

Graph-based Global Illumination

Brian C. Ricks · Parris K. Egbert

Received: date / Accepted: date

Abstract Global illumination algorithms are used in numerous commercial and academic settings; however, these algorithms are still considered slow. We show that ray casting global illumination algorithms could converge faster by taking advantage of redundant ray casts, and we propose a technique for quickly identifying and reusing these ray casts. Graph-based global illumination, our general-purpose ray casting optimization, uses these variance-reducing techniques to accelerate rendering. Unlike other algorithms that reuse information, our algorithm can handle all BRDF types and does not have noticeable artifacts. By optimizing path tracing with graph-based global illumination, on average the new algorithm converges in half the time.

Keywords Global Illumination Optimization · Monte Carlo Methods · Path Tracing Optimization

1 Introduction

Global illumination research has improved the visual quality of rendering and has had some success in reducing rendering times, yet global illumination algorithms are still considered slow. Many light sample calculations done while approximating the rendering equation are repeated, and this information is rarely reused.

We believe the intelligent reuse of light samples will be a driving force in future global illumination research, and we have developed a new, general-purpose global

illumination optimization to do so. Our contributions include our new technique, which allows repeated light samples to be reused to reduce image variance, as well as our optimization techniques, which allow this to be done more quickly.

2 Light Sample Reuse

Kajiya [8] used path tracing as an unbiased way to sample the rendering equation. We have found that with path tracing and its numerous extensions many light sample calculations are repeated or are highly similar. Reusing these samples will speed up rendering, with the theoretical capacity to speed up convergence in an n^2 fashion.

As an example, consider Figure 2 showing light paths starting at different pixels. Both light paths intersect near surface point β and then sample the light. When the algorithm finishes, there are two samples used in determining how much direct illumination arrives near β . If this information is not shared, each light path only has one sample. If this information were shared, each light path would have access to two light samples, as if there had been four light samples total. If a third light path intersected near β and this information were shared, the three direct light samples could be used in all the three light paths, as if there were nine ray casts. If n light paths intersect near the same surface point, then each light sample could be used n times, resulting in an n^2 growth in overall information. Reusing light samples in this way can rapidly increase information, thus reducing the variance in the rendered image.

B. Ricks
Brigham Young University, Provo, Utah, USA
E-mail: bricks@byu.net

P. Egbert
Brigham Young University, Provo, Utah, USA
E-mail: egbert@cs.byu.edu

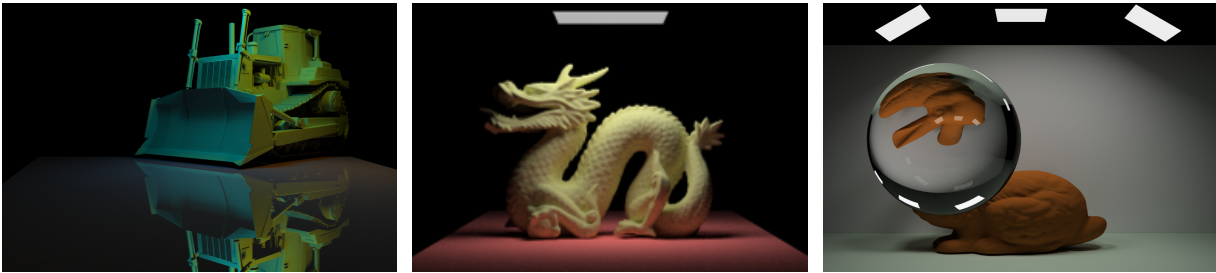


Fig. 1: By reusing ray casts, graph-based global illumination renders these and other images with reduced variance.

3 Implementation

We call our optimized path tracer graph-based global illumination and show that our optimization speed-up its convergence. Graph-based global illumination turns path tracing into a two pass biased Monte Carlo algorithm: First, visibility information is calculated and stored, and second, radiance transfer values are calculated and reused to generate an image. Our experiments show that optimizing path tracing in this way almost doubles its convergence speed.

