Chapter 2

Previous Work on the Radiosity Problem

2.1. The Physics of Global Illumination

We start by examining the physical basis for the simulation of global illumination in a computer graphics scene; first we introduce some important physical quantities, and then present the equations that govern light transport in such a scene.

2.1.1. Important Quantities

The key quantity in the physical simulation of light is *radiance*. This is defined as the amount of power radiated from a surface in a particular direction. It is measured in Watts radiated per unit area per unit solid angle, namely, $\mathrm{Watt}/\mathrm{m}^2\mathrm{sr}$.

We are primarily interested in the simulation of diffuse illumination; that component of global illumination that is view independent. For this, the physical quantity of *radiosity* is often more useful. It measures the amount of power radiated from a surface over all directions, and thus is measured in Watts per unit area, $\mathrm{Watt}/\mathrm{m}^2$. Radiosity is usually taken to refer to the light radiated by a surface. The light incident on a surface integrated over all directions is known as *irradiance*, and has the same units as radiosity.

Confusingly, the term "radiosity" is often used in computer graphics to refer explicitly to finite-element methods for solving for the transfer of radiosity in an environment. In fact, as we shall see, the various methods used to solve the

radiosity *problem* are independent of that problem, which is simply a more constrained version of the more general global illumination problem. A wide spectrum of approaches to solving this problem have been taken, from purely finite-element methods to purely stateless Monte Carlo methods.

Both radiance and radiosity are known as radiometric quantities; they are measured with respect to a specific wavelength, and are thus independent of the human visual system. All radiometric quantities have corresponding *photometric* ones, which are instead integrated over all possible wavelengths, weighted with the response of the human visual system. The photometric equivalent of radiance is luminance, and that of radiosity is illuminance. These quantities are importantly mostly when considering the transformation of radiometric quantities calculated for the scene into those suitable for use on a display device; this dissertation will largely ignore such issues. (However, see [Tumb99] for a full discussion of the issues involved.)

Appendix A has a full list of physical quantities and symbols used in this dissertation.

2.1.2. The Rendering Equation

If we ignore participating media, and concentrate solely on the interaction of light with scene surfaces, the global illumination problem can be summarized by the following *integral equation*:

$$L(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}) + \int_{\Omega} \rho(\mathbf{x}, \vec{\omega} \rightarrow \vec{\omega}_i) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_{\mathbf{x}} d\omega_i \qquad (1)$$

This is known as the *rendering equation* [Kaji86]. Basically, it states that the radiance emitted from some surface point $\mathbf{x}$ in the direction $\vec{\omega}$ is equal to the radiance the surface itself emits in that direction, L_e, plus the integral over the hemisphere of the incoming radiance that is reflected in that direction, ρL_i. To be more precise:

- $L(\mathbf{x}, \vec{\omega})$ is the total radiance leaving $\mathbf{x}$ in the direction $\vec{\omega}$;
- $L(\mathbf{x}, \vec{\omega})$ is the radiance directly emitted from $\mathbf{x}$ in the direction $\vec{\omega}$;
- $\rho(\mathbf{x}, \vec{\omega} \rightarrow \vec{\omega}_i)$ is the fraction of radiance incident from direction $\vec{\omega}_i$ that is reradiated in direction $\vec{\omega}$;
- $L_i(\mathbf{x}, \vec{\omega}_i)$ is the radiance incident on $\mathbf{x}$ from the direction $\vec{\omega}_i$;
- $\theta_{\mathbf{x}}$ is the angle between the surface normal at $\mathbf{x}$ and $\vec{\omega}_i$;

- Ω is the hemisphere lying above the tangent plane of the surface at $\mathbf{x}$.

For a summary of more general global illumination models, see [Glas94].

This integral equation is known as a Fredholm equation of the second kind, as it follows the form

$$b(s) = e(s)\int_{\alpha}^{\beta} \kappa(s, t)dt, \tag{2}$$

where α and β are fixed, and κ is known as the *kernel*. It cannot be solved analytically, except for the most trivial of cases.

If we assume that all surfaces reflect light diffusely, i.e., $L(\mathbf{x}, \vec{\omega}) \equiv L(\mathbf{x})$, the integral over the hemisphere collapses, and we have

$$L(\mathbf{x}) = L_e(\mathbf{x}) + \rho(\mathbf{x})\int_{\Omega} L_i(\mathbf{x}, \vec{\omega}_i)\cos\theta_{\mathbf{x}} d\omega_i \tag{3}$$

which, given $d\omega = ((\cos\theta)/r^2)dA$, and $B(\mathbf{x}) = \pi L(\mathbf{x})$, can be rewritten as the *radiosity equation*:

$$B(\mathbf{x}) = B_e(\mathbf{x}) + \rho(\mathbf{x})\int_{\mathbf{y} \in S} G(\mathbf{y} \rightarrow \mathbf{x})V(\mathbf{y} \rightarrow \mathbf{x})B(\mathbf{y})dA_{\mathbf{y}}, \tag{4}$$

where

$$G(\mathbf{y} \rightarrow \mathbf{x}) = \frac{\cos\theta_{\mathbf{x}}\cos\theta_{\mathbf{y}}}{\pi\|\mathbf{x} - \mathbf{y}\|^2} \tag{5}$$

and

- $B(\mathbf{x})$ is the total radiosity at point $\mathbf{x}$;
- $B_e(\mathbf{x})$ is the emittance at point $\mathbf{x}$;
- $\rho(\mathbf{x})$ is the reflectance at point $\mathbf{x}$;
- $V(\mathbf{y} \rightarrow \mathbf{x})$ is the visibility between points $\mathbf{x}$ and $\mathbf{y}$;
- $G(\mathbf{y} \rightarrow \mathbf{x})$ is the geometry kernel between points $\mathbf{x}$ and $\mathbf{y}$.

This is the equation we must solve to find the global illumination of a purely diffuse scene.

2.2. Solving Integral Equations

Given that global illumination is governed by an integral equation, it makes sense to look at the various methods for solving such equations. Unfortunately, even the simplest global illumination scene is unsolvable analytically, and we must look to numerical methods for solving integral equations. We start by looking at numerical techniques for solving integrals, known as quadrature methods, and then explain how they can be applied to solve an integral equation.

2.2.1. Monte-Carlo Quadrature

Commonly, quadrature methods estimate an integral of some function $F(x)$ by forming a weighted sum of samples of that function:

$$\int F(x)dx \approx \frac{\sum w_i F(x_i)}{\sum w_i} \tag{6}$$

The sample points, x_i, are known as abscissas. In some situations they are fixed beforehand, but in most situations, including the ones we face in computer graphics, we get to pick the abscissas.

In pure Monte-Carlo integration, we assume nothing about the integrand: the abscissas are picked randomly, and the weights w_i are set to one. It is also possible to take advantage of some knowledge of the integrand via *importance sampling*, where the sample points are drawn not from a uniformly distributed random variable, but from a probability distribution $p(x)$ that matches some known factor of the function being integrated. For instance, if we are evaluating the integral in **Equation 1**, we can choose our probability distribution according to the $\rho(\ )\cos\theta$ section of the integral, as this part of the function is known a priori. In this approach the weights are set to $w_i = 1/p(x_i)$.

The Monte Carlo approach is particularly effective in dealing with high dimensional integrals, and integrals with discontinuities in the integrand, and has the advantages of being simple to implement, and almost no memory cost. However, the convergence rate of Monte Carlo, $O(\sqrt{n})$, is poor, and is the major reason Monte Carlo methods are associated with computationally costly calculations. It is possible to do better convergence-wise, especially with smooth integrands. The other chief drawback of Monte Carlo methods is that the error in their approximation presents itself as noise; in places where we expect the results of an integral to be smooth over some domain, this can be highly distracting.

2.2.2. Deterministic Quadrature Methods

Deterministic quadrature methods work by picking some set of x_i and w_i so that, if some assumption about the integrand is true, the estimate of the integral is exact. The simplest possible such method is to assume that the integrand is constant, and estimate the integral with a single sample of $F(x)$. More general rules are possible, such as those given by Gaussian quadrature formulas. For m samples, these rules are guaranteed to be exact for polynomials up to degree $2m$.

The disadvantage of such methods is that they do not handle discontinuities well, and they suffer from dimensional explosion. If a particular rule requires m samples in one dimension, the corresponding n-dimensional rule will require m^n samples; this soon becomes unwieldy above three or four dimensions.

The kernel of the radiosity equation consists of one largely smooth factor, the geometry term G, and one factor that can be highly discontinuous, the visibility term V. **Figure 6** shows the radiosity function for a trivial scene configuration that illustrates the effect of both terms. The tension between these two factors is one of the reasons there are so many competing methods for simulating diffuse illumination; Monte Carlo methods do much better with the visibility part of the kernel, and deterministic methods with the geometrical part of the kernel. Used inappropriately, Monte Carlo methods lead to objectionable noise in the smooth parts of the function shown, and deterministic methods can result in the illumination discontinuity in the middle of the patch being unnaturally smeared.

2.2.3. Solving The Rendering Equation

Of course, it does not suffice to have techniques for solving the integral in **Equation 1**, because that integral is cast in terms of an unknown, namely the quantity we're trying to find in the first place, the scene radiance function. However it is possible to restate the equation recursively, in terms of an integral operator:

$$L = L_e + \Re L \tag{7}$$

This allows us to solve for L by using a generalisation of the geometric series,

$$\sum_{r=0}^{\infty} a^r = \frac{1}{1-a}, \tag{8}$$

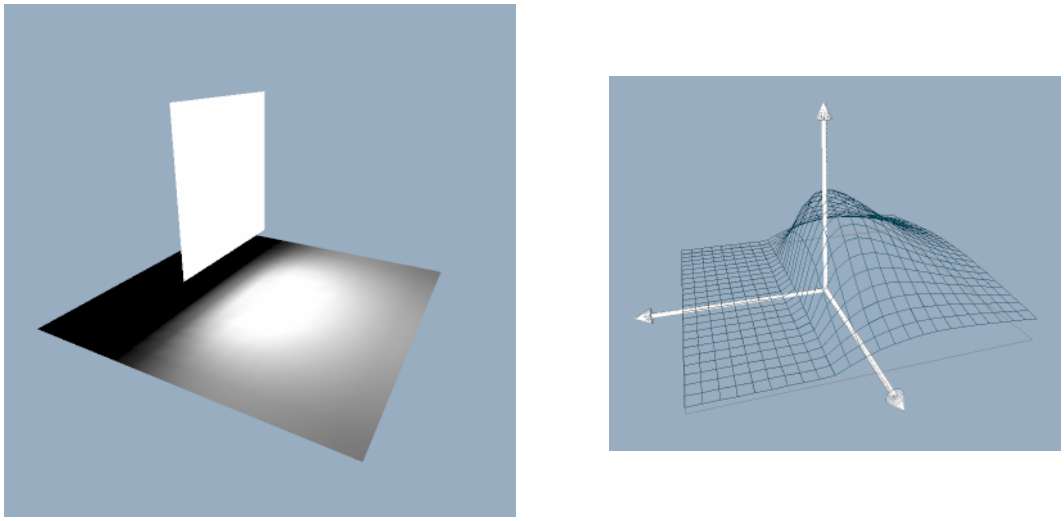


Figure 6: The radiosity kernel. Shown on the left is a simple configuration of a diffuse emitter and a diffuse reflector. On the right is a plot of the irradiance across the receiving patch; it is largely smooth, even with the two polygons in close proximity, except where we cross behind the light source, and it no longer becomes visible. In practice the distance between light source and receiver is greater, and the geometric part of the radiosity kernel is even smoother than this.

called the Neumann series:

$$L = L_e + \Re L_e + \Re^2 L_e + \ldots . \tag{9}$$

This series is guaranteed to converge only if the spectral radius of $\Re$ is less than one, but this is guaranteed for any reflection operator that conserves energy.

