Hierarchical Radiosity with Multiresolution Meshes

Andrew J. Willmott

21 November 2000

CMU-CS-00-166

School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890

Thesis Committee:
Paul Heckbert, Chair
David O'Hallaron
Steven Seitz
Francois Sillion

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This work was supported by NSF grants CCR-9505472, CCR-9357763, and DMI-9813259, and the Schlumberger Foundation.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or the United States government.

Keywords: global illumination, hierarchical radiosity, face cluster hierarchies, multiresolution models.

Abstract

The hierarchical radiosity algorithm solves for the global transfer of diffuse illumination in a scene. While its potential algorithmic complexity is superior to both previous radiosity methods and distributed ray tracing, for scenes containing detailed polygonal models, or highly tessellated curved surfaces, its time performance and memory consumption are less than ideal.

My thesis is that by using hierarchies similar to those of multiresolution models, the performance of the hierarchical radiosity algorithm can be made sub-linear in the number of input polygons, and thus make radiosity on scenes containing detailed models tractable. The underlying goal of my thesis work has been to make high-speed radiosity solutions possible with such scenes.

To achieve this goal, a new face clustering technique for automatically partitioning polygonal models has been developed. The face clusters produced group adjacent triangles with similar normal vectors. They are used during radiosity solution to represent the light reflected by a complex object at multiple levels of detail. Also, the radiosity method is reformulated in terms of vector irradiance. Together, face clustering and the vector formulation of radiosity permit large savings. Excessively fine levels of detail are not accessed by the algorithm during the bulk of the solution phase, greatly reducing its memory requirements relative to previous methods. Consequently, the costliest steps in the simulation can be made sub-linear in scene complexity.

I have developed a radiosity system incorporating these ideas, and shown that its performance is far superior to existing hierarchical radiosity algorithms, in the domain of scenes containing complex models.

This dissertation is dedicated to my grandfather, Rowland Harman

Acknowledgements

First, I would like to thank my advisor, Paul Heckbert, for support and advice during the course of my study at CMU. He has provided a substantial well of experience to draw on, and his comments and suggestions on numerous drafts of this work have helped give it more focus and direction than it would otherwise have had.

I'd also like to thank my committee members, Andy Witkin, Francois Sillion, Steven Seitz, and Dave O'Hallaron, all of whom provided valuable insights, comments, and feedback along the way. Andy left CMU for Pixar after my thesis proposal, but provided academic direction and wisecracks before then.

Many thanks are due to Michael Garland, for early help with both multiresolution models and his QSlim software, the starting point for my doctoral research. I had many useful discussions with Michael about integrating these methods within a hierarchical radiosity system. In particular, after my comments on how I was essentially treating the vertex hierarchy as a cluster hierarchy over the dual of the original model, by associating surface elements with each vertex, Michael went away and adapted his edge-collapse algorithm to produce this hierarchy directly, resulting in the face cluster algorithm that much of this thesis is based on. This new algorithm, which applies techniques from geometry simplification to surface hierarchies, should prove useful for many applications besides radiosity.

Thanks to the graphics crew for entertainment and conversation during my time at CMU, especially my officemates, Scott Draves, Sebastian Grassia, and Jovan Popovic, and also Jeff Smith, for a steady stream of dry wit. Thanks to Joel Welling of the PSC for machine access over the years, and Greg Ward for help with Radiance. And thanks to those at the University of Auckland, most especially Richard Lobb, for getting me started on the graphics path during my undergraduate education.

There are many friends who made my stay in Pittsburgh enjoyable. I would especially like to thank Rob O'Callahan, Scott Davies, David Rochberg, and Herbie Lee and Ted Wong, my housemates during that time, for conversation, moral support, and dinners. Special thanks also to Lin Chase, for hardware support when my laptop died during Christmas in England. Chapter 4 is dedicated to you, Lin!

Many thanks are due to my family, whom I've seen so little of in the last few years. My Grandfather always encouraged my interest in things technical, and hence this thesis is dedicated to him.

Finally, my love and deepest gratitude to Alma Whitten, whose support and encouragement was crucial to the completion of this work.

Note: Where possible, images in this dissertation have been corrected for a display device with a gamma of 1.8.

Contents

Chapter 1

Introduction

1.1. Computer Graphics

Computer graphics has grown at an astounding rate over the last three decades. In the 1970s, frame-buffers capable of displaying digital images were rare and expensive. These days it is rare that we see an image or video sequence that has not been manipulated in some way by a computer, and computer graphics has become ubiquitous; it underpins the most commonly used computer interfaces, drives many of the applications, and is fundamental to almost all computer-based entertainment.

In three-dimensional computer graphics, we are concerned with producing an image or an animation of some 3D world, that may be either purely imaginary, or may be trying to mimic some real-world counterpart. This process features a number of stages:

Modelling. Geometrical representations of the scene are produced, consisting of sets of points, faces, and other geometrical entities. These representations can be constructed by hand, modelled mathematically, or even scanned from real-world objects with a laser range scanner.

Animation. The various models are composed into a *scene*, and then (usually individually) animated. That is, the position and motion of each object in the scene, including the camera, is specified over time.

Lighting. Lighting a scene involves the selection of reflectances, materials and texture maps for the models in the scene, and the positioning of virtual light sources to illuminate the scene. (The former could arguably fall under the "modelling" category, but here I follow the conventions of the animation industry.)

Rendering. The rendering stage, the one I am primarily concerned with, is responsible for taking the three dimensional scene specification produced by the previous three stages, and producing the two dimensional representation of it as seen by the camera. This can be separated into two sub-problems; determining which parts of the scene correspond to which parts of the output image (perspective projection, depth sorting), and determining how scene surfaces look after the effects of lighting and reflection have been taken into account (shading).

In *photorealistic* computer graphics, we attempt to model the real world, and to produce a rendering of the result that matches the corresponding photograph of the real world. In *physically-based* rendering (and indeed animation), we attempt to reproduce the physical processes involved in this.

In what follows, we are primarily interested in the photorealistic, physically-based rendering problem, and, specifically, that part of the problem known as *global illumination*.

1.2. Global Illumination

We can define global illumination simply as, the propagation of light from light sources throughout the environment defined by our scene. This is in contrast to local illumination, which only considers a light source, and those surfaces it lights directly. Global illumination applies this local operation repeatedly, treating lit surfaces as light sources in turn, allowing multiple bounces of light through the scene and eventually to the camera. Consider **Figure 1**, for instance; the difference between the globally-lit scene with shadows and the direct-lit unshadowed scene more typically seen in interactive graphics is almost as great as the difference between direct lighting and no lighting.

The simulation of global illumination is one of the greatest challenges of realistic image synthesis; it requires modelling the transfer of light through the environment being rendered, including the effects of interreflection between objects. Currently there are two classes of methods for calculating global illumination; Monte Carlo-based and Finite Element-based. The former is usually loosely referred to as *ray tracing* and the latter as *radiosity*. Broadly speaking, Monte-Carlo based methods are better suited to simulating the interreflection between highly

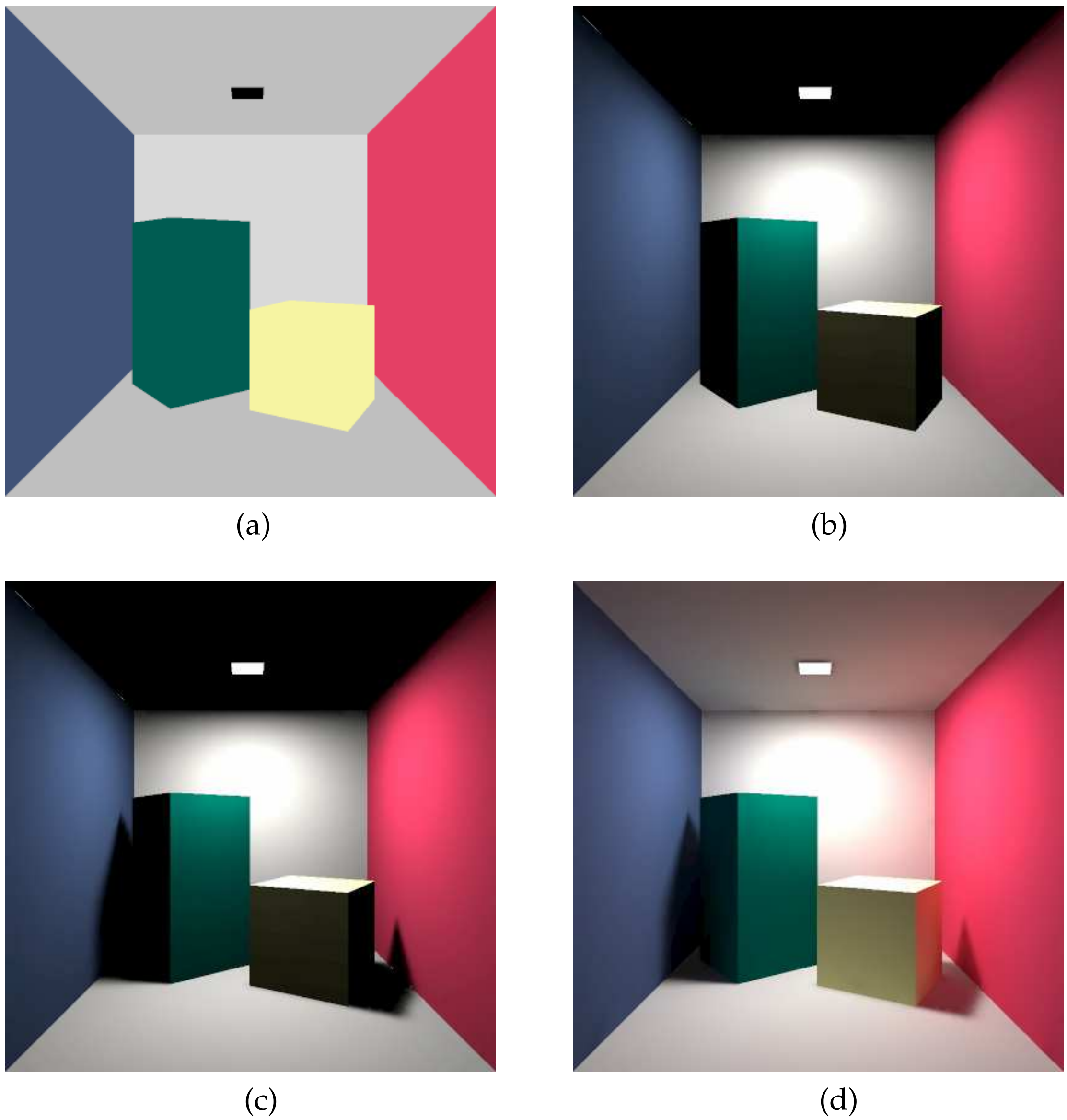


Figure 1: The need for global illumination. A variant of the famous "Cornell Box" scene is shown under several different lighting conditions. (a) No lighting; flat shading only. (b) Direct lighting only; notice the completely black ceiling and cube sides. (c) Direct lighting with shadows. (d) A complete solution for propagated light in the box. Notice the illuminated ceiling, and the cube's sides are now lit by the red side wall.