3.1 First Pass: Storing Visibility

The first pass calculates visibility as it recursively shoots rays into the scene and stores the information for reuse later. Our optimization focuses on identifying close points without adding meaningful bias and on storing visibility for easy reuse.

We discretize our scene into axis-aligned voxels (dashed circles in Figure 2) with the assumption that points in the same voxel can share visibility information. To prevent n^3 memory consumption, voxels are created dynamically, and since we find voxels around surfaces in a scene, this means we consume only about n^2 memory usage if there is no participating media. When a ray collides with a point, the algorithm determines which voxel that point would be in if the voxel existed, and that voxel’s center becomes the key to a hash. If a voxel already exists at that location, it is used to store information; otherwise, a new voxel is created and added to the hash table. Each voxel stores pointers to the other voxels to which it is visible. This generates a sparse visibility graph where entry (vx, vy) indicates whether vx and vy are visible.

In Figure 2, say that voxel $v1$ surrounds β . As our algorithm proceeds, the first light path would collide with β while calculating the first light path. Since there is no voxel for β , $v1$ would be created. The second light path then collides near β . Since β is already contained in $v1$,

this second light path would use $v1$ to store the results of future ray casts. Since $v1$ is in both light paths, both light paths would automatically know about both direct light samples. Voxels were only used to store similar ray casts, they do not affect the starting point of future ray casts.

3.2 Second Pass: Generating a Final Image

The second pass runs recursively starting at the camera and uses information in the visibility graph to quickly calculate a final image. For Figure 2, the second pass would start at the left pixel and use information from the first pass to determine if it is visible to point α . Searching the voxels visible to α , the recursive function would find that β had been found when doing a ray cast starting at α . The recursive function would find two light samples in voxel $v1$. The direct lighting of β would then be calculated, cached, and used to calculate the radiance transfer from β to α and then to the camera pixel. When calculating the right pixel, the recursive function would find that the radiance from the light source to β had already been calculated, so the recursion would stop and the cached value would be used.

Notice the optimizations in our method: instead of only two direct light samples in the two light paths, the equivalent of four direct samples were used, thus increasing the information available in an n^2 way. Also, the recursion in the second light path stopped early because of the caching. If in the example more light paths had intersected in $v1$, the optimization would be even more dramatic. Note also that radiance transfer information can be cached at any depth.

To cache radiance transfer, a list of color values is added to each voxel pointer. This list is a sparse graph where entry (vx, vy, Δ) represents radiance transfer from vy to vx with a light depth of Δ . To handle various BRDFs, there is a radiance transfer list for each BRDF. This second pass can run significantly faster

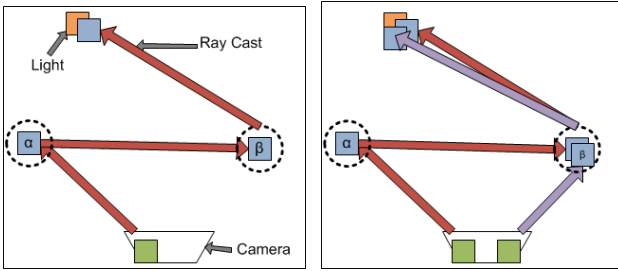


Fig. 2: Example of redundant ray casts. On the left, a light path hits α , then β , and then samples the light source. On the right, a second light path collides near β and samples the light source. Our algorithm effectively shares these similar ray casts.

than the visibility pass because there are no expensive ray casts and because when a radiance sample is reused, the recursion stops, thus pruning out huge parts of the recursive tree. Table 1 shows how much faster path tracing converged when optimized using graph-based global illumination.

The use of graphs in graph-based global illumination consumed about 8 GB of RAM to generate our test images in Table 1. For very large images, we render until we run out of memory, save the image, and then start over, doing a weighted average each time.

