Global Illumination Using Local Linear Density Estimation

Bruce Walter * Philip M. Hubbard † Peter Shirley ‡
Donald P. Greenberg
Cornell Program of Computer Graphics

Abstract

This paper presents the density estimation framework for generating view-independent global illumination solutions. It works by probabilisticly simulating the light flow in an environment with light particles that trace random walks originating at luminaires and then using statistical density estimation techniques to reconstruct the lighting on each surface. By splitting the computation into separate transport and reconstruction stages, we gain many advantages including reduced memory usage, the ability to simulate non-diffuse transport, and natural parallelism.

This paper also describes how several theoretical and practical difficulties can be overcome in implementing this framework. Light sources that vary spectrally and directionally are integrated into a spectral particle tracer using non-uniform rejection. A new local linear density estimation technique eliminates boundary bias and extends to arbitrary polygons. A mesh decimation algorithm with perceptual calibration is introduced to simplify the Gouraud-shaded representation of the solution for interactive display.

CR Categories and Subject Descriptors: I.3.0 [Computer Graphics]: General; I.3.6 [Computer Graphics]: Methodology and Techniques.

Additional Key Words and Phrases: realistic image synthesis, density estimation, regression, decimation, particle tracing.

*[bjw,pmh,shirley,dpg]@graphics.cornell.edu Program of Computer Graphics 580 Rhodes Hall Cornell University Ithaca, NY 14853

†current address: pmh@cs.wustl.edu Department of Computer Science Campus Box 1045 Washington University One Brookings Drive St. Louis, MO 63130-4899 pmh@cs.wustl.edu http://www.cs.wustl.edu/p̃mh/

‡current address: shirley@cs.utah.edu Department of Computer Science 3190 Merrill Engineering Building University of Utah Salt Lake City, UT 84112

1 Introduction

View-independent global illumination is a difficult problem that requires solving for detailed lighting information on each of many interacting surfaces. Traditional methods have tried to attack the entire problem at once, and the resulting complexity has limited the maximum problem size and solution quality that can be handled. This paper shows how this complexity can be reduced by splitting the computation into distinct transport and reconstruction stages. We call this approach the *density estimation framework* because statistical density estimation techniques play a crucial role. The framework consists of three phases: particle tracing, which simulates the global transport of light; density estimation, which reconstructs the lighting on each surface; and decimation, which reduces these estimates to a more compact and efficient form. The input is a geometric description of an environment along with its associated radiometric material properties, and the outputs are *illumination meshes* which represent the lighting on each surface. This framework has several advantages including greatly reduced memory usage, non-diffuse light transport, software modularity, and natural parallelism.

We also show how this framework can be implemented in a reasonably general and robust manner and detail the design decisions and techniques that we use in our current implementation, which is a significant advance over our initial work [29]. We can handle arbitrary polygonal models, including very general material properties, and produce piecewise-linear estimates of the luminous and chromatic exitance on all diffuse surfaces for fast hardware display. The particle tracing phase is spectrally based for improved physical accuracy and includes techniques to sample complicated emission and scattering functions. The density estimation is based on a new local linear technique which eliminates boundary bias and can handle arbitrary polygons. The decimation draws on simple perceptual principles to reduce the artifacts it creates as it simplifies the illumination meshes.

In Section 2 we discuss why we think the density estimation framework is a good idea. A overview of our implementation is given in Section 3 followed by a description of each of our three phases: particle tracing in Section 4, density estimation in Section 5, and mesh decimation in Section 6. Some results are presented in Section 7 and the conclusion in Section 8. We have also included some appendices for readers and potential implementers who want more detail. Appendix A describes the non-uniform rejection for particle tracing, Appendix B develops the local linear density estimation technique, and Appendix C describes the user study that calibrated the mesh decimation.

2 Motivation

Computing view-independent global illumination solutions involves solving for the lighting on the surfaces in a model by simulating the physics of light. Conceptually we can think of this process as consisting of two parts: light transport

and lighting function representation. Light transport is the flow of light between surfaces. Since light from each surface potentially interacts with every other surface, we say that the transport has a high inter-surface or *global complexity*. Representation is the process of constructing an estimate of the lighting function on each surface. For view-independent solutions we would like these estimates to be sufficiently detailed for direct display. The lighting may be quite complex with detailed features such as shadows and caustics. Thus, whatever representation we choose (e.g., piecewise linear or wavelet), it must be complex enough to capture these features if they exist.[1] The lighting representation has a high intra-surface or *local complexity*.[2]

2.1 Finite Element Methods

Traditionally, view-independent global illumination solutions have been computed by finite element methods. These methods work by interleaving the computation of the global light transport and the local lighting representation. They iteratively compute a portion of the global transport and update local representations until some convergence criteria are reached. This combination of operations involving high global and high local complexity causes an explosion in resource consumption in terms of both memory and time.

Much research has gone into improving the basic finite element method. Several techniques (e.g., hierarchical radiosity [12] and clustering [31]) greatly reduce the time required at the expense of additional data structures and greater memory usage. Consequently, it is usually memory that limits the maximum input size and solution quality. To overcome this problem, researchers have tried various ways to reduce global or local complexity. Discontinuity meshing [19] attempts to precompute the potential locations of shadows to allow for a more compact local representation. This can produce dramatically better shadows, but it does not handle other lighting features, such as caustics and shadows from secondary sources, and does not scale well to large environments. Teller et al. [32] try to partition the environment into small, weakly interacting subsets to lower the effective global complexity. If such a partitioning can be found then the computation can be ordered to use virtual memory efficiently, however such a partitioning may not exist (e.g., a hotel atrium). Others (e.g., [26, 31] reduce the local complexity by abandoning the idea of displaying the solution directly. Instead, the solution is computed at low resolution and a computationally expensive local-gather display pass is required to display the solution. This puts them in the realm of multi-pass methods rather than the view-independent methods that are our primary focus here. Multi-pass methods are further discussed in Section 2.3.

[1] Note that it is often possible to find a very compact representation after the estimate is computed, but it is extremely difficult to find an appropiate one beforehand.

[2] Our definition of the terms local and global complexity is different than that used by Teller et al. [32].

2.2 Density Estimation Framework

The density estimation framework avoids this combination of high local and global complexity by splitting light transport and lighting representation into separate computational stages. In the transport stage we compute the flow of light between surfaces without ever explicitly reconstructing the lighting on surfaces. Particle tracing is a natural and robust way to simulate this light flow. The representation stage then uses information from the transport stage to explicitly reconstruct the lighting on each surface. Since the intensity of the lighting is proportional to the density of light particles, the reconstruction is a density estimation[3] problem.

Particle tracing has been used by many other researchers [1, 2, 22] to compute illumination and Heckbert [14] first noted that reconstructing lighting from particles is a density estimation problem. Since then a variety of different density estimation techniques have been applied, including histograms [14], kernel methods [3, 6], and splines [24]. Our density estimation technique is new, but the fundamental difference between our framework and previous work is the separation of the transport and reconstruction stages.

The particle tracing stage computes a statistical simulation of global light transport. Since light particles do not interact there is no need to explicitly reconstruct the lighting function, and instead we simply record some information about the particle histories. Thus the particle tracing can work directly with the raw input geometry and has high global complexity but minimal local complexity.

The lighting reconstruction stage uses the recorded particle histories to estimate the lighting function on each surface. Because all of the global transport was handled in the previous phase, we can reconstruct the lighting on each surface independently. Reconstruction on a surface has high local complexity but no global complexity.

Because each stage has only high global or high local complexity but not both, the individual stages require fewer resources than finite element methods, especially in terms of memory. But we have not completely escaped the complexity problem. The combined high global and high local complexity is contained in the voluminous particle history data. Thus, careful attention must be paid to how the particle histories are stored and accessed. The particle tracing only writes particle data and the density estimation only needs the particle data associated with one surface at a time. The only operation we ever need to perform on the entire particle data set is a sort by surface; otherwise, the computation can be structured so that the particle data is processed in a predictable and largely sequential order. This means that we can efficiently store the particle data using secondary storage such as a hard disk, which is typically 100 times cheaper than physical memory (RAM). The efficient use of secondary storage is the key that makes our framework feasible.

[3] Density estimation is the problem of estimating a unknown density function from a set of discrete samples drawn according to the density function [30].

2.3 Multi-pass Methods

Global illumination calculations and methods can roughly be divided into view-independent computations, which are valid regardless of viewpoint, and view-dependent computations which are needed to produce a particular view or image. Purely view-dependent methods compute only the information needed to produce a desired image, but must start over the beginning to compute the next image. Also they generally compute each pixel independently which results in redundant computation of low spatial frequency components. On the other extreme, view-independent methods try to precompute as much view-independent lighting information as possible so that only a minimal amount of subsequent work is required to produce each image. This precomputation can be problematic though as appearance aspects that have a high spatial frequency or are angularly dependent are expensive both to compute and to store.

Multi-pass methods (e.g., [3, 26, 17]) combine both approaches. They split the lighting into various components, some of which are precomputed by a view-independent phase while others are computed anew for each image. Generally they try to divide the lighting into low and high frequency components by using distinctions such as direct vs. indirect lighting and specular vs. diffuse reflection. Each component is computed by the method which seems best suited to its expected characteristics.

The work of Jensen [17] is particular relevant here as he makes use of a particle tracing phase similiar to ours. He uses two particle tracing phases to estimate two lighting components: diffusely reflected indirect light and specularly reflected indirect light. A view-dependent phase then performs density estimation to query these components, filters the results through a local-gather operation, and combines them with estimates of other lighting components. This method often works quite well. When the distinctions between different lighting components are clear and when only a small number of images need to be generated, then Jensen's method is much faster than the density estimation framework presented here.

While the methods presented in this paper can be used as part of a multi-pass method (e.g. in Plate I) similiar to Jensen's work, we have chosen to emphasize its use as a view-independent method. One reason is that we are interested in interactive walkthroughs. Multi-pass methods are not currently useful for interactive walkthroughs as their display computations are simply too slow to achieve the required display rates. We also believe that the kinds of distinctions used in multi-pass methods are sometimes difficult to make. For example, if we move away from "bare bulb" lights and direct sunlight toward complex lighting fixtures and areas light by reflected sunlight, the distinction between direct and indirect light becomes unclear and unhelpful. Thus it is interesting to explore methods that do not rely on such distinctions. Lastly we think that because all lighting is computed by a simple statistical particle tracing, our method is better suited for a relatively simple and precise error analysis than other current methods. This aspect though will not be emphasized here.

A few other difference between our work and Jensen's are worth mentioning.

Jensen does not ever explicitly reconstruct the illumination function on a surface in our sense. Instead a new density estimation calculation is performed each time he wants to query information computed by a particle tracing. To make this feasible, he needs to be able to keep all his particle data in memory, which puts a strict limit on the number of particles. He also uses a very simplistic density estimation technique, which suffers from errors due to boundary effects, surface curvature, and nearby surfaces, but the resultant artifacts are mostly filtered out by his local-gather display computations. In our work we need to be more careful since our density estimation results are directly displayed.

2.4 Diffuse vs. Non-diffuse

The extent to which we can handle non-diffuse materials is a common point of confusion because it differs from most other methods. We do properly handle the scattering of light by non-diffuse materials in our transport stage. This means that the effect of light scattered from a non-diffuse object onto another surface is correctly handled. However the recontruction phase computes quantities like spectral irradiance and luminous exitance that do not depend on viewing direction. These are only sufficient to completely describe the appearance of diffuse surfaces. This means that the appearance of the non-diffuse surfaces themselves will not be correct when using our method as a view-independent one. If a ray tracing display phase is used, then it can fill in this missing information at non-diffuse surfaces and display them correctly, but the method becomes a multi-pass method.

An example of this can be seen in Plates I and J. The ground plane in Plate J is displayed exactly as it was computed by the density estimation process including the complex lighting effects caused by the very non-diffuse glass plates. Computing the appearance of the glass plates themselves, however, required some extra work which was performed by a ray tracing display computation. The object at the right is Plate J is actually a glass prism, but its appearance is not correct because a ray tracing display phase was not used for this image. Notice that we still capture the dispersion rainbox it causes onto the diffuse screen at the left. This distinction can also be seen in the bottom two rows of Plate E if one ignores the overlayed meshing. The left images were displayed using a ray tracing pass and the right using a direct hardware rendering.

3 Overview

Our implementation is composed of three phases:

1. **Particle-tracing phase:** Power-carrying particles, each with a specific wavelength, are emitted from each luminaire using an appropriate spectral radiant intensity distribution. They are then tracked as they travel through the environment until they are absorbed. Each time a particle hits a surface it is probabilistically absorbed or scattered in a new direction according to the BRDF (Bidirectional Reflectance Distribution Function)

of the surface. A list of all particle "hit points" where particles hit surfaces is generated and saved.

2. **Density-estimation phase:** The stored hit points are used to construct approximate lighting functions on each surface. The illumination on a surface is proportional to the density of the hit points; therefore this is a density estimation problem. In order to reconstruct luminous and tristimulus exitance[4], the hits are weighted by the surface reflectance and the CIE XYZ response functions and local linear density estimation is applied. The result is a Gouraud-shaded, or piecewise linear, mesh of triangles with three color channels suitable for direct display.

3. **Mesh decimation phase:** The initial mesh is generated conservatively and in most places it is much denser than is actually required. This mesh is decimated by progressively removing vertices as long as the resulting change is below a perceptually-based threshold. The simplified mesh requires less storage and can be displayed more quickly. For best results the decimation can be tuned for the characteristics of a particular display device.

