Global Illumination via Density-Estimation

Peter Shirley Bretton Wade Philip M. Hubbard David Zareski Bruce Walter
Donald P. Greenberg

Program of Computer Graphics, Cornell University, Ithaca, NY, USA.

1 Introduction

This paper presents a new method for the production of view-independent global illumination solutions of complex static environments. A key innovation of this new approach is its decomposition of the problem into a loosely coupled sequence of simple modules. This approach decouples the global energy transport computation from the construction of the displayable shaded representation of the environment. This decoupling eliminates many constraints of previous global illumination approaches, yielding accurate solutions for environments with non-diffuse surfaces and high geometric complexity.

Our algorithm produces a view-independent *display mesh* that represents the irradiances on surfaces in a form that allows direct display of the shaded surface. Most traditional radiosity algorithms also use a *computational mesh* to represent intermediate results in the light transport calculation (*e.g.*, the piecewise-constant global solution of Smits *et al.* [17]). Typically, a single representation is used for both the computational and display meshes (*e.g.* the static mesh used by Neumann *et al.* [11] and the adaptive mesh used by Teller *et al.* [18]).

Very few display mesh solutions have been produced for environments with more than a few thousand initial surfaces. The only implementation we are aware of that has produced a display mesh for more than 10,000 initial surfaces is the system by Teller *et al.* [18], which was run on a model with approximately 40,000 initial surfaces. Teller *et al.* argue that the reason for these surprisingly small limits is the high memory overhead of the data structures associated with the computational mesh.

To solve this problem, we draw on an observation by Lischinski *et al.* [10], that the computational mesh and the display mesh have different purposes and characteristics and therefore should be decoupled. Our method is based on the idea that once the display mesh and computational mesh are decoupled, the computational mesh can be replaced with a simpler data structure based on particle tracing. This replacement allows for the solution of larger models with more general reflectance properties.

Our method is composed of three phases which operate without a computational mesh. The first phase uses particle tracing to record a list of particle "hit points" for each surface. The second phase uses these lists to generate a view-independent functional representation of surface irradiance. This process is called *density-estimation* because the representation is an approximation of the underlying density function that generated the hit point locations. The third phase converts the functional representation into a view-independent display mesh. A novel aspect of the third phase is its use of a geometric mesh-decimation algorithm to reduce the size of the display mesh.

In our method all diffuse surfaces will be portrayed accurately regardless of display method. In addition, the contribution of non-diffuse surfaces to the appearance of diffuse surfaces will be accounted for correctly. Non-diffuse surfaces in this mesh will be portrayed accurately if displayed in a view-dependent second pass.

2 Background

Previous global illumination techniques have one or more of the following significant limitations:

– **High intermediate complexity (memory overhead)**. Current radiosity methods use large data structures to accelerate visibility computations. These data structures are usually the limiting factor in performing large radiosity simulations [18]. In practice, an algorithm that stores more than a few hundred bytes per polygon in physical memory will not be practical.

– **Difficulty with local complexity**. In cases where the model has a very high "global complexity" (large numbers of surfaces), but a limited "local complexity" (only small subsets of surfaces are mutually visible), partitioning can be used to decompose the model into subsets which can be solved separately [18]. But if any subset has a high local complexity, then partitioning may not reduce the subproblems to a solvable size. This is a problem in an environment such as a hotel atrium.

– **Quadratic time complexity**. Any algorithm that computes interactions between all pairs of N objects will require at least $O(N^2)$ time. This limits their utility in dealing with large models.

– **"Ideal" specular effects**. Many radiosity algorithms can only use the "virtual image method" [13], which is practical for solving models with only a limited number of ideal, planar, specular objects. Real models have windows, gloss paint, and metal luminaire-reflectors, and in general, non-diffuse surfaces.

– **Lack of parallelism**. Most existing radiosity methods were designed as serial algorithms, and cannot be easily adapted for parallel computation. This limits their ability to take full advantage of one of today's typical computing environments: a local area network of high-speed workstations sharing a common file system.

These limitations need to be overcome in a single system if the use of view independent global illumination solutions are to become widespread. Keeping the memory overhead low favors a Monte Carlo particle-shooting approach [12], which only requires a ray-tracing acceleration structure. Handling local complexity in sub-quadratic time suggests either a clustering approach [17] or a Monte Carlo shooting approach[1]. A Monte

[1] No global illumination algorithm has been proven to be sub-quadratic, but there is empirical evidence that both Monte Carlo shooting algorithms [15] and clustering algorithms [17] are sub-quadratic for reasonably "well-behaved" environments.

