# Caching Secondary Rays for Fast Approximation of Diffuse Global Illumination

Lawrence Kesteloot

January 20, 1995

**Abstract**

This paper describes a better calculation of the ambient term for ray-traced diffuse surfaces. This is done by ray-tracing the whole scene from each intersection point over a hemisphere and averaging the resulting image to estimate the ambient term. Since the ambient term is most useful for first-generation rays, only simple path-tracing is used for rays of second and later generations. In addition, since first-generation rays' intersections with objects have a good amount of locality, previous data is used to optimize the hemisphere calculation at each pixel.

## 1 Introduction

Traditional ray-tracing[2] uses a constant factor to estimate the ambient light's contribution at a given surface point. Kajiya used path-tracing[1] to estimate the global illumination term using Monte Carlo techniques. To generate his example pictures he shot 40 paths through each pixel. This amount of oversampling is necessary to reduce the variance in the Monte Carlo sampling, but is fairly computationally expensive.

I propose to calculate the illumination at a surface point by ray-tracing the entire scene from that point. The luminance at that surface point is the average of the luminance in the scene, weighted by the bidirectional reflection distribution function (BRDF). If surfaces in the scene are diffuse or near-diffuse, then only the first generation needs to be computed this way, since there is little or no specular term to be calculated. (A perfect mirror, for example, would require us to perform the ambient calculations for the second-generation rays after having reflected off the mirror.)

Ideally, for a diffuse surface we would integrate over the hemisphere at the surface point in question:

$$I = \int_0^{\frac{\pi}{2}} \int_0^{2\pi} L(\phi, \theta) \cos\theta \sin\theta \; d\phi \; d\theta$$

or, equivalently,

$$I = \int_0^1 \int_0^{2\pi} L(\phi, \sin^{-1} z) \cos\theta \; d\phi \; dz$$

I break the hemisphere into cells using the latter formula, thus avoiding the former's oversampling of the sphere's apex and compensation with the seperate sine term. A jittered ray is shot through each cell and evaluated using path tracing for a few generations to get a good approximation of the energy coming through that cell. The cells, weighted by the Lambertian cosine term, are averaged and the result is used as the illumination for the surface point for the first generation ray.

Notice that the intensity returned by the ray through a cell is not multiplied with the traditional $1/r^2$ term, area of the light source, or any other geometric factor—the ray's intensity is used directly, modulated only by the Lambertian cosine term. This is because the sampling of the hemisphere takes into account the distance to the light as well as the light's orientation and size. (Is it more accurate to say that the $1/r^2$ term and other geometric factors try to approximate the light's solid angle on the hemisphere, and here the ray represents the solid angle of the cell, not of the entire object that the ray intersected.)

## 2 Intersection Cache

If the hemisphere is tesselated into sufficiently many cells, a small amount of error in the calculation of a few cells does not significantly affect the resulting average. We can optimize the calculation of the hemisphere's average by caching intersection points of previously traced paths along with the paths' color. For each first-generation ray, I find the orientation of the hemisphere of the closest intersecting surface and project all the points in the cache onto the hemisphere, storing the cache entry's color in the appropriate cell. If several cache points fall into the same cell, the points that are closest to the surface point are kept and the rest are not used for this hemisphere. If any cell on the hemisphere has no cache points projected onto it, a new ray is shot through it and its result is added to the cache.

One problem with this approach is that much potentially useful information is lost when several cache points project onto the same cell. This can be solved by tesselating the hemisphere more finely, but many more cells will have no cache points at all, greatly increasing the computation for the hemisphere. A good compromise is to have the hemisphere be tesselated into super-cells, with each super-cell composed of, say, 2x2 cells. A new ray is only shot through a super-cell if the entire super-cell is empty. This approach allows the hemisphere to be tesselated more finely without incuring the cost of more traced rays. The cells in each super-cell are averaged together before averaging all the super-cells so as to not bias super-cells with more samples.