An illustration of the three phases is shown in Figure 1. The final result is a view-independent illumination solution such as that shown in color plate A. The solution quality depends significantly on the number of particles, as illustrated in color plate K. Each particle hit point consists of a surface identifier, the collision location, and a wavelength, encoded using twelve bytes per hit. After particle tracing, these hit points are sorted by the surface identifier for efficient access by the density estimation phase. The sorting is performed using standard techniques and reduces the storage requirements to eight bytes per hit point. The basic dataflow for the density estimation framework is illustrated in Figure 2.

3.1 Other Benefits

Besides the reduction in complexity, this framework has several other benefits. It naturally divides the software into three modular pieces corresponding to the three phases. Once the format for data flowing between the phases is chosen, they can be implemented, debugged, and maintained independently. In our case each phase was implemented by a different person with very little coordination required.

[4]There is a difficult terminology issue that arises when simulating sensor response to radiometric quantities for cases other than the "photometric" sensor. An example of this difficulty is that the weighted integral of spectral radiant exitance is called "luminance" if and only if the weighting function is the human luminace response curve. If it some other weighting function, such as the response of a human color channel, there is no standard term for the result of the weighted integral. When using the triple of standard CIE chromatic weighting functions, $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$, we substitute the term "tristimulus" in place of "luminous" to construct an analog of a photometric quantity. Although this terminology is not standard, it smooths our discussion considerably.

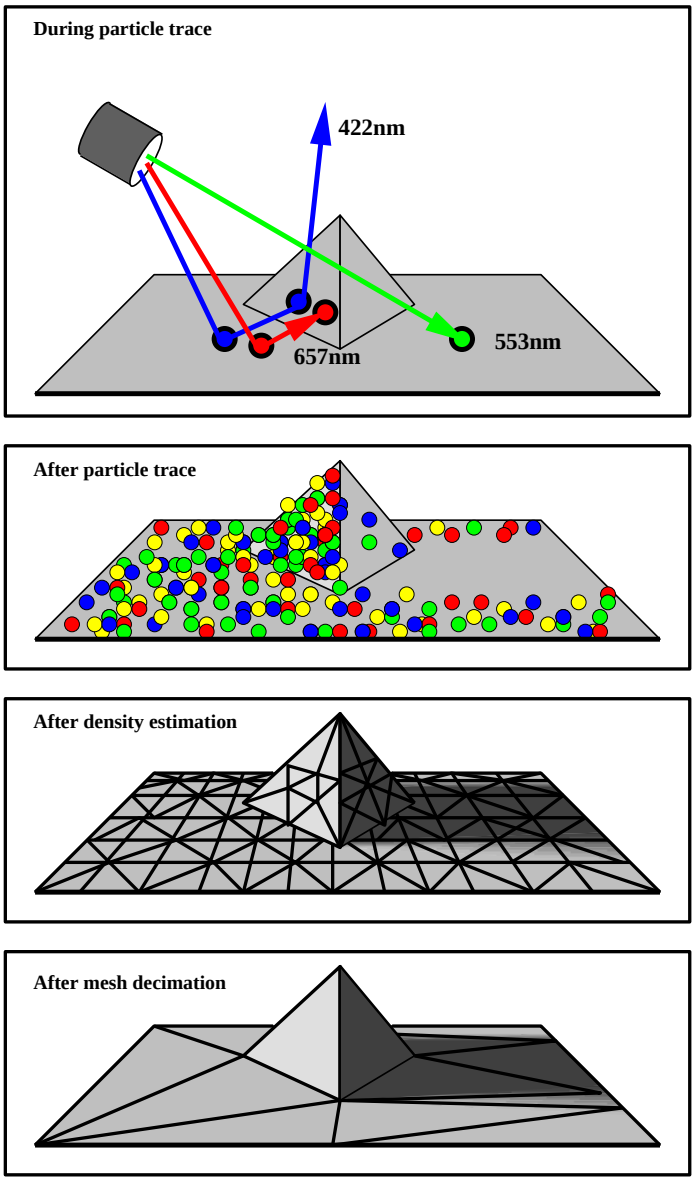


Figure 1: Overview of the density estimation algorithm.

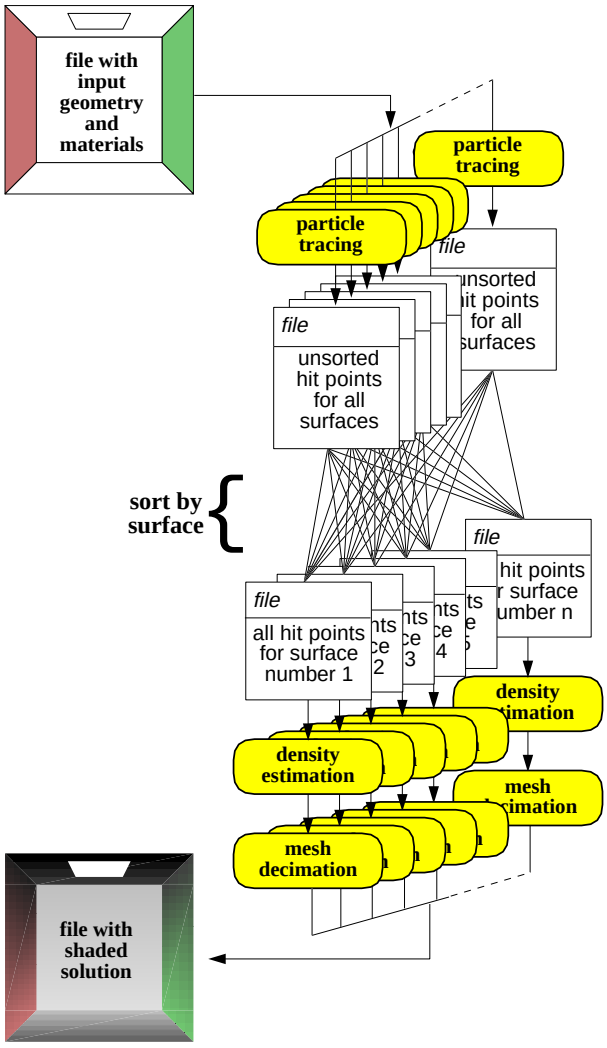


Figure 2: Overview of the density estimation algorithm.

This modularity also makes it much easier to optimize each phase for particular sub-goals in pursuit of the overall design goals. In our case we can optimize the particle tracing for physical accuracy of the light tranport, the density estimation for perceptual accuracy within the bounds of our chosen representation, and decimation for high compression rates while maintaining perceptual quality.

Another benefit is that this framework naturally contains easily exploited coarse-grain parallelism. Finite element methods are difficult to parallelize because they mix local and global operations. In our framework, it is easy to trace particles in parallel since particles do not interact. After sorting, lighting reconstruction on different surfaces can also be done in parallel since surfaces do not interact in that phase. As shown in Figure 2 communication is only required during the sorting. Zareski et al. [38] demonstrated this parallelism on a cluster of workstations.

3.2 Design Trade-offs

In the process of turning the general framework into a working implementation, we have had to make numerous design decisions and trade-offs. To help us make rational choices, we use architectural walkthroughs as our driving application. Our goal is to produce a view-independent global illumination solution which is perceptually accurate for a given input model and which can be displayed at interactive rates on current graphics hardware.

One of the first design decisions is what range of physical phenomena to model. We want to include all phenomena that are likely to be important in the input model with its potentially detailed material specifications. Fortunately, particle tracing is well suited to simulating a geometric optics approximation that includes nearly all the important lighting phenomena for architectural environments. We currently handle phonomena such as complicated emission functions, non-diffuse scattering, and dispersion; others, such as participating media and polarization, could be added, but long-range wave effects such as interference would be very difficult.

We currently restrict our input models to polygonal geometry only. Polygons are a simple and common geometry format that works well for many architectural models. There would be several advantages to being able to work directly with curved surfaces, but that is left as problem for future research.

As an output format we use Gouraud-shaded triangles, since this is the most widely supported format for interactive shaded display. Because Gouraud-shaded triangles' colors do not depend on viewing angle, they correspond to directionless lighting quantities such as irradiance or luminous exitance which completely specify the lighting only on purely diffuse surfaces. Diffuse-only output is a reasonable approximation if most of the surfaces are matte, or nearly diffuse, which is often true for architectural models. When display time is not critical, ray tracing can be used to display the solution and fill in some of this missing lighting information on non-diffuse surfaces. In principle, it is possible to solve for the directional lighting quantities needed for non-diffuse surfaces. But this would double the dimensionality of the density estimation

problem and necessitate finding good representations and decimation techniques for directional quantities. This is another area left for future research.

It is important to note that this diffuse-only approximation is introduced during the density estimation phase and does not affect the accuracy of the transport phase. For example, consider a metal reflector in a lamp fixture. The solution file will not have enough information to directly display the metal reflector correctly, but it will correctly contain the reflector's effect on the lighting at other surfaces. See color plate H for an example comparing diffuse and specular reflectors in a recessed light fixture. Another example of important non-diffuse transport is glass (e.g., windows). Color plate I illustrates that the effects of glass can be complex; it is not sufficient to simply remove the glass, which is a common practice.

For the particle tracing phase we have chosen to use particles that carry constant power and have a single wavelength. Ideally, we would like to have low power particles in regions that are dark or contain important features, such as shadow boundaries, and higher power particles elsewhere. However, in the absence of considerable *a priori* knowledge about the solution, using equal power particles is a reasonable compromise. In many cases, carrying a spectrum around with each particle would reduce the noise in solution for a fixed number of particles. We believe that this is offset by the advantages of a simpler particle tracer and reduced storage required per particle. It also makes it easy to simulate phenomena like the dispersion rainbow in color Plate J.

At each surface we could record either all incident particles or only reflected particles. Recording the reflected particles would most closely correspond to the luminous exitance and chromaticity[5] that we are solving for and would avoid storing large amounts of data on dark surfaces. We use the incident particles instead for two reasons: the incident particles are a superset of the reflected particles giving us more particle data for a given amount of work; and incident particles also allow us to reconstruct illuminance instead of luminous exitance on textured surfaces and thus avoid having to include the texture in the illumination mesh [4, 9] which would preclude effective decimation. We will still reconstruct the luminous exitance on untextured surfaces.

3.3 Improvements over Prior Implementation

The basic three-phase framework is similiar to the one we proposed earlier [29]; however that system was a proof-of-concept implementation and too limited for general use. In this paper we present several improvements that make the system more general and useful.

- The particle tracing has been changed from an RGB color space to spectral radiometry for better physical accuracy.
- We show how to sample non-trivial BRDFs and emission functions using rejection techniques with reasonable efficiency.

[5] Together these are equivalent to the tristimulus exitance used elsewhere in this paper. We switch terminology here due to a lack of appropiate standard terminology.

- A new density estimation technique based on locally-weighted linear least-squares regression handles boundary bias problems.
- This new local linear technique has been extended to arbitrary polygons, removing the previous restriction to rectangles only.
- The number of perceptible artifacts created during decimation have been reduced by algorithmic improvements.
- Perceptually-based heuristics and user studies provide a systematic way to choose decimation parameters.

4 Particle Tracing

The particle tracing phase operates much like the classic particle tracers of the heat transfer literature [33]. It takes as input the geometry, reflectance information, and emission information for each surface, and produces a file with a list of particle-surface interactions.

The particle tracing program processes n particle paths, where n is either set by the user, or is determined *a posteriori* by some termination criterion such as a minimum number of particle-surface interactions. For each particle, we choose a random position, direction, and wavelength according to the properties of the luminaires in the scene. At a surface it is either absorbed or scattered according to the BRDF of the surface. If scattered, the particle continues in a straight line in a new direction until it strikes the next surface. This process is repeated until the particle is probabilistically absorbed.

4.1 Choosing particle positions and directions

The emissive properties of the luminaires in the scene are described by the emitted spectral radiance defined over positions, directions, and wavelengths: $L_e(\mathbf{x}, \omega, \lambda)$, where $\mathbf{x}$ is a point on a surface, ω is a direction, and λ is a wavelength.

First we choose a particle position, direction, and wavelength with a probability density function $p_e(\mathbf{x}, \omega, \lambda)$. Because p_e need not be related to L_e, the power of outgoing particles must be adjusted to maintain correct overall emissive power distribution. The expression for the power ϕ carried by a particle that is randomly generated with density p_e from point $\mathbf{x}$, direction ω, and wavelength λ is:

$$\phi = \frac{L_e(\mathbf{x}, \omega, \lambda) \cos\theta}{n p_e(\mathbf{x}, \omega, \lambda)},$$

where θ is the angle between the direction ω and the surface normal at $\mathbf{x}$. As stated earlier, in our implementation we force all particles to carry the same power, which is guaranteed if we choose the appropriate p_e:

$$p_e(\mathbf{x}, \omega, \lambda) = \frac{L_e(\mathbf{x}, \omega, \lambda) \cos\theta}{\Phi},$$

where Φ is the total emitted power from all luminaires given by:

$$\Phi = \int_X \int_\Omega \int_\Lambda L_e(\mathbf{x}, \omega, \lambda) \cos\theta d\Lambda \; d\omega \; d\mathbf{x},$$

where Λ is the range of relevant wavelengths (we use 400 to 700 nanometers), Ω is the hemisphere of exitant directions, and X is the set of points on all surfaces.

