1 Bidirectional Path Tracing

The Light Transport equation is a self dual, or a self adjoint, process that can swap the incoming and outgoing directions and get the same result. This property is called reciprocity and this symmetric property is a fundamental point of light transport. We have the equation:

$$L_r = KGL_r + L_o$$

K is the BRDF or our rendering operator, which is reciprocal. G is the visibility, which is obviously symmetric. We can think of this equation in terms of symmetric matrices and we get:

$$L_r = (1 - KG)^{-1} L_o$$
$$L_r = ML_o$$

This means that the equation boils down to one operator that is self-adjoint.

When we are tracing a ray, we are figuring out values for a pixel. The value for a pixel v_i is computed by integrating the reflected radiance weighted by a weighting function w_i:

$$v_i = \int\int w_i(x, w) L_r(x, w) dA d\mu$$
$$v_i =< w_i, L_r >$$

From before, we saw that $L_r = ML_o$, so we can substitute and get:

$$v_i =< w_i, ML_o >$$
$$v_i =< M^* w_i, L_o >$$

We see that the symmetric term can be applied to either side, since it's symmetric. We can think of this as all light sources weighted by M or as shooting light from the viewer through a pixel and weighted by M according to L_o, the light emitter.

Now we have the tools to take the path tracing algorithm and substitute:

> From eye, trace rays proportional to w_i. For each intersection, send ray to light and add the BRDF times emission and then send ray proportional to BRDF, then recurse.

This is good for viewing through glass, but not for the caustics caused by glass. If we shoot a ray where a caustic would be, we would shoot a ray to the light source and see that it would be blocked by the glass object or would hit the light source itself. We wouldn't be able to figure out the path that would cause the caustic.

If we took the above statement and substituted L_o for w_i and we send the ray to the eye instead of light, we would get a method that performs well for caustics, but not for glass:

> From light, trace rays proportional to L_o. For each intersection, send ray to eye and add the BRDF times importance and then send ray proportional to BRDF, then recurse.

This is because for caustics, we would trace rays from the light that took the correct paths through the glass object and create caustics. What we want to do is called splatting. Each pixel that we render has a set of directions that we want to know the values for. Trace a path from light and see what rays we want and the pixels they correspond to.

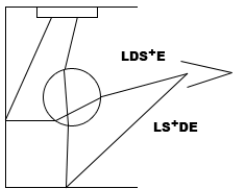

Figure 1: Different Paths

Now we have two methods, one's good for the LDS^+E path and the other is good at the LS^+DE path. Since these two paths are dual, we can use multiple importance sampling to get the best of both worlds. This is (almost) bidirectional path tracing.

To get the full bidirectional path tracing algorithm, we generalize this to allow paths to be generated from both ends at once. Care just needs to be taken to make sure that more than one bounce is taken from either side. Once both pathes are are diffuse surfaces, we can then just join them up.

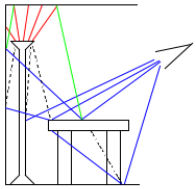

Figure 2: Exception Case; Everything in eye view needs at least two bounces to be illuminated

2 Metropolis Light Transport

The basic idea is that we use a random walk through the distribution that we want, and with enough random walks, we get what we are looking for. The system is setup as a Markov Chain, with the probabilities only relying on the previous state. We have a domain Ω, with a sample sequence $(x_1, x_2, x_3, ...)$ and transition probability of $K(x|y)$, which it the probability of $x_{i+1} = y$ given $x_i = x$. The formal equation:

$$p_i(x) = \int K(x|y)p_{i-1}(y)dy$$

To apply this to ray tracing, we have some path in the scene, and we just mutate the path. This might seem silly at first, but for scenes where the only source of light is a door in the room, and no rays directly go into that room, this method works considerably better than other heuristics. This is because when it finds a path into the other room, it can hold on to that path and mutate it and get good samples. For simple scenes, it's not as fast as other methods and is obviously harder to implement, since it's hard to keep track of all of the mutations.

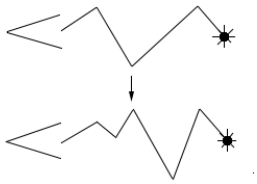

Figure 3: Mutation of Path