specular (mirror-like) surfaces, and finite element methods to diffuse (matte-like) surfaces (see **Figure 2**).

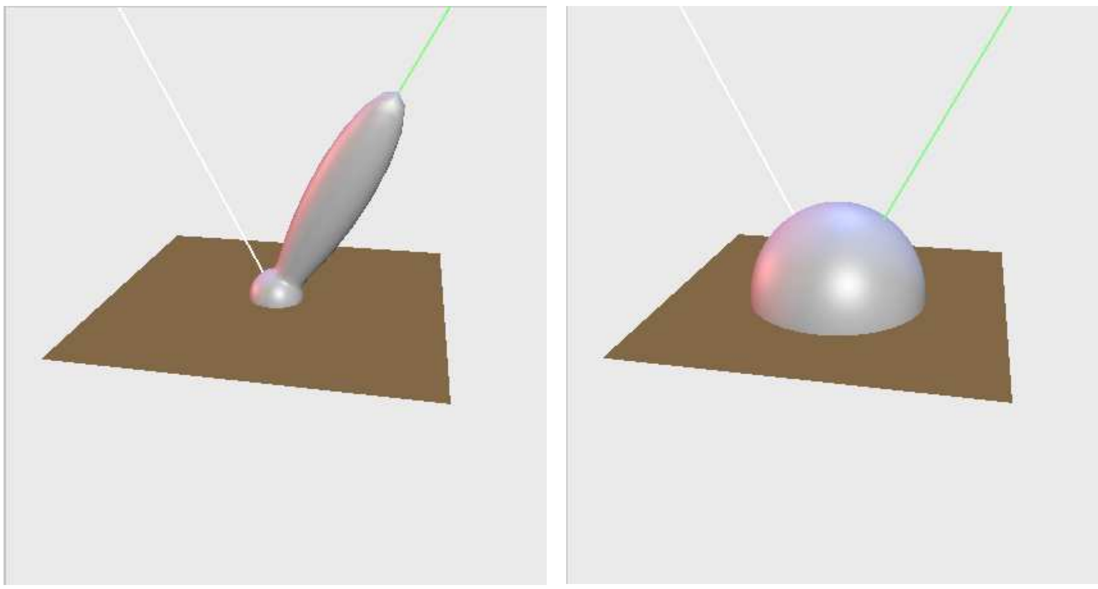


Figure 2: Different reflectance characteristics. Shown are polar plots of outgoing radiance, for light incident along the white ray. Left, a specular surface reflects light directionally, usually close to the angle of reflection. Right, a diffuse surface reradiates light equally in all directions.

1.2.1. Radiosity Methods

The basic radiosity algorithm is limited to simulating scenes with diffuse surfaces; this is both a strength and a weakness. It is a strength because any diffuse-only global illumination solution is *view independent*. This means that the output of most radiosity methods, instead of being a single image, is illuminated geometry. This geometry can then be quickly rendered from any viewpoint, without repeating any global illumination calculations; indeed, without doing any lighting or shading calculations at all. On the other hand, the weakness of radiosity as a global illumination method is precisely that it ignores any specular reflection in the scene. Such effects must be added afterwards, with a ray-tracing pass. Even then, this does not give us a full global illumination solution, as there is no chance for mixing specular and diffuse light bounces.

Still, radiosity methods that produce illuminated geometry are worth pursuing for a number of reasons. The interreflection of diffuse light is an important phenomenon in realistic image synthesis, particularly for indoor scenes, where a significant fraction of the light illuminating a surface comes not directly from a light source, but indirectly, by bouncing off one or more other surfaces. Although some ray-tracing methods that cache diffuse samples [Ward88] are currently faster than those radiosity methods used in industry, the hierarchical nature of state-of-the-art radiosity systems promises better performance, if only the dependence of those systems on a scene's geometrical complexity, as opposed to its inherent illumination complexity, can be removed. Finally, as well as illuminated geometry, many radiosity methods are capable of producing a global representation for the light flow in a scene. This is useful both as a guide to a ray-tracing postprocess, and as an aid to initialising more complicated Monte Carlo-based global illumination simulations.

1.2.2. Applications

Radiosity's areas of application include the following.

Architectural and Lighting Design

One of the major applications of radiosity methods is the prototyping of various architectural and lighting designs. A big advantage of a general radiosity solution is that its output is illuminated geometry, suitable for interactive walkthroughs. Such walkthroughs can be used to gain a more realistic sense of space and light in an architectural design before that design is actually implemented. Lighting fixtures can be rearranged, interior walls relocated, materials changed, and the results evaluated.

Radiosity methods have been used extensively in these areas, from building and luminaire design to theatre lighting design; an example is shown in **Figure 3**.

Virtual Sets

It is becoming commonplace to build "virtual sets" for television shows, such as sports shows, game shows, or news broadcasts. These sets must be combined with live footage in real-time, but a high level of visual realism can still be

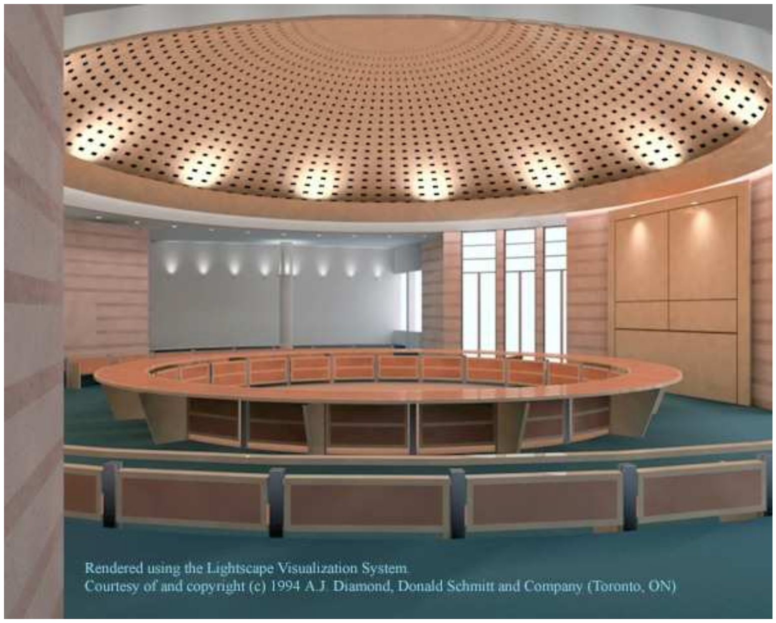


Figure 3: Radiosity used for architectural and lighting design. Note the soft lighting and the large number of light sources. (Picture from Lightscape.)

attained by mixing a precalculated diffuse-light solution with hardware lighting for (view-dependent) specular effects.

3D Games

Today, convincing realism is an important factor for the success of a computer game title. More and more games use techniques from realistic image synthesis such as precomputed global illumination, shadow maps, as well as advanced mirror and refraction effects to increase the illusion of reality. An example is shown in **Figure 4**. The game level shown is reminiscent of some office scenes that have appeared in global illumination papers, although the low polygon count and obvious discretisation artifacts make clear its interactive nature.

Arguably, this area is a more promising one for global illumination algorithms than computer animation for movies or television. The level-of-detail

Figure 4: A screen-shot from the interactive 3D game, Quake. The level shown in this picture features precomputed global illumination effects, in spite of needing to be rendered at interactive rates in graphics hardware. (Picture from ID Software.)

trade-off in a typical piece of such computer-generated animation is tilted more towards detail and less towards interactive rendering times, making the barrier to entry for such algorithms much higher. This may not make sense at first, as surely, without the need to be interactive, there is more time to amortize the cost of any illumination algorithm. Unfortunately, it is a rule in Computer Graphics that scene complexity expands to fill the rendering time available, and, as illumination algorithms scale at best linearly (and usually worse than linearly) with scene complexity, there is little time left over for such effects, except in small, limited situations.

Sky Illumination and Outdoor Scenes

Radiosity methods can be very useful for outdoor sky illumination; the sky is essentially a large, diffusely-emitting (blue!) light source [Daub97]. While outdoor scenes are generally considered to feature mainly direct lighting from the sun, contributions from the sky are still significant in any scene featuring shadows, as is reflected light from building walls, or scenes from an overcast day. An example is shown in **Figure 5**.