Carlo shooting approach also allows for more general specular transport, and possesses inherent parallelism, as each shot can be processed independently. Because specular transport through glass is important in many applications, we have chosen to pursue the Monte Carlo approach.

Although Monte Carlo radiosity schemes have been applied with great success using *a priori* computational meshes [11], there has been little success generating adaptive computational meshes. Appel [1] traced particles from the source to estimate direct lighting. Arvo [2] extended this idea to include illumination reflecting from mirrors before striking surfaces. Heckbert [8] extended Arvo's work to include adaptive meshing, and was the first to observe that this was a form of density-estimation. Chen *et al.* [3] used a *kernel-based* density-estimation technique to deal with *caustic maps*, and our density-estimation work can be considered an extension of their caustic map techniques to account for all illumination effects in a scene. Collins has one of the most sophisticated density-estimation techniques for global illumination [5], but his method does not account for multiple diffuse reflections.

Our strategy is similar to Heckbert's and Collins', but differs in that the meshing is delayed until all Monte Carlo particle tracing has been completed. This allows us to use all the information collected while estimating surface irradiances to generate a good display mesh. The additional storage due to stored hit-points can be processed sequentially and therefore need not be simultaneously resident in real memory.

3 Description of Density-Estimation Algorithm

The algorithm is composed of three basic phases:

1. **Particle-tracing phase:** Power carrying particles are emitted from each luminaire using an appropriate radiant intensity distribution, and are then tracked through the environment until they are probabilistically absorbed. A list of all particle "hit points" is generated and saved.

2. **Density-estimation phase:** The stored hit points are used to construct an approximate irradiance function $H(u,v)$ for each receiving surface.

3. **Meshing phase:** The approximate irradiance function $H(u,v)$ is further approximated to a more compact form $\bar{H}(u,v)$ that can be used for efficient hardware rendering or ray tracing display. If the desired output is a set of Gouraud-shaded polygonal elements for interactive display and walk-through on a conventional graphics workstation, then $\bar{H}(u,v)$ will be piecewise linear.

The algorithm is outlined in figure 1. Note that although the environment is right-left symmetrical, the solution is not. This is because of the randomness introduced by the particle tracing.

3.1 Particle-Tracing Phase

We begin the particle-tracing phase by totaling the power emitted by all luminaires Φ. We then generate approximately n "particles", each carrying power $\phi = \Phi/n$. We use the

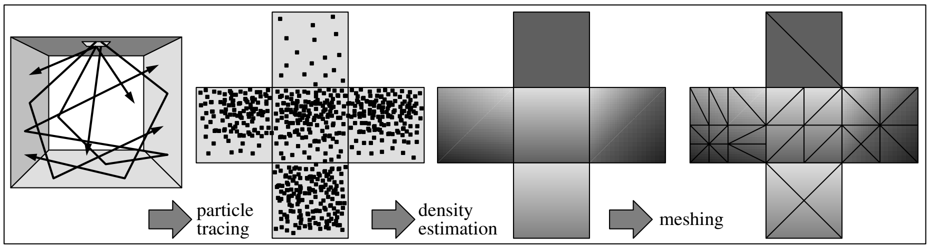

Fig. 1. Overview of the density estimation algorithm. The surfaces in the room are depicted "unfolded" in the three figures on the right.

traditional particle approximation where the particles obey geometric optics, and have an associated color.

For each luminaire ℓ_i with emitted power Φ_i, we trace an expected $N_i = n\Phi_i/\Phi$ rays. Since N_i is not necessarily an integer, we trace at least $\lfloor N_i \rfloor$ rays, and sometimes trace an additional ray with probability $N_i - \lfloor N_i \rfloor$. Each of these rays is sent with a probability density function that is determined by the emission characteristics of the luminaire:

$$p(x, \omega) = \frac{L_e(x, \omega) \cos\theta}{\int_X \int_\Omega L_e(x, \omega) \cos\theta d\omega dx}$$

where p is a probability density function for ray generation, x is a point on the luminaire, ω is a direction, L_e is the emitted surface radiance, θ is the angle between ω and the surface normal at x, X is the set of points on the luminaire, and Ω is the set of outgoing directions on the hemisphere oriented with the surface normal.