In computer graphics, this infinite sum has an obvious analog; each application of the $\Re$ operator represents a single bounce of light from every emitting surface in the scene. The sum represents successive "bounces" of illumination from the original light sources (L_e) throughout the scene, until a steady state is reached.

Path Tracing

We can extend the use of Monte Carlo integration techniques to this iterative solution, via the mechanism of random walks, in an approach known as path tracing. Rather than using Monte-Carlo sampling to apply the reflection operator to the entire scene, and then repeating that operation, a random walk is used to simulate repeated applications of the reflection operator to successive samples of scene illumination. A particle is probabilistically emitted from a light source in the scene, and then "traced" through the scene, being reflected at appropriate points by scene surfaces. At each successive bounce, either the new direction of the particle is chosen according to a probability distribution that matches the shape of the reflectance function ρ for that surface, or the path is terminated and the particle is "absorbed" by the surface. This process can be thought of as simulating the paths of large numbers of photons through the scene.

Finite Element Methods

Another approach to the problem is the use of the finite element method. Rather than concentrating on the propagation of light particles through the scene, this method tries to solve for the light emitted by the scene's surfaces. It works by breaking the domain up into a finite number of zones called elements, each having its own, local set of basis functions. The illumination problem then becomes a problem of stating the relationship between each individual element and the other elements in the scene, and deriving from that the light emitted over each element.

While the finite element method can be applied to the full rendering equation (**Equation 1**), it is more natural to apply it to the radiosity equation (**Equation 2**). Once the scene has been discretized, the radiosity equation becomes a simple linear system,

$$\mathbf{b} = \mathbf{e} + R\mathbf{b} \tag{10}$$

where $\mathbf{b}$ is a vector of radiosities per element, $\mathbf{e}$ is a vector of emittances, and R is a matrix, whose effect is analogous to that of the reflection operator $\mathfrak{R}$. (If there is more than one basis function per element, these will be tensors.) This system of equations is usually solved in a similar way to **Figure 9**; by repeated multiplications of the initial radiosity estimate $\mathbf{e}$ by $(I + R)$.

2.2.4. Global Illumination Methods

There are a wide variety of global illumination methods that can be applied to solving the radiosity problem. They vary in what quadrature methods they use, how they store any intermediate results, and what they output; either a view-dependent image, or a (usually) view-independent mesh. I will try to span the possible approaches with a review of the most salient papers.

Stateless Ray-Tracing

A purely stateless path-tracing method, such as stochastic ray-tracing [Kaji86, Lang94] proceeds as described above; photons are propagated through the environment, eventually reaching the view point. (For efficiency reasons, direct lighting is often treated as a special case, and path tracing used to estimate only the indirect illumination.) Where the reflection function is highly directional, i.e., specular, good results can be achieved with a moderate number of samples. When the reflection function is such that the entire hemisphere above any surface "hit-point" must be sampled evenly, a much larger number of samples is needed to achieve the same level of accuracy.

This approach produces very noisy results on diffuse scenes (see **Figure 7**). (Of course, as mentioned, it performs very well on specular scenes, for which finite-element based radiosity techniques do just as poorly.) The noise can be decreased by increasing the number of photons emitted, but only slowly; this is known as the law of diminishing returns of Monte-Carlo sampling.

State of the art path-tracing methods use several different sampling strategies, each targeted towards a particular aspect of the global illumination problem, and combine them in ways that are provably good [Veac94, Veac95]. These methods do improve results for the middle ground of semi-diffuse scenes measurably, but pure diffuse scenes still suffer from noise. (Again, see **Figure 7**.)

Finite-Element Methods

As discussed previously, finite element methods work directly on the mesh of the input scene. The input scene is polygonized if it is not already, and then subdivided into a number of discrete surface elements. The problem of radiosity transfer between these elements is then treated similarly to other radiative transfer problems; the interactions between elements, known as *form factors*, are found, and, given an initial set of emittances and reflectivities for each element, used to iteratively propagate energy though the environment until the solution reaches a steady state. These methods are particularly effective when the radiosity function

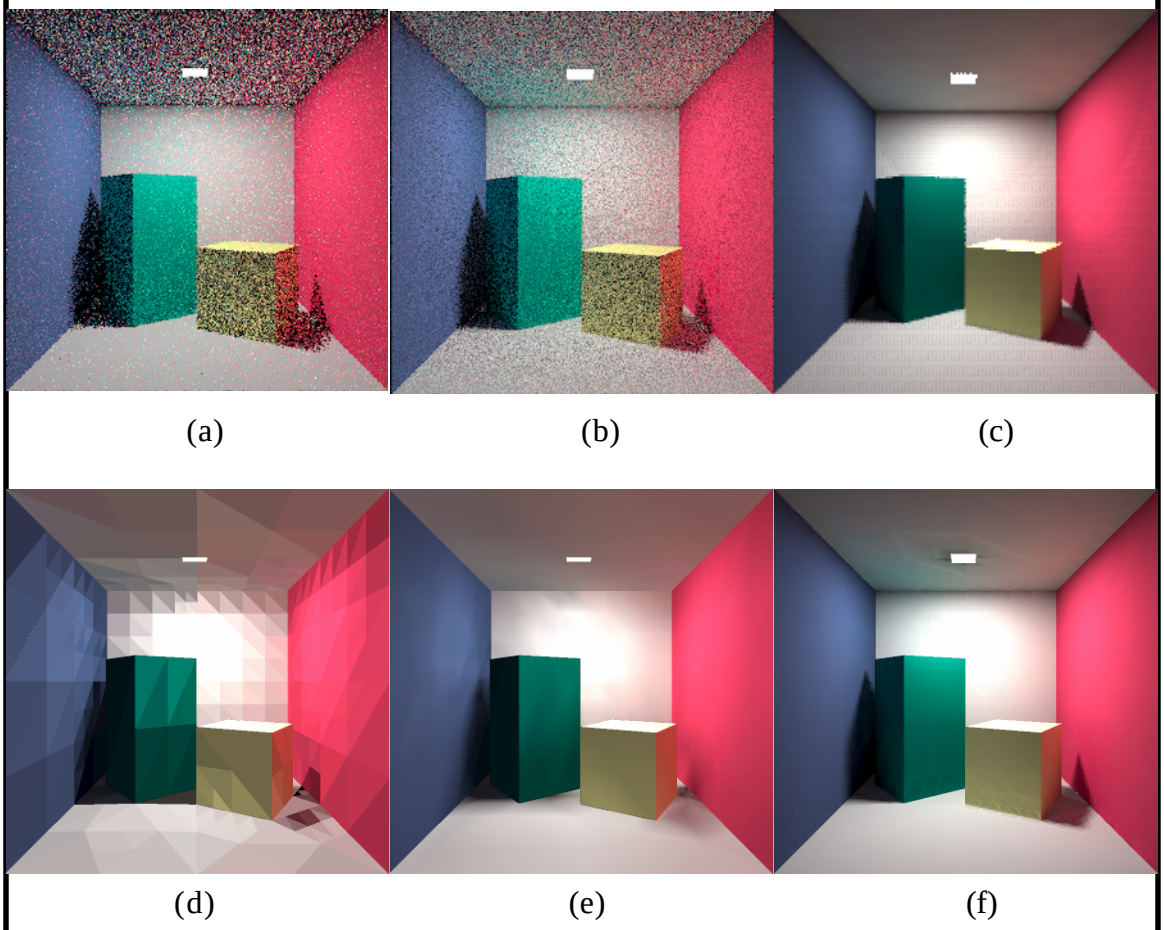


Figure 7: Global Illumination methods for the Cornell box scene. Top, three ray-tracing methods; (a) stochastic ray-tracing, (b) bi-directional ray-tracing, and (c) diffuse sample caching (Radiance). Bottom, hierarchical radiosity with (d) the Haar basis, (e) the Haar basis and a smoothing pass, and (f) a linear basis.

The radiosity methods each took several seconds to render the scene, sample caching fifteen seconds, stochastic ray-tracing twenty seconds, and bidirectional ray-tracing a minute.

varies only slowly across each element, as is often the case in diffuse environments.

Sample Caching Methods

Pure Monte Carlo methods suffer because they do not take advantage of the coherence in the radiosity function; illumination in a purely diffuse scene tends to vary only slowly across surfaces, as a direct consequence of the incoming radiance integral covering the entire visible hemisphere for any point on the surface. Ward takes advantage of this by using diffuse sample caching in his Radiance program. Whenever an expensive diffuse illumination sample is calculated, it is stored in a volumetric spatial data structure (an octree). If another sample is needed from a point nearby on the same surface, the sample is simply reused, with appropriate filtering.

Another technique that uses sample caching is the "photon map" [Jens96]. Whereas Radiance calculates and reuses diffuse samples on demand, as rays are traced out from the view-point, the photon map works in the other direction. It uses a path-tracing approach in which as usual power-carrying particles are emitted from each light source, and tracked through the scene until they are absorbed, but then stores their position and incident direction in a spatial data structure. (The genesis of such photon-tracing methods is a paper on simulating the particle model of light by Pattanaik and Mudur [Patt92].) In a second, image producing pass, when a ray traced from the eye hits a diffuse surface, nearby "photons" are filtered to estimate the local irradiance of the surface. The technique has the advantage that it is extremely flexible, and good at producing caustic effects. Its major drawback is that a large number of photons must be emitted to build up the map, and there is little way of telling a priori exactly how many will be sufficient to produce a desired image.

Mesh-Based Monte Carlo Methods

Rather than caching particles in a volumetric data structure, it is possible to store the results of random walks in a surface-based data structure. Heckbert takes just this approach with his "adaptive radiosity textures" [Heck90]. Photons are traced from light sources, and, when they hit a diffuse surface, accumulated in texture maps. These maps are adaptively subdivided when their resolution is insufficient.

Another such method is density estimation [Shir95, Walt97b]. This technique extends the sample caching methods in order to generate view-independent global illumination solutions; the output is a traditional surface mesh with vertex radiosities. It proceeds much like the photon map, but adds a phase, called the "density-estimation" phase, where the stored photons are used to construct an approximate irradiance function for each surface. This irradiance function is then used to construct the output mesh via Delauney triangulation.

