

Interactive Rendering of Globally Illuminated Glossy Scenes

Wolfgang Stürzlinger, Rui Bastos

Dept. of Computer Science, University of North Carolina at Chapel Hill
{stuerzl | bastos}@cs.unc.edu

Abstract. Global illumination simulates all transfers of light in a scene. The results of the simulation are then used to generate photo-realistic images. Scenes with diffuse surfaces only can be displayed in real-time using the results of radiosity methods. Images of scenes with more general surfaces are created with methods based on ray tracing but do not achieve interactive frame rates.

This paper presents a new algorithm for the display of globally illuminated scenes at interactive speeds. A photon tracing phase computes an approximation to the global illumination. The rendering phase splats the contribution of each photon hit onto the corresponding surface taking reflectance properties and viewing direction into account. Results demonstrate that this method allows to render images of globally illuminated scenes with glossy surfaces at interactive frame rates.

1 Introduction and Background

This paper presents a new method for rendering globally illuminated scenes with arbitrary surfaces at interactive frame rates. Traditional rendering handles either diffuse surfaces only or cannot take global illumination into account.

Graphic workstations support in hardware the display of diffuse shaded surfaces for real-time rendering of scenes. Globally illuminated scenes with diffuse surfaces can be displayed at interactive frame rates using the results of radiosity methods [4].

Shirley *et al.* [11] [18] proposed a global illumination algorithm for diffuse, mirror-like, and refractive surfaces based on particle tracing and density estimation. The algorithm shoots photons from the light sources, following their paths until they are absorbed, and stores the information where the photons hit surfaces. The illumination for a surface is then approximated by examining the density of photon hits on the surface and creating a mesh that represents the irradiance function. The mesh is simplified and is then used in the final rendering step to generate images. Advantages of this method are:

- It can handle diffuse surfaces, perfect mirrors, and perfect refractors;
- The total complexity is less than quadratic, which allows the application to large scenes;
- The method parallelizes easily. For both the particle tracing and rendering phases, each photon or viewing ray is independent and can be traced separately. The density estimation and simplification phases can be applied to multiple surfaces in parallel.

Diefenbach *et al.* [5] describe a hardware based approach to render glossy, mirror-like, and translucent surfaces with direct lighting and shadows. Their approach handles global illumination effects for diffuse, mirror-like and translucent surfaces.

In the real world most surfaces are non-diffuse and reflect light depending not only on the outgoing direction but also on the incoming direction. The term “glossy” is used to describe such surfaces. The appearance of glossy surfaces can be approximated using Phong-lighting available on current high-end graphics workstations.

Both approaches discussed above do not handle global illumination effects for glossy surfaces. A generalization of Shirley’s approach was presented by Stürzlinger [14]. The modified algorithm takes the incoming direction of the photons into account and stores a simplified form of the incoming radiance function for each glossy surface. The rendering phase uses this information to compute the final image. Disadvantages are the relatively high memory consumption and the slow rendering speed.

Rendering globally illuminated scenes with arbitrary surfaces can be achieved with methods based on ray tracing, but they do not allow interactive frame rates [12]. Some of these methods [7] [16] also use the information stored in a photon simulation and avoid meshing altogether. Their main disadvantage is the requirement to evaluate many rays per pixel to reduce the variance in the final image.

The motivation for the new method described here is to achieve interactive rendering for globally illuminated glossy scenes. The method is based on particle-tracing and uses a hardware splatting approach to compute the final image.

2 Interactive Rendering of Glossy Illuminated Scenes

The new method is split into two phases: particle-tracing and rendering. The particle-tracing phase stores hit records of the photon simulation. The rendering phase uses the hit records to reconstruct the global illumination of the scene.

2.1 Particle-tracing phase

Particle tracing simulates how light interacts with a scene. Initially each light source i emits a number of photons proportional to its power Φ_i . The direction of each photon depends on the emission distribution function associated with its light source. Diffuse light sources emit using a uniform distribution function.