At each surface the particle is probabilistically reflected, transmitted, or absorbed based on $\rho(x, \omega, \omega')$, the surface's bidirectional reflectance distribution function (BRDF).

3.2 Density-Estimation Phase

After completing the particle-tracing phase, we have associated with each surface a set of hit points with incoming power ϕ. It seems logical to guess a reasonable irradiance from the local denseness or sparseness of these hit points. For example, where the density of these points is high, we expect a high irradiance. As pointed out by Heckbert [8], this is a classic *density estimation* problem, where we attempt to guess a plausible density function given a set of non-uniform random samples[2]. Before getting to the details of how we apply density estimation, we first establish that the radiometric quantity we wish to estimate is the irradiance.

A Lambertian surface has a BRDF that is a constant R/π for all incoming/outgoing direction pairs, where R is the reflectance (ratio of outgoing to incoming power). This

[2] Note that this density estimation problem has a set of sample locations, but does not have function values at these locations. This is different from the problem of reconstructing a signal from a sampled function. It is easy to get these two problems confused. Ironically, the strategy of placing kernels at the hit points is very similar, but in density-estimation the kernels are *not* scaled.

implies that a Lambertian surface will have a constant surface radiance for all incoming/outgoing direction pairs. Ultimately, we wish to approximate this surface radiance for all Lambertian points.

For a particular parametric Lambertian surface with reflectance $R(u, v)$ and irradiance $H(u, v)$ (incident power per unit area at (u, v)), the radiant exitance $M(u, v)$ (outgoing power per unit area at (u, v)) is $R(u, v)H(u, v)$. Because the radiance of a Lambertian surface is $L(u, v) = M(u, v)/\pi$, the relationship between irradiance H and radiance L can be expressed by the following equation [3]: $L(u, v) = R(u, v)H(u, v)/\pi$. This equation implies that we can store the irradiance and reflectance at each point and later reconstruct the radiance. This is convenient because the reflectance may change quickly, while the irradiance may change slowly, allowing the irradiance to be stored in a coarse mesh. These ideas are based upon the "patch-element" radiosity work of Cohen et al.[4].

For a given surface, a list of hit point locations (u_j, v_j) is stored. Each of these points has the same power, ϕ. The irradiance function represented by this list is a set of "delta" functions where a finite amount of power strikes an infinitely small area. In one dimension, this is essentially taking a set of n samples x_i and noting that a possible density function f is:

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} \delta\left(x - x_i\right)$$

This sum of spikes would be a bad guess if we know f is smooth. Instead we could replace the δ functions with smooth "kernel" functions $k_1(x - x_i)$, where k_1 has unit volume. This generates a smoother estimate for f. An example kernel estimate is shown in Figure 2.

Using kernel functions on the hit points is analogous to the idea of "splatting" in volume rendering [20], and is similar to the illumination ray tracing of Collins [5]. Silverman [16] notes that whatever properties the derivatives of k_1 have will be shared by f, so we can ensure a smooth estimate for f by choosing a smooth k_1.

Good choices for k_1 are similar to the choices used for splatting or pixel filtering. As in those applications the kernels should be centered at the origin, have limited support (non-zero region), and should be roughly "lump" shaped. If the volume of the function we are approximating, A, is not unity (so the function is not a probability density function), we can compensate by multiplying the sum by A.

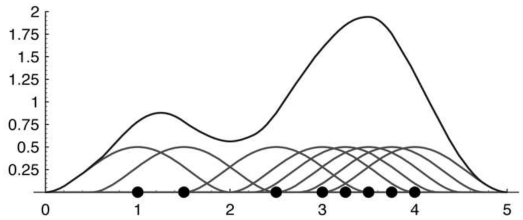

Fig. 2. Kernel estimate showing individual kernels.

[3] See the recent texturing work of Gershbein et al. [6] for a more detailed analysis.