When a particle strikes a surface, it is scattered with some probability s or absorbed with probability $1 - s$. If scattered, the outgoing direction is chosen randomly with probability density function $p_r(\omega)$. If the incoming particle has power ϕ, the power ϕ_r of the scattered particle is:

$$\phi_r = \phi \frac{f_r(\omega_i, \omega, \lambda) \cos\theta}{s p_r(\omega)}, \tag{1}$$

where ω_i is the incident direction of the particle before it hits the surface, and f_r is the spectral bidirectional reflectance distribution function (BRDF). To ensure that $\phi_r = \phi$, we can choose s and p_e such that their product is equal the numerator in Equation 1. The most straightforward way to do this is to choose $s = R$ and

$$p_r(\omega) = \frac{f_r(\omega_i, \omega, \lambda) \cos\theta}{R},$$

where R is the integrated hemispherical reflectance associated with the incident direction and f_r:

$$R = \int_\Omega f_r(\omega_i, \omega, \lambda) \cos\theta \; d\omega.$$

In our implementation we must choose positions, directions, and wavelengths for emitted particles according to p_e, and directions for scattered particles according to p_r. Because the particle tracing is providing samples in a high-dimensional space, the potential benefits of stratification are modest and we have chosen to use unstratified random sampling. The general strategies for generating non-uniform random variables are covered in Appendix A. Our job is made easier by the fact that in our current models the spatial, directional, and spectral emission properties are separable on any single luminaire. For opaque specular surfaces, we have the special case where f_r is a delta function [5], so the scattered direction is deterministic and is generated as a special case. For dielectrics, we probabilistically choose between a reflected and refracted ray; the probability of reflection varies with incident angle and wavelength according to Fresnel's equations [11]. The angle of refraction depends on wavelength to include dispersion.

We do not take polarization effects into account, although it is straightforward to do so [20]. Other phenomena such as fluorescence could also be added provided that they are linear with respect to the incident light [10]. Nothing in the density estimation or mesh decimation phases would need to be changed if polarization or fluorescence were to be added.

5 Density Estimation

After completing the particle tracing phase, we have a long list of locations where light particles hit surfaces. The particle tracing is designed so that the density of hit points is proportional to the amount of light striking the surface. What we need is a way to recover the density function from the hit points. This type of problem is called density estimation and is common enough that there is an entire field in statistics devoted to it [30, 35]. In this section we will discuss the kernel density estimation method, some heuristics for choosing appropriate parameters for this method, and a new modification to this method that eliminates the problem of boundary bias in a clean way.

5.1 Kernel Density Estimation

The histogram is the oldest and most widely known density estimation technique. It consists simply of dividing the domain of the problem into regions, or bins. We then form a piecewise constant approximation by counting the number of data points in each bin and dividing by the size of the bin. The simplicity of histograms is appealing, but there are other methods that generally produce better approximations to the real density function.

In the statistics literature, kernel density estimation seems to be the most widely used and recommended density estimation technique. The density function at a point is estimated by taking a weighted sum of nearby points and then dividing by the area under the weighting (or *kernel*) function. Usually the kernel is normalized to have unit area so that the division is unnecessary.

It is traditional to split the choice of a kernel function into the general shape of the kernel (in a canonical form, written as K) and a kernel width parameter h, which is usually called the *bandwidth*. The kernel scaled by the width h is written as K_h and defined as:

$$\mathrm{K}_h(\mathbf{x}) = \frac{1}{h^d}\,\mathrm{K}\left(\frac{\mathbf{x}}{h}\right) \tag{2}$$

where d is the dimension of domain. It is also traditional to assume the density function has unit volume, though this can easily be generalized to other densities by a scaling factor[6]. Given n data locations $\{\mathbf{X}_1 \ldots \mathbf{X}_n\}$ the estimated function $\tilde{f}$ is:

$$\tilde{f}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{K}_h(\mathbf{x} - \mathbf{X}_i) \tag{3}$$

We can think of this expression as a kernel centered at the estimation point used to weight nearby data points, or alternatively we can think of it as the

[6] This scaling factor is independent of the spatial distribution of data points and must be known from some additional information. In our case we can find the scaling factor because we know the amount of spectral power carried by each particle.

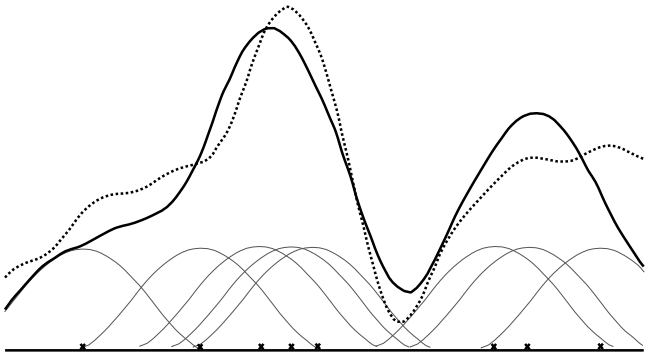


Figure 3: Kernel Density Estimation: data points (x's) are distributed according to a density function (dashed) and kernels (gray) centered on the data points are summed to form an estimate (solid) of the density function.

sum of n kernels centered on the data points[7] as shown in Figure 3. Thus our function estimate is simply a sum of suitably translated kernel functions. Essentially we are blurring the data points to produce a smooth estimate where the bandwidth parameter controls the amount of blurring.

Once a kernel function and bandwidth are chosen, it is quite straightforward to implement Equation 3. Unfortunately, there is a fundamental problem with kernel density estimation called boundary bias.

5.2 Boundary Bias

The cause of boundary bias is that the kernel method does not differentiate between regions that have no data points because the density function is near zero, and regions which have no data points because they lie outside the domain where we have information about the function. Effectively it assumes that the function goes to zero everywhere outside of the domain; consequently, there is a strong bias toward zero in regions that are within a bandwidth of the boundary of the domain. In our application this would show up as a noticable darkening near the edges of surfaces (see Figure 4).

This might not seem like a big problem because it only affects boundary regions, but in applications like ours, the boundary region can be a large fraction of the total domain. This is especially true for complex scenes, which often consist of many small polygons. A variety of techniques have been used to reduce boundary bias. One simple technique is to create "phantom" data outside the domain by reflecting data across the boundary [29]. Unfortunately, this only partially corrects for the boundary bias and is only easy to implement for shapes which can tile the plane (e.g., rectangles). In the statistics literature, when the boundary bias problem is dealt with at all it is usually by using modified kernels

[7] We will break this symmetry when we introduce the local linear method, but the intuition built here is still useful.

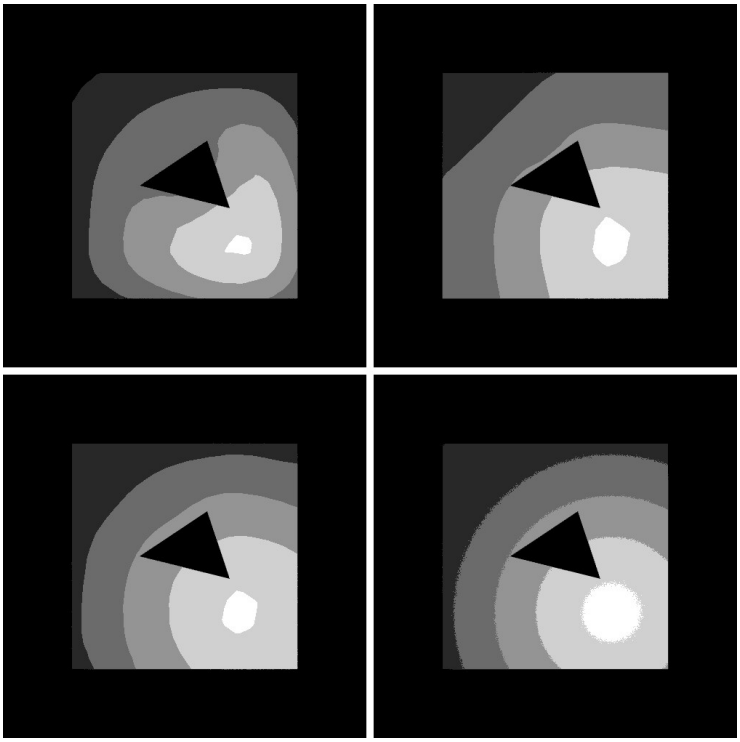


Figure 4: Illumination contours on a polygon with a hole. Reconstructed from $\approx$ 23,000 hit points using kernel density estimation (top left), local constant (top right), and local linear density estimation (bottom left), along with a path traced reference image (bottom right). The constant and linear methods are examples of local polynomial methods introduced in this paper. The reflection method of [29] is roughly similiar to the local constant method.

called boundary kernels (e.g., [35, p. 47]). The difficulty with boundary kernels is that they are derived in an *ad hoc* manner, and there is no general agreement as to which are the best.

We have developed a new method of eliminating boundary bias[8] (demonstrated in Figure 4) by adapting the locally-weighted polynomial least-squares regression method described below. The biggest advantage of this new method is that it has a clean conceptual basis. It also helps to build intuition about the behavior of kernel density estimation.

[8] Note that eliminating the boundary bias does not mean eliminating all bias in the boundary regions. Rather it means that the boundary regions are no more biased than the interior (non-boundary) regions.

5.3 Local-Weighted Polynomial Least-Squares Regression

Regression is the problem of reconstructing a function given noisy function estimates at a discrete set of points. It is closely related to density estimation and also suffers from boundary bias problems. In the regression literature, the locally-weighted polynomial least-squares regression technique has recently become very popular, in part because it eliminates boundary bias in a natural way [13]. At each point where we want to estimate the function we fit a polynomial to the function estimates using weighted least-squares. We use a kernel function to give large weight to nearby data and little or no weight to more distant data. We then use the value of the polynomial at that point as our estimate for the function value. Note that we fit a different polynomial at every point at which we estimate the function. This method is different from the kernel method in that it uses a weighted least-squares fit rather than a weighted sum.

In applying this method we are free to choose what degree polynomial to use for our local least-square fits. For density estimation problems, it turns out that odd-order polynomials do not have boundary bias, and that low order polynomials are usually better at filtering out noise. For these reasons we use linear, or first-order, polynomial fits.

From basic calculus, any smooth function appears linear or straight if you look at a small enough piece of the function. Thus, if we can use fits that are local enough, we should be able to fit any smooth function accurately. In practice this is not always achieved and the estimates will be biased to the extent that the function does not look like a straight line, or linear, over the region of a local linear fit. The bias will be greatest in regions of high curvature and at discontinuities in the function. In boundary regions we simply have less nearby data for our linear least-squares fits, which makes the results noisier but does not introduce any additional bias.

In order to apply this technique to our problem we need to transform this regression method into a density estimation method. We can turn a density estimation problem into a regression problem by performing a histogram on the data. Each histogram value is then an estimate of the density function at the bin center, and we can apply the locally-weighted linear regression method. This process is illustrated in Figure 5. This would still leave us with the problem of how to form the histogram bins and how many to use. Instead we eliminate the histogramming step by taking the limit as the number of histogram bins goes to infinity. We call this method *local linear density estimation* and the details are presented in Appendix B. One surprising result is that local linear density estimation is identical to kernel density estimation in the interior, and thus we can still use the intuition we have gained about kernel density estimation. However in boundary regions this new method automatically adapts to eliminate boundary bias.

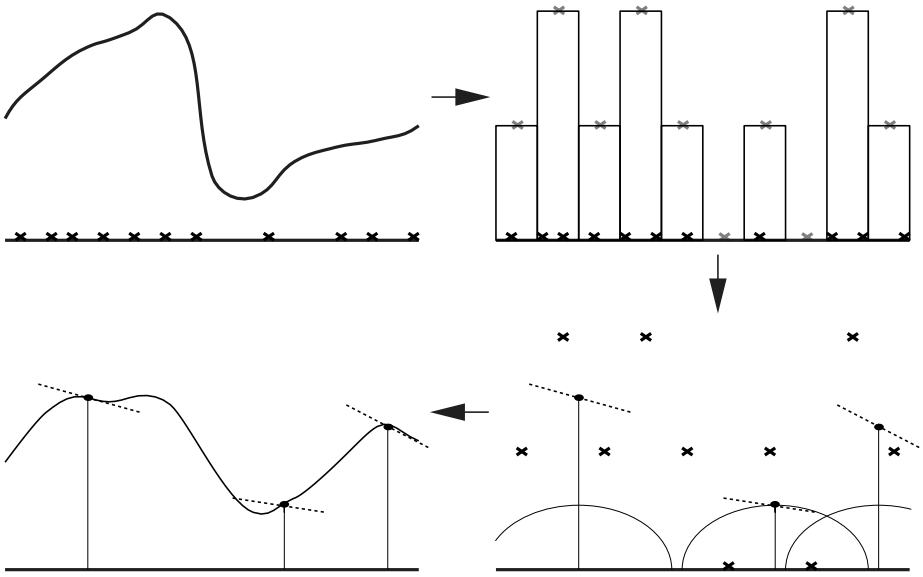


Figure 5: Density estimation performed using locally-weighted linear regression with a finite number of bins. Samples from a density function (top left) are histogrammed (top right),and then local linear regression is applied (bottom right) to produce the estimated density function (bottom left).

5.4 Choosing a bandwidth

In order to use local linear density estimation we need to choose a kernel and decide how to set the bandwidth parameter. Since this method is so closely related to kernel density estimation, which is easier to analyze, we will use some theoretical results about the kernel method to motivate our choices.

The conventional wisdom in the statistics literature [30, p. 43] which is borne out by our own experience [34], is that the exact shape of the kernel makes very little difference in the quality of the density estimation. However, we can reduce the computational costs by choosing a kernel that has compact support and is simple to calculate. For these reasons we have chosen a standard kernel known as the 2-D Epanechnikov kernel [30, p. 76]:

$$\mathrm{K}(\mathbf{x}) = \begin{cases} \frac{2}{\pi}(1 - |\mathbf{x}|^2) & \text{if} |\mathbf{x}| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Choosing a good bandwidth is much more difficult. To do this intelligently we need to understand the trade off between bias and variance. It is a well known result [35, p. 97] that the expected value of kernel density estimation is given by:

$$E\tilde{f}(\mathbf{x}) = \int \mathrm{K}_h(\mathbf{x} - \mathbf{y}) f(\mathbf{y}) d\mathbf{y} \tag{5}$$

In other words, our estimate $\tilde{f}$ converges, not to the correct function f, but rather to the correct function convolved with the kernel. This results in bias if the kernel width is greater than zero and implies that we want our bandwidth to be as small as possible.