3.3 Differences From Previous Algorithms

We now compare and contrast graph-based global illumination with other algorithms. Note that our algorithm is a ray casting optimization, which means it can easily extend algorithms like Metropolis Light Transport [11] and bidirectional path tracing [9] without competing with them.

Bekaert et al.’s [1] work recognized the importance of reuse in rendering. In their work, groups of light paths were calculated together. When a secondary light ray was found for one path, the other paths shot rays towards that same location. Although this addresses the same problem as our work, the algorithm is very different from ours. Our algorithm breaks the rendering process into two distinct passes, which allows up to an n^2 increase in information. Bekaert et al.’s algorithm only shares between small groups of light paths. Also, since their algorithm shares information only across a few light rays, it has distinct artifacts. Our results have no such artifacts.

Irradiance caching is a popular technique for diffuse scenes (see [12] and [4]). These algorithms work where the appearance of diffuse surfaces changes slowly. Since

diffuse bleeding is so computationally demanding, irradiance caching samples diffuse lighting and interpolates those values across the surface. Graph-based global illumination differs from irradiance caching because irradiance caching does best on scenes where “indirect illuminance tends to change slowly over a surface” [12] while graph-based global illumination can effectively render any type of light interaction regardless of BRDF. For example, irradiance caching is known to do poorly on scenes with bright, localized diffuse lighting. As shown in Table 1 and Figure 1, graph-based global illumination renders these types of scenes accurately.

Radiosity algorithms (see [5], [3], and [10]) do well on diffuse surfaces and other work has extended the capacity of radiosity to reflective scenes [2]. Radiosity is similar to graph-based global illumination in that both break down the scene into patches. However, unlike radiosity, graph-based global illumination optimizes ray casting algorithms so it is designed to handle any BRDF type.

Photon mapping [7] is a two pass algorithm that sends photons out from light sources. As photons collide with surfaces, their locations and intensities are stored and used in the second pass to approximate the light intensity at that location. An extension [6] works especially well for light sources behind glass. Photon mapping is similar to graph-based global illumination since it stores information about ray casts. However, photon mapping focuses on storing radiance values while graph-based global illumination focuses on both visibility and radiance values. Also, photon mapping does a pass starting at the light sources and then a pass starting at the camera. Graph-based global illumination starts at the camera in both passes and only casts rays in the first pass, thus making the second pass go quickly.

Graph-based global illumination is a novel global illumination algorithm optimization capable of rendering a wide variety of scenes quickly. Although it shares some similarities with previous algorithms like those just discussed, the way we cache visibility and radiance transfer presents a new paradigm for thinking about sampling a scene. Additionally, graph-based global illumination is designed as a general-purpose optimization that can benefit any ray casting algorithm.

4 Results and Future Work

We compared the speed of graph-based global illumination with path tracing in speed and accuracy tests to determine if path tracing converged faster with our optimization. We chose 12 scenes that contained different lighting effects, primitives, and surface area as shown

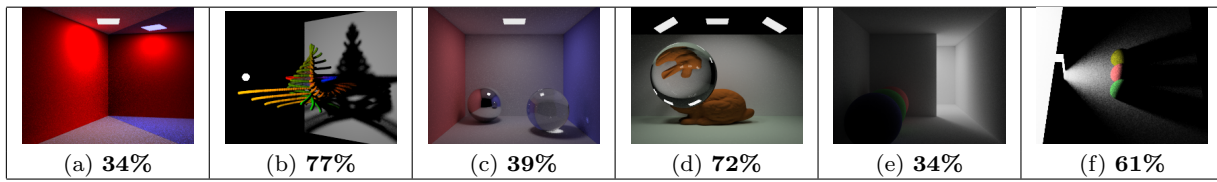


Table 1: Example images from our tests showing how much faster path tracing converged when optimized with graph-based global illumination. Results lower than 100% meant our optimization sped up convergence. Across all 12 test scenes, our algorithm averaged a 52%, meaning graph-based global illumination ran almost twice as fast.