In two dimensions, the irradiance function can be estimated as:

$$H_i(\mathbf{x}) = \frac{\Phi}{n} \sum_{j=1}^{n_i} \delta\left(\mathbf{x} - \mathbf{x}_j\right)$$

Where $\mathbf{x}_j$ is the position of the jth hit point. We can replace the delta functions with n_i "kernel" functions k_j, and note that $\Phi/n = \phi$:

$$H_i(\mathbf{x}) = \phi \sum_{j=1}^{n_i} k_j\left(\mathbf{x}_j\right) \tag{1}$$

The kernel functions have the conflicting requirements of being narrow enough to capture detail, and being wide enough to eliminate the random mottling caused by the irregular pattern of the hit points. We can use a scaling parameter h to widen or narrow the filter. Because narrowing the filter will decrease its volume, we also increase the height of the kernel to keep its volume constant:

$$H_i(\mathbf{x}) = \frac{\phi}{h^2} \sum_{j=1}^{n_i} k\left(\frac{\mathbf{x} - \mathbf{x}_j}{h}\right) \tag{2}$$

Note that for Equation 2 to represent irradiance, k must have unit volume. On a locally planar surface with orthogonal length parameters (u, v), this is straightforward to guarantee. A more complex form would be needed to conserve energy on more complex surfaces.

3.3 Meshing Phase

At first it seems logical to render the approximate $H(u, v)$ directly, but the number of sample points is large enough that any method that attempts to randomly access all the points in the environment will not be practical for large environments. Instead, we need to reduce the amount of information needed to specify an approximate irradiance function.

The most obvious strategy is to sample $H(u, v)$ at a finite set of locations and use some type of polynomial elements to interpolate between these values. Ideally, the sample points should be chosen so that they are dense only where the irradiance function has many features.

Once we have a more compact representation of $H(u, v)$ we can display an image of the illuminated surfaces using either 3-D graphics hardware or ray tracing. For Gouraud shading we should approximate $H(u, v)$ with piecewise-linear elements, but for ray tracing we can use higher-order elements.

3.4 Parallel Execution

Each of the three phases above are ideally suited to take advantage of parallelism. In the particle-tracing phase, particle paths can be computed independently, and once the particle hit points for each surface are grouped together, the density-estimation and meshing phases can compute the shading of each surface independently.

4 Implementation

We have implemented the algorithm in C++ as three separate serial programs that communicate using files. The first program reads the input geometry, performs the particle-tracing phase, and writes out the hit points. The second program reads the hit points and geometry, performs the density-estimation phase, and then performs the initial pass of the meshing phase by generating a finely-tessellated display mesh. The third program reads this display mesh and performs the final pass of the meshing phase by decimating the mesh so that it can be displayed more quickly without significant loss of image quality.

We have also implemented a parallel version of the algorithm in C++ as two separate parallel programs [21]. The first is a parallel version of the serial particle tracer, and the second is a combined, parallel version of the both the serial density-estimation and mesh-decimation programs. The second parallel program also performs the hit point sorting.

The three serial programs mentioned above are described in the remainder of this section. Our goal has been to implement each component of the density-estimation framework as simply and conservatively as possible. This strategy has allowed us to explore the basic strengths and weaknesses of density-estimation without getting bogged down with low-level issues. Our implementation should therefore be considered a proof-of-concept which leaves open many avenues of investigation that will improve on our results.

4.1 The Particle-Tracing Program

This program implements the particle-tracing phase exactly as described in Section 3.1. Rays are emitted from random locations on each luminaire in a directional-distribution determined from the emitted surface radiance of the luminaire.

Each time a ray hits a diffuse surface, the surface id (4 bytes) and fixed-point representations of the uv coordinates (2 bytes each) are written to a file. This means we can store approximately 125 million hit points on a one gigabyte disk. This code is run once for each of the red, green, and blue channels.

The only memory overhead in the particle-tracing phase is the uniform-subdivision ray-tracing efficiency structure (approximately 140 bytes per patch in our implementation, or about 7 million patches per gigabyte).

An alternative way to store hit points would be with surface normal information in a 3D data structure, as is done by Ward [19] and by Jensen and Christensen [9]. This would raise memory usage, but would elimiate some problems associated with models that are hard to parameterize.

4.2 The Density-Estimation and Initial-Meshing Program

Ideally we would like to make a density-estimation of the irradiance function of the ith surface $H_i(\mathbf{x})$, and then output a piecewise linear approximation with as few linear elements as are needed to accurately represent $H_i(\mathbf{x})$. However, because generating a concise, piecewise-linear approximation is a hard problem, and because we wanted to determine the potential accuracy of density-estimation techniques before attempting to solve the most difficult problems suggested by the technique, we adopted a strategy of over-meshing the solution, and then decimating this mesh.

The density-estimation program begins by sorting all hit points by surface id in each of the red, green, and blue channel files produced by the particle-tracing program. Each surface is then processed in series by examining all its hit points, calculating a density estimate of its irradiance function $H_i(\mathbf{x})$, and then outputting an "over-meshed" point-sampled approximation to $H_i(\mathbf{x})$.