Since we use a finite amount of data, our solution will also have variance, or noise. The variance is given by:

$$\mathrm{Var}\tilde{f}(\mathbf{x}) = \frac{V}{n}\left(\int \mathrm{K}_h(\mathbf{x}-\mathbf{y})^2 f(\mathbf{y})d\mathbf{y}\right) - \frac{1}{n}\left(\int \mathrm{K}_h(\mathbf{x}-\mathbf{y})f(\mathbf{y})d\mathbf{y}\right)^2 \quad (6)$$

where $V = \int f(\mathbf{y})d\mathbf{y}$ integrated over all surfaces. We can get the leading term in the variance by neglecting the second term above, using the Taylor expansion, $f(\mathbf{y}) \approx f(\mathbf{x}) + (\mathbf{y}-\mathbf{x})f'(\mathbf{x})$, using Equation 2, and assuming a symmetric kernel so that $\mathrm{K}(\mathbf{x}) = \mathrm{K}(-\mathbf{x})$ to find:

$$\mathrm{Var}\tilde{f}(\mathbf{x}) \approx \frac{V\int \mathrm{K}(\mathbf{y})^2 d\mathbf{y}}{nh^2} f(\mathbf{x}) \quad (7)$$

From this we can see that to reduce the noise we want to use a large bandwidth. The bandwidth controlled tradeoff between bias and variance is illustrated in color plate B.

Our heuristic solution to this dilemma is to choose a bandwidth just large enough to reduce the noise to a imperceptible (or at least tolerable) level. Visual noise perception corresponds roughly to contrast or relative error [9], therefore we could achieve a roughly constant level of noise everywhere by making the variance proportional to $f(\mathbf{x})^2$. This would require a continuously changing bandwidth, but our implementation is currently limited to using a constant bandwidth over a single surface[10]. Instead we will make the variance proportional to $\bar{f}_i f(\mathbf{x})$ where $\bar{f}_i$ is the average value of f over the ith surface. Thus our heuristic for setting the bandwidth on the ith surface is:

$$h = \sqrt{\frac{CA_i}{n_i \pi}} = \sqrt{\frac{CV}{n\bar{f}_i \pi}} \quad (8)$$

where A_i is the area of the ith surface, n_i is the total number of hit points on this surface, and C is our user settable parameter for adjusting the noise level which corresponds to the average number of data points within the support of a kernel. We typically use values for C in the range of four thousand to sixteen thousand.

5.5 Reconstructing perceptual functions

We have now presented all of the machinery necessary to apply density estimation to the hit points from the particle tracing phase. Recall that each hit point

[9] This fact is also used in choosing parameters for the decimation phase.

[10] This bandwidth restriction makes the implementation easier and the computation faster, but it does not achieve the best solution quality.

consists of a location $\mathbf{X}_i$ and a wavelength λ_i, and represents a fixed amount of power ϕ. We could apply density estimation directly to these hit points, ignoring their wavelengths, and reconstruct the incident power density (irradiance), but this is not the function we want.

Our goal is to display the lighting solution for human observers. Thus we want to reconstruct the values which most closely correspond to what a human observer would perceive. First we weight the particles by the surface reflectance to get the reflected power (spectral radiant exitance). Since our visual system perceives luminance and color rather than power, we further weight the particles to account for this. The standard for human color vision is the CIE 1931 Standard Colormetric Observer [18], which describes color vision in terms of the XYZ tristimulus values. We can calculate the X, Y, and Z values by weighting particles by the spectral response functions, $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$ respectively.

We have noticed that the most significant lighting features are usually shadows and highlights, which are largely luminance features. Thus it makes sense to use a smaller bandwidth for the luminous, or Y, channel and accept some additional luminance noise in exchange for better resolution of these important features, but changing the bandwidth also changes the bias. If we simply compute the X, Y, and Z channels at different bandwidths, this changing bias shows up as distracting color shifts at these features. Instead we compute all three channels at a larger bandwidth to estimate the color chromaticities [18], x and y. These chromaticities are then combined with an additional luminous channel computed at a smaller bandwidth to find the XYZ tristimulus values. This allows us to enhance the luminous resolution without introducing unwanted color artifacts. This technique was used in Color Plates A, F, and G.

We also need to decide how to represent the solution. Since our goal is to display the solution, we sample the reconstructed function at a mesh of points and use a piecewise linear approximation. This has the advantage that the elements can be rendered in hardware as Gouraud-shaded triangles. By the nature of the density estimation, any features which are much smaller than the bandwidth will be strongly filtered or blurred out in the reconstruction. Thus we can create a mesh of points which is dense enough to capture the features which are reconstructed. We are not especially worried about generating too fine a mesh in this phase since the mesh will be decimated in the next phase. For textured surfaces we do not want to include the texturing in the illumination mesh as this would preclude effective decimation. Instead we compute these surfaces as if they were white, or perfect diffuse reflectors, and then multiply by the texture at display time [4, 9]. This is an reasonable approximation since our textures are currently RGB and not spectral.

Once we have chosen the bandwidth and generated the illumination mesh, we can perform the density estimation at all of the mesh vertices simultaneously using a single pass through the hit point data. Since we cannot assume that we can fit the hit point data into physical memory, using only a single pass makes the implementation simpler and faster. The drawback is that we can only use, at most, a few bandwidths per surface. We believe that in many cases using greater bandwidth variation would be worth the additional implementational

complexity, but this is left as future work.

6 Mesh Decimation

The mesh created by the density estimation phase captures the details in the solution, but it may be prohibitively large for most applications. An interactive walkthrough of the solution requires a mesh with few enough triangles to be rendered in real-time. To meet this requirement, the decimation phase provides various levels of reduction in the number of triangles while maintaining as much accuracy and detail as possible. These qualities are defined in perceptual terms, because the goal of a walkthrough is display. The decimation phase therefore exploits a simple, perceptually motivated calibration technique to minimize the artifacts introduced as the mesh is reduced.

This perceptual component of the decimation phase augments an approach based on geometric simplification. Each surface in the scene is planar, so the mesh that represents the solution on this surface is two dimensional. We call this 2-D mesh the surface's *illumination mesh*, M_I. For each vertex of this mesh, the decimation phase adds a third coordinate, $z(L)$, corresponding to the *luminance*, L, of the illumination at that vertex. This process transforms M_I into a 3-D geometric mesh, M_G. To each such M_G, the decimation phase applies an algorithm that simplifies 3-D geometry. The effect of such an algorithm is the removal of vertices whose luminances can be approximated by interpolation from nearby vertices. Removing the third coordinate of the simplified M_G transforms it back into a simplified illumination mesh, M_I'. Color plate C illustrates this process. Hughes et al. [16] describe an alternative approach that does not build a 3-D geometric mesh. We believe, however, that the use of this mesh has the advantages of aiding intuition and allowing the reuse of previous work in geometric simplification.

As part of our earlier work [29], we developed a basic implementation of this approach, which simplifies M_G with an extended version of the Schroeder et al. geometric simplification algorithm [27]. In its basic form, this algorithm estimates the *cost* of each vertex, V, in terms of how much V's removal would change the mesh; if the cost is below a threshold, the algorithm removes V and re-triangulates the resulting hole. Our earlier work extends this algorithm in two ways. The first extension prohibits changes to M_G that would produce overlapping triangles when M_G is transformed back into M_I'. To avoid this problem, the algorithm removes a vertex only when it can re-triangulate the resulting hole without creating overhanging ledges (i.e., triangles whose normal vectors have negative z components). The second extension reorganizes the algorithm to remove vertices in order of increasing cost. The algorithm uses a priority queue to maintain this ordering efficiently. The remainder of this section describes additional extensions that we have subsequently found important for reducing perceptible artifacts in M_I'.

Perceptual issues are significant in our revised approach to computing the cost of removing a vertex, V. Using the standard Schroeder et al. approach [27],

the cost is the length of the 3-D vector from V to an approximation for the new mesh without V. In our context, however, this length mixes the spatial units of M_I's two dimensions with the luminance units of the added dimension, z. Our earlier approach [29] requires a user-specified parameter to scale these units relative to each other, but in general, there is no good way to choose this parameter. Our current approach avoids the scaling problem by using only the z component of the vector.

With this approach, the cost measures the change in luminance. It is well known that people respond to luminance changes in a nonlinear manner. There are many models for this nonlinear response, as summarized by Sezan et al. [28], but the accuracy of these models depends on the task and visual conditions. In our earlier work [29] we tried a standard model, the cube root function, but we have subsequently obtained better results using our own empirical model tailored to our specific application. We conducted informal user studies, described in Appendix C, to determine conservative and liberal estimates of the perceptible change in luminance, $\Delta(L)$, for a baseline luminance, L. The goal is then to make the added dimension, z, meet the following constraint:

$$z(L + \Delta(L)) - z(L) = 1 \tag{9}$$

With this constraint, removing a vertex with cost below 1 should produce a change below the empirical perceptual threshold. The decimation phase is thus automatically calibrated, freeing the user from guessing cost thresholds at run time.

The data on perceptible luminance changes from our user studies roughly fits a linear model,

$$\Delta(L) = aL + b \tag{10}$$

A problem with this data is that it is in the units of the display monitor, as opposed to the units of the simulated luminances known to the decimation phase. The relationship between these units depends on the "white point," L_w, which is the simulated luminance that will be displayed at the monitor's brightest level. Since L_w is a parameter that the user may adjust when the decimated mesh is displayed, we use a two-pass approach to decimation. The first pass is conservative enough that its results should look acceptable at any L_w. It uses a lower bound on $\Delta(L)$ that does not change with the scaling produced by L_w. The simplest way to derive this lower bound is to set $b = 0$. In this case, a solution to Equation 9 is

$$z(L) = \frac{1}{\log(1 + a)} \log(\max(L, L_b)) \tag{11}$$

where L_b is the "black point," the lowest possible luminance; we use $L_b = 10^{-7}\text{candelas/m}^2$, which is below the luminance for a moonless overcast night as quoted by Glassner [11]. We use $a = 0.063$, the value determined by the conservative user study with our display conditions, as described in Appendix C.

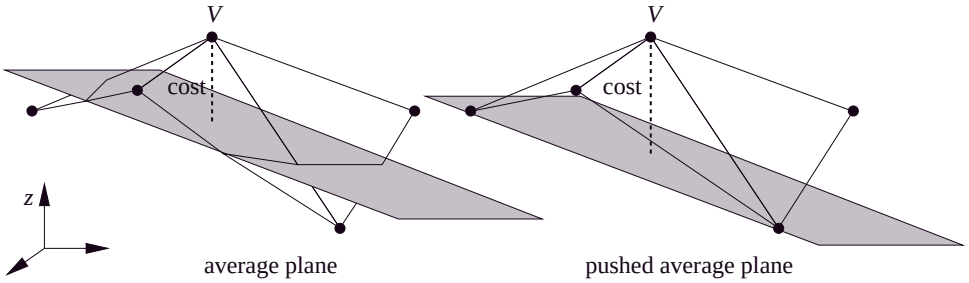


Figure 6: The plane that averages the faces adjoining V, and that plane pushed to the furthest neighboring vertex.

The second pass, which can be run once L_w has been chosen, performs a more liberal decimation of the results from the conservative pass. The solution to Equation 9 for this case is

$$z(L) = \frac{1}{\log(1+a)} \log(a \min(L/L_w, 1) + b) \tag{12}$$

with our liberal user study indicating $a = 0.13$ and $b = 0.0017$ for our viewing conditions. The decimation phase can take a user-specified parameter, ϵ, and remove vertices with costs below ϵ. For $\epsilon > 1$, this extension allows additional passes that further simplify the results of the liberal decimation, for situations in which simplification is more important than perceptual accuracy.

Once M_G is built with the appropriate z, measuring the cost of a vertex, V, requires approximating what the mesh will be after the removal of V. The standard Schroeder et al. algorithm uses a local, planar approximation, with the plane being the area-weighted average of the faces adjoining V. We obtain better cost estimates by pushing this plane away from V along its normal vector until it reaches V's furthest neighboring vertex, as shown in Figure 6. This extension tends to make the cost more conservative in regions containing shadow boundaries, thus reducing artifacts in those regions.

The final step in the removal of V is triangulation of the hole opened by the removal. Schroeder et al. [27] describe a simple greedy algorithm that chooses triangulation edges to maximize triangle aspect ratios. In our experience, fewer artifacts result if the greedy algorithm instead chooses edges that minimize differences between the old and new meshes. To measure the difference, we project each candidate edge, E, onto the triangles from the old mesh and measure the change in luminance along E, as illustrated in Figure 7.

Each of these new features of the decimation phase improves the quality of the results, but it is difficult to rank their relative importance in isolation. Their advantages are most apparent when they are used together. Color plate D shows the floor of a small room in three forms: undecimated (55129 vertices, 108710 triangles); decimated with our old approach [29] (265 vertices, 470 triangles); and decimated with the new approach (265 vertices, 457 triangles). We

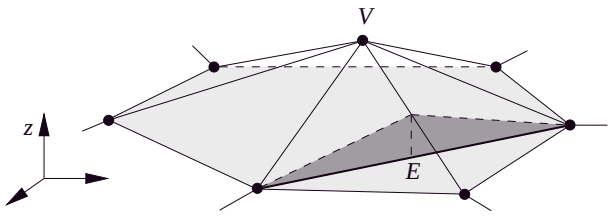


Figure 7: The dark region indicates the change in luminance along a candidate edge, E, projected onto the old mesh for a vertex, V.

constrained the old approach to remove the same number of vertices as the new approach. Even so, the new approach shows fewer artifacts and looks significantly more like the undecimated image. Section 7 shows the results of the new algorithm on larger, more complicated scenes.