1.3. Contents of Dissertation

This thesis is primarily concerned with making the use of finite-element methods for global illumination tractable for scenes containing large models. In the next chapter, I give a brief overview of the radiosity problem, the methods commonly used to solve it, and the hierarchical radiosity method that my work builds on. Chapter 3 then presents the basic face cluster radiosity method. It starts with some motivation, and presents the core of the basic algorithm; the use of face cluster hierarchies, and the use of vector irradiance.

Chapter 4 presents the theoretical justification and error analysis of the radiosity part of the algorithm. Chapter 5 presents some analysis of Garland's original face cluster construction algorithm, along with my modifications to it to improve its speed and memory use, as well as the hierarchies generated, and some examples of its use.

Details of my implementation of face cluster (and hierarchical) radiosity are presented in Chapter 6, and in Chapter 7 I present results for several variants on both a simple test scene, and a more realistic "museum scene".

Finally, we round off with conclusions in Chapter 8.

Figure 5: Radiosity used for outdoor scenes. Note that, without taking indirect lighting into account, the areas in shadow would be completely black. (Picture from Lightscape.)

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*:

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

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 $\vec{\omega}$;
- $\rho(\mathbf{x}, \vec{\omega} \to \vec{\omega}_i)$ 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*:

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

where

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

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 $\mathbf{x}$;
- $V(\mathbf{y} \to \mathbf{x})$ is the visibility between points $\mathbf{x}$ and $\mathbf{y}$;
- $G(\mathbf{y} \to \mathbf{x})$ 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,

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

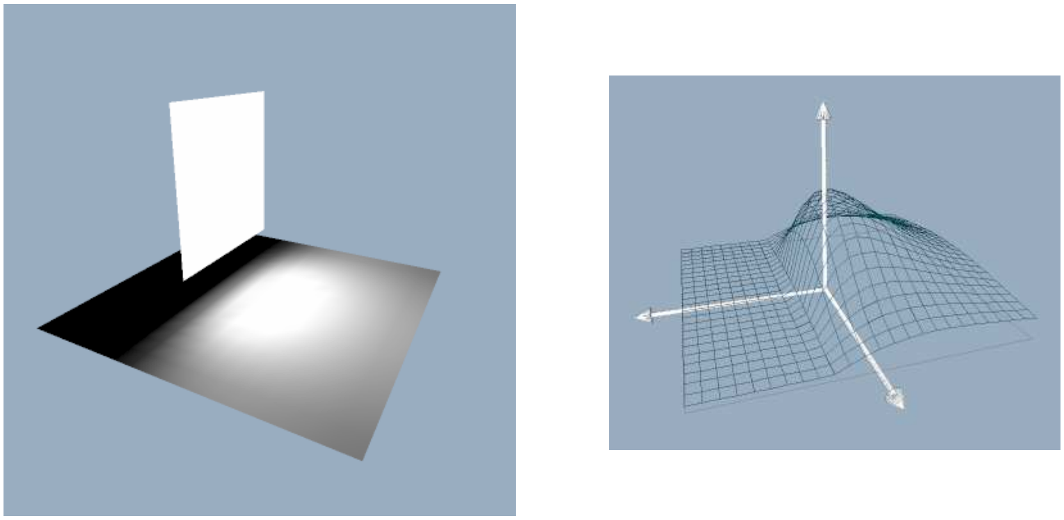


Figure 6: The radiosity kernel. Shown on the left is a simple configuration of a diffuse emitter and a diffuse reflector. On the right is a plot of the irradiance across the receiving patch; it is largely smooth, even with the two polygons in close proximity, except where we cross behind the light source, 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:

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

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

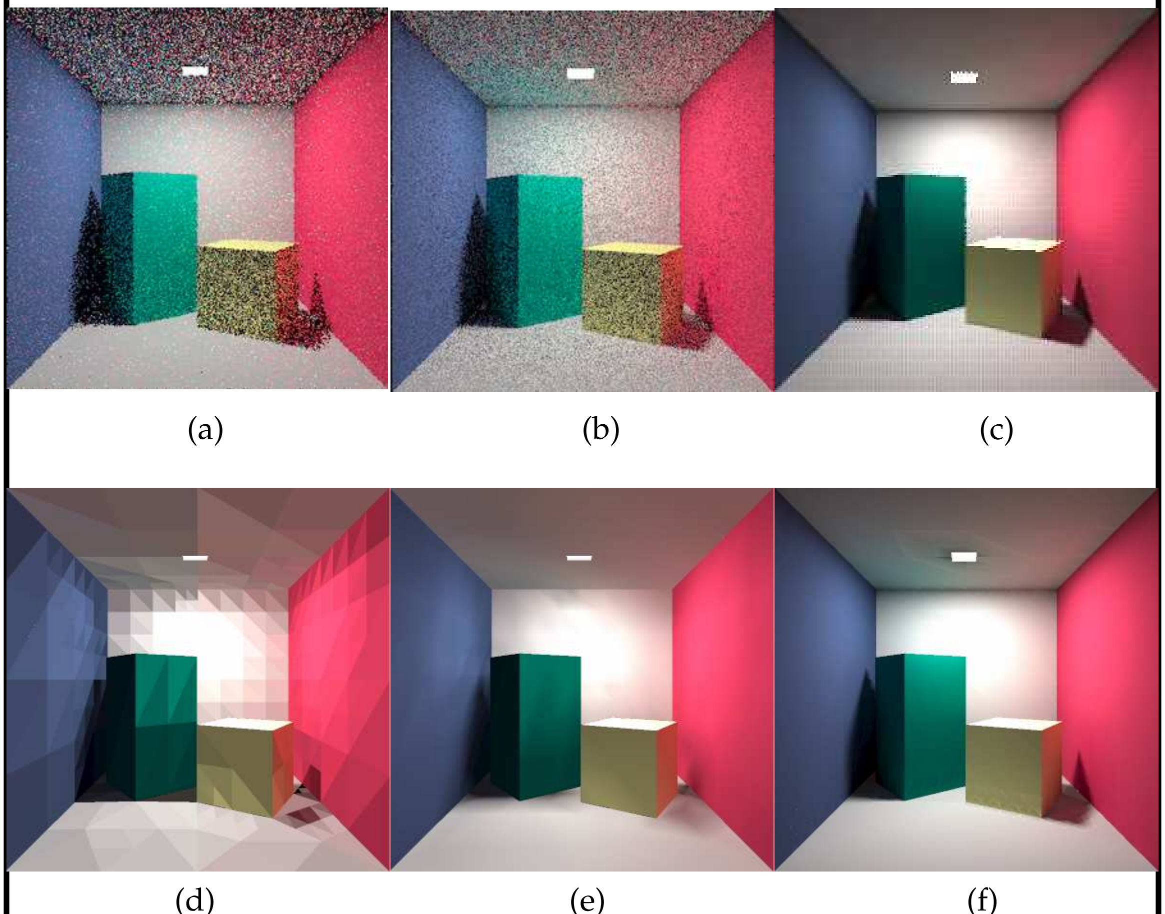


Figure 7: Global Illumination methods for the Cornell box scene. Top, three ray-tracing methods; (a) stochastic ray-tracing, (b) bi-directional ray-tracing, and (c) diffuse sample caching (Radiance). Bottom, hierarchical radiosity with (d) the Haar basis, (e) the Haar basis and a smoothing pass, and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

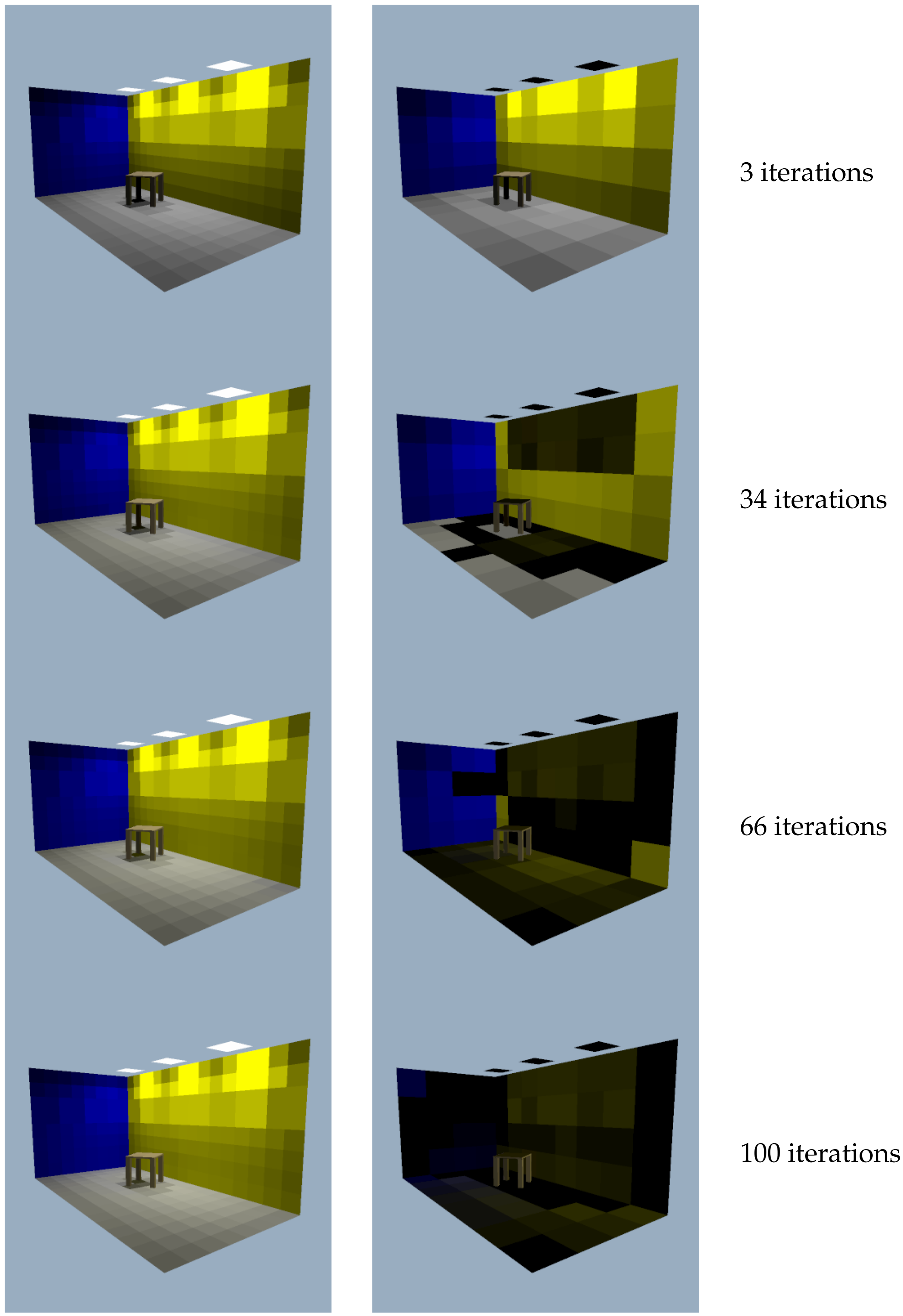


Figure 8: Shooting steps. Left: radiosity, right: unshot radiosity.