The current implementation is restricted to rectangular surfaces only. It samples each rectangle on a n_u by n_v lattice, and outputs this approximation as a triangular mesh with $2(n_u - 1)(n_v - 1)$ elements and $n_u n_v$ vertices, with irradiance values at each vertex. We use triangles as mesh elements instead of rectangles because it simplifies the mesh-decimation algorithm.

We have chosen to use Silverman's K_2 kernel function [16]:

$$K_2(u, v) = \frac{3}{\pi}\max\left(0, (1 - ||\mathbf{x} - \mathbf{x}_j||^2)^2\right) \tag{3}$$

The width of the kernel function, h, is chosen to relate it to the average distance between sample points on the ith surface. This is approximated by $C_1\sqrt{A_i/n}$, where A_i is the area of surface i, n is the number of sample points, and C_1 is a positive constant that controls the width of the kernel. The desired spacing between adjacent grid points is chosen as $C_2\sqrt{A_i/n}$, where C_2 is typically less than C_1. Since each polygon can be meshed independent of other polygons, we only need keep this grid in memory for one polygon at a time, and paging has not been a problem on any of our runs. Near the boundary we use the *reflection method* [16] to avoid darkening near the edge of polygons.

The trade-off between noise and blurring as controlled by C_1 is shown in Figure 3. In this figure C_2 has been set to 8, which is small enough to not affect the images. We have found useful values of C_1 range from 10 to 40, and useful values of C_2 to range from $C_1/10$ to $C_1/2$. It is important to note that surfaces that receive fewer particles will get wider kernels and coarser meshes. This avoids the under-sampling problems of traditional illumination ray tracing [2] in a manner similar to Collins [5]. Unlike Collins, we do not require any coherence between adjacent particle paths, so we can choose appropriate kernel sizes for data that includes diffuse interreflection.

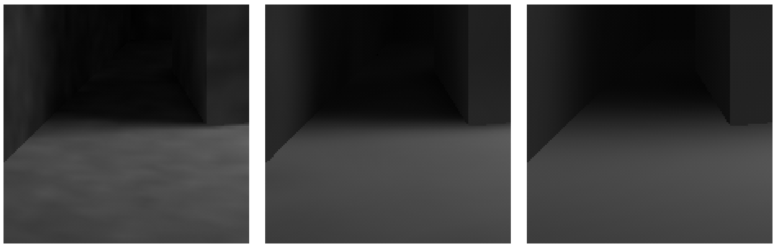

Fig. 3. Different noise/blur effects for $C_1 = 10, 30, 100$.

4.3 Irradiance-Mesh-Decimation Program

The final program decimates the triangulated mesh produced by the density-estimation program. Its goal is to eliminate as many mesh elements as possible without compromising mesh accuracy.

In order to take advantage of the rich literature in geometric mesh decimation, we transform our 2-D surface mesh, which we call an *illumination mesh*, into a 3-D mesh. The X and Y axes correspond to the surface's parametric space, and the Z axis corresponds to "brightness." This mesh is a height field because there are no "ledges" that overhang other parts of the mesh.

To this 3-D mesh we apply geometric decimation techniques. As long as the result is still a height field, it converts to a decimated illumination mesh by simply ignoring the Z coordinate. To decimate the 3-D mesh, we use the algorithm of Schroeder *et al.*[14]. It uses two heuristics to approximate curvature at a mesh vertex. If curvature is low enough, it removes the vertex and re-triangulates the resulting "hole" in the mesh.

This decimation algorithm does not necessarily produce a height field, so we added a simple enhancement. When re-triangulating a hole, the enhanced algorithm checks for triangles on "overhanging ledges"; if it finds any, it does not remove the vertex.

Another disadvantage of Schroeder's algorithm is that it does not consider the decimation to be an optimization task. It uses a fixed threshold on curvature, decimating all vertices which cause curvature changes below the threshold. We turned the algorithm into an optimization task by using a *priority queue*. The queue holds all vertices ranked by the change in curvature their decimation would cause. The new algorithm decimates vertices in the order they come off the queue, thus minimizing the change in curvature with each decimation. Maintaining the queue is efficient, since each access takes only $\mathrm{O}(\log n)$ time for a mesh of n vertices.