These images are not completely free of artifacts. Some artifacts are due to the output media; to calibrate the decimation phase, we conducted a user study on video monitors instead of the printer that produced these images. Other artifacts are evidence that further work could improve the decimation phase. Currently, the cost estimate for the removal of a vertex considers effects on the mesh's luminance but not its color. Humans are more responsive to changes in luminance than to changes in color, so this approach is appropriate in most cases, but there can be situations in which it would produce visible artifacts. One solution would be to add a mechanism that detects these artifacts and then cancels the vertex removal that caused them. The decimation phase would also benefit from mechanisms that consider perceptual issues beyond luminance and color. An important issue is discontinuities in shading that produce perceptible artifacts, such as Mach bands. Removing a vertex from an illumination mesh may increase these artifacts, so the decimation phase needs a way to predict this increase when computing a vertex's removal cost. These improvements in cost estimation might also work better in conjunction with a more advanced geometric decimation algorithm such as Hoppe [15].

7 Results

We have implemented all three phases of the algorithm including small-scale parallel versions of particle tracing and density estimation. Timings are approximated for an equivalent serial execution on Hewlett-Packard 9000 735/755 workstations (SPECint95 3.27 SPECfp95 3.98).

In order to test the accuracy of our solutions, we have rendered a simple scene (color plate E) using an undecimated density estimation solution, a path tracing implementation, and Ward's Radiance [36] system. Note that the density estimation solution was ray traced in order to display the specular surfaces correctly. It is reassuring that all three methods produce very similiar results. Also shown are two decimated versions of the density estimation which are

nearly identical to the undecimated image but which use far fewer triangles.

We have also included solutions for three architectural models displayed using hardware scan conversion: the Science Center (color plate F), the Alto Library (color plate A), and Falling Water (color plate G). Surface and triangle counts have been rounded to two significant digits. To choose bandwidths we used $C = 16,000$ for the chromaticities and $C = 8,000$ for the additional luminance channel.

The Science Center consists of about 4,000 initial surfaces. The particle tracing took 17 hours to produce 200 million particle hits. The density estimation took 11 hours and produced one million triangles. Conservative decimation reduced this to 150,000 triangles in 30 minutes, and liberal decimation further reduced this to 33,000 triangles in an additional 8 minutes.

The Alto Library consists of 13,000 initial surfaces. Particle tracing produced 200 million hit points in 47 hours. Density estimation took 10 hours to produce 1.5 million triangles. Conservative decimation reduced this to 330,000 triangles in 32 minutes and liberal decimation further reduced it to 100,000 triangles in 10 minutes.

The Falling Water model consists of 140,000 initial surfaces. Particle tracing took 47 hours to produce 225 million hits. Density estimation produced 1.7 million triangles in 11 hours. Conservative decimation reduced this to 490,000 triangles in 47 minutes. Liberal decimation reduced this to 180,000 triangles in an additional 20 minutes. This is the largest model we have tried, and it required only sixty megabytes of RAM. Conventional wisdom suggests that this is an order of magnitude less memory than is required by conventional view-independent global illumination techniques.

8 Conclusions

The density estimation framework has many advantages which make it a natural tool for computing view-independent global illumination. By separating the global transport from the local lighting representation, this framework decomposes the problem into distinct phases with greatly reduced complexity. Other benefits include non-diffuse transport, software modularity, and natural parallelism.

There are several difficulties in turning the potential of this framework into a real working system. In this paper we have presented techniques which allow each of the three phases to be implemented for general polygonal models.

In particle tracing, we have implemented a faithful simulation of geometric optics including spectral effects and non-trivial BRDFs. In density estimation, we have shown how boundary bias can be eliminated using the local linear method and how this method can be efficiently implemented on arbitrarily shaped polygons. The density estimation phase translates from the physical to the perceptual domain and we have derived perceptually motivated heuristics for setting the density estimation parameters. Finally, we have shown how mesh decimation can be used to produce output which is small enough to be dis-

played at interactive rates, and we have found perceptually informed heuristics which make the decimation effective while preserving perceived image quality. We believe that our system as presented is a very appealing way to produce view-independent global illumination solutions.

8.1 Problems and Future Work

While the present work represents a significant step toward making the density estimation framework a practical method for computing view-independent global illumination solutions, it still suffers from a number of problems and limitations. It is our belief that there is still a lot of opportunity to improve each phase of the method and that such improvements will greatly reduce many of these problems. The following is a partial list of current problems.

- Large computational requirements. While we believe that our method is at least competitive with other view-independent methods, it may be too expensive for people who are not used to or do not require view-independent methods.
- Appearance of non-diffuse surfaces. An expensive display computation is required to get the appearance of non-diffuse surfaces right, and such computations are currently too slow for use in interactive walkthroughs.
- Curved surfaces. The method is currently limited to polygonal models.
- Small and/or narrow polygons. The method relies on the ability to collect and smooth lighting information over nearby surface regions in order to reduce the noise. If a polygon is too small to permit this then its solution will contain an unacceptable amount of noise.
- Large variation in illumination over a surface. The bandwidth controls bias/noise tradeoff, and its proper value varies with illumination level. We are currently limited to a single bandwidth per surface even if the illumination level varies considerably over a surface. This can result in excessive blurring in brighter areas and excessive noise in darker areas of a polygon.
- Large variation in illumination over the scene. The spatial resolution of the method varies with illumination level. If some important regions are much darker than others, we may exhaust our space for storing particle hits before we get enough particle hits in the dark regions.
- Large surfaces. In parallelizing the density estimation, we currently assign each surface only one processor. If one surface receives a large proportion of the total particle hits, it causes a load balancing problem in the parallel implentation.

Some of the future improvements that we expect are:

- Weighted particles. By allowing particles to carry different amounts of power, it becomes possible to use importance sampling techniques in the particle tracing. The computation could then be steered to use more particles in important regions and fewer in unimportant regions.
- Error analysis. We believe that a simple but precise error analysis is feasible for this method. This would not only tell us how good our solution is, but could also be used to set parameters and steer the computations to produce better results.
- Continuously varying bandwidths. This would give finer control over the noise in the solution and allow a more aggressive reduction of the bias without introducing visible noise.
- Curved surfaces. We believe that the local polynomial density estimation methods will provide a nice mathematical basis for extending the density estimation technique to curved surfaces.
- Reconstructing non-diffuse appearance. It is possible to include some angularly dependent quantities in the density estimation reconstruction, though it is probably infeasible to reconstruct (or store) complete non-diffuse information for surfaces such as mirrors or glass. Exploring these trade-offs is a wide open area for future research.
- Better perceptual model. The decimation still produces some visual artifacts. A more complete model of perceptual error could help avoid this artifacts and achieve greater levels of decimation. The density estimation could also benefit from an improved perceptual model.

A Generating nonuniform random variables

Given a one-dimensional probability density function $p(x)$ with domain $[a, b)$, and a *canonical* random number ξ that is uniformly distributed on $[0, 1)$, a random number x' with density p is given by:

$$x' = P^{-1}(\xi),$$

where P is the cumulative probability distribution function associated with p:

$$P(x) = \int_a^x p(z)dz.$$

We call this method the *integration-inversion* method. This method extends to multidimensional random variables as described in Glassner [11].

Unfortunately, it is not always possible to find analytic forms for P or P^{-1}. To avoid CPU-intensive numerical integration and function inversion, *rejection* techniques can be used. The simplest rejection technique uses an upper bound C for p such that $p(x) \leq C$ everywhere.

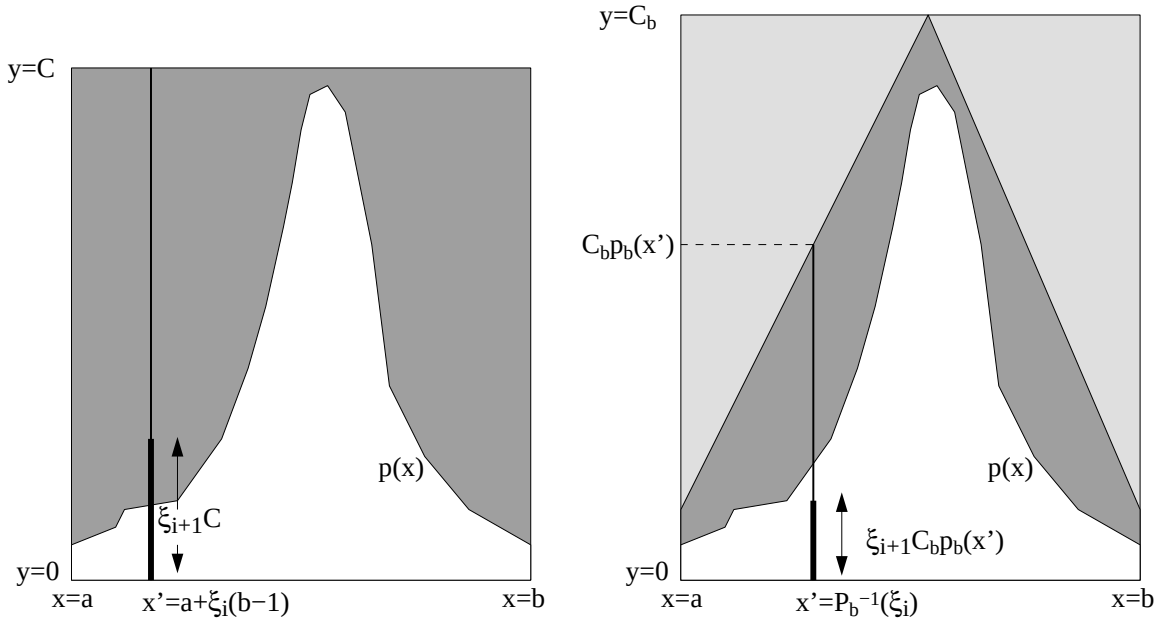


Figure 8: The same random seed is used for left: (a) uniform rejection, where the transformed point is in the grey region and is rejected, and right: (b) nonuniform rejection, where the transformed point is in the white region and is accepted.

A random number with density p can be generated from a series of canonical random pairs (ξ_i, ξ_{i+1}) by taking $x' = a+(b-a)\xi_i$ if $C\,\xi_{i+1} < p(x')$, and *rejecting* this pair otherwise. This process is repeated until a pair is not rejected. This process is illustrated in Figure 8(a), where (ξ_i, ξ_{i+1}) are scaled to form a random point in a rectangle with width $b-a$ and height C. If this point is above p then it is rejected. If it is below p then its x coordinate is used as the random variable having density p.

There are two problems with this simple rejection method. The first is that it is hard to stratify the samples, which is why the integration-inversion technique is usually used. The second is that it will take an expected $(C-1)$ rejections before a random number is successfully generated. Stratification is not an important issue for high-dimensional problems, and the inefficiency of rejection can be addressed with *non-uniform* rejection. Non-uniform rejection uses a bounding probability density function $p_b(x)$ and constant C_b such that $p(x) \le C_b p_b(x)$ everywhere.

The rejection process first generates a random number x' with density p_b using integration-inversion. This number is accepted if $\xi_{i+1} C p_b(x') < p(x')$. By choosing an appropriate p_b the number of rejections can be dramatically reduced over simple inversion for p that have high peaks. This process is illustrated in Figure 8(b), where ξ_i is used to generate an x coordinate with density p_b, and ξ_{i+1} is used to assign a y coordinate such that the (x, y) pair is a uniform random 2D point in the area under the curve $y = p_b(x)$. If the point is below p then its x coordinate is used as the random variable having density p.

B Local Linear Density Estimation

In this appendix we describe how to turn the locally-weighted linear least-squares regression method into a density estimation method. Regression problems can be stated as follows: given a set of points $\{\mathbf{X}_i\}$ and corresponding (possibly noisy) estimates $\{Y_i\}$ of an unknown function at these points, try to reconstruct the function. This is similiar to density estimation except that in density estimation we only have the points $\{\mathbf{X}_i\}$, not the estimates, and we have to try to reconstruct the function from the density of the points. We can transform a density estimation problem into a regression problem by chopping the domain up into a large number of small bins and performing a histogram on the data. We can then use the bin centers as our points $\{\mathbf{X}_i\}$ and the histogram values give us our $\{Y_i\}$ estimates of the density function. The histogram value Y_i is just the number of data points in bin i divided by its area, A_i, and divided by the total number of data points, n. In itself this is not a very good approximation for the function, but we can use this as input for a regression method such as locally-weighted linear least-squares regression. This process is illustrated in Figure 5 for one dimension.