Each photon is traced through the scene until it is absorbed or leaves the scene. Whenever a photon hits a surface a random number is used to decide between absorption or reflection. If the photon is reflected, the bidirectional distribution function of the material is used to generate a random outgoing direction. Photon hits are stored on disk. The stored data includes: the incoming direction, a reference to the light source the photon was emitted from, the current number of “bounces” of the photon, and the accumulated attenuation of the photon until it hit the current surface. Storing the attenuation factor ρ_ϕ , instead of its power ϕ , enables the system to change the number of photons emitted by a light source without having to modify previous hit records. Whenever the contribution of a photon is needed it can be recomputed by dividing the power of the

light source Φ_i by the current number of emitted photons m_i and weighting the result with the attenuation factor:

$$\phi = \frac{\Phi_i}{m_i} \rho_\phi \quad (1)$$

This scheme allows for incremental refinement of the solution, as the number of photons need not be fixed *a priori*. It also provides selective increase of the number of photons emitted by each light source. But this introduces bias in the solution.

2.2 Rendering phase

The directional dependent radiance $L_o(x, \omega_o)$ seen from a viewing direction ω_o for a surface point x is reconstructed from the photon hits stored in the first phase. To take the characteristics of the surface into account, the contribution of each photon ϕ_j with incoming direction ω_i is scaled by the bidirectional reflection distribution function f_r (BRDF) of the surface. To achieve a smooth reconstruction of the outgoing radiance function over the surface, the contribution of each photon is multiplied by a kernel function k positioned at the photon hit location x_j :

$$L_o(x, \omega_o) = \frac{1}{h} \sum_{j=1}^n \phi_j f_r(x, \omega_i, \omega_o) k\left(\frac{x-x_j}{h}\right) \quad (2)$$

where k is a unit volume kernel function (e.g. Silverman's kernel function [13]) and h is a scaling factor for the kernel. Comparable expressions have been used successfully by other researchers [7],[11]. The size of the kernel support is adjusted per surface. It depends on the number of photon hits on the surface and the area of the surface. It is controlled by the following expression:

$$h = C \sqrt{\frac{A}{n}} \quad (3)$$

where A is the surface area, n is the total number of hit points on the surface, and C is a scaling constant. This expression ensures that the reconstruction improves with the number of photons, as the kernel support gets proportionally smaller. If sufficient photons are used in the particle tracing phase the kernel support for each photon covers only a small number of pixels in the final image. Previous work [14] has shown that two million photon hits suffice to generate realistic looking images for a scene with four thousand surfaces. Further experiments with other interior scenes indicated that the number of photons needed does not increase proportionally to the number of surfaces.

The key observation to achieve interactive rendering frame rates is that equation (2) corresponds to "splating" the contribution of each photon onto its surface. All quantities but the BRDF in equation (2) are constant per surface. The BRDF has to be reevaluated for each photon for the current viewpoint. Splating is performed by rendering a textured triangle and adding the pixel values to the framebuffer. The texture corresponds to a regular sampling of the kernel function. The needed triangle geometry and the kernel texture for the splats are precomputed. The algorithm proceeds as follows:

Preprocessing phase:

- Compute the kernel texture.
- For each surface:
 - Determine the size of a kernel splat using equation (3).
 - For each photon hitting the surface:
 - Create a triangle on the plane of the surface and size it to enclose the kernel support. Map the kernel texture onto the triangle.

Rendering phase:

- Render the scene into the depth-buffer.
- Clear the framebuffer.
- For each surface:
 - Construct a stencil mask using the depth-buffer values, thereby restricting rendering to the visible parts of the current surface.
 - For each photon hitting the surface:
 - Set the color of the triangle to the power of the photon multiplied by the BRDF w.r.t. the viewing direction for that surface point.
 - Add the splat to the framebuffer using blending.

Each splat is drawn as a three-dimensional polygon thereby generating the correct perspective effects.

The main drawback of the above algorithm is that its performance is directly proportional to the number of splats which can be rendered per time-unit. Because of limitations of current high-end graphics hardware (see section 4) the method does not achieve high frame rates. Also the splatting can blur shadow boundaries.

2.2.1 Optimized Rendering Phase

Direct illumination accounts for the most prominent illumination features [2]: highlights and shadows. By factoring out the direct illumination from the global illumination solution, the number of photons which need to be rendered can be significantly reduced. Small enough area light sources can be simulated by point light sources without introducing significant error. The image quality improves as highlights and shadows are depicted more accurately. Direct illumination from larger area light sources can be simulated by multiple point light sources.