The main challenge in using the Schroeder *et al.* algorithm is picking the exact mapping from the illumination mesh to a purely geometric, 3-D mesh. The geometric mesh has one set of units (spatial, *e.g.*, meters) but the illumination mesh has two: spatial for the surface's two parametric dimensions (X and Y), and "brightness" for the third (Z). The mapping must balance the two sets of units, since the decimation algorithm cannot distinguish between them.

We map the two sets of units to a common scale, and choose a single maximum-allowable error, ϵ, in this scale. We scale the spatial dimensions so that the smallest important feature has width ϵ. We scale the "brightness" dimension by using

$$z = \left(\frac{R(x,y)H(x,y)/\pi}{w_0} \right)^{1/3},$$

where $R(x,y)$ is photometric reflectance, $H(x,y)$ is illuminance, w_0 is the white-point, and the exponent is due to Stevens Law [7]. We clamp z to 1.0, and choose w_0 so that $z = \epsilon$ corresponds to the largest allowable "brightness" error. In practice, determining the scaling parameters requires some trial and error. Fortunately, in our experience we found useful parameter values after only two or three attempts.

Decimation works well in practice. Figure 4 shows an example in which decimation removed 90% of the triangles produced by the density-estimation program. We achieved similar decimation rates in our other tests.

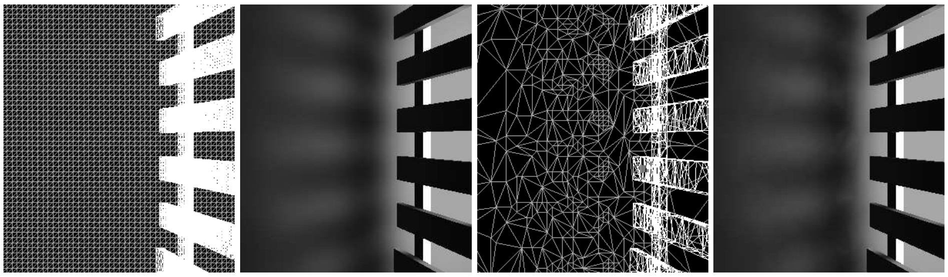

Fig. 4. Left: undecimated mesh. Right: 90% decimated mesh.

4.4 Sample results

Figure 5 shows a ray-traced image of a view-independent density-estimation solution. The environment depicted has sixteen light sources forming a compact lattice to the left of the image. This image illustrates how we correctly handle transport chains that include glass and specular reflection from metals.

Notice how light passes through the glass and then through the fence, reflects from the steel, and casts a green streak on the floor. The shadowing is stored in the illumination mesh, so only the colors of the metal and glass need to be calculated during ray tracing.

One characteristic of the error in our solutions is that the red, green, and blue channels of our images have uncorrelated oscillations, which causes a visible colored texture if the kernels are too narrow. Eliminating this colored texture increases the blurring of desired features such as shadow boundaries.

Figure 6 (left) shows a hardware-rendered decimated version of a model with approximately 10,000 initial surfaces. The decimated mesh has approximately 12% of the triangles in the original mesh.

Figure 6 (right) shows the solution of a model with approximately 150,000 original polygonal surfaces. This solution is roughly four times larger than the largest previous solution produced by radiosity methods which compute a display mesh *a posteriori*. The undecimated display mesh contained about 4.2 million triangles and the decimated mesh had approximately 400,000 triangles. In all, the solution took approximately 37 hours to compute on an HP model 755 100MHz PA-Risc 7100 workstation. A total of 96 million particles were traced (32 million in each of the red, green, and blue channels).

Preliminary investigations indicate time can be reduced by more than one order of magnitude by using our parallel workstation cluster [21].

The proportion of time spent in each of the three programs was: particle-tracing: 49%; density-estimation and initial-meshing: 29%; mesh-decimation: 22%.

5 Conclusion

We have presented a new global illumination method based on density estimation. The method is straightforward to implement, can attack much larger problems than previous techniques, accounts for specular transport, and is designed to use parallelism efficiently in each of its sequential phases. In addition to these new capabilities, this method also

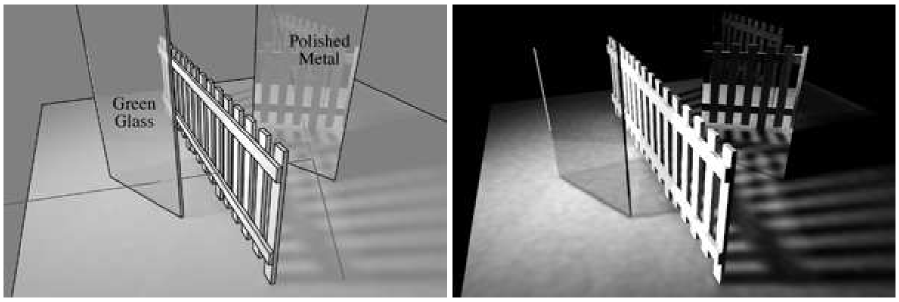

Fig. 5. Fence with green glass filter and polished metal reflector (ray-traced).