in Table 1. For each scene, we increased the rays per pixel to see how the two algorithms compared in terms of speed and error in a total of 240 tests. To generate ideal comparison images, we ran a path tracer for up to fifteen hours for each image. This means even though our optimization adds bias, its error result was calculated against an unbiased image, which makes our results even more impressive. To speed up the testing process and create a constant memory demand, each image was rendered at a resolution of about 256^2 pixels. We ran the test on a quad core CPU with 16GB of RAM.

For each scene we graphed error versus the number of rays per pixel for path tracing with and without our optimizations. By comparing these curves, we found the ratio of how much faster graph-based global illumination converged. If the ratio was less than 1 (i.e. $<100\%$), then graph-based global illumination sped up path tracing. A result of .5 (i.e. or 50%) meant that graph-based global illumination would have arrived at the same level of error as path tracing in only half the time. In every scene the ratio was less than one, meaning graph-based global illumination sped up the convergence of path tracing in every test scene. On average, our path tracer optimized with graph-based global illumination achieved convergence in about half the time.

Scenes with large specular reflections ran particularly fast in graph-based global illumination because it is designed to reuse information after it has been rendered. If a wall is shown on the left part of an image (as in Table 1a) and the reflection of the wall appears on the right, then there will be a large amount of information reuse. Graph-based global illumination also excelled at scenes with diffuse bleeding since there is so much information that can be reused without adding significant bias. These results validate our contributions since they show that using repeated light samples can reduce the variance in global illumination and they show in practice how this can be done quickly.

In future work we would like to reduce the memory requirements for large images. If this can be done, then our algorithm’s speed-up could be used on machines

with less RAM or on images with a large resolution. Also, we are interested in the advantages of interpolating radiance transfer values across voxels on diffuse surfaces with slow changes in luminance. Finally, we are excited to see how much faster other ray casting algorithms, like Metropolis Light Transport and bidirectional path tracing, converge when optimized with graph-based global illumination.

References

1. Bekaert, P., Sbert, M., Halton, J.: Accelerating path tracing by re-using paths pp. 125–134 (2002)
2. Christensen, P., Lischinski, D., Stollnitz, E., Salesin, D.: Clustering for glossy global illumination. *ACM Transactions on Graphics (TOG)* **16**(1), 33 (1997)
3. Cohen, M.F., Chen, S.E., Wallace, J.R., Greenberg, D.P.: A progressive refinement approach to fast radiosity image generation. *SIGGRAPH Comput. Graph.* **22**(4), 75–84 (1988)
4. Dutre, P., Bekaert, P., Bala, K.: Advanced global illumination. AK Peters, Ltd. (2006)
5. Goral, C.M., Torrance, K.E., Greenberg, D.P., Battaile, B.: Modeling the interaction of light between diffuse surfaces. *SIGGRAPH ’84* **18**(3), 213–222 (1984)
6. Hachisuka, T., Ogaki, S., Jensen, H.W.: Progressive photon mapping. *ACM Trans. Graph.* **27**(5), 1–8 (2008)
7. Jensen, H.W.: Realistic Image Synthesis Using Photon Mapping. A K Peters, Ltd. (2001)
8. Kajiya, J.: The rendering equation. In: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pp. 143–150. ACM (1986)
9. Lafortune, E., Willems, Y.: Bi-directional path tracing. In: *Compugraphics 93*, pp. 145–153 (1993)
10. Sloan, P.P., Kautz, J., Snyder, J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In: *SIGGRAPH ’02*, pp. 527–536. ACM, New York, NY, USA (2002)
11. Veach, E., Guibas, L.: Metropolis light transport. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* pp. 65–76 (1997)
12. Ward, G., Rubinstein, F., Clear, R.: A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics* **22**(4), 85–92 (1988)