We will work in two dimensions, denoting 2D points in boldface and their two spatial coordinates by the subscripts u and v. Let $\mathbf{x}_i$ be the center of the ith histogram bin, with area A_i and histogram value Y_i. Let m be the number

of bins, and $\mathrm{K}_h(\mathbf{y})$ be the kernel function scaled for our choice of bandwidth. Then we can write the local least squares fit at point $\mathbf{x}$ as a matrix equation using the following matrices. Let B be the matrix whose columns are the basis functions for a polynomial fit, given in our case by:

$$B = \begin{bmatrix} 1 & [\mathbf{x}_1 - \mathbf{x}]_u & [\mathbf{x}_1 - \mathbf{x}]_v \\ \vdots & \vdots & \vdots \\ 1 & [\mathbf{x}_m - \mathbf{x}]_u & [\mathbf{x}_m - \mathbf{x}]_v \end{bmatrix} \tag{13}$$

W be a diagonal matrix of weights:

$$W = \begin{bmatrix} \mathrm{K}_h(\mathbf{x}_1 - \mathbf{x})A_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathrm{K}_h(\mathbf{x}_m - \mathbf{x})A_m \end{bmatrix} \tag{14}$$

and β be the vector of coefficients for the linear polynomial:

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_u \\ \beta_v \end{bmatrix} \tag{15}$$

where the fitted polynomial is $\beta_0 + \beta_u[\mathbf{y} - \mathbf{x}]_u + \beta_v[\mathbf{y} - \mathbf{x}]_v$. Note that our function estimate at this point, $\mathbf{x}$, will just be the value of β_0. Finally, let Y be the vector of computed histogram values:

$$Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_m \end{bmatrix} \tag{16}$$

If the histogram values happened to lie exactly on a plane, we could solve for β using the equation $B\beta = Y$, but in general this equation will have no solution. We could instead find the closest solution in the least squares sense by using the normal equation $B^T B\beta = B^T Y$. However this would be a global least squares fit. We turn this into a local least squares fit by using a weighting matrix to give more influence to nearby values. The locally-weighted least squares fit which minimizes $\|W^{1/2}B\beta - W^{1/2}Y\|$ is given by:

$$B^T W B\beta = B^T W Y \tag{17}$$

We can multiply these matrices to find:

$$B^T W B = \begin{bmatrix} Q_{0,0} & Q_{1,0} & Q_{0,1} \\ Q_{1,0} & Q_{2,0} & Q_{1,1} \\ Q_{0,1} & Q_{1,1} & Q_{0,2} \end{bmatrix}$$

$$Q_{i,j} = \sum_k^m \mathrm{K}_h(\mathbf{x}_k - \mathbf{x})A_i([\mathbf{x}_k - \mathbf{x}]_u)^i([\mathbf{x}_k - \mathbf{x}]_v)^j \tag{18}$$

and:

$$B^T W Y = \begin{bmatrix} \sum_i^m \mathrm{K}_h(\mathbf{x}_i - \mathbf{x}) A_i Y_i \\ \sum_i^m \mathrm{K}_h(\mathbf{x}_i - \mathbf{x}) A_i Y_i [\mathbf{x}_i - \mathbf{x}]_u \\ \sum_i^m \mathrm{K}_h(\mathbf{x}_i - \mathbf{x}) A_i Y_i [\mathbf{x}_i - \mathbf{x}]_v \end{bmatrix} \tag{19}$$

All of these values could be calculated and the matrix equation solved by standard techniques, and indeed this procedure is often used in the statistics literature [25]. The difficulty with this is in choosing the number of histogram bins, m, to use. If we use too few bins, we will have thrown away too much spatial information, adding a complicated bias to our estimate. If we use too many bins, the computation becomes too expensive. Analyzing these trade offs is difficult and we know of no reliable automatic way to set this parameter.

Fortunately, there is a way to avoid this whole issue: take the limit as the number of bins goes to infinity. In the process, our sums turn into integrals and our histogram bins become delta functions if they lie exactly on a hit point or are equal to zero otherwise.

The matrices now become:

$$B^T W B = \begin{bmatrix} M_{0,0} & M_{1,0} & M_{0,1} \\ M_{1,0} & M_{2,0} & M_{1,1} \\ M_{0,1} & M_{1,1} & M_{0,2} \end{bmatrix}$$

$$M_{i,j} = \int_D \mathrm{K}_h(\mathbf{y} - \mathbf{x}) ([\mathbf{y} - \mathbf{x}]_u)^i ([\mathbf{y} - \mathbf{x}]_v)^j d\mathbf{y} \tag{20}$$

and:

$$B^T W Y = \begin{bmatrix} \frac{1}{n} \sum_j^n \mathrm{K}_h(\mathbf{X}_j - \mathbf{x}) \\ \frac{1}{n} \sum_j^n \mathrm{K}_h(\mathbf{X}_j - \mathbf{x}) [\mathbf{X}_j - \mathbf{x}]_u \\ \frac{1}{n} \sum_j^n \mathrm{K}_h(\mathbf{X}_j - \mathbf{x}) [\mathbf{X}_j - \mathbf{x}]_v \end{bmatrix} \tag{21}$$

where n is the number of hit points, $\mathbf{X}_j$ are the hit points, and D is the intersection of the support of the kernel and the domain (which is a polygon in our case).

One extremely important case is when the support of the kernel lies entirely within the domain. Since we use symmetric kernels, we can easily show that the off-diagonal elements of the matrix (20) are zero in this case. Recalling that kernels are normalized such that $\int \mathrm{K}_h(\mathbf{y}) d\mathbf{y} = 1$, our estimate reduces to the simple form:

$$\tilde{f}(\mathbf{x}) = \beta_0 = \frac{1}{n} \sum_j^n \mathrm{K}_h(\mathbf{X}_j - \mathbf{x}) \tag{22}$$

This is exactly equivalent to Equation 3 for standard kernel density estimation.

When we are in a boundary region then the matrix $B^T W B$ will generally not be diagonal and we will need solve it using Equations 17, 20, and 21. Thus by taking the limit, we have transformed a regression method into a density estimation method which turns out to be identical to the standard kernel method in non-boundary regions, but automatically adapts to eliminate boundary bias in boundary regions.

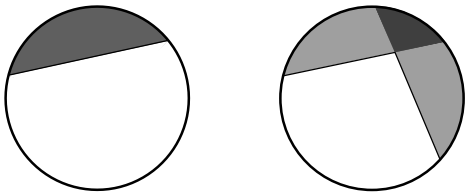


Figure 9: Dark shaded areas are the part of a kernel's support affected by an edge (left) and a vertex (right).

B.1 Computing Kernel Moments

There is one large difficulty with the local linear density estimation method derived above. Solving the integrals in Equation 20, which we will call *kernel moments*, is a non-trivial problem in boundary regions. We can take advantage of the fact that our surfaces are polygons to find analytic formulas for these integrals. We accomplish this by breaking the computation up into three classes: kernel moments when the support of the kernel lies completely within the polygon, modifications to those moments when an edge of the polygon crosses the support of the kernel, and modifications to the edge effects when two edges meet at a vertex within the support of the kernel (see Figure 9). It can be shown that these three pieces are complete in that when summed they suffice to calculate the kernel moments on any arbitrary polygon. Interestingly this construction is somewhat similiar to the polygon anti-aliasing filters used in [8].

The entire kernel, edge, and vertex formulas could be solved by a symbolic math package, such as Mathematica [37]. However the results turn out to be too complicated, so we will simplify the integrals before solving them. The first simplification is to notice how Equation 20 changes under rotations about the evaluation point, $\mathbf{x}$. We define three quantities $\mathcal{M}_0$, $\mathcal{M}_1$, and $\mathcal{M}_2$ by:

$$B^T W B = \left[\begin{array}{c|cc} M_{0,0} & M_{1,0} & M_{0,1} \\ \hline M_{1,0} & M_{2,0} & M_{1,1} \\ M_{0,1} & M_{1,1} & M_{0,2} \end{array}\right] = \begin{bmatrix} \mathcal{M}_0 & \mathcal{M}_1 \\ \mathcal{M}_1{}^T & \mathcal{M}_2 \end{bmatrix} \tag{23}$$

We could work out the rotation equations directly from the integrals, but its easier to notice that $\mathcal{M}_0$, $\mathcal{M}_1$, and $\mathcal{M}_2$ are zeroth, first, and second order tensors [7] respectively. Given a coordinate frame and an angle θ, we can express a point, $\mathbf{y}$, in coordinate frame rotated by θ as $\mathbf{y}' = R\mathbf{y}$ where:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{24}$$

We can express the tensor moments in the rotated coordinate frame by the following formulas:

$$\begin{aligned} \mathcal{M}_0 &= \mathcal{M}'_0 \\ \mathcal{M}_1 &= \mathcal{M}'_1 R \\ \mathcal{M}_2 &= R^{-1}\mathcal{M}'_2 R \end{aligned} \tag{25}$$

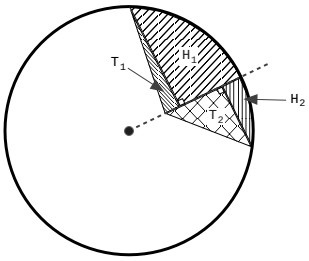


Figure 10: To simplify the computation vertex kernels are split into four regions: two right triangles (T_1 and T_2) and two half-edges (H_1 and H_2).

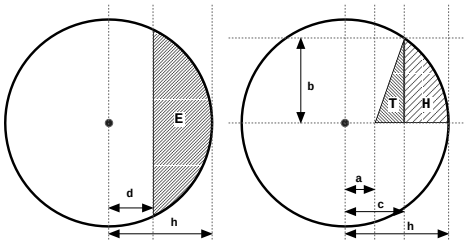


Figure 11: The three standard shapes for computing kernel moments: edges (E), right triangles (T), and half-edges (H).

Thus we can compute the edge and vertex moments in a canonical orientation and then transform them using the above equations. The second simplification we will use is to split the vertex moments into four pieces as shown in Figure 10. Now we only need to compute the moments for four simple shapes: the whole kernel, a vertical edge, a vertical half-edge, and an axis-aligned right triangle. The last three are shown in canonical position with their relevant parameters in Figure 11. Note that a, b, c, and d are all signed quantities. As shown they are all positive, but some care is required to get the signs correct in other cases. Let us define the following integrals for these pieces of the kernel moments:

$$M^W_{i,j} = \int_{-h}^{h} \int_{-\sqrt{h^2-u^2}}^{\sqrt{h^2-u^2}} \mathrm{K}_h(u,v) u^i v^j \, dv \, du \tag{26}$$

$$M^E_{i,j} = \int_{d}^{h} \int_{-\sqrt{h^2-u^2}}^{\sqrt{h^2-u^2}} \mathrm{K}_h(u,v) u^i v^j \, dv \, du \tag{27}$$

$$M^H_{i,j} = \int_{c}^{h} \int_{0}^{b\sqrt{\frac{h^2-u^2}{b^2}}} \mathrm{K}_h(u,v) u^i v^j \, dv \, du \tag{28}$$

$$M_{i,j}^{T} = \int_{a}^{c} \int_{0}^{b\frac{u-a}{c-a}} \mathrm{K}_h(u,v) u^i v^j \, dv \, du \tag{29}$$

Below are listed the results of solving these integrals for the Epanechnikov kernel (Equation 4) using Mathematica [37].

$$\begin{aligned} M_{0,0}^{W} &= 1 \\ M_{1,0}^{W} &= M_{0,1}^{W} = M_{1,1}^{W} = 0 \\ M_{2,0}^{W} &= M_{0,2} = \tfrac{h^2}{6} \end{aligned}$$

$$\begin{aligned} M_{0,0}^{E} &= \tfrac{2d^3e - 5deh^2 + 3h^4 \arccos(\frac{d}{h})}{3h^4\pi} \\ M_{1,0}^{E} &= \tfrac{8e^5}{15h^4\pi} \\ M_{0,1}^{E} &= M_{1,1}^{E} = 0 \\ M_{2,0}^{E} &= \tfrac{8d^5e - 14d^3eh^2 + 3deh^4 + 3h^6 \arccos(\frac{d}{h})}{18h^4\pi} \\ M_{0,2}^{E} &= \tfrac{-8d^5e + 26d^3eh^2 - 33deh^4 + 15h^6 \arccos(\frac{d}{h})}{90h^4\pi} \\ e &= \sqrt{h^2 - d^2} \end{aligned}$$

$$\begin{aligned} M_{0,0}^{H} &= \tfrac{2c^3b - 5cbh^2 + 3h^4 \frac{b}{|b|} \arccos(\frac{c}{h})}{6h^4\pi} \\ M_{1,0}^{H} &= \tfrac{4b^5}{15h^4\pi} \\ M_{0,1}^{H} &= \tfrac{(h-c)^3(3c^2 + 9ch + 8h^2)}{30h^4\pi} \\ M_{2,0}^{H} &= \tfrac{8c^5b - 14c^3bh^2 + 3cbh^4 + 3h^6 \frac{b}{|b|} \arccos(\frac{c}{h})}{36h^4\pi} \\ M_{1,1}^{H} &= \tfrac{b^6}{12h^4\pi} \\ M_{0,2}^{H} &= \tfrac{-8c^5b + 26c^3bh^2 - 33cbh^4 + 15h^6 \frac{b}{|b|} \arccos(\frac{c}{h})}{180h^4\pi} \end{aligned}$$

$$\begin{aligned} M_{0,0}^{T} &= \tfrac{b(c-a)(-a^2 - 2ac - 2c^2 + 5h^2)}{6h^4\pi} \\ M_{1,0}^{T} &= \tfrac{b(c-a)(-3a^3 - 6a^2c - 8ac - 8c^3 + 9ah^2 + 16ch^2)}{30h^4\pi} \\ M_{0,1}^{T} &= \tfrac{b^2(c-a)(-a^2 - 3ac - 3c^2 + 7h^2)}{30h^4\pi} \\ M_{2,0}^{T} &= \tfrac{b(c-a)(-6a^2 - 12a^3c - 17a^2c^2 - 20ac^3 - 20c^4 + 14a^2h^2 + 24ach^2 + 35c^2h^2)}{90h^4\pi} \\ M_{1,1}^{T} &= \tfrac{b^2(c-a)(-a^3 - 3a^2c - 5ac^2 - 5c^3 + 4ah^2 + 10ch^2)}{60h^4\pi} \\ M_{0,2}^{T} &= \tfrac{b^3(c-a)(-a^2 - 4ac - 4c^2 + 9h^2)}{90h^4\pi} \end{aligned}$$

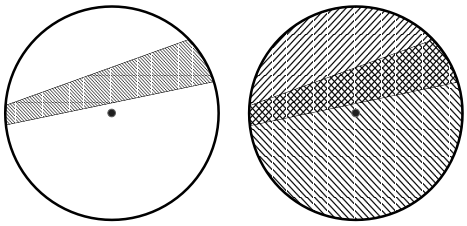


Figure 12: The shaded region on the left is a region that is outside the domain, or polygon, but inside the support of the kernel. When we naively include the effects of these edges we get the figure on the right, where the singly shaded regions have been subtracted once and the double shaded region have been subtracted twice. We can get back to the correct configuration on the left by simply adding back the entire area of support for the kernel once.