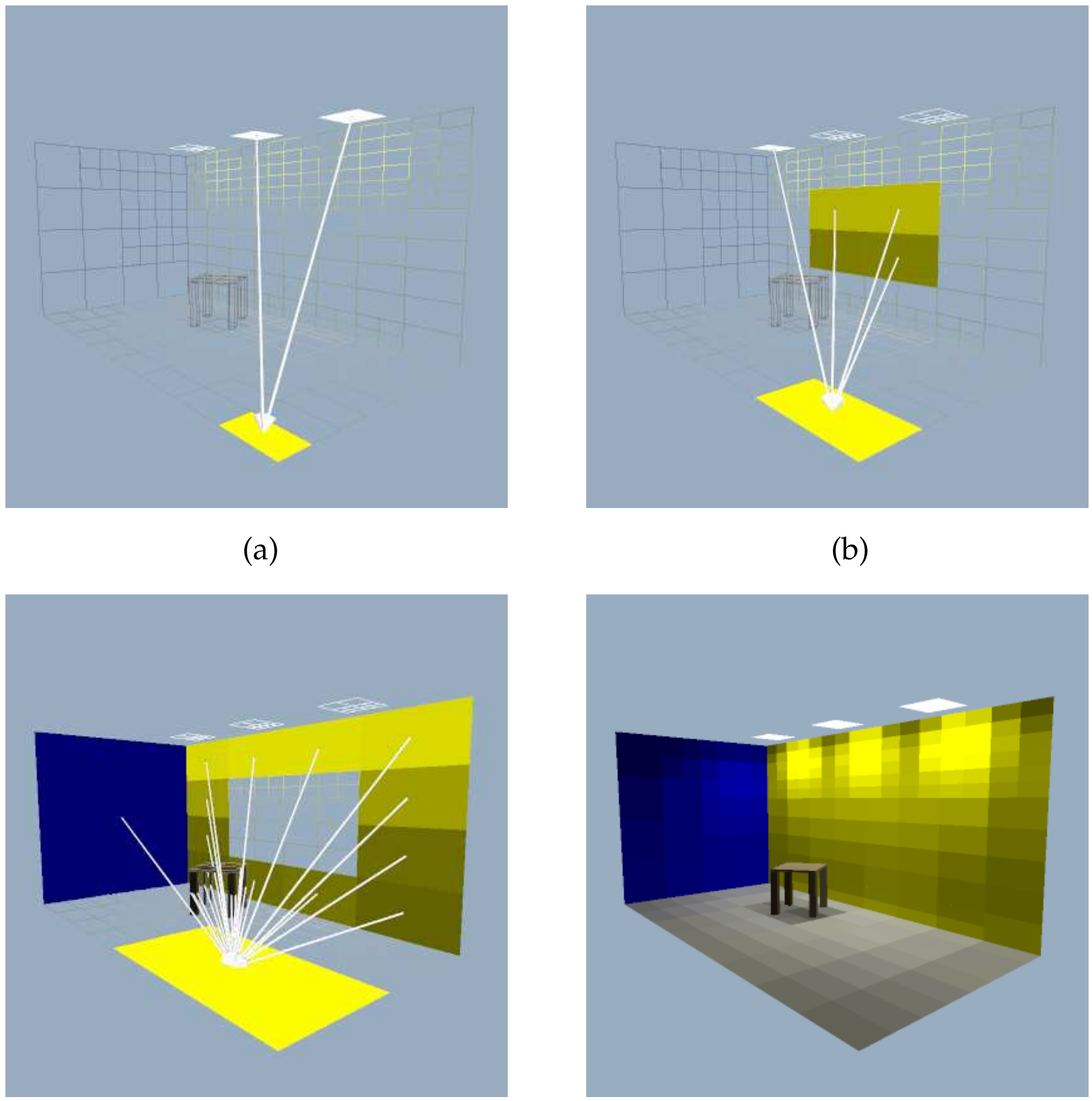
face, a given surface is meshed coarsely, but when reflecting to a nearby surface, it is meshed more finely.

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

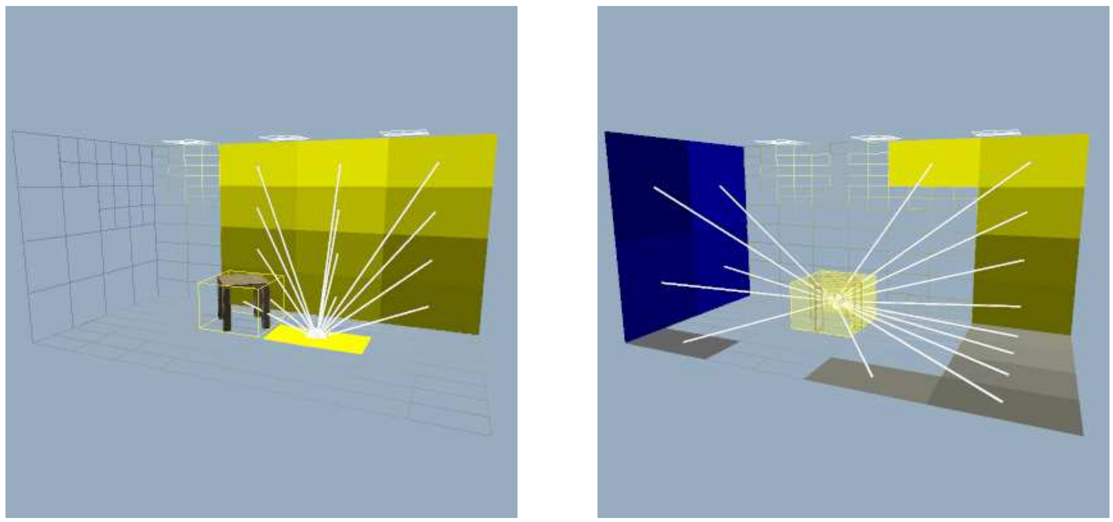
Classical hierarchical radiosity algorithms work well on scenes with a small number of large polygons, but the k^2 term means they become impractical in time and memory consumption for scenes of several hundred polygons.

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 viewpoint 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 overhead 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 algorithms, 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.

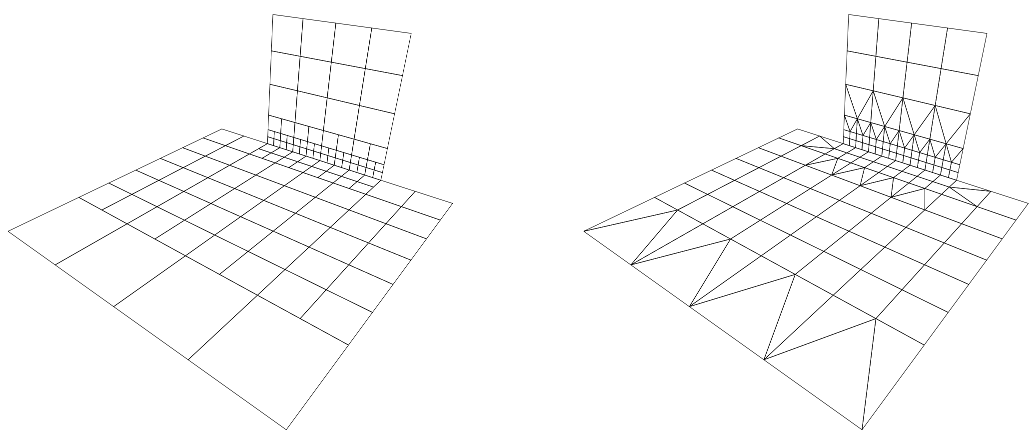


Figure 12: A quad-tree mesh. Left, the original mesh. Right, the mesh after anchoring “T-vertices” to eliminate illumination discontinuities.

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.

