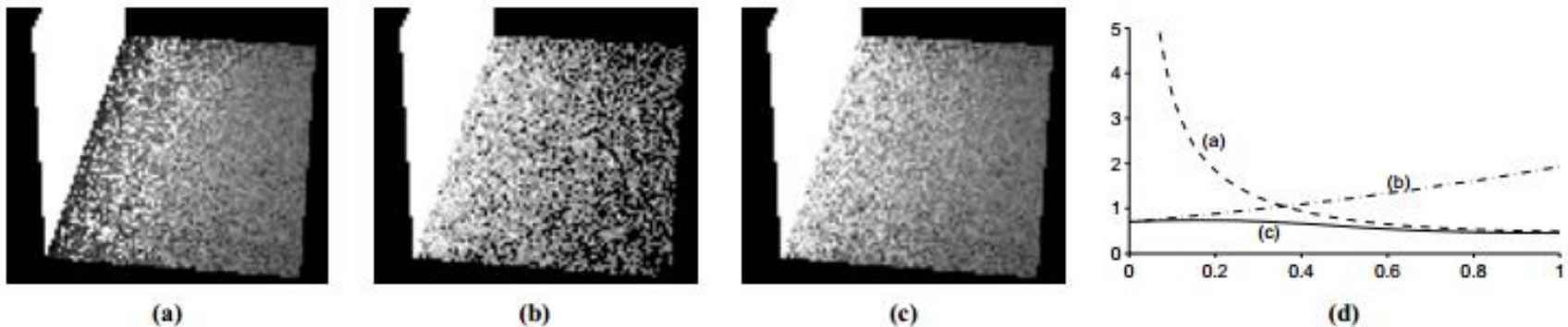
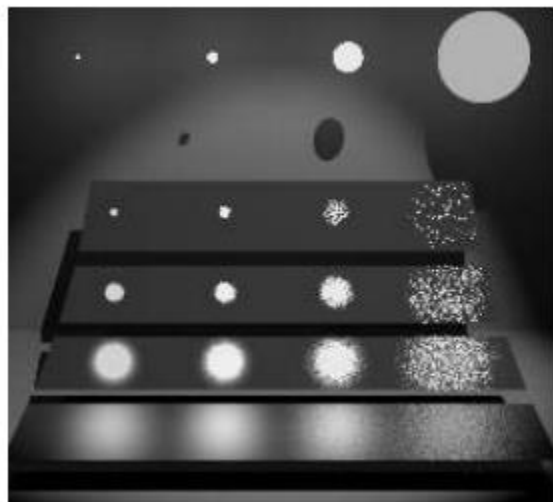
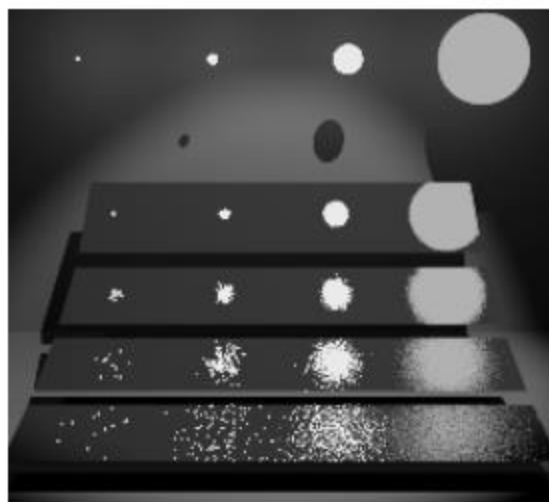


Figure 6: A simple test scene consisting of one area light source (i.e. a bright patch) and an adjacent diffuse surface. The images were computed by (a) sampling the light source according to emitted power, with $n_1 = 3$ samples per pixel, (b) sampling the hemisphere according to the *projected solid angle* $[3] \cos(\theta') d\sigma(\vec{\omega}')$ (see Fig. 1), with $n_2 = 6$ samples per pixel, and (c) a weighted combination of samples from (a) and (b) using the power heuristic with $\beta = 2$. (d) A plot of σ/μ (standard deviation divided by mean) as a function of distance from the light source, for $n_1 = 1$ and $n_2 = 2$.