Finally, finite-element methods have been adapted to use Monte-Carlo path tracing to calculate diffuse light transport [Shir91, Neum95]. This is similar to the photon map and radiosity texture approach, but, instead of the samples being stored in a spatial data structure or texture map, they are used to estimate the diffuse illumination of elements in the input mesh. One of the great advantages of this approach is the robustness of path-tracing in the face of large, complex scenes. The chief drawback of such Monte Carlo radiosity methods is the introduction of noise; it tends to produce a dappled effect on what would otherwise be smooth surfaces. Also, the method requires that scenes be pre-meshed a-priori to a reasonably fine resolution, often incurring more geometry storage than strictly necessary, and posing the difficult question of what resolution to use for the mesh. Some work has been done on driving mesh adaption by maintaining a "preview" mesh at finer resolution than the result mesh, and using the preview to decide where further subdivision is necessary [Tobl97].

Shirley has shown that, under certain assumptions, if a given scene is divided up into n elements, evaluating their diffuse illumination via Monte-Carlo path tracing requires $O(n)$ rays to be cast [Shir92]. It is generally accepted that the cost of a ray-tracing query is $O(\log n)$, leading to a total time complexity of $O(n \log n)$ for naive ray-tracing based methods.

Summary

A summary of the various methods is shown in **Table 1**.

Method	**Quadrature.**	**State**	**Fixed View**
Stochastic Ray-Tracing	Monte Carlo	None	Yes
Bi-Directional Path Tracing	Monte Carlo	None	Yes
Diffuse Sample Caching	Monte Carlo	Sample Cache	Yes
Photon Maps	Monte Carlo	Sample Cache	Yes
Adaptive Radiosity Textures	Monte Carlo	Adap. Texture	No
Density Estimation	Monte Carlo	Mesh	No
Random Walk Radiosity	QMC	Finite Element	No
Galerkin Radiosity	Galerkin	Finite Element	No
Importance-based Radiosity	Galerkin	Finite Element	Yes

Table 1: A Global Illumination Taxonomy.

2.2.5. Finite-Element vs. Monte Carlo Methods

For low to medium complexity scenes, the various finite element methods produce the best results for diffuse scenes. For more complex scenes, Monte Carlo methods have proven faster: Ward's sample caching method is currently the most stable, fast and robust method for such scenes.

We might reasonably ask, what is the point in further research into finite-element methods, given that they only solve for diffuse light transport? For a start, Monte Carlo-based techniques have their drawbacks as well:

- They are noisy, especially when the scene is animated
- They are not as adaptive as Finite Element Methods.
- Of the various methods, only density estimation produces illuminated scene geometry, as opposed to a single image. (Although Ward's method does cache diffuse illumination samples between runs, so that calculations are reused between viewpoints.)

The Achilles heel of radiosity methods is their time complexity, and their reliance on the input scene to define the element mesh; arguably this overconstrains the solution. There are a number of reasons to continue investigating finite element methods, however:

- The hierarchical nature of state-of-the-art radiosity algorithms is appealing.
- The algorithm can be used as a pre-processing stage for a full global illumination solution. Hierarchical radiosity methods produce a set of links which summarize the transport of light in the given environment. These links can be used to guide sampling in a more general Monte Carlo-based renderer.
- The ability to amortize the cost of calculating diffuse illumination over the course of animation or walkthrough is valuable.
- While not physically correct, calculating static diffuse illumination and then adding specular stuff reflections during a post-pass can produce highly realistic images, as evidenced by the output of the commercial Lightscape renderer [Auto].

Currently, radiosity methods have a strong niche in medium-complexity virtual worlds, for games and architectural walkthroughs. The complexity of these environments is limited by the need for interactive rendering rates, so the poor performance of radiosity methods on more complex scenes is irrelevant. However, this will not always be the case. The work contained in this dissertation was pri-

marily motivated by the observation that, unless the dependence on input geometry of finite element methods could be overcome, the method would most likely be written off as a historical curiosity as scene complexities continued to rise. At the very least, radiosity must be able to compete with caching ray-tracing's complexity and robustness.

2.3. An Introduction to Finite-Element Radiosity

The finite-element radiosity algorithm is the most commonly used algorithm for simulating diffuse interreflection [Cohe93]. It subdivides surfaces in the environment into a mesh of *elements*, and then sets up and solves a large system of linear equations in order to compute the *radiosity* (the amount of emitted or reflected light) at each surface point. Early radiosity algorithms used a fixed mesh of elements, and, as each element can potentially illuminate every other element, had costs that were quadratic in the number of elements in the scene. These algorithms did a poor job of exploiting the sparseness of the element-to-element interaction kernel; most objects are visible to only a small subset of the other objects in the scene, and the transfer of radiosity also obeys an inverse square law, so much of this kernel is either zero, or of small magnitude.

In this section we shall introduce the most important finite-element algorithms for the radiosity problem. We start with the most commonly used radiosity algorithm in practice, progressive radiosity.

2.3.1. Progressive Radiosity

All radiosity methods trace their roots from matrix radiosity, which explicitly computes a large *form factor* matrix and solves several large systems of equations [Gora84, Nish85, Cohe85]. First, the scene is discretized into patches, then form factors between every possible pair of patches are computed. The resulting $n \times n$ linear system of equations is typically solved using an iterative method such as Gauss-Seidel iteration [Stra86]. Computing the n^2 form factors is the most costly step, as each one requires one or more visibility tests. The amount of storage required is $O(n^2)$, which makes the method infeasible for anything other than trivial scenes.

Because it is so expensive in time and space, matrix radiosity is mainly a historical curiosity, though it is occasionally useful for theoretical investigations. Its immediate successor, however, was progressive radiosity, which is probably the most widely used, and certainly the most robust, finite-element radiosity method. This technique progressively refines an image by computing the matrix

and the solution incrementally [Cohe88]. It iteratively "shoots" light from the brightest light sources and reflective surfaces, computing just one column of the form-factor matrix at a time. This is a variant of Southwell's relaxation technique for solving linear systems [Gort93a, Shaw53]. Although this relaxation technique has been largely superseded by other iterative techniques in the numerical methods literature [Barr94], it has proven useful for radiosity simulations, because it reduces the amount of form-factor storage needed to $O(n)$. The technique also has the great advantage that the illumination that will make the greatest difference the final image is calculated first, starting with direct lighting—see **Figure 8** for an example of the results. To make this possible, a per-patch record of the amount of unshot radiosity left in the scene must be kept, but this results in only an addition $O(n)$ storage.

Progressive radiosity is usually used in conjunction with substructuring, which introduces a two-level hierarchy to the mesh; the coarser patches shoot light to a finer set of elements [Cohe86]. The elements are subdivided adaptively in regions of high radiosity gradient, such as shadow boundaries. Substructuring is generally regarded as being very desirable since it helps capture fine detail illumination, such as sharp shadow boundaries, without any consequent penalty in terms of the number of shooting patches to be dealt with.

If s shooting steps are used, the time cost of progressive radiosity is $O(snv)$. In practice the number of shooting steps is smaller than the total number of patches, so $s < n$. Still, the method is usually significantly superlinear in the number of elements n.

2.3.2. Hierarchical Radiosity

The most significant theoretical modification of the original matrix radiosity algorithm is the hierarchical radiosity algorithm [Hanr91, Cohe93, Sill94b]. This takes advantage of the sparseness of the interaction kernel by employing techniques similar to the *n-body algorithm* used in gravitational simulations [Appe85, Barn86, Gree87]. By treating interactions between distant objects at a coarser level than those between nearby objects, the hierarchical radiosity algorithm reduces the cost from quadratic to linear in the number of elements used.

These methods employ multilevel meshes to represent the radiosity function, and allow inter-patch interactions to take place between arbitrary levels of the mesh hierarchy [Gort93b, Stol96, Schr96]. Thus, unlike previous methods, hierarchical radiosity does not use a fixed mesh, but a mesh that is adaptive in resolution to the other surfaces "viewing" it. When reflecting light to a distant sur-

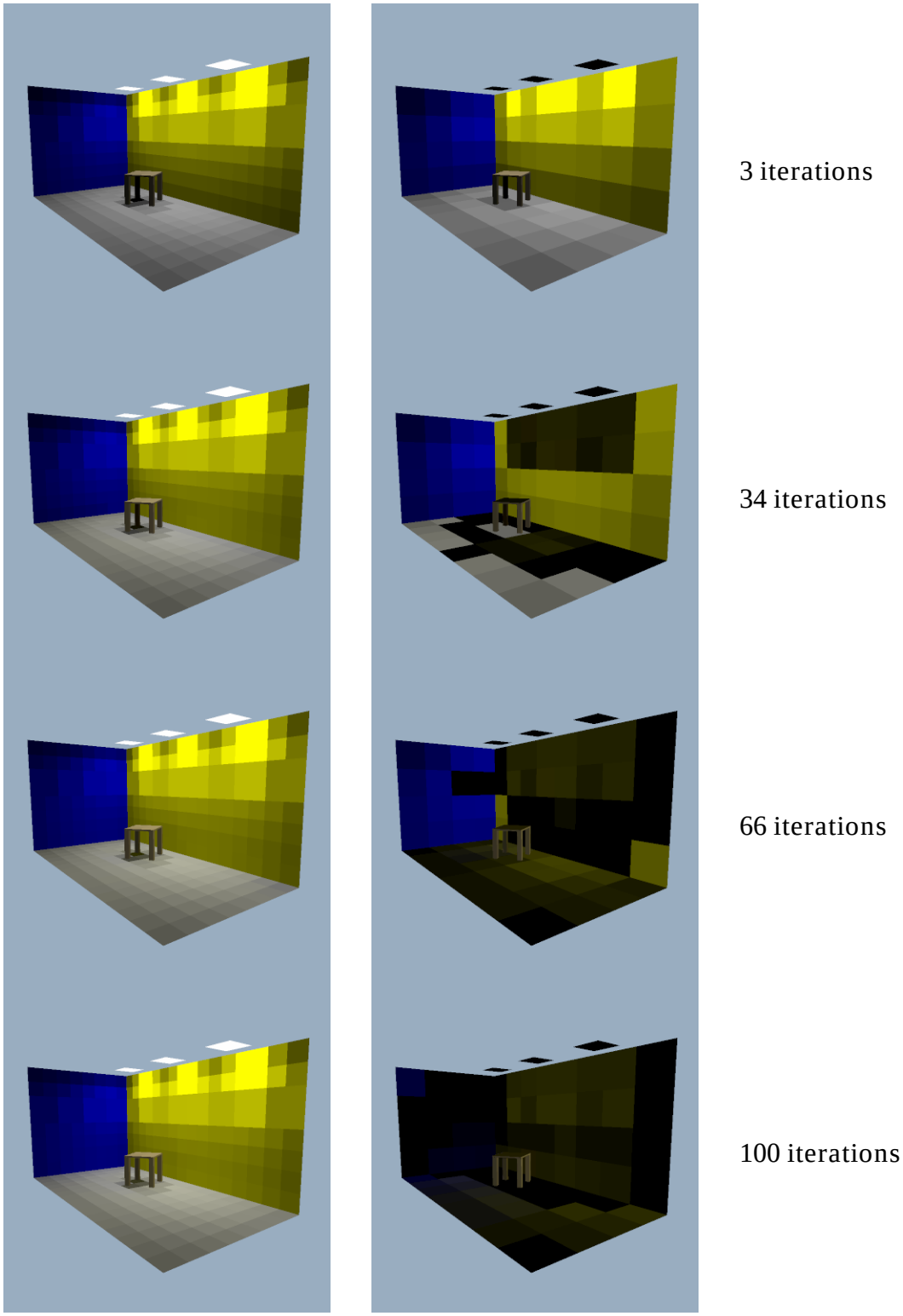


Figure 8: Shooting steps. Left: radiosity, right: unshot radiosity.

