Path Tracing

Steve Marschner
Cornell University
CS 667 Spring 2007, 11 October

[These notes are meant to supplement the lecture and the text, and they focus on writing down details clearly rather than on high-level explanations.]

Path tracing is a recursive-sampling method for solving the rendering equation. In class I developed a sequence of path tracing algorithms, progressing from downright useless to reasonably capable. The version numbers here refer to the version numbers of the algorithms written in pseudocode on the lecture slides.

Version 0: brute force recursive sampling

The idea of path tracing is to use Monte Carlo to compute the illumination integral for a surface point, but to make a recursive call to get *all* radiance incident on the surface rather than just the direct radiance. This zeroth version uses uniform sampling in projected solid angle. It is arrived at as follows:

$$L_r(x,\omega) = \int_{H^2} f_r(x,\omega,\omega')\, L_i(x,\omega')\, d\mu(\omega')$$

integrand

probability = uniform $\frac{1}{\pi}$

$$g = \frac{f}{p} = f_r(x,\omega,\omega')\, L_i(x,\omega') \cdot \pi$$

recursively evaluated.

Version 0.5: Russian Roulette

The biggest problem with version 0 is that the recursion does not terminate. To make it terminate, without introducing an arbitrary depth limit, we can use *Russian Roulette*: when we are evaluating the integral we replace a fraction of the samples with zero (i.e. terminate some paths) and increase the weight of the remaining samples to preserve the mean. In the abstract, in terms of a estimator g over a random variable X, this is easy to state:

$g(X)$ an estimator for L

$$g'(X) = \begin{cases} \frac{1}{q} g(X) & \text{with probability } q \\ 0 & \text{with probability } (1-q). \end{cases}$$

$$E\{g'(X)\} = q \cdot \left(\frac{1}{q} g(X)\right) + (1-q) \cdot 0 = g(X)$$

and when we implement this idea in the path tracer we arrive at the algorithm 0.5 in the slides. It is identical to algorithm 0 except it sometimes doesn't make the recursive evaluation. As long as the probability of continuing is bounded away from 1, we have a program with a finite expected running time.

Version 0.75: BRDF sampling

We can improve things by doing importance sampling according to the BRDF rather than the uniform projected solid angle sampling.

Version 1.0: direct illumination

The first big improvement in variance comes from handling direct illumination specially. Just as we saw that sampling direct illumination using uniform sampling produces high variance, sampling the sum of direct and indirect in the same way produces high variance (at least when there are light sources in the scene—a scene lit by a uniform background will render just fine with BRDF sampling). To ket to Kajiya's algorithm we separate the integral into direct and indirect and use two samples:

$$L_r(x,\omega) = \int_{H^2} f_r(x,\omega,\omega') \left[L_i^\circ(x,\omega') + L_i^+(x,\omega')\right] d\mu(\omega')$$

$$= \underbrace{\int_{H^2} f_r(x,\omega,\omega') L_i^\circ(x,\omega') \, d\mu(\omega')}_{\text{sample according to luminaires} - p_L} + \underbrace{\int_{H^2} f_r(x,\omega,\omega') L_i^+(x,\omega') \, d\mu(\omega')}_{\text{sample according to BRDF} - p_B}$$

$$g_L = \frac{f^\circ}{p_L} = f_r(\omega,\omega') L_i^\circ(\omega') / p_L(\omega') \quad ; \quad g_B = \frac{f^+}{p_L} = f_r(\omega,\omega') L_i^+(\omega') / p_B(\omega')$$

This means we trace two rays, one by luminaire (L) sampling and one by BRDF (B) sampling. The L ray goes toward a luminaire and its radiance value is the *emitted* light from its direction. We don't recurse on the L ray (called a *shadow ray*). The B ray (the *indirect ray*) goes in some arbitrary direction (maybe toward a luminaire, maybe not) but in either case its radiance value is

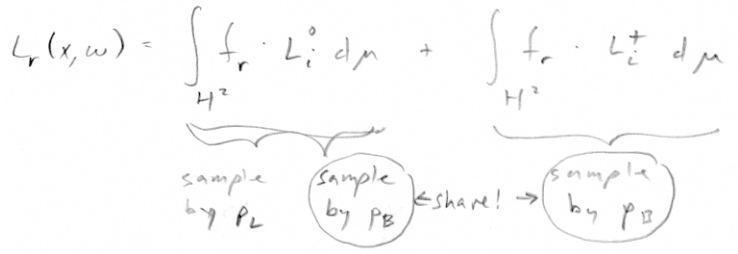
the *reflected* light (recursively estimated) of the surface it hits. We don't include emission in the B rays. In the example code on the slide, this is done by having the caller trace the ray and then call reflectedRadianceEst (rather than rayRadianceEst, which would have included emitted light).

Version 1.0m: direct by multiple importance

We got the best (or at least most robust) results for direct illumination by using multiple importance sampling to combine luminaire and BRDF sampling. There's nothing to stop us from going ahead and pasting that same code into the direct lighting evaluation in our path tracer. That's what I've done in version 1.0m on the slides.

Version 1.1: sharing the BRDF ray

The only problem with version 1.0m is that it is doing extra work tracing BRDF rays. For each reflection it generates *three* rays by the time it is done: an L and a B for direct, and then later another B for indirect:



This is wasteful, because those two samples don't need to be independent. They are not samples of the same estimator; they are samples contributing to two estimators we are adding together. So we can save work by tracing a single B ray and using it to sample both emitted and reflected light. The weighting is important: the contribution of emitted light is weighted against the luminaire sample using Veach & Guibas's balance heuristic, whereas the contribution of reflected light is just normalized as its own separate estimate and added in.

Doing this in the pseudocode results in a monolithic reflectedRadianceEst function that is perhaps harder to read, but performs well.

Note that none of these methods will do a good job of sampling paths that undergo specular transport between a small light source and a diffuse surface (that is, "caustic" paths).