# 3 Noise Reduction

The above algorithm illuminates surface points for first-generation rays using point sampling, and is therefore subject to the usual aliasing artifacts. This is not very noticeable when the surface point is not in direct view of a light source because the intensities in the cells have low variance. Surface points that are in direct view of light sources, however, are likely to receive the majority of their intensity from the light source. Cells mapped to lights may be several dozen times brighter than cells mapped to non-emitting surfaces, and sampling differences between neighboring screen pixels can result in noticeable noise in the image.

Since most of the noise comes from sampling the light source, we can replace the area under the integral curve that corresponds to the lights with an analytical estimate:

$$I = I_s + \sum_{\text{lights}} g \frac{I_l A \cos\theta'}{2\pi r^2}$$

where $I_s$ is the average of the hemisphere without the lights' contributions, $g$ is a geometric term representing visibility, $I_l$ and $A$ are the intensity and area of the light source, $\theta'$ is the angle between the ray to the light source and the light's normal vector, and $r$ is the distance to the light. (Notice that here we do use the geometric factors because the ray is not random or evenly distributed across the hemisphere but rather aimed directly at the light source.) Distribution ray-tracing is used here to simulate penumbras, with 10 rays shot to each light. Removing the light's contribution to the hemisphere is implemented by clipping each cell's intensity to 1, with the assumption that most non-emitting surfaces would contribute less and most lights would contribute more. The resulting hemisphere is now acting as a better approximation to the tradition constant ambient term.

# 4 Results

The test model (18 polygons) was a white room with a square light suspended from the ceiling and a colored cube on the floor. For an image of size 512x512, brute-force path-tracing with 40 paths per pixel and 5 generations per path took one hour on an HP700 workstation and cast 100 million rays altogether, including those biased towards the light source. No variance-reduction techniques such as hierarchical sampling were used; these would have greatly reduced the noise at the cost of an increased rendering time.

My algorithm took twice as long on the same model and workstation and at the same resolution, but shot one-fourth as many rays. If the scene were more complicated, then the fewer rays shot by my algorithm would reduce the gap in the running times. An average of one percent of the sphere cells had to be traced

at each pixel. (Notice that the cache-hit ratio increases with larger images since screen pixels have more locality in first-generation surface intersections.)

A more complex scene (161 polygons) of a large hall with a table and two sofas, six light sources (four windows and two spot lights), took only a little longer to render (three hours) and the results of diffuse-diffuse interactions were clearly visible on some walls and pillars, under the table, and in dimly lit areas. The noise in bright areas could have been reduced by using a variance-reducing scheme when randomly sampling the light sources.

# 5 Future Work

In my implementation, the hemisphere is tesselated equally in azimuth and zenith, which causes each cell to be four times wider than it is tall. Sampling four times more in azimuth will cause the cells to be fairly square, which may reduce the sampling noise.

The cache was allocated to hold twice as many points as there were cells on the hemisphere. Too small a cache causes too many new rays to be cast and too big a cache wastes time projecting points that overlap in cells. A wide range of cache sizes should be tested to find the optimum cache-hit ratio for the work done.

My cache-projection loop was not optimized and probably took up the majority of the time with two square roots and one arctangent. A simple table-lookup would probably greatly speed up the algorithm, as would using a hemicube instead of a hemisphere.

Instead of keeping a cache, a standard image-warping algorithm could be applied from the previous pixel's hemisphere to the current one. This would be faster than projecting the cache but might result in more empty cells.

# 6 Acknowledgements

I would like to thank Leonard McMillan for the original idea for this algorithm and countless helpful suggestions during its implementation, and Robert Wheeler and Jonathan McAllister for many useful discussions on form factors and the bahavior of light.

# References

[1] Kajiya, J. T. "The Rendering Equation," *Computer Graphics 20,* 4(1986), 143–150.

[2] Whitted, T. "An Improved Illumination Model for Shaded Display," *CACM 23,* 6(1980), 343–349.