The direct illumination of a point light source onto each surface is rendered using simulated Phong-shading and shadow maps. Current hardware systems support only per-vertex Phong-lighting for point light sources. To overcome this limitation, each surface is subdivided in a fine regular mesh during setup. This mesh is displayed using per-vertex Phong-lighting and bilinear interpolation. As long as each mesh element covers only a few pixels Phong-shading is simulated accurately.

Shadow maps [10] are used to account for occlusion of the light source. The original polygons of the scene are rendered into a depth texture from the view point of the light source. This depth texture is used during rendering to identify shadowed parts of the scene. Multiple light sources require multiple shadow maps.

Indirect illumination is taken into account by adding only contributions of photons which have been reflected more than once. The modified algorithm is (new parts are emphasized for clarity):

Preprocessing phase:

- Compute the kernel texture.
- For each surface:
 - Determine the size of a kernel splat using equation (3).
 - For each photon hitting the surface:
 - Create a triangle on the plane of the surface and size it to enclose the kernel support. Map the kernel texture onto the triangle.
- **Construct a shadow map for each light source.**
- **Construct a fine triangular mesh for each surface by regular subdivision.**

Rendering phase:

- Render the scene into the depth-buffer.
- Clear the framebuffer.
- **For each light source:**
 - **Activate the corresponding shadow map.**
 - **Add the direct illumination to the framebuffer by rendering all surfaces as a triangular mesh with Phong lighting and shadow maps.**
- For each surface:
 - Construct a stencil mask using the depth-buffer values, thereby restricting rendering to the visible parts of the current surface.
 - For each photon hitting the surface **with more than one bounce:**
 - Set the color of the triangle to the power of the photon multiplied by the BRDF w.r.t. the viewing direction for that surface point.
 - Add the splat to the framebuffer using blending.

Because the appearance of a diffuse surface does not depend on the viewing direction such surfaces are handled by precomputing the effects of the photons hits in the preprocessing phase and displaying the surface using a Gouraud shaded mesh or textured surfaces [3].

3 Implementation

In the following a few noteworthy details of the implementation are described for each phase. The scenes are specified using the MGF file format [17]. Ward's surface model [15] is used for glossy surfaces. Shading of glossy surfaces is simulated in hardware with the fit Phong model introduced by Diefenbach [5]. All example scenes use small area light sources, which were replaced by point light sources with equivalent power for both phases of the algorithm. This avoids problems with mixing hardware and software lighting.

3.1 Particle-tracing phase

The information stored on disk for each photon hit includes the surface number, the position on the surface (u, v) , the incoming direction, a reference to the light source the photon was emitted from, the number of “bounces”, and the attenuation factor for the photon due to previous reflections. Each hit point record is stored in a compressed format using 21 bytes.

3.2 Rendering phase

Reconstruction is performed using “splatting”. The contribution of each photon hit is taken into account by rendering a triangle on the plane of the corresponding surface. The triangle is constructed to enclose the support of the kernel centered at the hit point. A kernel texture is mapped on the triangle. It is modulated by the color of the reflected photon to approximate its contribution in the direction of the viewer. The “splatting” is performed using hardware blending functions set up to add rendered pixel values to the framebuffer. The kernel function is precomputed as a texture of 64×64 pixels. Mip-mapping is used for filtering of small triangles. See figure 1 for the geometric relationships of the triangle, the kernel support, and the kernel texture.

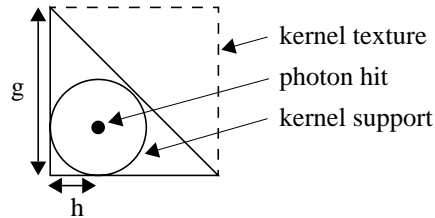


Figure 1. Geometric relationship of triangle and kernel support.

In figure 1, h is the scaling factor defined in equation (3) and g is given by $(1/2 + \sqrt{2})h$. The constant C controls the relative size of the kernel support. Its values range from 30-50, which agrees with values reported previously [11] [14].

A four dimensional table is used to speed up the evaluation of the BRDF. Note also that careful scaling is needed as well as a sufficient number of bits per pixel has to be available to avoid under- and overflow effects when adding photon contributions to the framebuffer.

4 Results

The presented method was implemented on an SGI Onyx Infinite Reality with 2 raster managers in OpenGL. The current implementation uses 12 bits per color channel. A simple scene of a reflective plane lit by colored emitters was used to test the method of section 2.2. Further tests were performed on the Cornell box (with glossy material for the larger cube) and the office scene available in MGF format [17]. See figure 2 for images of the scenes used.