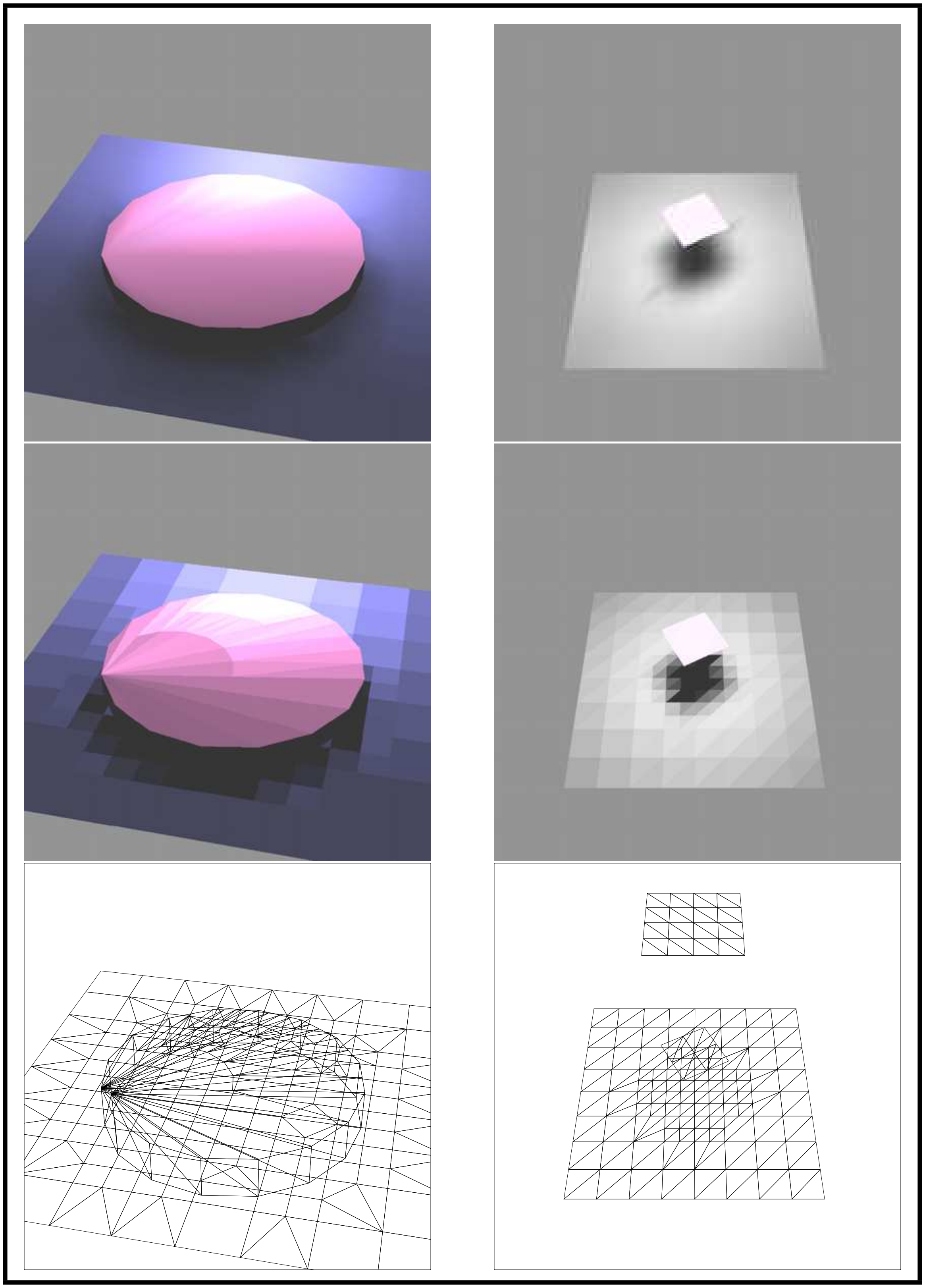


Figure 13: Effect of the underlying base mesh. The base mesh can have a large effect on the result of the radiosity algorithm. **Left:** a poorly-tessellated disc leads to streaking in the well-lit area. **Right:** the orientation of the initial triangulation of the ground square affects the shadow cast 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

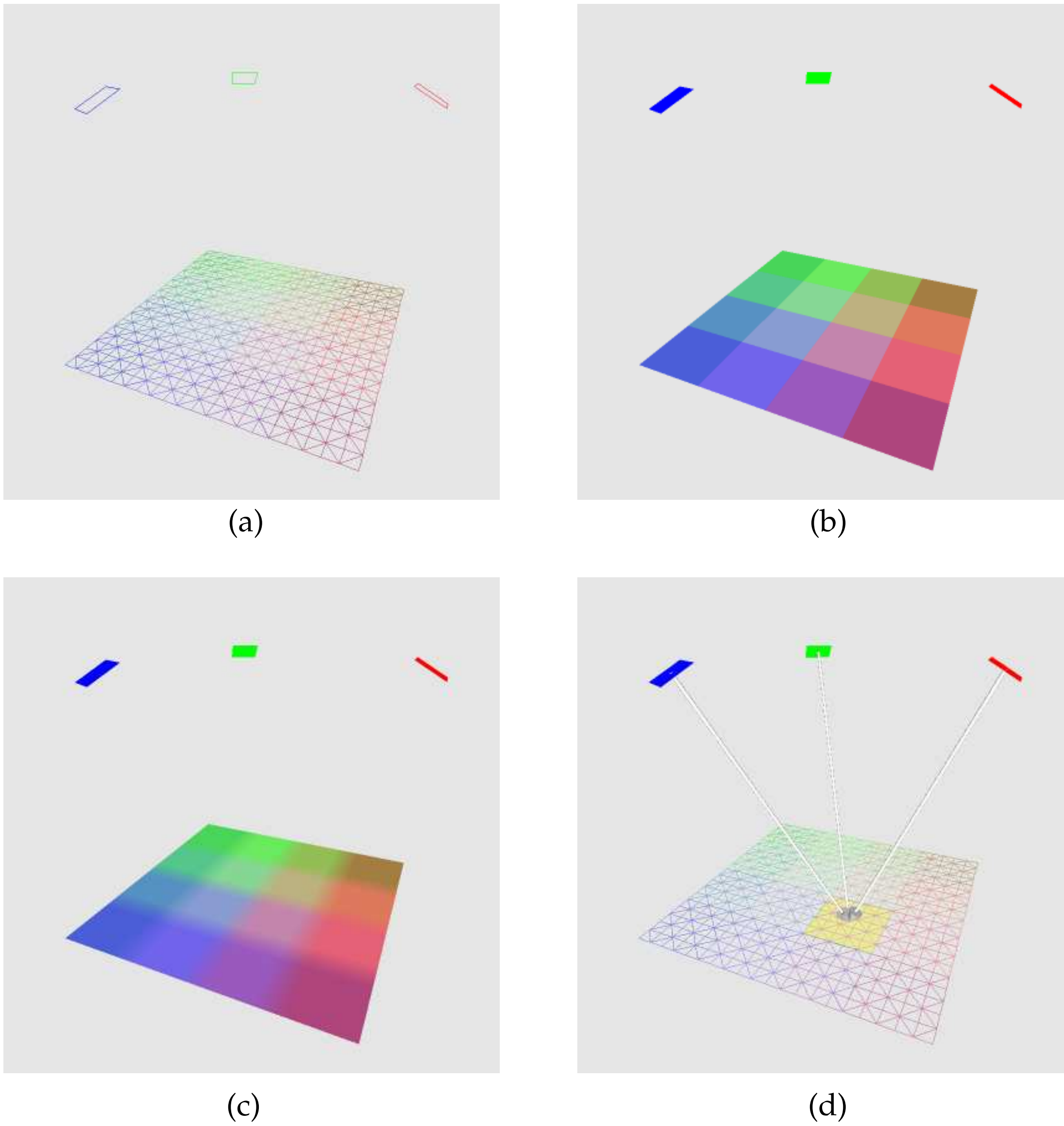


Figure 14: Clustering Artifacts. Using volume clustering on a dense mesh lit by three lights (a) speeds up calculations, but also leads to interpolation problems because of the constant cluster basis functions, as shown in (b). Turning on gouraud shading (c) does not do a good job of fixing this, as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