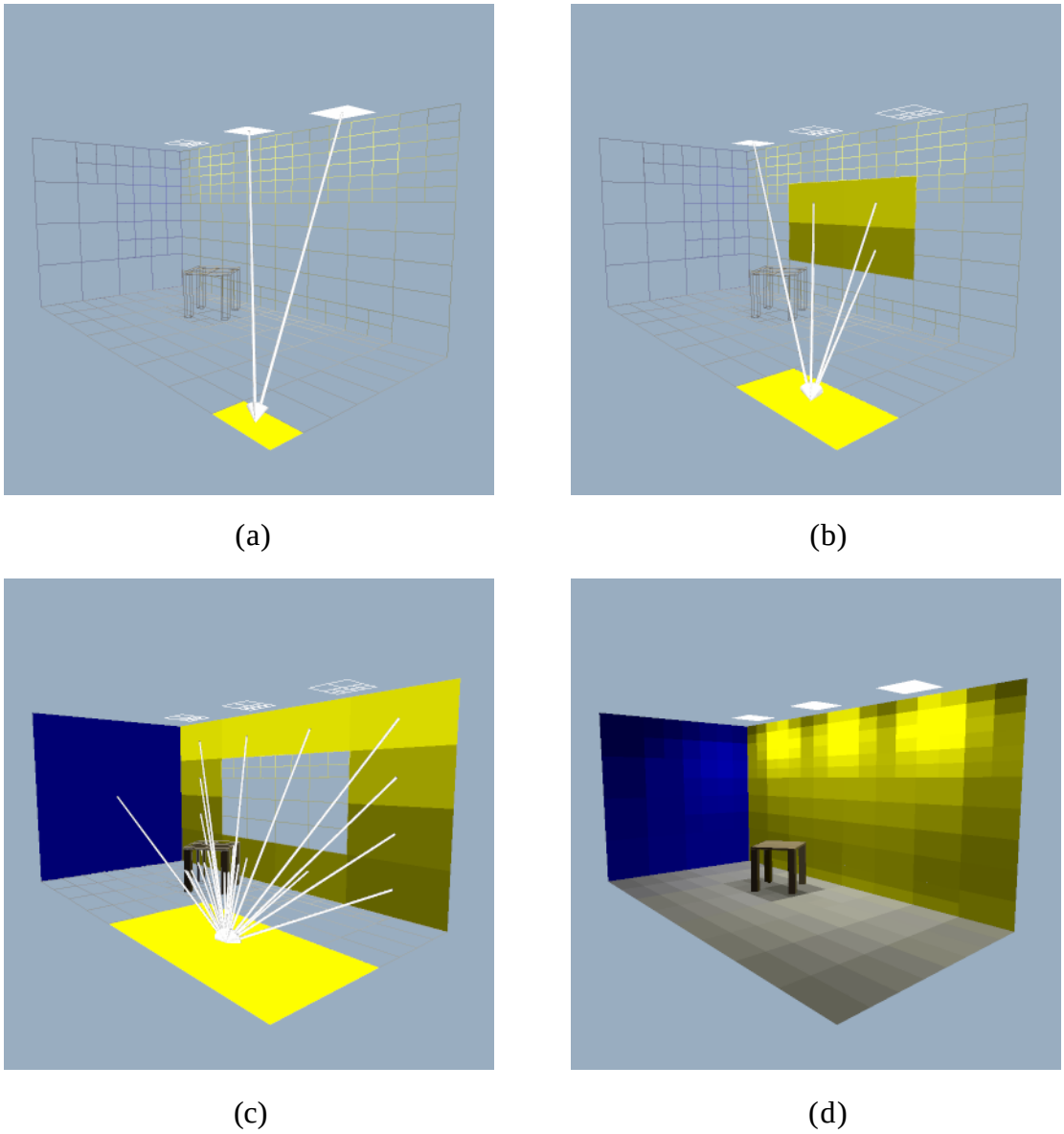
face, a given surface is meshed coarsely, but when reflecting to a nearby surface, it is meshed more finely.

The commonly used adaptive mesh representation is the *quad-tree*; this is used to represent each input polygon in the scene. **Figure 9** shows how nodes in various levels of a quad-tree mesh interact with the rest of a simple scene. Each transfer of energy between two different nodes in the quad-tree is represented by a data structure called a *transport link*, which contains some representation of the fraction of radiosity transferred, and some estimate of the error in that representation.

Hierarchical radiosity has a cost linear in the number of final solution elements, n. Unfortunately, because an initial light transport link from each polygon to every other polygon must be computed, the cost is also quadratic in the number of input polygons, k. The cost is thus $O(k^2 + n)$.

Hierarchical radiosity can be regarded as a specific instance of a more general class of *wavelet radiosity* algorithms, although it was originally developed without reference to wavelet techniques. Whereas the original HR algorithm assume the radiosity was constant across each mesh element, Various basis functions can be used to represent the radiosity across an element.

While hierarchical radiosity was inspired by the original n-body algorithm of Appel, and is similar in spirit to its widely-used successor in the astrophysics community, the Barnes-Hut algorithm, the difference in application area has lead it in different directions. (The parallel class of algorithms in the physical simulation community are referred to as “treecodes”.) Chiefly, the difference is due to the presence of occlusion in the illumination problem. In the radiosity problem, we must deal with any occluding objects between two interacting mesh elements, and also with tangential occlusion; a surface, by its nature, emits radiosity above the tangent plane only. Not only does this introduce discontinuities into the interaction kernel, but evaluating possible occlusion is an expensive operation; much more so than evaluating either the forces of gravitational attraction or indeed the rest of the rendering equation. The upside of occlusion is that it promotes sparseness in the kernel. The number of radiosity interactions in, say, an office building, will be fewer than the equivalent gravitational configuration, due to the various offices being largely compartmentalised.



(a) (b)

(c) (d)

Figure 9: Hierarchical Radiosity. The transfer of radiosity takes place between several different levels of the mesh hierarchy. Strong light sources and immediate neighbours contribute to an element at the finest level of the mesh (a), medium-strength contributors to its parent element (b), and low-strength, distant sources to its parent (c). These contributions are summed to form the final result (d). The white arrows shown represent transport links, each of which contains an estimate of the form-factor and occlusion between the two elements it connects.

2.3.3. Hierarchical Radiosity with Clustering

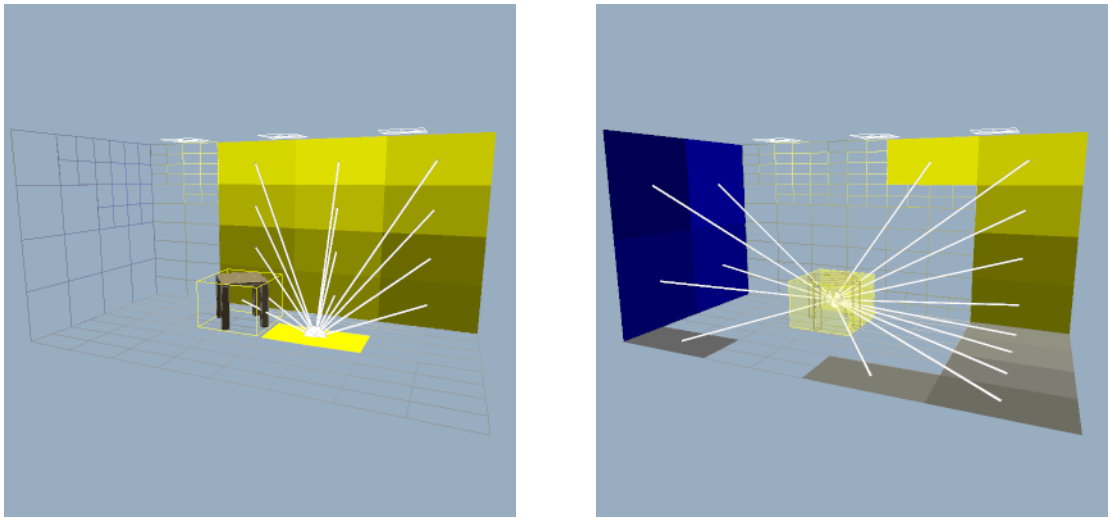
Classical hierarchical radiosity algorithms work well on scenes with a small number of large polygons, but the k^2 term means they become impractical in time and memory consumption for scenes of several hundred polygons.

To combat this problem, clustering methods for hierarchical radiosity were developed [Chri97, Gibs96, Sill95b, Sill95a, Smit94]. These methods group the input polygons into *volume clusters*, building a hierarchy above the input polygons that culminates in a root cluster for the entire scene. The lower nodes in this hierarchy are elements in quadtrees, as before, but the upper nodes are *volume clusters*. These are octree or k-d tree boxes containing a set of disconnected polygons with potentially varying normal vectors and reflectances. With the use of volume clusters, we now have a complete, singly-rooted simulation hierarchy, which requires only a single initial illumination link; thus the k^2 term in the complexity vanishes. (See **Figure 10**.)

Several methods for handling the light incident on a cluster have been explored. The simplest is to assume the clusters are isotropic, and thus sum the incoming light from all directions. This approach, called beta links by Smits, turns out to be fast, $O(k+n)$, but inaccurate. A more successful alternative is to push the light down to the leaves of the tree whenever it is gathered across a link. (Smits' alpha links) [Sill95a, Smit94, Stam97a]. This raises the cost of the algorithm to $O(k\log k+n)$. A third alternative, proposed by Sillion and Christensen [Chri97, Sill95b], is to represent a cluster as a point that emits and reflects light according to a directional distribution. Both latter methods require light to be pushed down the tree, as with alpha links. Christensen's algorithm appears to be asymptotically the fastest, achieving good quality results in $O(k+n)$ time. Although any of these clustering methods is significantly faster than classical hierarchical radiosity, the need to touch all of the input polygons on each solver iteration can cause their working set to be excessively large for complex scenes.

2.3.4. Importance-Based Radiosity

Smits first introduced the concept of importance-based radiosity methods [Smit92]. In such methods, as well as radiosity being distributed from light sources, a quantity known as importance is distributed from the view-point. The refinement process then targets those elements with high radiosity and importance. In this way, the solution generated is optimised for the particular view-point chosen. This saves considerable time over generating the standard view-



(e) (f)

Figure 10: Hierarchical Radiosity with volume clustering. With the addition of volume clustering, interactions can also take place between clusters of polygons rather than the individual polygons. (e) The floor patch from (c) now interacts with the table as a whole. (f) The table cluster's incident illumination is made up of all but the closest patches and the light sources.

independent solution, at the cost of having to perform more solver iterations whenever the viewpoint changes.

This thesis concentrates primarily on methods for producing view-independent solutions for geometry, so we will ignore importance from here on in. It should be noted, however, that the importance approach could be applied to the central vector radiosity algorithm presented later, if a view dependent solution were acceptable.

