Multiple Importance Sampling

Steve Marschner
Cornell University
CS 5630 Spring 2026

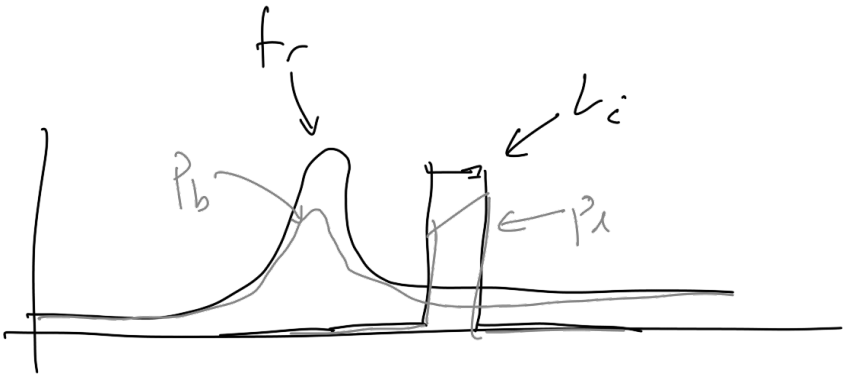
In computing local illumination (and in other cases) we have an integration problem with at least two plausible sampling strategies: sampling the BRDF and sampling the lights. In some cases one of these works pretty well, in some cases the other. For instance, rendering on an overcast day works pretty well with BRDF sampling: this will remove most of the variation from the integrand, since the incident radiance doesn't vary that much. On the other hand, if our scene has all diffuse surfaces then sampling light sources works well (if we do it well); it leaves only the variation in the integrand due to the cosine factor, which is usually relatively minor.

But there are cases where neither approach works well—for instance on a sunny day in a scene where surfaces are sometimes shiny. In that case we will find that in some areas of the image the light source sampling works best whereas in others the BRDF sampling is best. In the 90s this was a big problem for renderers and people tried all kinds of heuristics to switch sampling schemes, but now there is a general solution available that works well enough in a wide range of situations, known as multiple importance sampling, or MIS. The technique and the name come from a paper by Eric Veach in 1995.

Sampling from the average PDF

MIS is about intelligently taking weighted averages of the estimates from different estimators, in a way that helps ensure variance will be under control as long as any of the estimators is good for a particular feature of the integrand. Although there is a bit more to MIS, the simplest form of it, which Veach called the "balance heuristic" can be understood in simple terms, and that's what I'll describe here.

Suppose we have an integrand that is a product of two functions f_r and L_i which can sometimes be very peaky (I'm thinking of BRDFs and light sources but drawing in 1D):



We have pdfs p_b and p_l available that are good for sampling either of these things (maybe approximately). But in cases where most of the variation comes from one or the other of these, using the wrong pdf will produce very high variance since it hits high values with low probability. To avoid having to choose, why not use the average of these two pdfs? This is easy to implement: flip a coin, then sample from one of the two pdfs. The pdf of samples generated by this process is $(p_b + p_l)/2$ and the corresponding estimator is:

$$g_m(\omega) = \frac{f(\omega)}{p(\omega)} = 2\frac{L_i(\omega)\, f_r(\omega)\, \omega \cdot \mathbf{n}}{p_b(\omega) + p_l(\omega)}$$

This is different from the two estimators I would otherwise have to choose between:

$$g_b(\omega) = \frac{f(\omega)}{p_b(\omega)} \quad ; \quad g_l(\omega) = \frac{f(\omega)}{p_l(\omega)}$$

It's surprising what a small change to the computation this is … the only difference is that we divide by a somewhat different pdf. But it's an important difference.