Assembling these pieces into complete kernel moments is an exacting task, and a careful case analysis is required to get the various signs correct. We used a simple Monte Carlo integrator to check against while debugging our implementation and strongly recommend this practice. Also in some cases it is necessary to add whole kernel moments in order to get the correct result. An example is shown in Figure 12. These cases are easily detected since when finished, $M_{0,0}$ should always be non-negative.

Implementing an analytic kernel moment calculator is quite complicated, but the runtime costs are minimal, because the the complex cases rarely occur in practice. In our implementation the moment calculations take a negligible fraction of the total run time.

Other local polynomial methods such as local constant and local quadratic can easily be derived in a similiar manner. Local quadratic and higher orders would improve convergence in relatively smooth regions, but the kernel moments become harder to compute and the artifacts at discontinuities would get worse. Local constant is simpler to derive and to compute, thus it might be preferable in some cases such as when exploring extensions to curved surfaces or when the local linear computations are considered too expensive.

C Mesh Decimation User Study

The decimation phase, described in Section 6, estimates the cost of removing a vertex from an illumination mesh in terms of a change in luminance. To help it determine the perceptual importance of a luminance change, the decimation phase uses a function, Δ, of luminance, L, with the property that the difference between luminances $L + \Delta(L)$ and L is just perceptible. We derived $\Delta(L)$ by conducting informal user studies. Users participating in the studies determined the just-perceptible change in luminance, $\Delta_i(L_i)$, for a baseline luminance, L_i, drawn from a set of sample luminances. Given the data points, $\Delta_i(L_i)$, we constructed the function $\Delta(L)$ by linear regression.

A session of the user study proceeds as follows. The user sits at a workstation whose monitor is black except for a window roughly four inches square. The window displays a gray-scale noise pattern whose lowest luminance is L_i. The pattern of the noise comes from Perlin and Hoffert's turbulence function [23], which contains a broad band of spatial frequency components. The height of the noise above its lowest luminance, L_i, is a parameter controlled by the user. When the user first sees the pattern, the height is zero, making the pattern a uniform gray of luminance L_i. The user can increment the height by one displayable gray-level by pressing a mouse button. We instruct the user to increment the height until the noise just reaches a *termination threshold* of no longer looking like "a wall with diffuse reflectance under uniform illumination;" at this point, the height is the experimental value of $\Delta_i(L_i)$. If the user inadvertently passes this threshold, he or she has no option of decrementing the height, but he or she may reset it zero and begin again; this approach ensures that the user's visual system is adapted to the baseline luminance, L_i, which should encourage conservative estimates of $\Delta_i(L_i)$. When the user is satisfied that he or she has reached the termination threshold, he or she presses a mouse button to record $\Delta_i(L_i)$. The study then progresses to the next L_i, chosen randomly. Our study involved a total of 20 values for L_i.

In this study, we intentionally describe a conservative termination threshold when instructing the user. This study therefore produces data points, $\Delta_i(L_i)$, that lead to a conservative version of the function $\Delta(L)$, and thus to a conservative simplification of the illumination mesh in the decimation phase. As Section 6 describes, it is also useful to have a more liberal version of $\Delta(L)$ which allows the decimation phase to further simplify the illumination mesh. To derive this liberal $\Delta(L)$, we repeat the user study with different instructions for the termination threshold. In the second study, we instruct the user to consider the noise to have reached the termination threshold when it looks like "an annoying deviation from a wall with diffuse reflectance under uniform illumination."

We conducted our user studies in a room with low ambient lighting, making the workstation monitor the main source of illumination. To calibrate the radiometric transfer function of the monitor, we used the gamma-correction model proposed by Motta [21]. This model involves two parameters whose values are chosen by users in two simple visual tests conducted before the user studies begin. The first test allows the user to choose the lowest monitor luminance that is distinguishable from black, and the second test asks the user to match gray levels to three dither patterns that approximate shades of gray.

The participating users in our studies were three of this paper's authors. For each version of the study, we compared the data from the three users and chose the most conservative $\Delta(L)$. Given this limited participation, and the informal nature of our experimental procedure in general, our studies should be interpreted as pilot studies that give useful information but not final conclusions. More rigorous versions of these studies are an important topic for future work.

Acknowledgements

This work was sponsored by the Cornell Program of Computer Graphics, the NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219), and NSF Division of Computer and Computation Research (CCR-9401961). The work was performed on workstations generously provided by the Hewlett-Packard Corporation. Special thanks to David Ruppert for providing his expertise on density estimation and pointing us to the locally-weighted polynomial regression literature. Thanks to Jonathan Shewchuk for making his *Triangle* meshing code publicly available at http://www.cs.cmu.edu/ quake/triangle.html. Also thanks the modelers, Ben Trumbore, Jim Ferwerda, Sebastian Fernandez, Ken Torrance, Bretton Wade, Dave Zareski, and our many anonymous reviewers.

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] Michael 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] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press, Boston, MA, 1993.

[6] Steven Collins. Adaptive splatting for specular to diffuse light transport. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 119–135, June 1994.

[7] D. A. Danielson. *Vectors and Tensors in Engineering and Physics.* Addison-Wesley, New York, 1991.

[8] E. A. Feibush, M. Levoy, and R. L. Cook. Synthetic texturing using digital filters. *Computer Graphics (SIGGRAPH '80 Proceedings)*, pages 294–301, July 1980.

[9] 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.

[10] Andrew S. Glassner. A model for fluorescence and phospherescence. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 57–68, June 1994.

[11] Andrew S. Glassner. *Principles of Digital Image Synthesis.* Morgan-Kaufman, San Francisco, 1995.

[12] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, July 1991. ACM Siggraph '91 Conference Proceedings.

[13] T. Hastie and C. Loader. Local regression: Automatic kernel carpentry. *Statist. Science*, 8:120–143, 1993.

[14] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics*, 24(3):145–154, August 1990. ACM Siggraph '90 Conference Proceedings.

[15] Hugues Hoppe. Progressive meshes. *Computer Graphics (ACM Siggraph '96 Conference Proceedings)*, pages 99–108, August 1996.

[16] Merlin Hughes, Anselmo A. Lastra, and Edward Saxe. Simplification of global-illumination meshes. In *Proceedings of Eurographics '96*, 1996.

[17] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques '96*, pages 21–30. Springer-Verlag/Wien, 1996.

[18] Deane B. Judd and Gunter Wyszecki. *Color in Business, Science, and Industry.* Wiley, New York, 1975.

[19] 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.

[20] C. Lo, B.J. Palmer, M.K. Drost, and J.R. Welty. Incorporation of polarization effects in Monte Carlo simulations of radiative heat transfer. *Numerical Heat Transfer, Part A*, 27:129–142, 1995.

[21] Ricardo J. Motta. Visual characterization of color crts. In *Device-Independent Color Imaging and Imaging Systems Integration*, pages 212–221. SPIE, February 1993.

[22] S. N. Pattanaik. *Computational Methods for Global Illumination and Visualisation of Complex 3D Environments.* PhD thesis, Birla Institute of Technology & Science, Computer Science Department, Pilani, India, February 1993.

[23] Ken Perlin and Eric M. Hoffert. Hypertexture. *Computer Graphics*, 23(3):253–262, July 1989. ACM Siggraph '89 Conference Proceedings.

[24] R.A. Redner, M.E. Lee, and S.P. Uselton. Smooth b-spline illumination maps for bidirectional ray tracing. *ACM Transactions on Graphics*, 14(4), October 1995.

[25] D. Ruppert. Personal communications. 1995.

[26] Holly Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. In *Proceedings of Graphics Interface '93*, pages 227–236, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

[27] 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.

[28] M. Ibrahim Sezan, Kwok-Leung Yip, and Scott J. Daly. Uniform perceptual quantization: Applications to digital radiography. *IEEE Transactions on Systems, Man and Cybernetics*, 17(4):622–634, August 1987.

[29] Peter Shirley, Bretton Wade, David Zareski, Philip Hubbard, Bruce Walter, and Donald P. Greenberg. Global illumination via density estimation. In *Proceedings of the Sixth Eurographics Workshop on Rendering*, pages 187–199, June 1995.

[30] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, London and New York, 1986.

[31] 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.

[32] 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.

[33] J. S. Toor and R. Viskanta. A numerical experiment of radiant heat interchange by the Monte Carlo method. *International Journal of Heat and Mass Transfer*, 11:883–897, 1968.

[34] Bretton Spencer Wade. Kernel based density estimation for global illumination. Master's thesis, Program of Computer Graphics, Cornell University, January 1996.

[35] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman and Hall, London and New York, 1995.

[36] Gregory J. Ward. The RADIANCE lighting simulation and rendering system. *Computer Graphics*, 28(2):459–472, July 1994. ACM Siggraph '94 Conference Proceedings.

[37] Steven Wolfram. *Mathematica: a system for doing mathematics by computer*. Addison-Wesley, New York, 1991.

[38] 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.

Plate A Hardware rendered solution for library model.

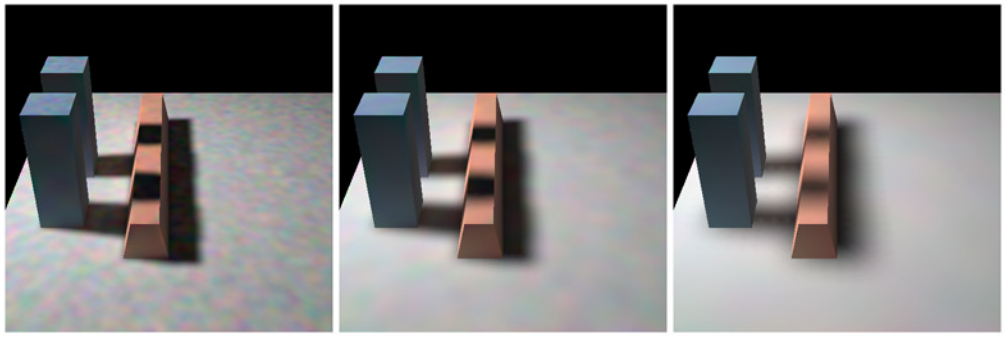


Plate B Bias vs. variance tradeoff as bandwidth goes from small (left) to large (right).

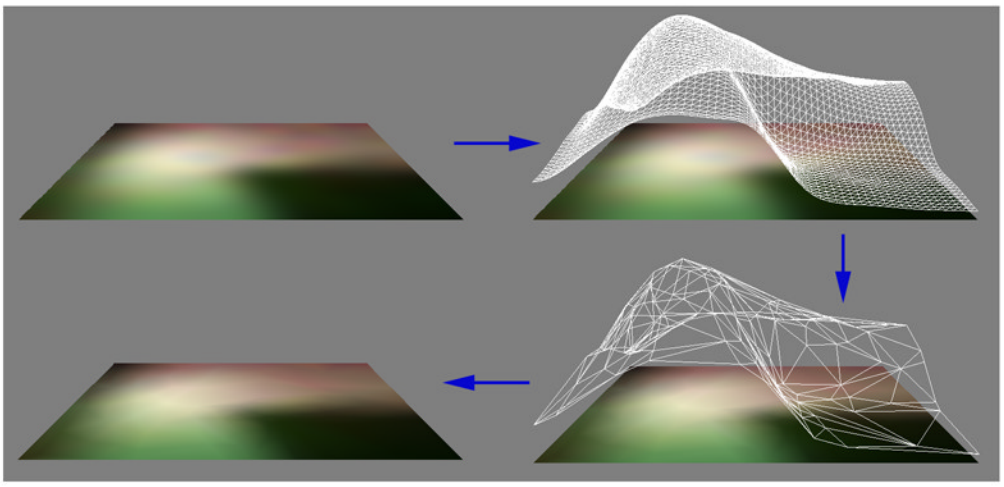


Plate C Transforming an illumination mesh into a geometric mesh for decimation.