Fig. 6. Left: Room with 10,000 initial polygons. Right: Room with 150,000 initial polygons.

retains many of the historical advantages of radiosity methods, including view independence of diffuse components, physical accuracy, and capability for progressive refinement. Future work should include investigating other methods for density estimation that do a better job of reconstructing shadows and other details in the illumination function, and better methods of decimation or mesh optimization.

Acknowledgments

Thanks to Jim Arvo for his many helpful suggestions. Special thanks to Gene Greger who created all of the models depicted in the paper. This work was supported by the NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219) and by NSF CCR-9401961 and performed on workstations generously provided by the Hewlett-Packard Corporation.

References

1. A. Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computing Conference*, pages 37–49, 1968.
2. James Arvo. Backward ray tracing. *Developments in Ray Tracing*, pages 259–263, 1986. ACM Siggraph '86 Course Notes.
3. Shenchang Eric Chen, Holly Rushmeier, Gavin Miller, and Douglass Turner. A progressive multi-pass method for global illumination. *Computer Graphics*, 25(4):165–174, July 1991. ACM Siggraph '91 Conference Proceedings.
4. Micheal F. Cohen, Donald P. Greenberg, David S. Immel, and Philip J. Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics & Applications*, 6(2):26–35, 1986.
5. Steven Collins. Adaptive splatting for specular to diffuse light transport. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 119–135, June 1994.
6. Reid Gershbein, Peter Schröder, and Pat Hanrahan. Textures and radiosity: Controlling emission and reflection with texture maps. *Computer Graphics*, pages 51–58, July 1994. ACM Siggraph '94 Conference Proceedings.
7. E. Bruce Goldstein. *Sensation and Perception*. Wadsworth Publishing Co., Belmont, California, 1980.
8. Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics*, 24(3):145–154, August 1990. ACM Siggraph '90 Conference Proceedings.
9. Henrik Wann Jensen and Niels Jorgen Christensen. Bidirectional monte carlo ray tracing of complex objects using photon maps. *Computers & Graphics*, 19(2), 1995.
10. Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. *Computer Graphics*, pages 199–208, August 1993. ACM Siggraph '93 Conference Proceedings.
11. László Neumann, Martin Feda, Manfred Kopp, and Werner Purgathofer. A new stochastic radiosity method for highly complex scenes. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 195–206, June 1994.
12. S. N. Pattanaik. *Computational Methods for Global Illumination and Visualization of Complex 3D Environments*. PhD thesis, Birla Institute of Technology & Science, February 1993.
13. Holly E. Rushmeier. *Realistic Image Synthesis for Scenes with Radiatively Participating Media*. PhD thesis, Cornell University, May 1988.
14. William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.
15. Peter Shirley. Time complexity of monte carlo radiosity. In *Eurographics '91*, pages 459–466, September 1991.
16. B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1985.
17. Brian E. Smits, James R. Arvo, and Donald P. Greenberg. A clustering algorithm for radiosity in complex environments. *Computer Graphics*, 28(3):435–442, July 1994. ACM Siggraph '94 Conference Proceedings.
18. Seth Teller, Celeste Fowler, Thomas Funkhouser, and Pat Hanrahan. Partitioning and ordering large radiosity calculations. *Computer Graphics*, 28(3):443–450, July 1994. ACM Siggraph '94 Conference Proceedings.
19. Gregory J. Ward. The radiance lighting simulation and rendering system. *Computer Graphics*, 28(2):459–472, July 1994. ACM Siggraph '94 Conference Proceedings.
20. Lee Westover. Footprint evaluation for volume randering. *Computer Graphics*, 24(4):367–376, August 1990. ACM Siggraph '90 Conference Proceedings.
21. David Zareski, Bretton Wade, Philip Hubbard, and Peter Shirley. Efficient parallel global illumination using density estimation. In *Proceedings of the 1995 Parallel Rendering Symposium*, 1995.