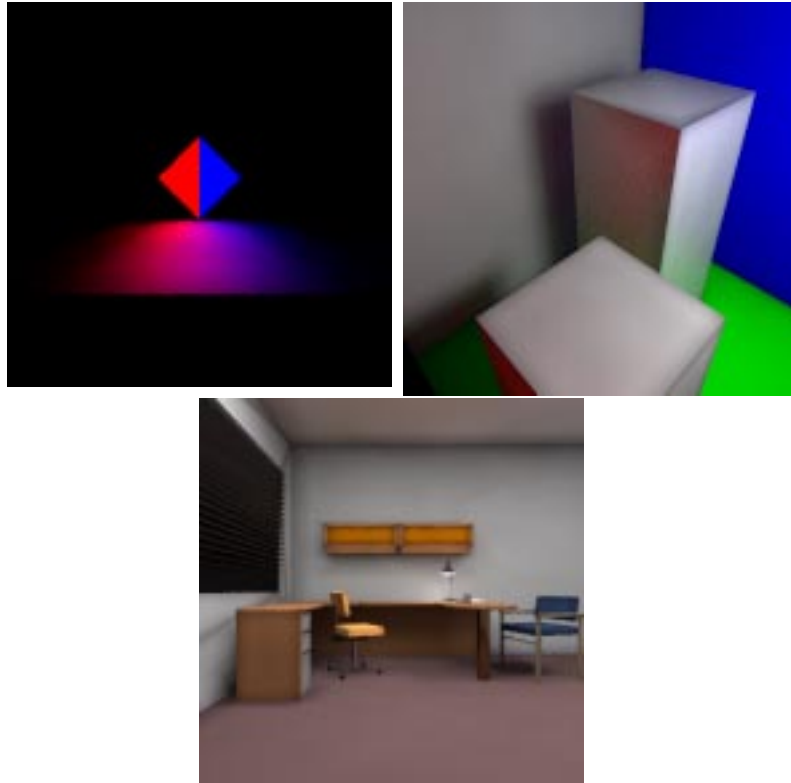


Figure 2. Test scenes.

In figure 3 one million photon hits for the Cornell box scene are shown as three dimensional points with colors proportional to the photon attenuation factor (see section 2.1). Note that both direct and indirect hit records are shown.



Figure 3. Photon hits visualization.

Average frame rates achieved along a predetermined camera path are summarized in table 1.

Table 1: Performance statistics for example scenes.

scene	polygons	photons	sec/frame
Simple	3	0.5M	2.15
Glosbox	18	1.0M	4.21
Office	4072	2.5M	10.5

Performance statistics for the optimized rendering method presented in section 2.2.1 are summarized in table 2.

Table 2: Performance statistics for optimized rendering method.

scene	polygons	indirect photons	sec/frame
Glosbox	18	0.25M	1.18
Office	4072	0.75M	3.46

The main factor determining the performance of the method is the time needed to draw the triangles with the kernel textures. The measured average performance for rendering splats is about 246k triangles per second. This is remarkably different from the 10M triangles per second specified by SGI. The discrepancy is due to the fact that only certain “paths” through the graphics hardware, corresponding to certain parameter configurations, are heavily optimized. When the operands or operations are slightly different, performance drops dramatically. This has been verified by other researchers as well [1].

Note that both scenes in table 2 use relatively small light sources which can be approximated accurately with point light sources. As the direct illumination is computed using hardware, less photons are needed to generate images. Performance improves almost proportionally to this decrease. Additionally, image quality has improved as the most important illumination is depicted more accurately.

The number of photons necessary to generate images without artifacts is difficult to determine *a priori*. But the incremental refinement scheme introduced in this paper allows to increase the number of photons until enough photons have been traced.

5 Conclusions and Future Work

This paper presents a new algorithm for the interactive rendering of globally illuminated glossy scenes. By storing the incoming direction for each photon in the particle tracing phase, the radiance leaving each visible surface can be approximated quickly in

the rendering phase. Unfortunately, the limitations of current high-end graphics hardware prevent the method to achieve real-time rendering frame rates.

The presented system will be ported to PixelFlow hardware [9]. Due to deferred and programmable shading capabilities, significant performance gains are to be expected. The possibility of implementing a splat as a primitive operation of the graphics hardware is very promising for improved frame rates.