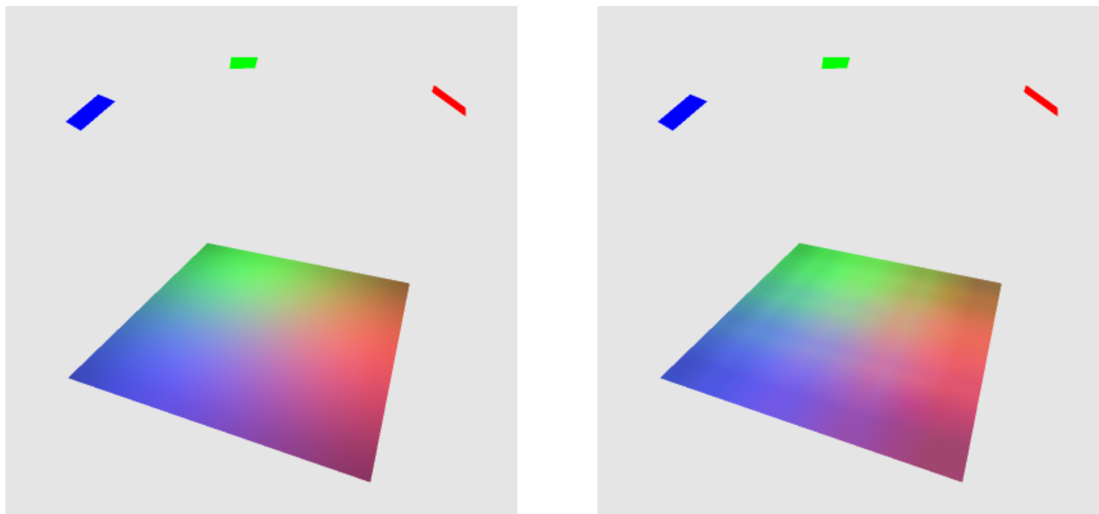
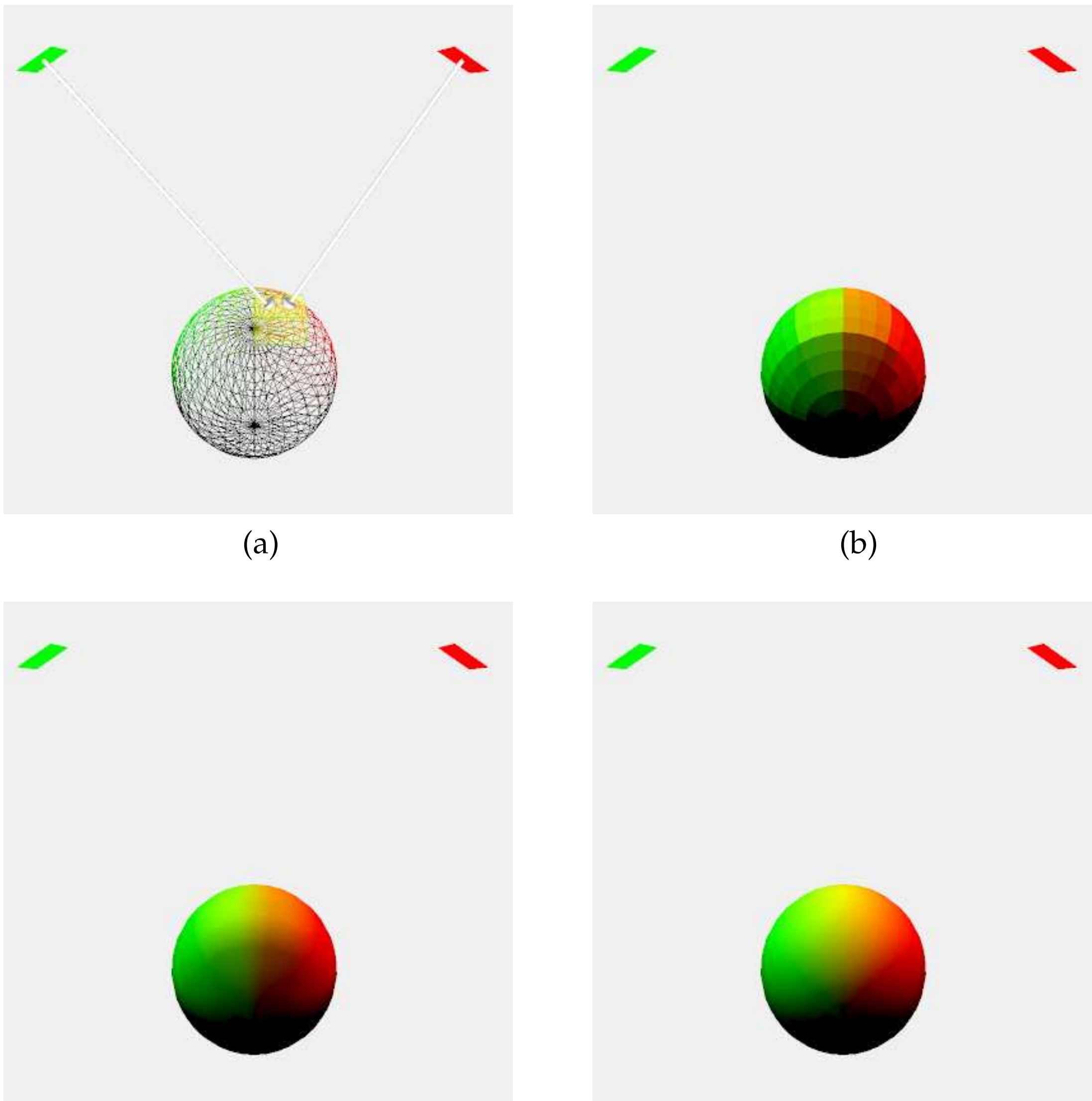


Figure 15: Clustering Artifacts cont. Even when the error threshold is reduced, the clustered solution (f) has aliasing artifacts not present in the reference solution (e).

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-



(a) (b)

(c) (d)

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.

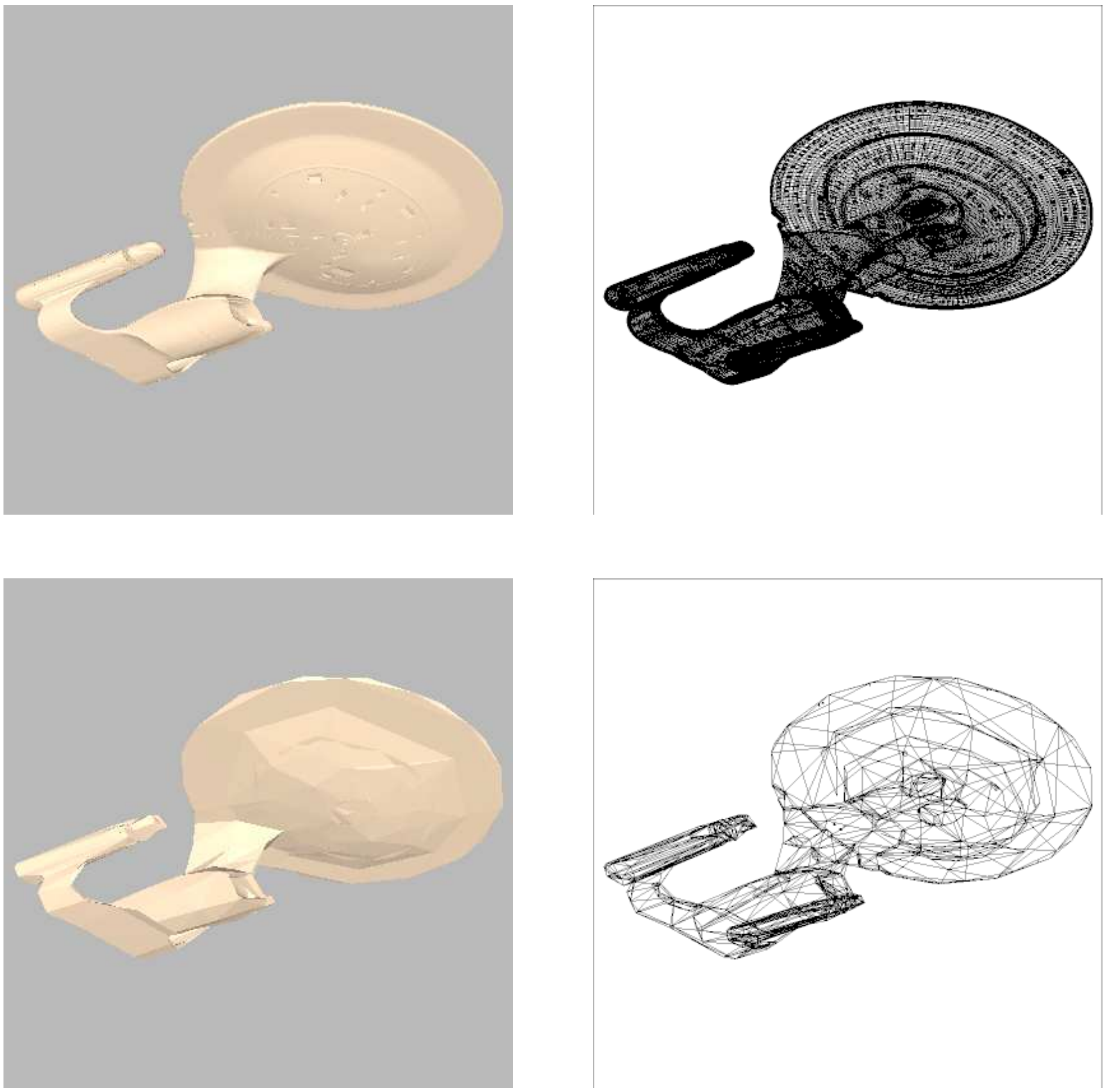


Figure 17: Highly-detailed models. Top: The model of the 'starship enterprise' pictured is composed of 203,878 triangles. The model shown is a medium resolution version of the original model. (Models captured by Cyberware scanner or equivalent device can sometimes reach complexity levels of the order 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 algorithm 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, $s < n$.

Chapter 3

Face Cluster Radiosity

3.1. Introduction

The best hierarchical radiosity methods, using volume clustering, permit scenes of moderate complexity (several hundred thousand input polygons) to be simulated in times ranging from ten minutes to a few hours, depending on the level of accuracy chosen for the simulation, and the geometric arrangement of the scene. Unfortunately, current radiosity techniques, even with clustering, use excessive memory and their speeds are not competitive with other, less realistic rendering methods. We would like to be able to apply radiosity methods to the complex scenes common in special effects. Such scenes routinely use objects each employing 100,000 polygons or more. We therefore seek an enhancement to the hierarchical radiosity algorithm that will permit very complex scenes—scenes with millions of input polygons—to be economically simulated on a standard computer.

One of the greatest difficulties with existing radiosity methods is that their memory use is at least linear in the number of input polygons. This is not a problem if the scene is small, but if the input polygons cannot fit in physical memory, the algorithm will thrash and performance will degrade dramatically. To deal with very complex scenes, we need methods which in practice have memory and time cost that is sub-linear in the number of input polygons. This is possible because for many such scenes, the geometric detail represented by the input polygons is much greater than that needed for the radiosity simulation.