2.3.5. Linkless Radiosity

It has been observed that, particularly with wavelet radiosity, the cost of storing transport links can be prohibitive [Will97b, Will97a]. The overhead per link goes up as M^4, where M is the order of the wavelet algorithm used, whereas the over-

head per element only goes up as M^2. Worse yet, whereas elements can often be represented with tensor product bases, with attendant savings in space and time, the geometric kernel governing transport between elements has no such inherent symmetry, and thus cannot.

A number of researchers have proposed methods for reducing or eliminating the storage of links in a hierarchical radiosity method [Stam98, Cuny00]. Such algorithms usually use a modified form of the shooting algorithm, which has the advantage that it reuses links much more rarely than the standard gather-type algorithm for hierarchical radiosity. They then either regenerate transport links as needed, or use a fixed-size cache to store the links; when there is no room left in the cache, the link judged to be least likely to be reused is ejected.

There are some drawbacks to the linkless radiosity methods. There is the increased computation time due to recalculating links, especially given the cost of visibility calculations. Also, not keeping all the radiosity links necessary for the final solution can reduce the utility of the algorithm. A complete list of links is an informative data-structure about scene illumination in its own right. Without it, there can be no final gather stage. It is also potentially useful in deciding where to place light maps, shadow maps, and environment maps. However, linkless radiosity does make the memory overhead of the various wavelet methods much more tractable; for these methods its drawbacks are easily outweighed by its benefits.

The mesh-based random-walk methods of Tobler and Shirley mentioned previously in**Section 2.2.4** can be viewed as another way of avoiding explicit transport links.

2.4. Discussion of Hierarchical Radiosity

In this section we examine some aspects of the hierarchical radiosity algorithm in closer detail, in order to supply enough background to carry the reader through the oncoming chapters on face cluster radiosity.

2.4.1. Link Refinement

Hierarchical radiosity is a top-down algorithm. We start with a small set of initial links, representing light transport in the scene at the coarsest possible level, and iteratively refine those links until our transport representation is fine enough enough to capture the desired illumination effects. Usually, on each iteration we decide whether to refine a given link based on whether some estimate of its approximation error is less than a preset threshold, ε. In early radiosity algo-

rithms, this error was taken to be the magnitude of the transport, or form-factor, itself. As well as working reasonably well, this had the advantage that it was easy to show theoretically that for any given ε, each element in the solution mesh could have at most a fixed number of links, this being a necessary precondition of the $O(n)$ solution process. (This has also proven to be true in practice, and for bases other than Haar; see "An Empirical Comparison of Radiosity Methods" [Will97b], Figure 76.) This is a consequence of the total form-factor of any element summing to unity. Ignoring discretisation error and assuming a closed scene, each element would then have $1/\varepsilon$ transport links associated with it.

More recent radiosity algorithms use either an estimate of the total power carried by the link, or the error in that power, to make refinement decisions. This has the advantage that transport links that affect the resulting solution the most are refined first. Stamminger et al. classify the possible approaches to estimating error into three categories: the previously-discussed form-factor approach, *bounding* methods which establish upper and lower bounds on the transport, and *sampling* approaches which use the variation in samples of the transport to estimate error [Stam97b]. Surprisingly, they find that there is no great difference in result accuracy between the different methods; most of the difference comes in implementation simplicity, robustness, and visibility handling.

When the decision is made to refine a transport link between two elements A and B, with respectively k_A and k_B sub-elements, a decision must also be made as to how to refine that link. Generally, there are three options:

- Subdivide the element A, and replace the link with links from B to the children of A. (k_A new links.)
- Subdivide the element B, and replace the link with links from A to the children of B. (k_B new links.)
- Subdivide both elements, replacing the link with links between all sub-elements. ($k_A \times k_B$ new links.)

Typically this decision is made by comparing the relative contributions of the two elements to the total error metric defined by the link, by comparing the projected areas of the elements along the direction of transport, or even by just comparing the total surface areas of the two elements. (In the area-based comparisons, the largest element is the one to be subdivided.) The first approach has the advantage of a better actual error estimate, the second the advantage of generality, being independent of the error metric, and the third the advantage of robustness. In some situations, the first two approaches can lead to infinite subdivision of one element, with no corresponding reduction in transport error. (An area-limit below

which no patch is subdivided is the usual approach to circumventing this problem.)

Refining the link "on both ends" is reserved for situations where it is known for certain that both ends of the link will end up being refined, such as a self-link between two clusters.

2.4.2. Solving the Radiosity System

To use our element hierarchy and set of links to solve for the radiosity of the leaf elements requires two stages:

Gather

Radiosities are "gathered" across every link in the system to find the corresponding irradiance on the receiving node. The per-link irradiances are summed for each node to produce the total irradiance for that node, but this still leaves us without a consistent representation of the radiosity over all elements.

The gather process is $O(l)$, where l is the number of links. Assuming that as usual the number of links is proportional to n/ε, its time complexity is $O(n)$.

Push-Pull

To get a consistent representation of radiosity at the leaves of the simulation, irradiances are summed down the hierarchy to produce a total irradiance at each leaf, This is known as the "push" phase. This irradiance is then converted to radiosity at the leaf elements by application of the reflection operator, and the resulting radiosities are "pulled" back up the hierarchy, usually via an averaging operation, to assign appropriate radiosities to coarser nodes in the hierarchy.

Because of the hierarchical nature of mesh, this approach to solving the radiosity equation is similar in spirit to the "multigridding" approach often employed in the numerical simulation of physical phenomena. The key feature of multigridding is that the system is solved for progressively finer levels of detail, with the results of each previous stage being used to initialise the next, and thus speed convergence. (When multiple iterations are used, the finer levels can also feed back to the coarser levels on the next iteration.) Wavelet radiosity differs in that this process is interleaved; because the gather operation itself is a multiresolution one, each iteration of the algorithm solves all "levels" of the mesh simultaneously.

The solver is much closer to n-body and multipole methods and barnes hut. One of the key differences is that these algorithms do not explicitly calculate

or store transport links, as they are much more lightweight to calculate than light transport, which often involves visibility computations. They also do not take advantage of rough estimates of the importance of the transport to influence the accuracy of the transport.

2.4.3. Interleaving Refinement and Solution

We have discussed how to refine the links that define the transport of light in our scene, and how to solve for the radiosities in the scene. The remaining question is, how do we mix these two actions to produce our final solution. Broadly, there are three possibilities:

Two Stage

In a two-stage solution, we first refine the links, and then solve for the radiosities; there is no feedback between the two stages. The links can be refined either according to a metric that only takes into account geometric considerations, or one that only takes into account the initial light sources in the scene. While this approach keeps things simple, it ignores the fact that the optimal link refinement is dependent on the final radiosity solution.

Interleaved

To create the ideal distribution of links, we need to know the radiosity solution a priori, so that we can ensure the error in the total power carried by each link is constant. (Put more simply, links that are formed between a bright source and a highly reflective receiver should have less error than those between a dim source or a less reflective receiver.) As is usually the case, this kind of inter-dependence problem can be addressed with a multigrid method. We can *interleave* link refinement and system solution, so that the current estimate of the radiosity solver is used to drive link refinement, which in turn results in better radiosity estimates.

Picking an appropriate solver iteration to interleave is simple; we simply run one iteration of the gather / push / pull process. Picking an appropriate refinement iteration is a little trickier, because refinement is defined recursively. The solution is to remove this recursion, and only allow the possibility of a particular link being refined once per iteration.

Scheduled

A problem with the interleaved approach is that the granularities of link refinement and system solution is not well matched. When considering solution

approaches, it is vital that, in order to reduce error, we try to arrange things so that radiosity is only ever transferred across a link after it has been refined as much as our error criteria demands. To do otherwise is to introduce error into the system. If the link underestimates the actual transfer, the system merely converges more slowly. If it overestimates the actual transfer, however, this can introduces excess radiosity into the system that in the best case will slow down convergence, and in the worst case can prevent convergence.

This can lead to problems with the interleaved approach, where links are not sufficiently refined before radiosity is gathered across them. An alternative is to run refinement until all links meet some error criterion ε_0, then iterate the gather/push/pull stage until the radiosity solution has converged, and then repeat the whole process for a lower criterion ε_1. Typically, this process is repeated for a fixed number of iterations, and a simple rule such as $\varepsilon_{i+1} = \varepsilon_i / \alpha$ used to set successive values of the refinement epsilon.

In some sense, this is like running the two-stage algorithm above with successively smaller epsilon values. However, the solutions and previous transport links are carried over from each previous cycle, so it takes much less time to converge to both an appropriate level of link refinement and a radiosity solution, than if it were being performed from scratch. After each stage, we reduce our current value of epsilon by dividing by a term α.

When used in conjunction with "lazy linking", and $\alpha = 1$, the method can be thought of as simulating successive global bounces of radiosity. In the first stage, we refine links according to just the light sources in the scene, and then solve for radiosity directly transported to scene surfaces. (The lazy linking ensures that radiosity is not transported between surfaces that were both unlit before the stage started.) The second stage further refines the links according to this "direct-only" radiosity solution, and then solves again, this time for the first bounce of light. In this way, we converge to a global radiosity solution. In practice, we set $\alpha < 1$ so as to speed the initial stage and get feedback more quickly.

The scheduled solution method provides the best and most stable results of the various hierarchical radiosity solvers. Some variant of it is used by most hierarchical radiosity implementations. The chief drawback of this approach is that there is no longer a natural stopping point to the solution. Generally, a fixed number of iterations is chosen, and solution is stopped after that many stages. Also, it is much more coarsely grained than the interleaved approach. Thus, the interleaved approach can still be useful for interactive applications, because it delivers results more quickly, and is easier to stop quickly in a consistent state.

2.4.4. Visibility

Dealing with the cost of visibility queries is hard, due to the discontinuous nature of occlusion, and the difficulty of characterizing the distribution of visibility discontinuities in a scene. As a result, many analyses of the complexity of finite element methods simply ignore it.

Generally, an occlusion query is $O(\log n)$ [Shir92]. The originally hierarchical radiosity paper showed that evaluating visibility required the casting of $O(n)$ rays. This is a consequence of the number of links being bounded by n/ε, and thus being $O(n)$, and the fact that we usually cast a fixed number of rays per link to evaluate its fractional visibility. (Because link refinement proceeds in a hierarchical fashion, the total number of links evaluated is linearly proportional to the final number of links used to calculate transport for the scene.)

Thus the time complexity of finite-element radiosity methods *including visibility testing* is potentially $O((k\log k)^2 + n\log n)$. In practice, approaches such as shaft culling [Hain94] and caching of shadow hits [Hain91], and the use of density grids [Fuji86, Sill94a], can make ray-tracing queries closer to $O(1)$ time for evaluating each link transports. Still, even if they are constant time, ray-tracing queries tend to be by far the most expensive part of evaluating the transport kernel, so some care must be taken to optimize their use.

2.4.5. Meshing for Radiosity

Generally the input scene to a radiosity method is polygonized, and the polygons are then split into a mesh of triangular or quadrilateral elements. There are two methods commonly used to adapt this mesh to a radiosity solution as it progresses.