Future work includes:

- Integration of Quasi-Monte Carlo Methods [8] into the particle-tracing phase;
- Handling of planar reflective surfaces to simulate mirror-like reflections;
- Determining in advance how many photons are necessary to approximate the illumination on a surface;
- Improving the appearance of the polygon edges similarly to [11];
- Handling of area light sources. One promising approach is described in [6] and the related documents.

Acknowledgments

This work was partially supported by the following grants: NIH/National Center for Research Resources P41RR02170-13 Interactive Graphics for Molecular Graphics and Microscopy, Fellowship FWF-J01317-TEC, Fellowship CNPq - Brasilia/BRAZIL process no. 201142/93-7.

Thanks to the anonymous reviewers for their helpful comments.

References

- [1] D. Aliaga, “*SGI performance measurements*”, personal communication.
- [2] R. Bastos, “*Radiosity Viewer with On the Fly Corrected Illumination*”, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, COMP 238 final project report, Dec. 1996.
- [3] R. Bastos, M. Goslin, H. Zhang, “*Efficient Radiosity Rendering using Textures and Bicubic Reconstruction*”, 1997 Symposium on Interactive 3D Graphics, pp. 71-74, April 1997.
- [4] M. F. Cohen, J. R. Wallace, “*Radiosity and Realistic Image Synthesis*”, Academic Press, 1993.
- [5] P. J. Diefenbach, N. I. Badler, “*Multi-Pass Pipeline Rendering: Realism for Dynamic Environments*”, 1997 Symposium on Interactive 3D Graphics, pp. 59-70, April 1997.
- [6] M. Herf, P. Heckbert, “*Fast Soft Shadows*”, Technical Sketches (SIGGRAPH '96 Visual Proceedings), p 145, July 1996.
- [7] H. W. Jensen, “*Global Illumination using Photon Maps*”, Proceedings of 7th Workshop on Rendering, pp 22-31, June 1996.
- [8] A. Keller, “*Quasi-Monte Carlo Radiosity*”, Proceedings of 7th Workshop on Rendering, pp 102-111, June 1996.

- [9] S. Molnar, J. Eyles, J. Poulton. “*PixelFlow: High Speed Rendering using Image Composition*”, Computer Graphics (Proceedings of SIGGRAPH ‘92), vol. 26, no. 2, pp 231-240, July 1992.
- [10] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, P. Haeberli, “*Fast Shadows and Lighting Effects Using Texture Mapping*”, Computer Graphics (Proceedings of SIGGRAPH ‘92), vol. 26, no. 2, pp 249-252, July 1992.
- [11] P. Shirley, B. Wade, P. M. Hubbard, D. Zareski, B. Walter, D. P. Greenberg, “*Global Illumination via Density-Estimation*”, Proceedings of 6th Workshop on Rendering, pp 187-199, June 1995.
- [12] F. X. Sillion, J. R. Arvo, S. H. Westin, D. P. Greenberg, “*A Global Illumination Solution for General Reflectance Distributions*”, Computer Graphics (Proceedings of SIGGRAPH ‘91), vol. 25, no. 4, pp 187-196, July 1991.
- [13] B. W. Silverman, “*Density Estimation for Statistics and Data Analysis*”, Chapman and Hall, London, 1985.
- [14] W. Stürzlinger, “*Global Illumination with Glossy Surfaces*”, International Conference in Central Europe of Computer Graphics and Visualization (WSCG) ‘97, pp. 543-551, Feb. 1997.
- [15] G. J. Ward, “*Measuring and Modeling Anisotropic Reflection*”, Computer Graphics (SIGGRAPH ‘92), pp 265-272, July 1992.
- [16] G. J. Ward, “*The RADIANCE Lighting Simulation and Rendering System*”, Computer Graphics (SIGGRAPH ‘94), pp 459-472, July 1994.
- [17] G. J. Ward, R. Shakespeare, I. Ashdown, H. Rushmeier, “*MGF Parser and Examples*”, <http://radsite.lbl.gov/mgf/HOME.html>.
- [18] D. Zareski, B. Wade, P. Hubbard, P. Shirley, “*Efficient Parallel Global Illumination using Density-Estimation*”, Proceedings of ACM Parallel Rendering Symposium, pp 47-54, Oct. 1995.