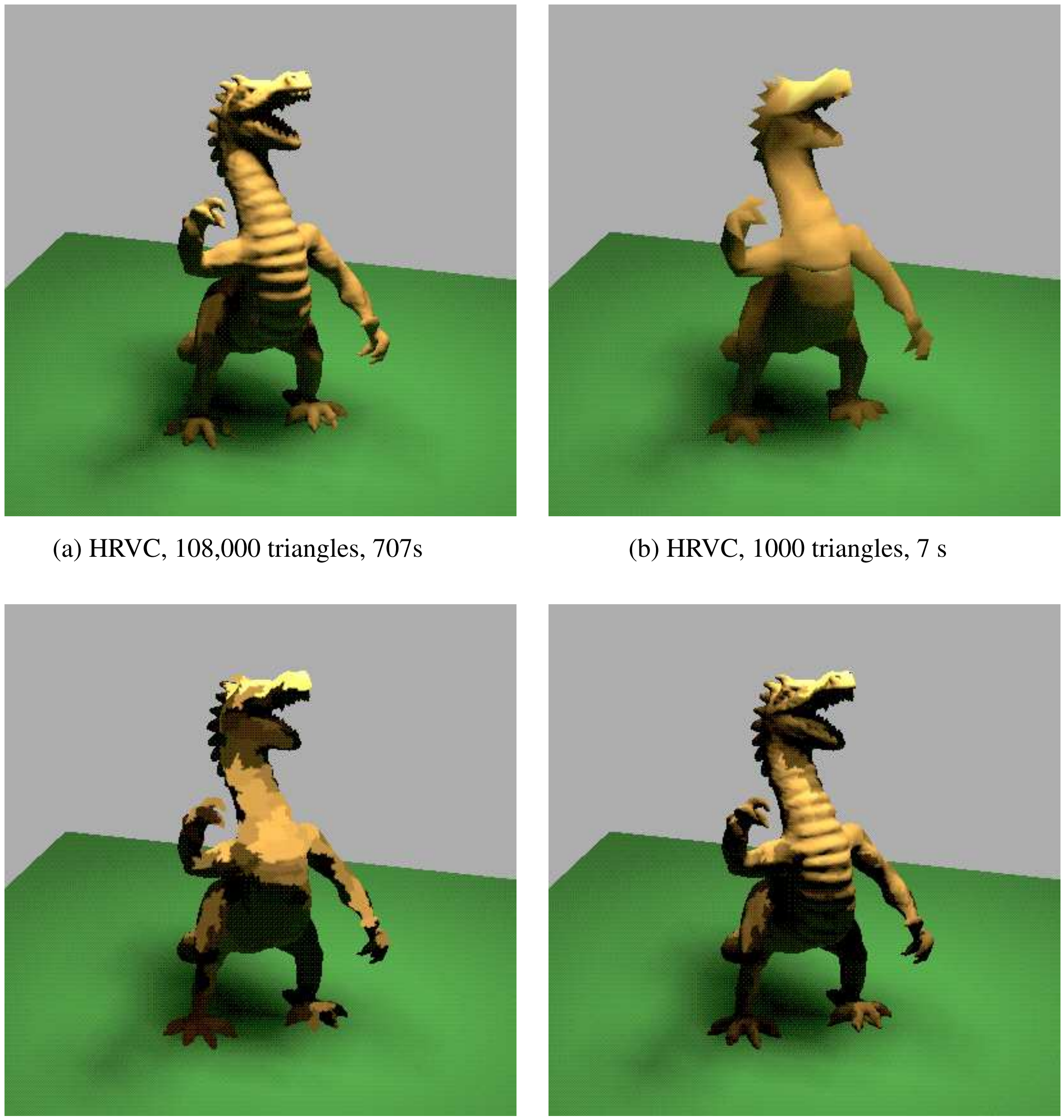
In this chapter I describe the core elements of the face cluster radiosity algorithm, a technique that achieves this goal. This algorithm grew out of research into adapting the radiosity method to use multiresolution models. Later chapters will deal with technical details and some of the assumptions behind the algorithm, but for now, I will focus on its overall structure.

3.1.1. Overview

A preview of the technique is shown visually in **Figure 19**. If one of the best existing radiosity algorithms (hierarchical radiosity with volume clustering) is used on a detailed model, a solution takes over ten minutes (**Figure 19**a)[1]. If, on the other hand, the input geometry is simplified by cutting the number of triangles by a factor of 100, and the same algorithm is applied to the simplified model, a solution can be calculated much more quickly (**Figure 19**b, 7 seconds). This is fast, but the accuracy and visual quality are poor. The face cluster radiosity technique allows a solution not much more expensive than this to be calculated and propagated to the fully detailed model, yielding **Figure 19**d. This is much faster than the full solution and almost as accurate.

The three main phases of the algorithm are preprocessing, solution, and postprocessing. Preprocessing converts the scene description into a multiresolution, hierarchical model. The time cost of this is super-linear in the number of input polygons, but preprocessing can be done on an off-line, object-by-object basis, so its memory costs are modest and its time costs can be amortized over multiple solutions. Next, one or more radiosity solutions are found. This is the costliest step, in practice. The solution phase is sub-linear in cost because it accesses only the coarsest levels of detail from the hierarchy that are necessary. Consequently, often large portions of the hierarchy need never be paged in during this phase, with huge physical memory savings. After solution, postprocessing evaluates the radiosity of the finest details of the scene. This requires linear time. The overall cost, being dominated by solution, is thus sub-linear in practice.

1. These times are highly dependent on accuracy settings, platform and implementation. The times quoted are meant to be illustrative, though care was taken to use similar settings for each algorithm.



(a) HRVC, 108,000 triangles, 707s

(b) HRVC, 1000 triangles, 7 s

(c) Constant FCR, 108,000 triangles, 7s

(d) Vector FCR, 108,000 triangles, 8s

Figure 19: Radiosity on a detailed dragon model. Face cluster radiosity (FCR) and hierarchical radiosity with volume clustering (HRVC) algorithms applied to a detailed dragon model.

3.2. Outline

In this section we present the ideas which lead to the development of the face cluster radiosity algorithm, and a rough sketch of the algorithm itself. We close with a more thorough analysis of the complexity of the new method and its relationship to previous work on hierarchical radiosity algorithms.

3.2.1. Motivation

The current state of the art in radiosity methods, hierarchical radiosity with volume clustering [Smit94], has a complexity that depends, among other things, at least linearly on the number of polygons in the input scene. The work presented in this chapter has a number of motivations, but the primary motivation can be summarised very simply:

> If we increase the number of input polygons in our scene by tessellating existing polygons[1], it should have no impact on the speed, memory use, or results of our radiosity simulation.

That is, we would like the performance of our radiosity algorithm to be *sub-linear* in the number of input polygons. This should hold if we simply tessellate polygons in the scene, but we would also like it to be largely true if we add small-scale detail to surfaces, perhaps by perturbing the tessellations slightly, that is unlikely to affect the broader radiosity computation. In other words, high-resolution detail in a scene should have minimal impact on the radiosity solution time.

This motivation arose from experiments with radiosity simulations on scenes that contained large scanned models; large enough to make the dependence on input polygons of the volume clustering system a problem. The observation that most of the polygons in such models are for high resolution detail, and don't affect radiosity computations much, led me to investigate the use of model simplification in radiosity. Model simplification is a process whereby the polygon count of an existing model is reduced, while trying to keep the same approximate shape as the original model [Heck] (see **Figure 20**). In particular, it lead to the use of multiresolution models [Hopp97].

A multiresolution model allows the resolution of an existing polygonal-mesh model to be lowered by different amounts at different points on that model. It is possible to have one region of the model be heavily simplified, and another

1. Tessellating means replacing an existing polygon with several smaller polygons that cover the same area.

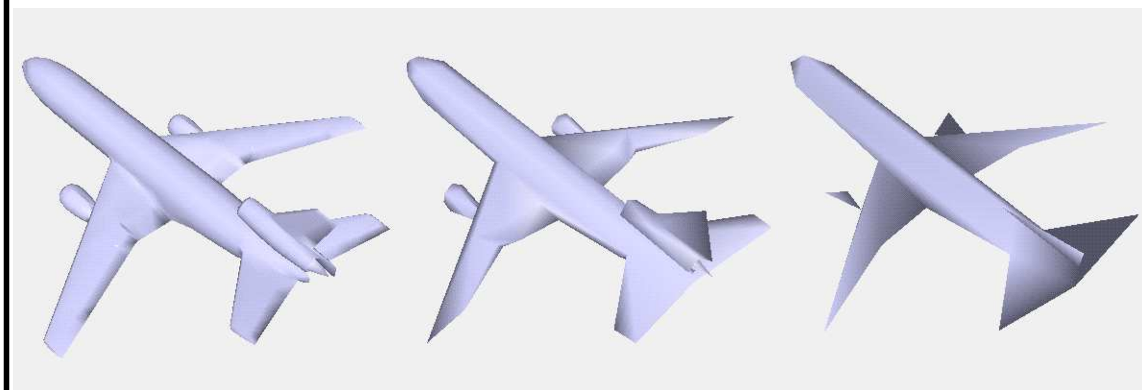


Figure 20: Progressively simplified models of a DC10 aircraft.

region simplified almost not at all, with the intermediate regions having corresponding gradations of resolution. Such models are generated by applying an iterative simplification process to the original model, and generating a corresponding tree of simplifications, with the original model's polygons at the leaves of the tree. Any cut through this tree gives a version of the model with a particular resolution.

Intuitively, we can envisage using such a model to remove the unneeded high-resolution detail of models before running our radiosity algorithm. Instead of running our radiosity simulation on a detailed model (as in **Figure 19**a), we would run it on a simplified model (**Figure 19**b), and then somehow apply the results back to the original model.

A better solution would be to combine both the multiresolution model and radiosity algorithms. By using the multiresolution hierarchies of the models directly in the element hierarchy of the radiosity solver, we could adjust the model resolution "on the fly" to match that needed by the radiosity algorithm. Apart from being more flexible, this would ensure that no manual selection of simplification level was necessary. This is vital, because it is difficult to guess a priori where the models can be simplified, as these decisions are best made using global information. To do a good job, we would need the results of the radiosity simulation we are trying to run.

Traditionally, below the level of the input polygons we add refinements to those polygons to capture any illumination detail that is at a higher resolution than those polygons, as shown in **Figure 21**. If we envisage a radiosity hierarchy where the root of the hierarchy represents the entire scene, and the input polygons lie at some particular level in the hierarchy, refinements can be represented by nodes below that line in the hierarchy.