Regular Refinement

The standard mesh-refinement technique employed in hierarchical radiosity algorithms (and progressive radiosity with substructuring) is the regular refinement of elements. For instance, a triangle can be refined into four elements by splitting each edge at its midpoint, and retesselating these points into four subelements. Such refinement operations on an original mesh element form a quadtree. Regular refinement has the advantage of simplicity, and independence—any particular element can be refined independently of another. It also makes refinement decisions straightforward; if the total estimated error across the element is above a certain threshold, we refine it. It has the disadvantage that, if an element is poorly shaped, or not well aligned to the current radiosity solution, the same will hold

for its subelements. Also, quadtree meshes can feature T-vertices, which lead to interpolation problems when rendering, as in **Figure 11**. This can be corrected by balancing and anchoring the mesh [Baum91]. An example of a quad-tree mesh, and its anchored equivalent, is shown in **Figure 12**.

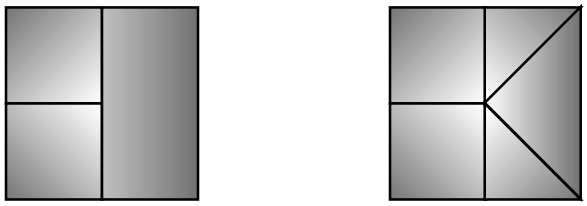


Figure 11: Shading discontinuities at a T-vertex.

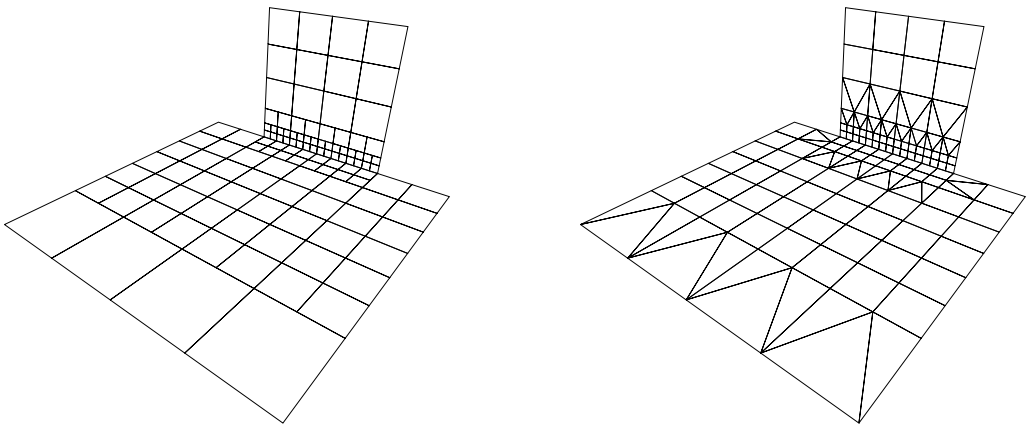


Figure 12: A quad-tree mesh. Left, the original mesh. Right, the mesh after anchoring "T-vertices" to eliminate illumination discontinuities.

Discontinuity Meshing

Discontinuity meshing takes adaptive methods using regular refinement a step further by computing where shadow edges or other visibility discontinuities will occur, and pre-splitting the mesh along such features. This can produce impressive results, especially when the shadows in question are relatively sharp. Its primary drawbacks are that it can be difficult to implement robustly, and it is an object space method, dependent on the edges in the scene. Most previous implementations of discontinuity meshing modify the mesh as a preprocess, and thus attempt to calculate all possible discontinuities. This makes them poorly suited to curved objects and objects with highly detailed polygonal silhouettes.

Discontinuity meshing is also best suited to surfaces that consist of a small number of large polygons. On a highly tessellated, curved surface, a large number of faces would have to be split along discontinuity lines.

2.4.6. Output Representations

When considering radiosity methods, it is useful to keep in mind the potential output targets for the simulation, especially view-independent ones. As much of the work of finding a global illumination solution as possible should be delegated to them, as they have the following advantages:

- Their brute force lends them stability.
- They are hardware assisted in many cases.
- They are implemented solidly by many scan-line renderers.

The most common of these output representations are listed below.

Gouraud-Shaded Polygons

Any renderer has support for drawing a polygon with the colour interpolated from colours specified at its vertices, and support for these primitives in hardware is now commonplace. Gouraud-shaded polygons are a standard output target for radiosity methods. The radiosity for each vertex can be calculated, and the renderer will take care of interpolating these radiosity samples. This is sometimes known as vertex lighting.

Point Light Sources

A number of attempts have been made to transform the output of a global illumination method into a set of point light sources for interactive viewing [Walt97a,

Kell97]. The effects of global illumination can be handled by so-called virtual light sources—lights placed so as to simulate reflected light, rather than where the direct light sources are located. For instance, the light reflected from a blue floor on to an object sitting on it by an overhead light could be simulated by placing a blue virtual light source beneath the floor pointing upwards. This type of light source is often used in computer animation, where the virtual light sources are placed manually by a skilled lighter to simulate a complete lighting solution.

Unfortunately, most graphics hardware can only handle a small fixed number m of point light sources quickly, and this number is often too small to get good results. One solution to this problem is to use multi-pass rendering, where the final image is composited from multiple renderings of the scene, each of which uses m point light sources.

Shadow Maps

Shadow maps can be calculated using a hardware z-buffer, but the actual mapping algorithm used to render the shadows is not in most hardware systems. (An exception is SGI's shadowX OpenGL extension.) While shadow maps initially only handled shadows cast by point light sources, they have been adapted in recent years to handle area light sources as well.

Light Maps

Light maps. These are akin to texture maps, but instead store the irradiance over a surface. They can then be blended with texture maps to produce the effect of shadows over the surface. It is becoming common for interactive games to use light maps to represent lighting detail. For interactivity reasons, it is essential that the polygon count in these scenes remain low; often the graphics cards used are heavily optimised for multi-texturing, and have significant amounts of texture memory, so it is natural to take advantage of this. Also, until recently, most PC graphics cards did not have onboard transformation and lighting, making it relatively expensive to represent lighting detail with extra geometry.

2.5. Problems with Finite-Element Radiosity Methods

The hierarchical radiosity with clustering algorithm described in the previous sections, while capable of impressive performances with scenes of medium complexity, has a number of drawbacks. We examine some of them here.

2.5.1. Mesh Artifacts

The regular refinement employed in refining input polygons means that the orientation and shape of these polygons can have a detrimental effect on interpolation results, both in areas of high radiosity gradient, and in the presence of shadows. **Figure 13** demonstrates some of these problems. The key problem is that, while we use the input geometry to directly define our finite element mesh, it is almost never designed specifically for that purpose. This often leads to tedious and expensive geometry clean ups before an acceptable solution can be generated. (One of the most compelling arguments that can be made for Monte Carlo path-tracing methods is that, leaving aside the noise, they are very robust in the face of scene complexity and tricky geometrical cases.) This sensitivity to the input scene geometry is in contrast to the use of finite-element methods in most other fields, where the solution mesh can be generated specifically to match the quirks of the solver being used, according to a much looser set of input constraints.

2.5.2. Volume Clustering Artifacts

Volume clusters assume that the incoming directional irradiance is constant across the contents of their cluster. This can lead to blocky results, such as those shown in **Figure 14** and **Figure 15**, when adjacent parts of an otherwise connected surface fall within different clusters. A major drawback of volume clustering techniques is that there is no easy way to overcome this problem. While radiosity can be interpolated across surfaces, doing so across volumes would not lead to good results, because of the varying orientations of surfaces within the cluster. The way we correct for those orientations, by pushing irradiance to the leaf polygons “on the fly” whenever we gather it across a link, makes it difficult to construct a higher-order (and thus smoother) approximation of the polygon’s irradiance.

One solution to the clustering artifacts shown here is to decrease the error threshold until links are at the resolution of the leaf polygons, but this negates many of the benefits of using clustering. The problem comes when the density of our input polygons becomes much greater than the illumination density; then this approach can become very costly.

Later, we will show how using surface-oriented clusters instead of volume-oriented clusters allows us to define an interpolation method that mostly overcomes these problems.

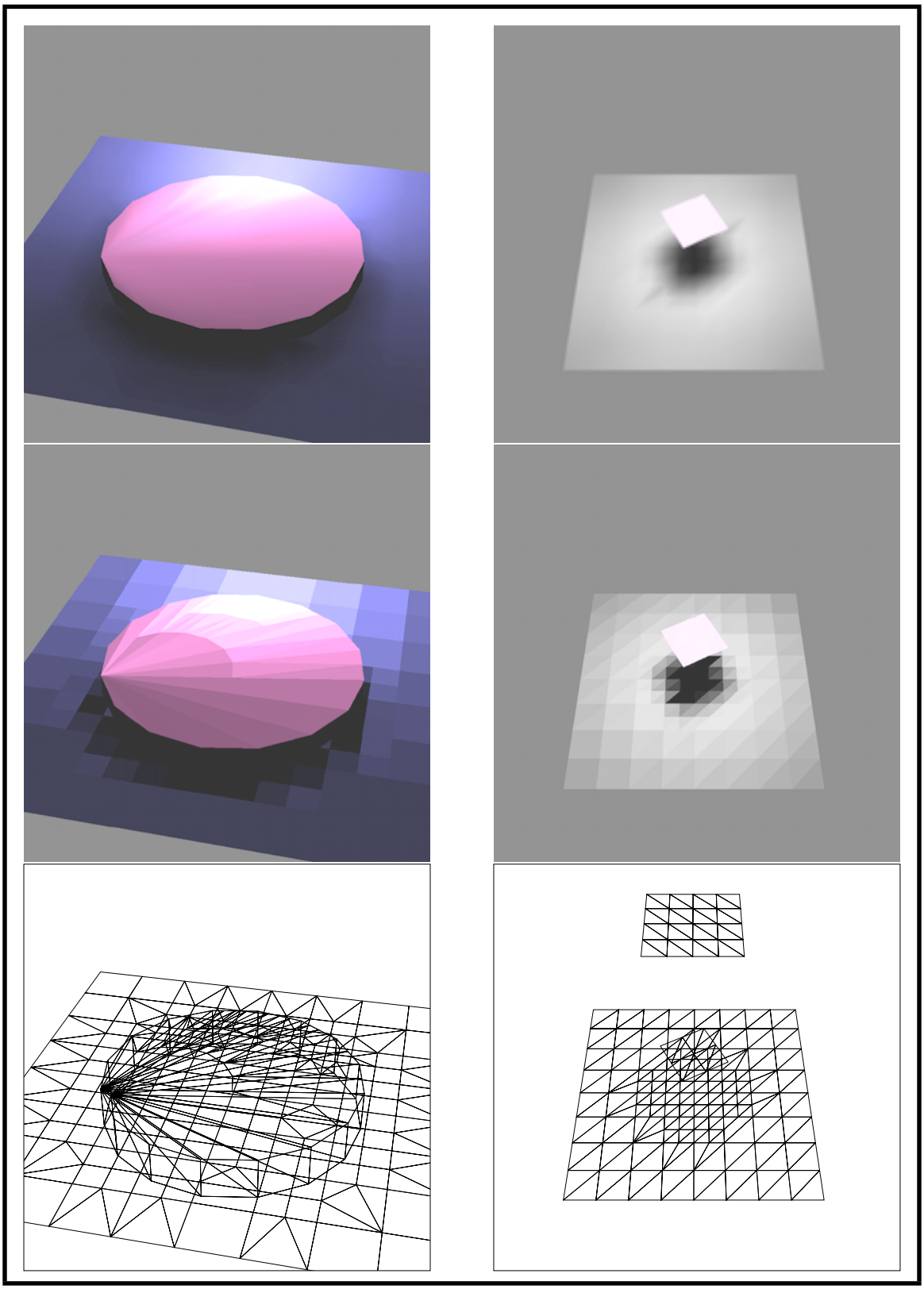


Figure 13: Effect of the underlying base mesh. The base mesh can have a large effect on the result of the radiosity algorithm. **Left:** a poorly-tessellated disc leads to streaking in the well-lit area. **Right:** the orientation of the initial triangulation of the ground square affects the shadow cast on it. Shown from top to bottom are the post-processed solutions, the original solutions, and the solution meshes. In post-processing, these meshes have been balanced and anchored before shading interpolation, but this still leaves us with noticeable shading artifacts.

