Recursive Ray Tracing

OUTLINE:

Classical Recursive Ray Tracing

Distribution Ray Tracing

Classical Recursive Ray Tracing

In a **ray casting** algorithm, for each pixel you shoot one ray from the eye through the center of the pixel, determine what it hits, and shade that intersection point. Total: one ray per pixel.

In a **classical recursive ray tracing algorithm** (see *Whitted, Communications of the ACM, June 1980*) you shoot one ray from the eye through the center of each pixel, determine what it hits, and trace additional rays toward light sources and recursively trace rays in specularly reflected and specularly transmitted directions, if any. This results in a **ray tree**. To prevent infinite recursion, you stop when ray depth exceeds some limit (maybe 3 or 5) or the coefficient (weight) drops below some threshold.

If there are L light sources, and maximum ray depth is D, then how many rays per pixel can you end up tracing, worst case?

With all of these extra rays, spatial subdivision optimizations for ray-surface intersection testing become even more important.

What Have We Assumed?

In classical recursive ray tracing, you assume:

assumption → consequence

trace one ray per pixel → we get aliasing.

smooth surfaces → specular reflection and transmission distribution functions have a narrow peak as a function of direction (light is reflected/transmitted in a very small cone of directions) so they are well approximated by a delta function, hence you can trace one ray to simulate each of them. We can't simulate rough specular surfaces well.

point light sources → can determine if a point is directly illuminated by a given light source by tracing a single ray toward the light. We can't simulate linear or area light sources.

pinhole camera → all rays pass through eye (camera) point. We can't simulate depth of field.

scene is static and/or shutter is instantaneous → objects don't move during frame time. We can't simulate motion blur.

Distribution Ray Tracing

In distribution† ray tracing:

To antialias, trace several rays per pixel (with jittering or other good random pattern).

To simulate rough surfaces, trace several rays throughout the cone of directions from which light is specularly reflected or transmitted.

To simulate penumbra, trace rays throughout the cone of directions subtended by the light source, averaging the results.

To simulate depth of field, trace rays from within a disk approximating the aperture of the camera.

To simulate motion blur, use a scene data structure that records motion information, and trace rays throughout the frame time.

More rays -- more motivation to use spatial data structures to optimize ray-surface intersection testing!

† a.k.a. "distributed" ray tracing, stochastic ray tracing