CS667 **Lecture 8: Path Tracing and Photon Mapping I** 19 February 2004

Andy Scukanec Lecturer: Steve Marschner

1 Path Tracing

So far, all of the algorithms we have discussed have been able to handle certain cases of light transport, but not others. The types of paths that each algorithm could handle can be thought of being represented as some number of diffuse and specular bounces that the light goes through between its source and the eye (or camera if you prefer). This can be represented as a regular expression: L (S | D)* E.

Path Type	Algorithm
L D? S* E	Ray tracing
L D* S* E	Radiosity with Final Gather
L D* E	Radiosity without Final Gather
L D+ S* E	Path Tracing
L [(S | D)* D D]? D* S* E	Bidirectional Path Tracing

- Ray tracing can follow rays along an infinite number of specular bounces from the eye, then assumes that light interacts diffusely with the surface.
- Pure radiosity assumes everything is diffuse, and as a result can only handle paths that have diffuse bounces.
- Radiosity with a final gather allows any number of diffuse bounces to be followed by any number of specular bounces in going from the source to the eye.
- Path Tracing has issues with some particularly difficult to find paths: essentially it cannot usefully handle specular to diffuse transport paths.
- Bidirectional tracing tries to alleviate these problems by splitting the path into a prefix and a suffix. This allows important paths to be found in more cases.

As a reminder, we were able to formulate the rendering equation by using a rendering operator $\mathbb{R}$. This operator was itself defined as a function of two linear operators $\mathbb{K}$ and $\mathbb{G}$:

$$\mathbf{R} = (\mathbf{I} - \mathbf{KG})^{-1}$$

We arrived at this definition by simplifying the following equation for radiance transport:

$$\begin{aligned} L_e &= L_e^0 + \mathbf{KG}L_e \\ L_e &= (\mathbf{I} - \mathbf{KG})^{-1} L_e^0 \\ L_e &= \mathbf{R}L_e^0 \end{aligned}$$

By expanding $\mathbf{R}$ using the infinite series expansion, we can approximate the true operator using a sum of integrals, one for each power of $\mathbf{KG}$.

$$\begin{aligned} \mathbf{R} &= (\mathbf{I} - \mathbf{KG})^{-1} = \mathbf{I} + \mathbf{KG} + (\mathbf{KG})^2 \ldots \\ \langle W^0, \mathbb{R}L_e^0 \rangle &= \langle W^0, L_e^0 \rangle + \langle W^0, \mathbf{KG}L_e^0 \rangle + \langle W^0, (\mathbf{KG})^2 L_e^0 \rangle + \ldots \\ &= \int_{\mathcal{M}}\int_{H^2} W_e^0 L_e^0 + \int_{\mathcal{M}}\int_{H^2} W_e^0 \int_{H^2} f_r(\cdot) L_e^0(\cdot) + \int_{\mathcal{M}}\int_{H^2} W_e^0 \int_{H^2} f_r(\cdot) \int_{H^2} f_r(\cdot) L_e^0(\cdot) + \ldots \end{aligned}$$

This of course can be used directly in simulating probabilistic radiance transport. We pick a point in each domain, evaluate the transport along that path, and use that as an estimate of the integral. There is an analogy here to Markov Chains. The original path tracer blindly walked from the eye through the scene. Most path tracers do a direct illumination calculation at each bounce and use the above method to estimate indirect illumination effects. Bi-directional path tracers trace a path from the light and from the eye and join up each pair of segments. Still, there are some paths that are particularly pathological (ie – water in a pool, or the table underneath a wine glass, L S* D S* E).

2 Photon Mapping

Photon mapping provides a way to handle many of the paths that are difficult for path tracing, but introduce bias. The photon mapping solution is to first propogate rays from the light source and deposit 'photons' at hit points. During a second rendering pass, the indirect lighting can then be calculated at a point by estimating the photon density within some area near the point of interest.

There are some design issues to be considered - should the photons play russian roulette along their paths to determine whether to continue to propogate or not? Should the power of the photon be adjusted at each bounce? There are two basic ways to handle photon mapping:

1. At each diffuse bounce, play russian roulette to determine if you continue to propogate or not. This russian roulette probability is proportional to the directional-hemispherical reflectance of the BRDF the photon just hit. If the photon is not absorbed, do another bounce. In either case, deposit a photon.
2. At each diffuse bounce, deposit a photon with the current power. Then attenuate the power by the BRDF at that point and continue. Stop when the power is lower than some ϵ.

Note that in both options, specular bounces are handled without any depositing of photons or termination of the bounce.