Why does this work? One way to see it is to think about what goes wrong when we have a poor match between the integrand and the sampling pdf. If the pdf was perfect—that is, a scalar multiple of the integrand—then every sample would produce the same value for the estimator; the variance would be low. Where the pdf is too high, the estimator will be below the mean, and the sampling procedure will produce extra samples in this area to compensate. Where the pdf is too low, the estimator will be above the mean and the sampler will produce fewer samples. These two cases are quite different, because the integrand is nonnegative. Too-high pdfs can produce at worst samples with zero contribution, or 100% error. But too-low pdfs can produce arbitrarily large estimator values, and one sample whose value is 1,000 times too high requires at least 1,000 samples whose value is too low to balance it out. So the real problem here is under-sampled peaks in the integrand.

Sampling from the average pdf can solve this effectively. If the integrand can only be high when at least one of the two factors is high, and the two pdfs are proportional to the two factors, then whenever we have a high value of f, one or the other of p_b or p_l will be high, so that we don't get the high-value-over-low-pdf that leads to the outlier sample values that produce high variance in images. Averaging the two separate estimators with equal weights would not achieve this.

Examples

Suppose we're in a sunny outdoor environment where there's a Lambertian diffuse sidewalk that's partly in the sun and partly in the shade of a building. In the shade, sampling the BRDF (aka. sampling uniformly in solid angle) is a good strategy since the environment doesn't vary that much in radiance. But using this strategy all the time is really bad, though: if we sample solid angle in the sunny area, about 1 in 10,000 rays will hit the sun, where the radiance is maybe a million times higher than in a typical direction. Since the pdf is the same, the estimator for these samples is a million times higher than the rest. We would need about a million samples to get the relative standard deviation down to 10%.

If we instead sample directions towards the sun half the time, and uniform directions half the time, about half the rays will hit the sun, and because the pdf for generating directions towards

the sun is high, the estimator will be reasonable even when we hit the sun "by accident" while sampling from the uniform distribution.

Weighted combinations of samples

With MIS it is common to take several samples at once, for instance one from each of two estimators, and then combine them. This is exactly the same as sampling the combined distribution twice, using a stratified random choice to decide which one to select. If we look at what happens here, the combined estimator is the average of two samples:

$$\frac{1}{2}\left(2\frac{f(\omega_1)}{p_1(\omega_1)+p_2(\omega_1)}+2\frac{f(\omega_2)}{p_1(\omega_2)+p_2(\omega_2)}\right)=\frac{f(\omega_1)}{p_1(\omega_1)+p_2(\omega_1)}+\frac{f(\omega_2)}{p_1(\omega_2)+p_2(\omega_2)}$$

We can write this as a weighted average of the two separate estimators:

$$\left(\frac{p_1(\omega_1)}{p_1(\omega_1)+p_2(\omega_1)}\right)\frac{f(\omega_1)}{p_1(\omega_1)}+\left(\frac{p_2(\omega_2)}{p_1(\omega_2)+p_2(\omega_2)}\right)\frac{f(\omega_2)}{p_2(\omega_2)}$$

or

$$w_1(\omega_1)\frac{f(\omega_1)}{p_1(\omega_1)}+w_2(\omega_2)\frac{f(\omega_2)}{p_2(\omega_2)}$$

Since we can see that $w_1+w_2=1$, this is a convex linear combination of the two estimators, and since both estimators have the same mean (i.e. the value of the integral), the average also has that mean. So it is a valid estimator for our problem.

Indeed, any other scheme for selecting weights, as long as they sum to 1, will produce a valid estimator. Veach explores several of these, the most successful of which is the "power heuristic":

$$w_i^{\text{power}}=\frac{p_i^\beta}{\sum_j p_j^\beta}$$

The original one (with $\beta=1$) is called the "balance heuristic."

Implementation in practice

The desire to implement MIS leads to a feature of sampling code that wasn't needed before MIS. In a basic single-estimator setup, you need a sample and the value of the sampling pdf at the same point, never at a different point. But for MIS the complete computation requires the value of the pdf that *did* generate the sample and also the value of the other pdf that *might have* generated that sample.