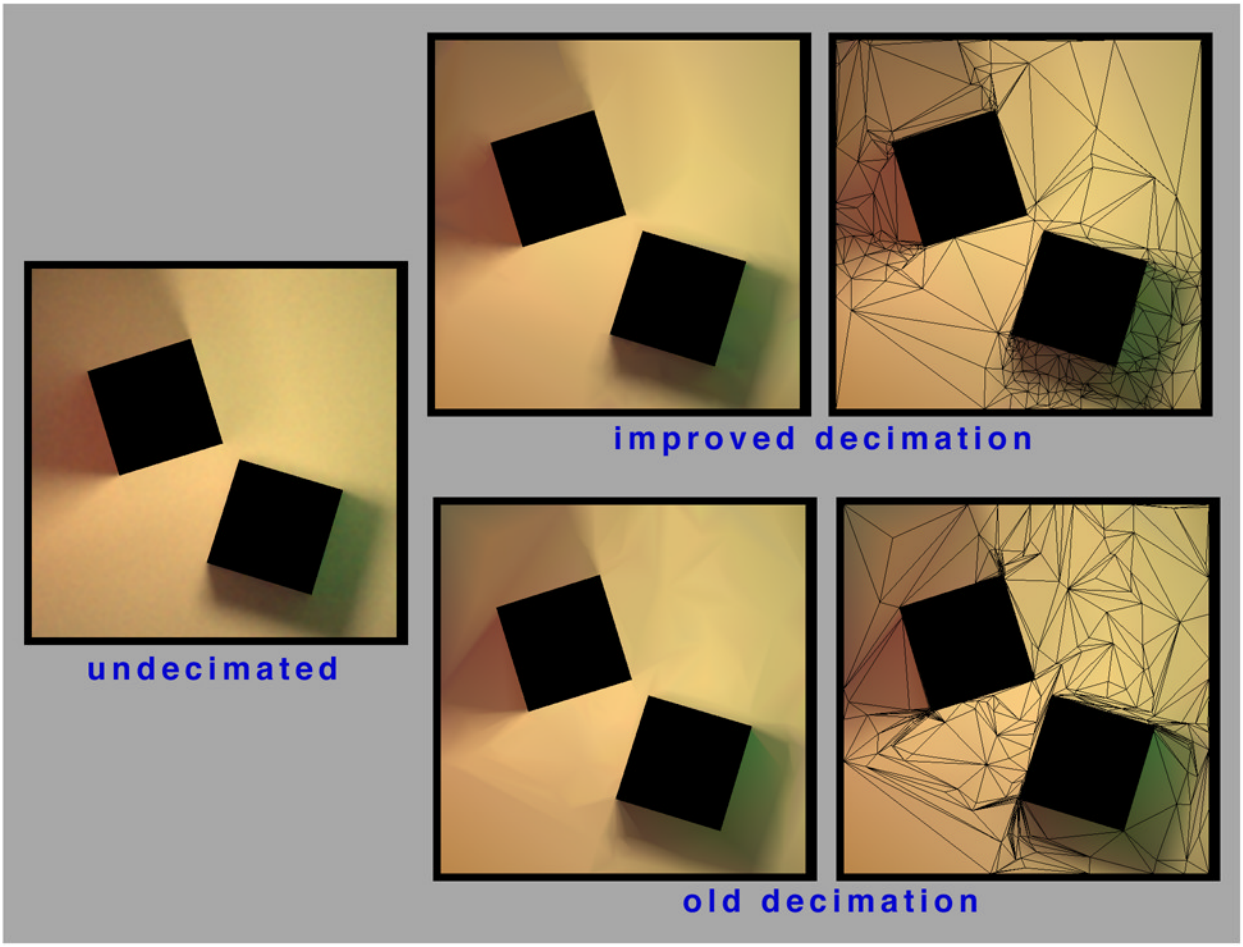
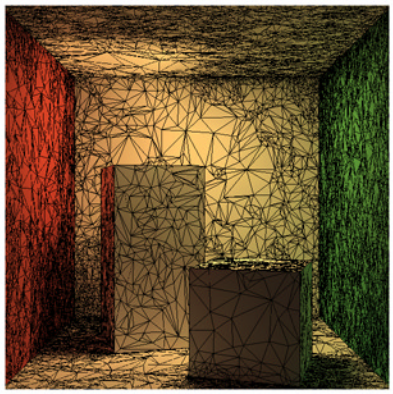


Plate D Comparsion of decimation techniques.

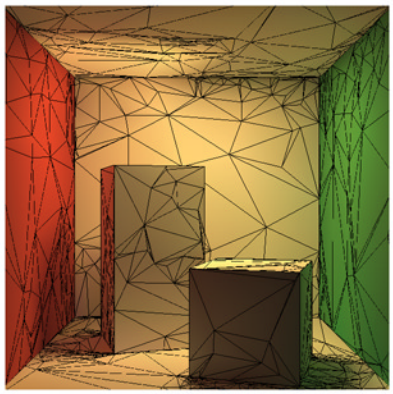
density estimation

path tracing

Radiance



conservative decimation



liberal decimation

Plate E Comparsion of rendering techniques.

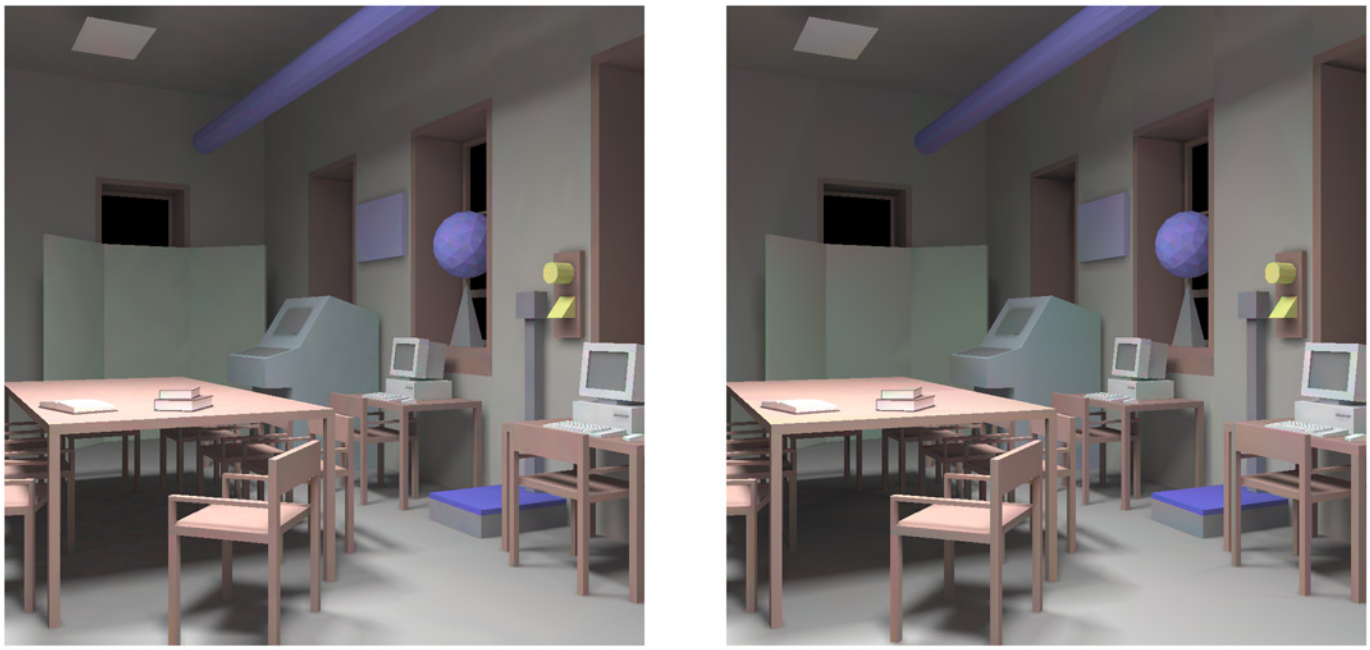


Plate F Hardware renderings of the Science Center model. Conservative decimation (left), liberal decimation (right).

Plate G Hardware rendering of the Falling Water model.

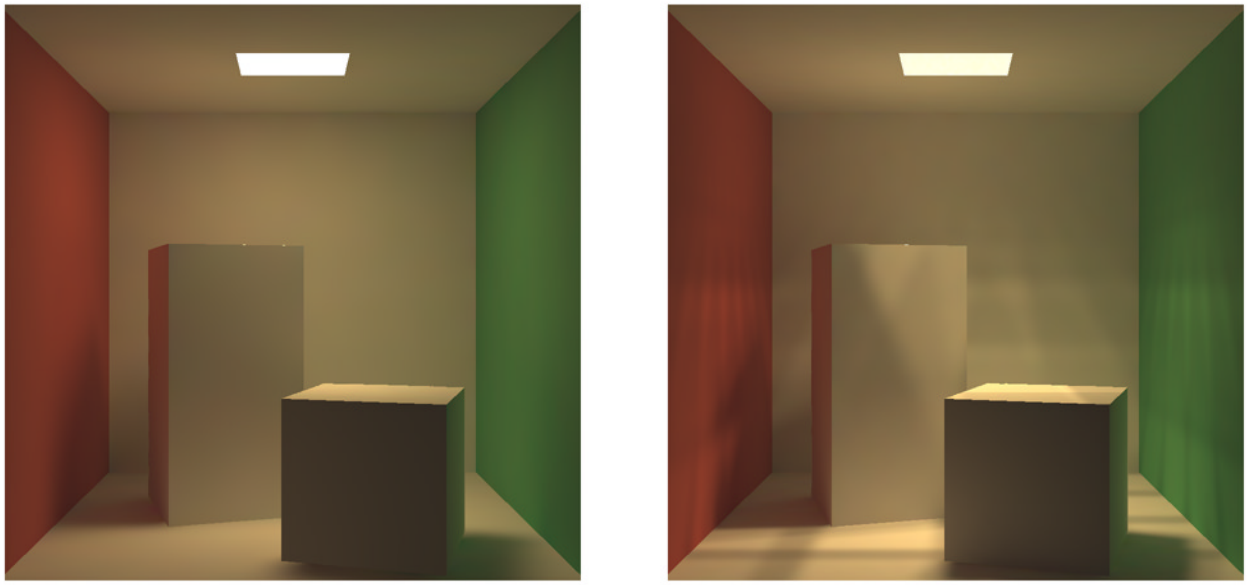


Plate H A recessed lighting fixture with diffuse (left) and specular (right) walls.

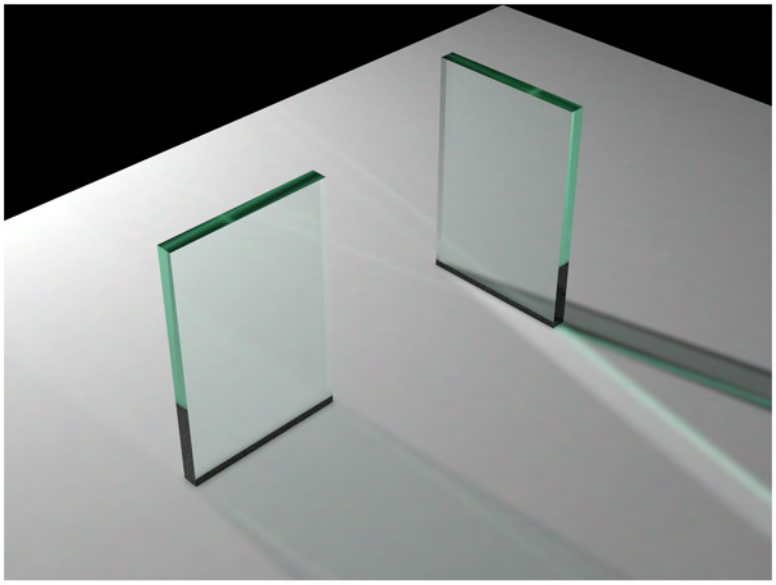


Plate I A solution with two panes of glass displayed using ray tracing.

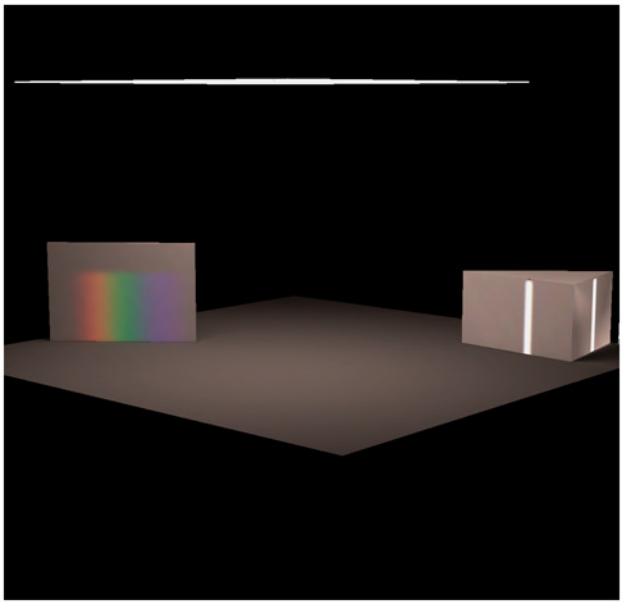


Plate J Rainbow from a prism.

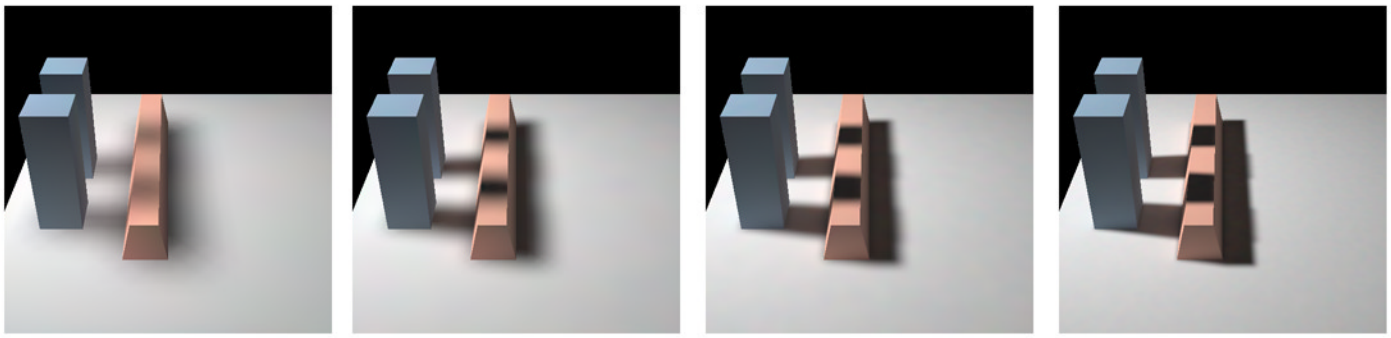


Plate K Solution convergence with increasing numbers of particles.