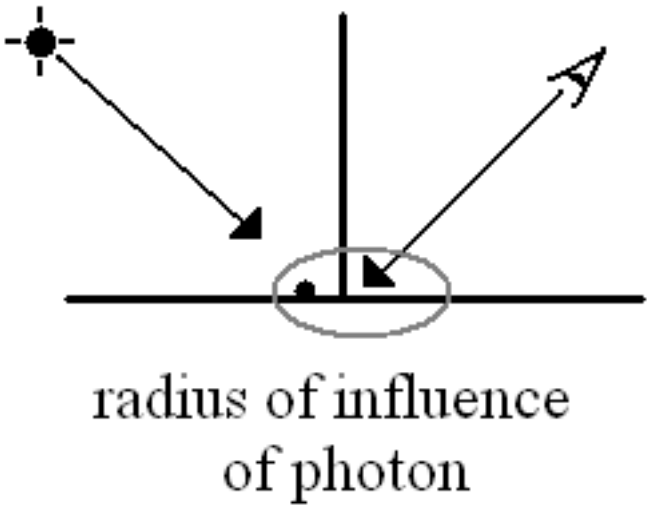
This technique will handle paths of the form L (S | D)* D+ S* E. Adding in a ray tracing solution (similar to the final gather performed for radiosity) would allow paths of the form L S* E as well.

Photon data to be stored, and the size:

Data Stored	Size
Position (in $\mathbb{R}^3$)	4 bytes * 3 = 12
Incident Direction (spherical)	1 byte * 2 = 2
Power (in RGBe format)	1 byte * 4 = 4
2 flag bytes	1 byte * 2 = 2
Total	20

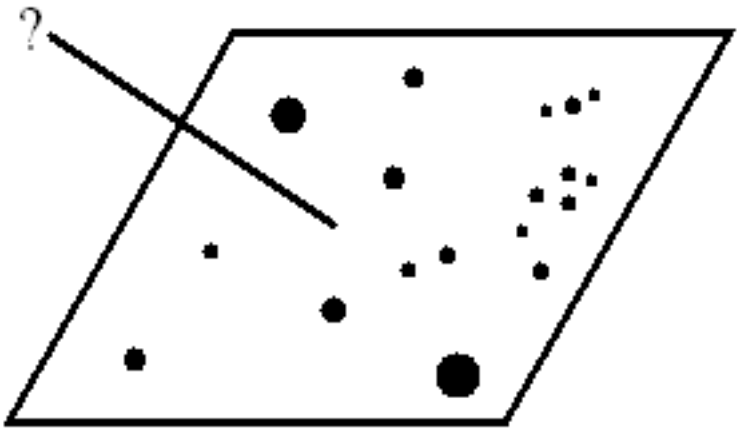
There can be problems in the photon mapping algorithm if the implementation does not consider what happens at corners and back sides of geometry. The second problem is easily solved by storing directions of the photon, and not counting its contribution if its 180° out.

The corner problem, illustrated by the picture below, is difficult to solve without remeshing the floor surface so that photons can't "bleed" across the wall (which is acting like a blocker).



2.1 Estimating Power using Photon Density

The problem of estimating power across a surface from nearby photons is really just a sampling theory problem. Given nearby evaluations of some function, we want to reconstruct our best guess as to what the function was at some point which is not necessarily a direct sample of the function. We can weight contributions using box or cone filters, or something more sophisticated like a gaussian, but its all really just sampling theory.



We want to obtain something like a weighted density estimation. We can divide the photons into bins to get a histogram of sorts, but this isn't really what we want.

We could average out the number of photons 'near' our point $\mathbf{x}$. This gives us a simple formula for computing the density of the photons ...

$$\tilde{p}(\mathbf{x}) = \frac{1}{N} \frac{\#\text{ of photons in A}}{|A|}$$

where N is the total number of photons. This is an estimate of the probability of a photon that started at the source winds up in the area A around a particular point. We could also come up with something a little bit more generic ...

$$\tilde{p}(\mathbf{x}) = \frac{1}{N}\sum_i h(|\mathbf{x} - \mathbf{x_i}|)$$

and put the restraint on h that it must integrate to 1. The first equation then is just a special form of the second. The second equation allows us to weight photons arbitrarily (presumably higher as they get closer to $\mathbf{x}$).

Finally, we have our estimate of irradiance, given the photon density, which can be converted to an estimate of radiance:

$$\begin{aligned} E(\mathbf{x}) &= \Phi\tilde{p}(\mathbf{x}) \\ &= \frac{\Phi}{N}\sum_i h(|\mathbf{x} - \mathbf{x_i}|) \\ &= \sum_i \Phi_i h(|\mathbf{x} - \mathbf{x_i}|) \\ L_e(\mathbf{x}) &= \sum_i f_r(\cdot)\Phi_i h(|\mathbf{x} - \mathbf{x_i}|) \end{aligned}$$

Note: $\mathbf{x}$ in the equations above is actually a 3D vector, meaning that there has to be some way of storing and retrieving arbitrary 3 dimensional points. A KD-Tree or some other acceleration structure works great for this sort of application.
Note: The Φ_i is for the weighted photon version, where the weight of each photon is attenuated at each bounce. For unweighted photons, each accounts for the power $\frac{\Phi}{N}$ where Φ is the total power of the source. For variable weight, we generalize this to a power Φ_i per photon; in the unweighted case $\Phi_i = \frac{\Phi}{N}$ for all i.