Multiple Importance Sampling



Light sampling



BRDF sampling



Multiple importance
sampling

Path Tracing Extensions

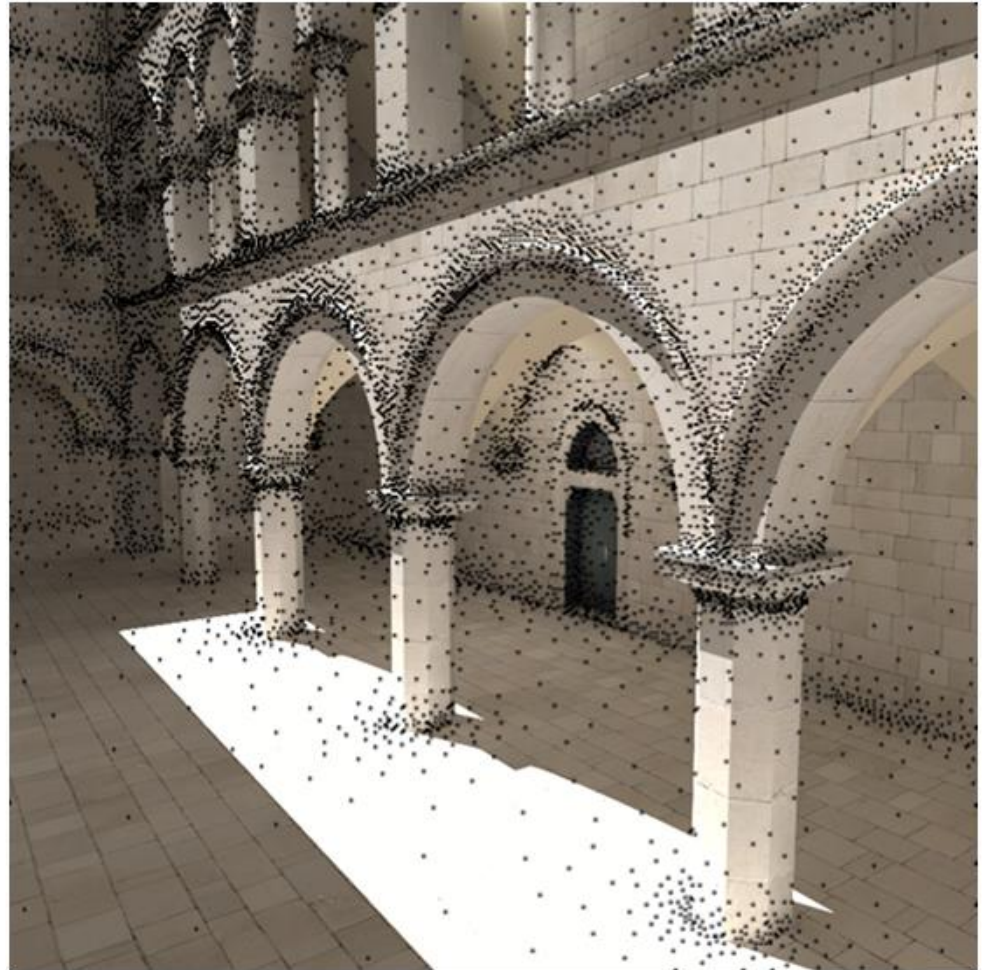
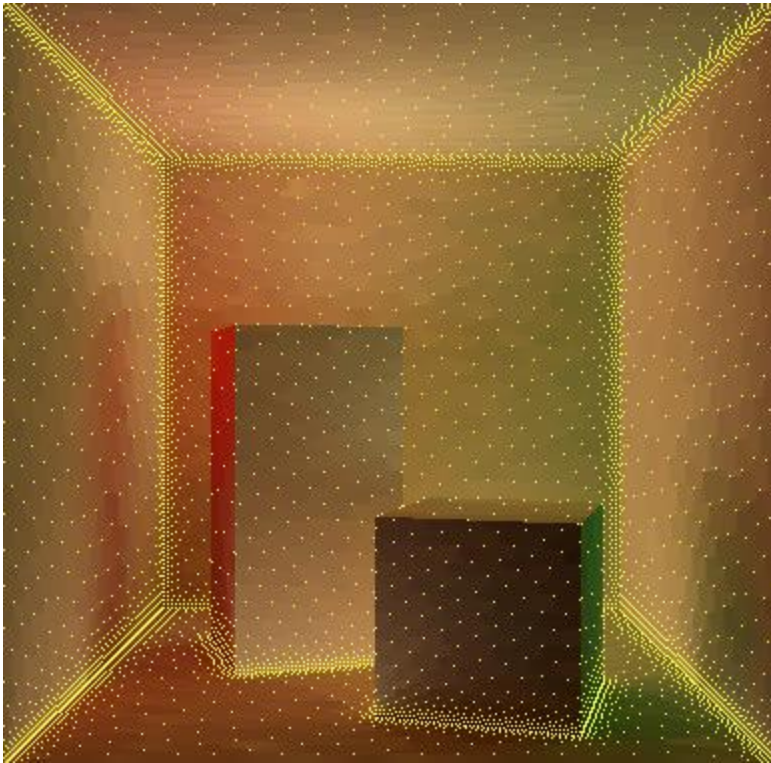
Path Tracing