This leads to somewhat expanded interfaces for objects in renderers, like lights and BRDFs, that are responsible for sampling strategies. For instance a BRDF with no need for MIS might have an interface like:

```
BRDF {
  eval(ω1 : dir, ω2 : dir) → f : 1/sr
  sample(ω1 : dir, seed : vec2) → [ω2 : dir, g : unitless]
}
```

where the sampling routine uses a sampling pdf that resembles the BRDF and returns both the selected direction and the value of f/p for that direction. It also supports evaluation because this is needed in contexts where the direction is not chosen by the BRDF (e.g. when the direction is chosen by a light source). Note that since only that ratio is needed to construct the simple importance-sampled estimator, the caller doesn't actually know or care about f or p separately.

On the other hand, to do the MIS calculation you have to have the pdfs of both sampling strategies for both samples. For this we extend the sampling function so that it reveals the pdf value and add a third method that can compute the pdf even when it didn't pick the direction.

```
BRDF {
  eval(ω1 : dir, ω2 : dir) → f : 1/sr
  sample(ω1 : dir, seed : vec2) → [ω2 : dir, g : unitless, p : 1/sr]
  pdf(ω1 : dir, ω2 : dir) → p : 1/sr
}
```

A similar setup applies to area lights which sample by area

```
AreaLight {
  eval(y : pt, ω : dir) → L : radiance
  sample(x : pt, seed : vec2) → [y : pt, g : intensity, p : 1/area]
  pdf(y : pt) → p : 1/area
  N(y : pt) → vec3
}
```

For area lights we need the surface normal to the light, here returned by a separate method.

MIS for direct lighting

To have a basically well-performing direct lighting calculation that is reasonably robust to a range of BRDFs and light source setups, you need MIS to combine the strategies of sampling from the light and sampling from the material (BRDF).

If we look at some pseudocode for each of these strategies separately, we'll see it's a matter of combining the two and adding the computation of MIS weights. If we are just sampling from lights, here is a representative shading method:

```
Color shade(x, V, brdf, N, seed) {
    result = black;
    g_y, y, p_y = light.sample(x, seed);
    ω = normalize(y – x);
    if visible(x, y) {
        G = (–ω · light.N(y)) / distSqr(x, y);
        g = g_y * G;
        f_r = brdf.eval(V, ω);
```

```
            result += g * f_r * ω · N;
        }
        return result;
    }
```

And for sampling from the BRDF:

```
    Color shade(x, V, brdf, N, seed) {
        result = black;
        g, ω, p_ω = brdf.sample(V, seed);
        y = light.intersect(x, ω);
        if (y and visible(x, y)) {
            L = light.eval(y, –ω)
            result += L * g * ω · N;
        }
        return result;
    }
```

The MIS code is almost just the concatenation of these two with some deduplication of names, plus computing the MIS weights:

```
    Color shade(x, V, brdf, N, seed) {
        result = black;
        g_y, y, p_y = light.sample(x, seed);
        ω_l = normalize(y – x);
        if visible(x, y) {
            G = (–ω_l · light.N(y)) / distSqr(x, y)
            g_l = g_y * G;
            p_l = p_y / G;
            f_r = brdf.eval(V, ω_l);
            p_b = brdf.pdf(V, ω_l);
            w_l = p_l / (p_b + p_l);
            result += w_l * g_l * f_r * ω_l · N;
        }
        g_b, ω_b, p_b = brdf.sample(V, seed);
        y_b = light.intersect(x, ω_b);
        if (y_b and visible(x, y_b)) {
            G = (–ω_b · light.N(y)) / distSqr(x, y_b);
            L = light.eval(y, –ω_b);
            p_l = light.pdf(y_b) / G;
            w_b = p_b / (p_b + p_l);
            result += w_b * L * g_b * ω_b · N;
        }
        return result;
    }
```

The bold parts are the ones that are substantially different. There is some messiness in this code because the two pdfs don't start out in the same measure—one is area on the light, the other is solid angle at the shading point. All the geometry factors (Gs) can be moved out of this code if

we make the light return its pdf in solid angle instead of area, or build some kind of adapter that provides that interface so that this code can be all in terms of solid angles.