2.5.3. Visibility

As discussed previously, the transport kernel consists of both geometric and visibility terms, and while finite elements on the whole handle the geometric term well, they can have problems with the discontinuities introduced by sharp shadows. Our perceptual sensitivity to illumination discontinuities makes shadow handling in the final output especially important.

This is especially a problem in the radiosity method, as for transport evaluation, we must bundle these two terms together. This leads to tension between the two; in cases where there are sharp shadow discontinuities, we end up refining heavily to capture the shadow, and light-weight basis functions (Haar, for example) work best. In cases where illumination is smooth, more heavy-weight basis functions—linear, quadratic or cubic—can represent the illumination with fewer coefficients. The problem is that the ideal level of refinement or resolution for capturing visibility effects is often different from that for capturing the unoccluded irradiance.

There have been a number of approaches to ameliorating this problem. One of the earliest was the use of “shadow masks” [Zatz93]. In this approach, visibility is calculated separately, and at a much higher resolution than unoccluded radiosity. This results in a shadow mask, which represents occlusion across the receiving patch, and can be post-multiplied into the radiosity function to produce good shadows. This approach has been refined by a number of other researchers [Slus94].

Another approach is to use what is called a “final gather” stage. The radiosity solution proceeds as normal, using basis functions tuned more towards the geometric transport term, and placing less emphasis on quality of visibility. After solution has been reached, a final solution iteration is performed in which the solution is recalculated using the existing solution, but reevaluating visibility at

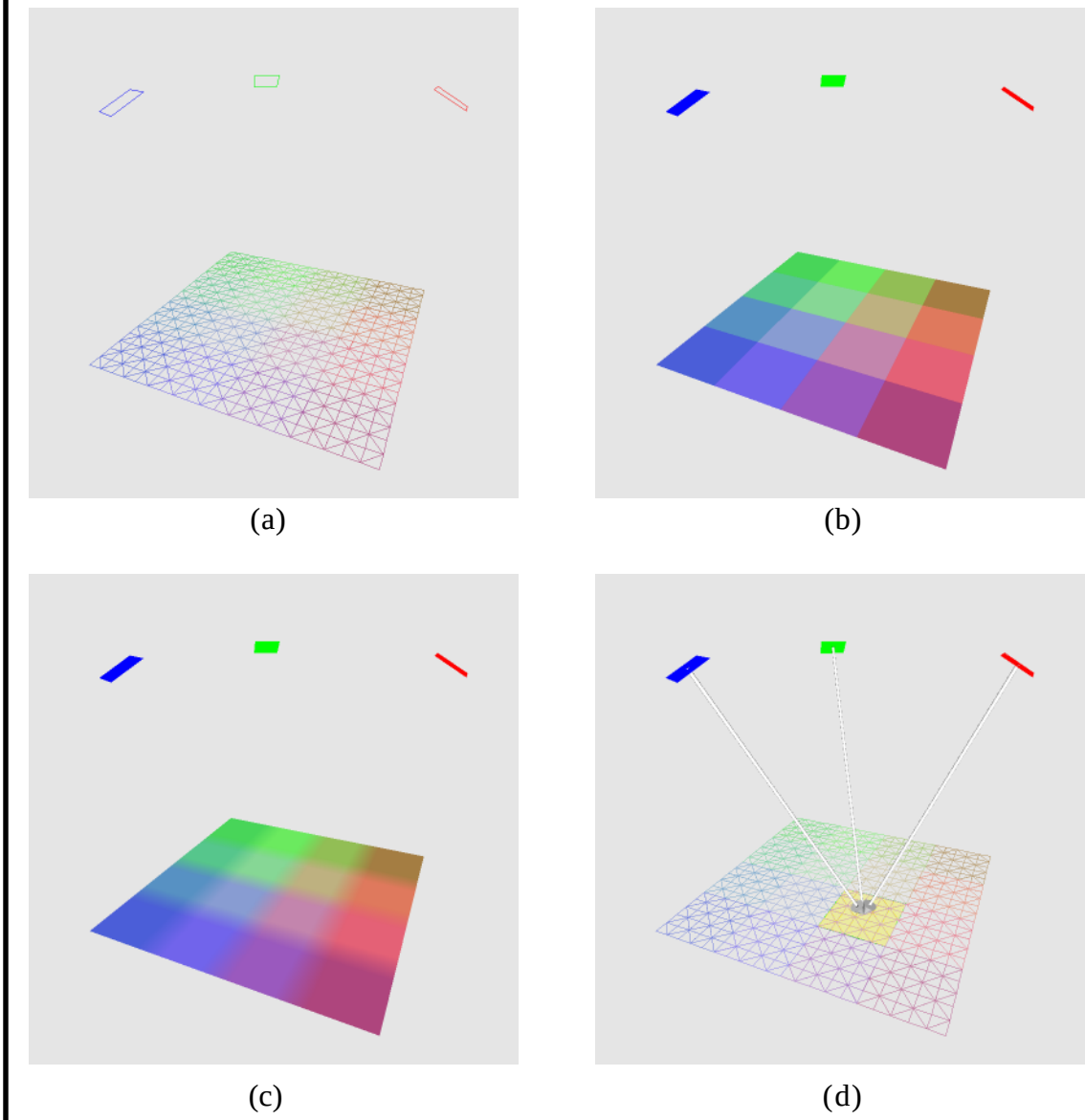
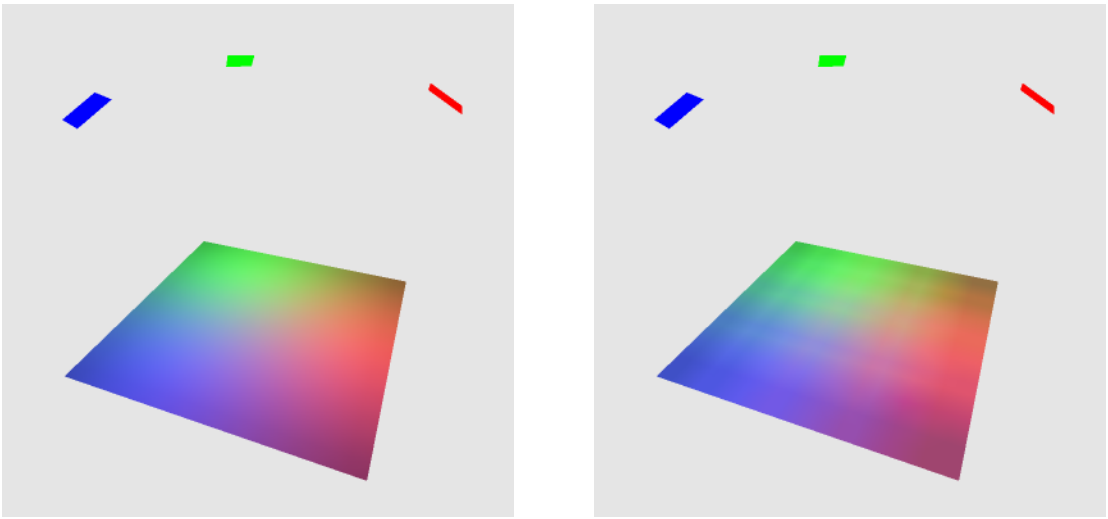


Figure 14: Clustering Artifacts. Using volume clustering on a dense mesh lit by three lights (a) speeds up calculations, but also leads to interpolation problems because of the constant cluster basis functions, as shown in (b). Turning on gouraud shading (c) does not do a good job of fixing this, as the mesh element size is much smaller than the cluster resolution. The contribution of the three lights to a volume cluster is shown in (d).



(e) (f)

Figure 15: Clustering Artifacts cont. Even when the error threshold is reduced, the clustered solution (f) has aliasing artifacts not present in the reference solution (e).

every vertex in the model, for geometric approaches, or every pixel in the final image [Rush88, Lisc93, Chri96]. This approach can produce very nice results; its main drawback is that it is very compute intensive, and is linear in the number of polygons in the illuminated geometry. For large scenes, this can make it prohibitively expensive [Lisc93].

2.5.4. Scenes with Detailed Surfaces

Large, highly-tessellated objects can cause problems when using hierarchical radiosity with volume clustering, beyond the clustering artifacts already discussed, simply because of the sheer number of input polygons in relation to the scene's illumination complexity. Because the algorithm is least $O(k \log k)$, when the natural level of illumination lies relatively high in the hierarchy, the push-to-leaves part of the algorithm's cluster gather stage becomes significant. An example of such an object can be seen in **Figure 17**. This occurs even though the general n-