- There are several standard extensions to path tracing that are typically used to get the most out of the technique
- We will look at some of the more popular methods

Irradiance Caching

- The direct illumination (LDE) in a scene typically has sharp discontinuities from shadows and angled surfaces
- The indirect illumination (LDD+E), however, is typically much smoother
- The idea behind irradiance caching is to compute and cache accurate indirect illumination at key points and use those cached values to interpolate
- There are many details in interpolating cached values and determining when to re-use nearby cache points and when to create new ones
- Irradiance caching was introduced by Greg Ward in 1988 and there have been many extensions and enhancements since then

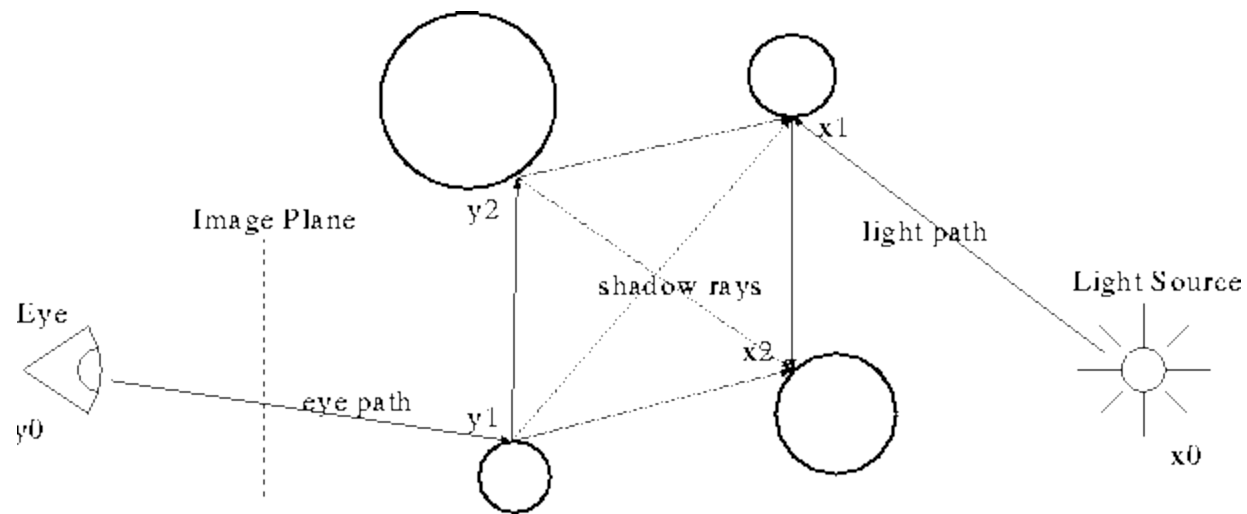
Irradiance Caching



Bidirectional Path Tracing

- *Bidirectional path tracing* (BPT) is an extension to path tracing
- A path is traced from both the camera and a light source
- Then, each vertex of the eye path is connected to each vertex of the light path to form all possible combinations of the two paths
- If the connecting ray is not blocked, then the path is added to the pixel, according to its weighted probability
- BPT is better at handling indirect lighting situations, and will generally do a better job than standard path tracing in most practical cases

Bidirectional Path Tracing



Bidirectional Path Tracing



Photon Mapping

- *Photon mapping* is a two-pass rendering system
- In the first pass, photons are emitted from the light sources and scatter through the environment
- At each bounce, they are either randomly reflected according to the BRDF of the surface or they are absorbed by the material, and their 3D position is added to the *photon map*
- Often, millions of photons are shot out, and need to be efficiently stored and managed
- In the second pass, the scene is path traced
- Certain components of the lighting are computed by with the standard path tracing approaches (such as direct lighting and specular reflections)
- Other components of the lighting can come from the photon map (such as highly specular paths like caustics)
- The photon map can also be used for the indirect diffuse components as well, and can be combined with a technique called *final gathering*
- Photon mapping combines with path tracing and can be balanced to gain the benefits that each approach offers

Photon Mapping



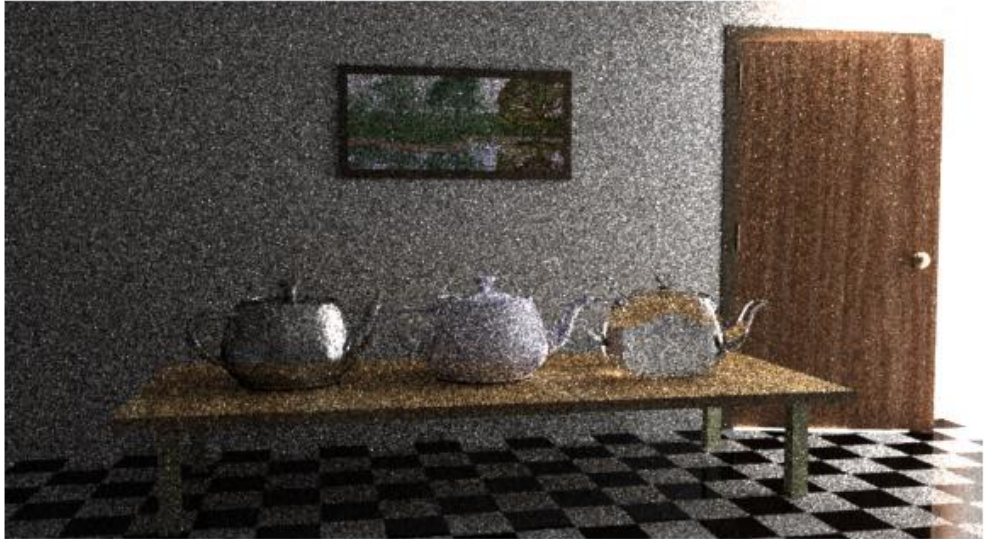
Metropolis Light Transport

- *Metropolis light transport* (MLT) is a rather exotic form of path tracing that has been gaining popularity lately, as it is notable for its ability to handle particularly complicated light paths
- For each pixel, an initial path is created that starts from the camera, bounces in the scene, and ends up at a light source
- Then, the path is modified (or *mutated*) according to a variety of possible rules
- Next, the path is either accepted or rejected according to a random decision, weighted by the relative contribution of the new path to the previous path
- The current path then adds its contribution to the pixel and the mutation process is repeated some desired number of times

```
x = x0
for i = 1 to n
    x' = mutate(x)
    a = accept(x, x')
    if (random() < a)
        x = x'
    record(x)
```

Metropolis Algorithm

BPT: 40 samples per pixel



MLT: 250 mutations per pixel



Ray Tracing References

- “An Improved Illumination Model for Shaded Display”, Turner Whitted, 1980
- “Distributed Ray Tracing”, Cook, Porter, Carpenter, 1984
- “The Rendering Equation”, James Kajiya, 1986
- “Bidirectional Path Tracing”, Eric Lafortune, Yves Willems, 1993
- “Rendering Caustics on Non-Lambertian Surfaces”, Henrik Wann Jensen, 1996
- “Metropolis Light Transport”, Eric Veach, Leonidas Guibas, 1997