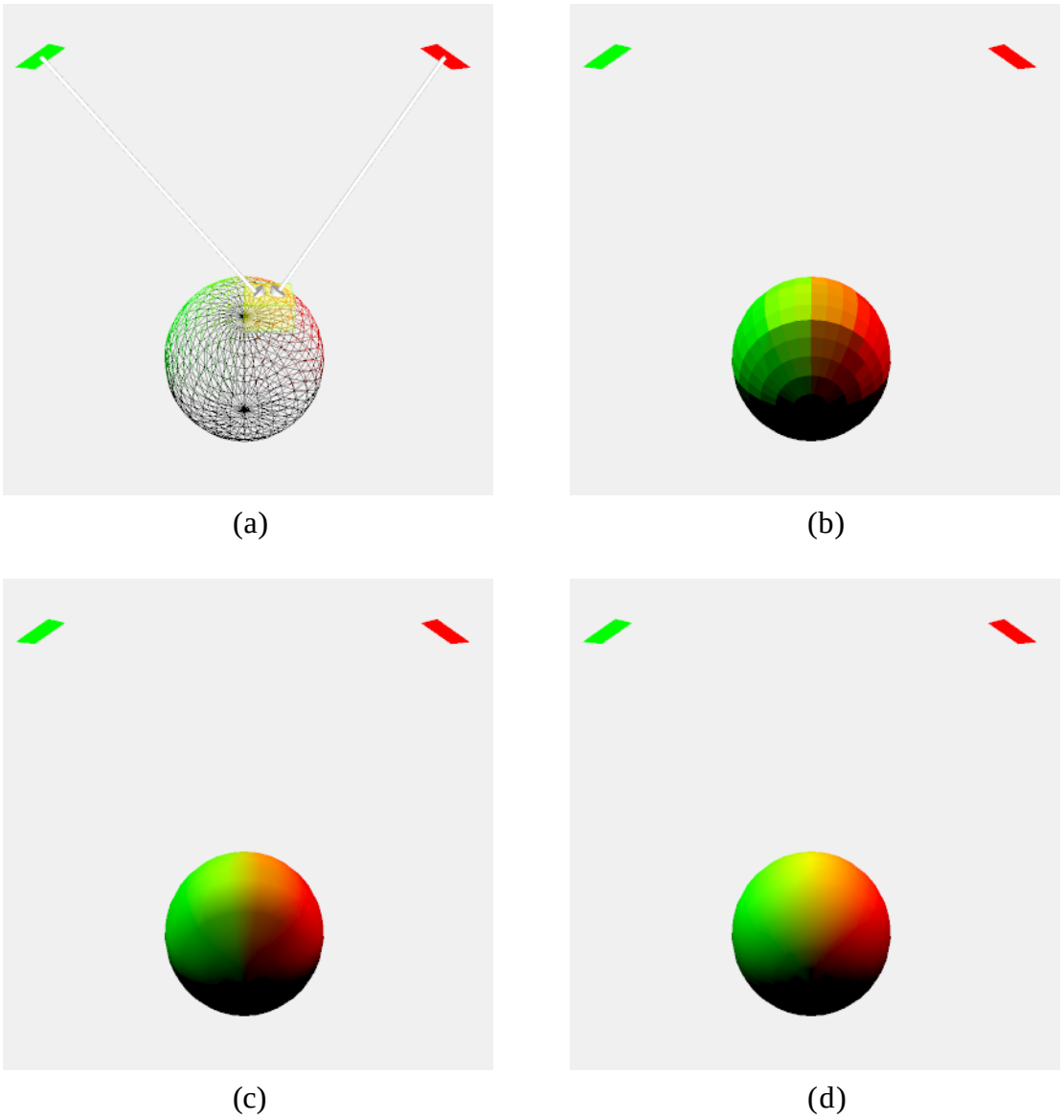


Figure 16: Clustering on a sphere. The cluster level at which light interaction is taking place in (a) leads to the "blocky" solution (b). When Gouraud shading is applied (c) the result still falls short of progressive radiosity (d). Another problem here is that usually volume clusters are axis-aligned, and are not fixed in an object's frame of reference. If we attempt to animate the sphere by rotating it or translating it, any clustering artifacts will stay fixed in space, rather than following the sphere.

body finite element method has the potential to be $O(n)$, where n is the number of elements used in the simulation, and $n \ll k$ in such situations.

There are similar problems with memory behaviour. Because we must push radiosity to cluster leaves to account for varying surface orientation, all input polygons must be touched at least once on each solver iteration, and thus the algorithm exhibits poor memory locality when $n \ll k$. There are other considerations too; a radiosity sample must be stored for all input polygons, for instance, rather than only for the n elements being used in the simulation, as we might wish. Also, if we wish to apply some kind of gouraud-shading post-processing, there is additional overhead in keeping track of the neighbours of all these polygons. Finally, the large number of polygons exacerbates the clustering artifacts mentioned previously; see **Figure 18**.

2.6. Radiosity with Detailed Models

The primary problem this thesis addresses is that outlined by the previous subsection; the application of finite element methods to radiosity scenes containing detailed surfaces, such as scanned models, or tessellated implicit surfaces or height fields. These objects may be geometrically simple, at least on a macroscopic level, but the sheer number of polygons necessary to represent them defeats most current finite element algorithms. To handle such models, we must look again at some of the standard assumptions concerning the radiosity problem.

2.6.1. Reexamining Assumptions

Assuming that our scene contains potentially many high-resolution meshes changes a number of our basic assumptions about the radiosity algorithm:

- A complexity of $O(k)$ in the number of input polygons is a good result.

This is no longer acceptable when k can be on the order of millions of polygons. Ray-tracing approaches are sublinear in time; empirically they are often close to $O(k^{1/2})$.

- The common case is that we need to refine polygons to capture illumination detail.

Now the common case is that the polygons in the input mesh are smaller than the illumination detail we wish to capture. Any solution that involves evaluating the radiosity function, especially visibility, on a per-polygon basis may well be too expensive.

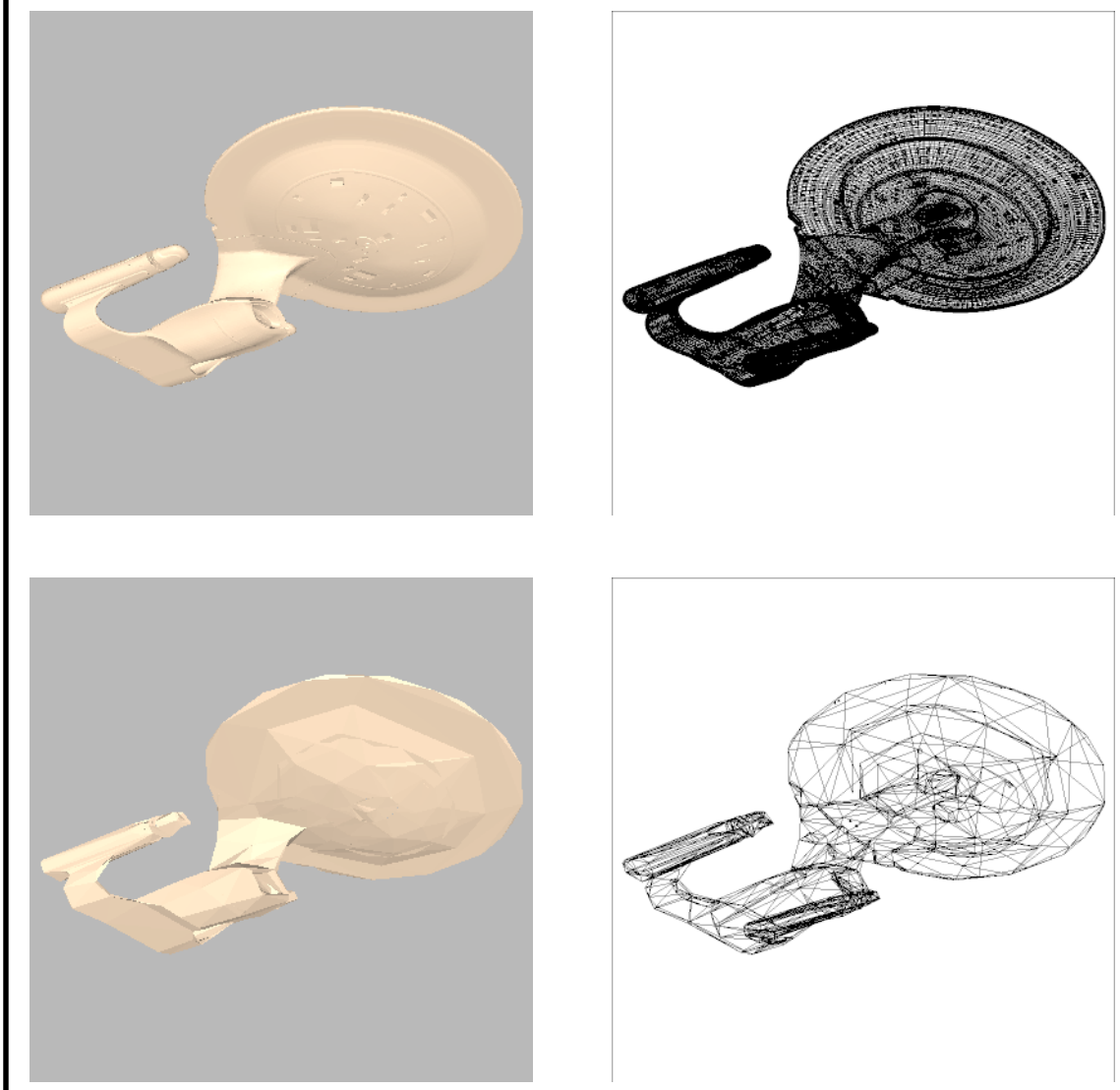


Figure 17: Highly-detailed models. Top: The model of the 'starship enterprise' pictured is composed of 203,878 triangles. The model shown is a medium resolution version of the original model. (Models captured by Cyberware scanner or equivalent device can sometimes reach complexity levels of the order of a million triangles.) Bottom: Such models can be simplified by a technique that creates a multiresolution hierarchy of the model. The simplified model shown here has 1000 triangles.

Figure 18: Clustering problems on a real scene. Note especially the gear stick; either side of the knob belongs to a different cluster, with effects similar to those seen in **Figure 16**. (From Hasenfratz et al. [Hase99].)

- Transfer links will wind up pointing to the leaves of the element hierarchy, i.e. the input polygons.

In our target scenes, the illumination complexity, which affects how far in the element hierarchy the solver must descend to produce a solution, is much smaller than the geometrical complexity, i.e., the number of surfaces in a scene. Hence, transfer links will wind up pointing closer to the top of the hierarchy, and rarely to the leaves of the models in question.

2.6.2. Discussion

In summary, this dissertation is concerned with the hierarchical radiosity algorithm, which solves for the global transfer of diffuse illumination in a scene. While its potential algorithmic complexity is superior to both previous radiosity

methods and ray tracing methods, it is more inflexible than pure ray-tracing algorithms, and the density and orientation of the polygons in the input scene can unduly affect the output of the method. Worse yet, the time complexity of the best current radiosity methods is at least linear in the number of input polygons; $O(k)$. To compete with Monte Carlo methods for detailed scenes, it is crucial that methods be developed that are sub-linear in geometric complexity, otherwise, with ever-increasing scene complexities seen in the computer graphics industry, radiosity will become restricted to a niche as an illumination method.

In tackling these problems, we will assume the input and output geometry is composed of:

- Polygonal meshes.
- Per-face materials; diffuse reflectance, texture maps.
- Face or vertex colours.
- Texture maps for detail.
- Bump maps for detail.
- Any part of the geometry can be a light source.

All of this is similar to standard model formats such as Alias/Wavefront's OBJ format, 3D Studio Max's binary format, and the research format MGF. It is also helpful to keep in mind what our scenes are typically composed of:

- Large flat surfaces, the easiest case. Typically represented with a small number of polygons; low complexity.
- Terrain geometry. Typically height-field meshes; medium to high complexity.
- Lighting fixtures. Low to medium complexity.
- Models (animals, objects, vehicles, etc.) Often detailed but largely connected polygonal meshes; medium to high complexity

Any radiosity method should ideally be able to handle this entire range of complexities, from small, large wall polygons, to tiny detail polygons making up a complex model.

In the next chapter, I will show how by using flexible surface hierarchies similar to certain model simplification hierarchies [Garl97], many of these concerns can be addressed, increasing both the speed and the quality of the basic algorithm. Specifically, by incorporating a data-structure similar to such multiresolution meshes into our radiosity simulation, we can reduce the complexity of the algo-

rithm to $O(s \log s + n)$, where s is the number of faces in the most simplified version of our scene. In complex scenes, $s \ll k$, and often, $s < n$.