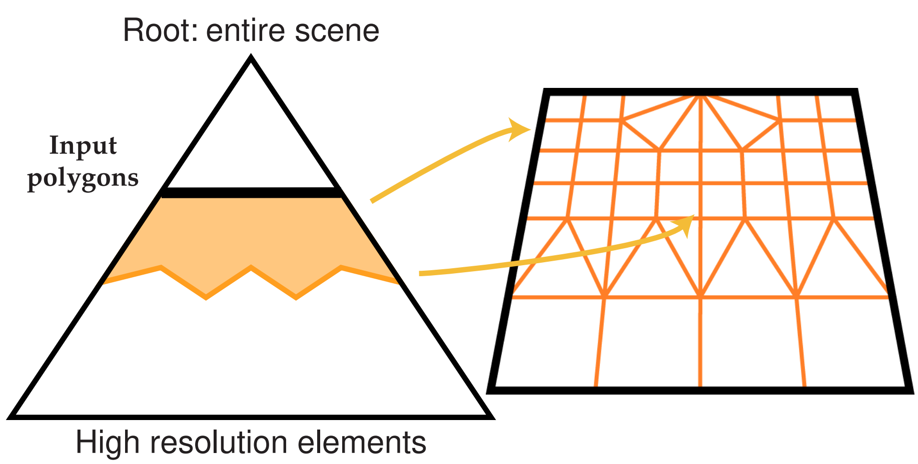


Figure 21: Refinement of polygonal elements in the detail hierarchy

In the same way, **Figure 22** shows how in turn we can use simplification to reduce the resolution of the scene above the input polygons; conceptually, these simplifications make up the nodes above the input polygons in the hierarchy. This also has the benefit of improving our representation of the scene at such levels. Volume clusters, which are traditionally used to represent versions of the scene coarser than the input polygons, are designed to approximate a cloud of unconnected polygons. For the connected, largely smooth surfaces that occur in many models, surface simplification provides a better and more natural approximation.

The particular multiresolution model format we use in our algorithm, as discussed in the next section, is called a face cluster hierarchy. **Figure 23** outlines how these face clusters fit into a traditional hierarchical radiosity with volume clustering algorithm. As indicated, they help bridge the gap between the surface

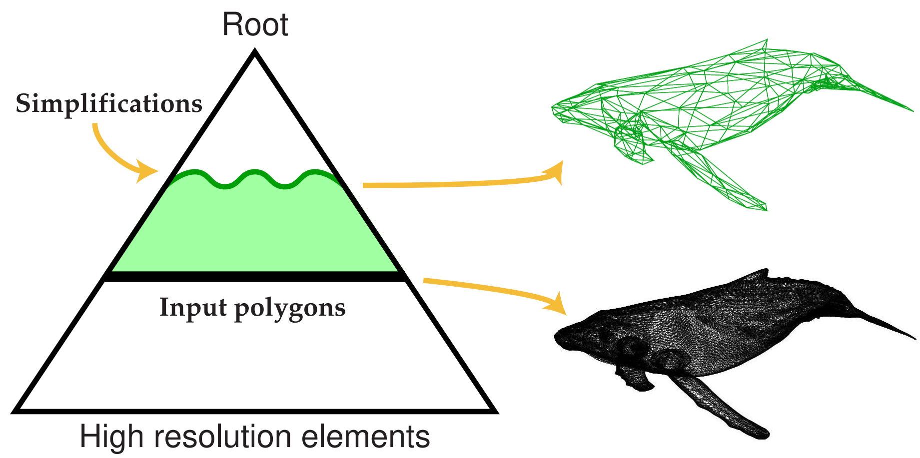


Figure 22: Simplification of polygonal elements in the detail hierarchy

representation of polygons, which are contiguous and have a single orientation, and the representation of volume clusters, which is best at modelling a collection of scattered, disconnected surfaces. Face cluster nodes are intended to model a contiguous, connected chunk of the surface, which is assumed to be largely coplanar, but to have varying surface normals.

With both a vanilla hierarchy, **Figure 23** (a), and with volume clustering (b), the solver must descend to at least the level of the k input polygons. This level gets ever further away from the root (as $\log k$ increases) as the detail of the scene is increased. That is, as we arbitrarily tessellate the scene, the solver must do more work. Once face clusters are introduced in (c), this is no longer the case; the solver must only descend as far as the leaves of the solution hierarchy, which remain at a constant level regardless of the tessellation of the underlying models. This has the further advantage that we need not store radiosity coefficients for all input polygons — only for those face cluster nodes within the solution hierarchy. In a simulation where a detailed model can be safely approximated for the purposes of radiosity computation by a much simpler model, this provides tremendous memory savings.

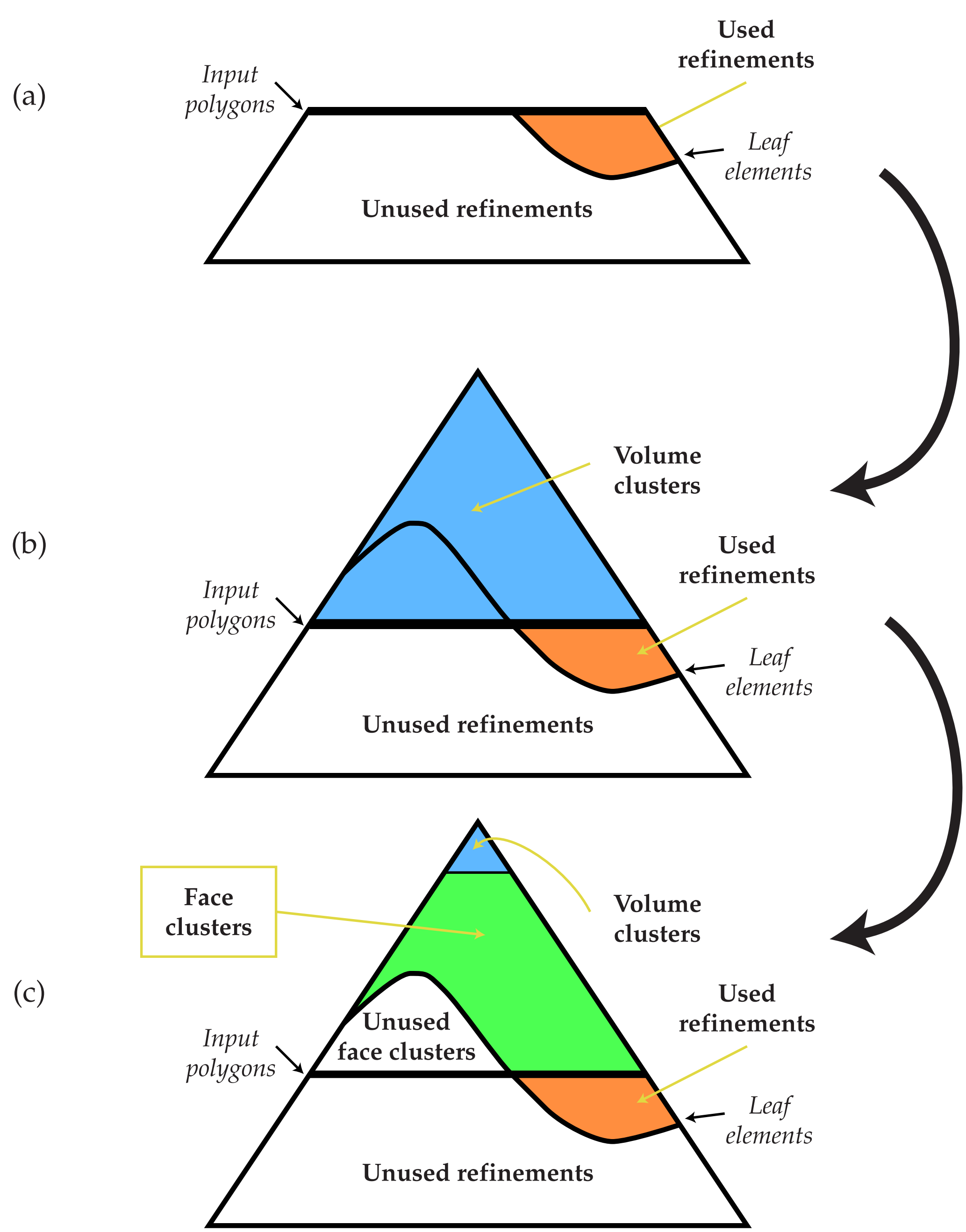


Figure 23: Adding face clusters to the hierarchy. The original polygon-only solution hierarchy (a) can be augmented with in turn (b) volume clusters and (c) face clusters.

3.2.2. Hierarchy Analysis

Figure 24 is a schematic comparison of variants of hierarchical radiosity on a scene with two large polygons A and B in close proximity, and eight small polygons C-J, more distant. Simple hierarchical radiosity (a) yields a forest of quadtrees. Polygons A and B are subdivided and some of their children are linked. The large number of links between the small input polygons C-J makes the algorithm inefficient. **Figure 24**b shows hierarchical radiosity with volume clustering. If cluster Q is sufficiently small and distant from cluster P then a single link between them suffices. Since a cluster can illuminate itself, a self link on Q is necessary as well. The algorithm is still slow because it is necessary to push light down to polygons C-J.

Face Cluster Radiosity. We propose that volume clusters be replaced by multiresolution models for all groups of input polygons that represent a surface. The use of such models allows pushing of light to the leaves to be avoided, and they often provide a better fit to the original surfaces than volume clusters. The particular multiresolution representation that we use, face clustering, groups adjacent faces that have similar normals, and thus can approximate largely planar surfaces well.

With this scheme, the data structures for elements coarser than and finer than the input polygons are more similar, since both face clustering and quadtree refinement yield a contiguous piece of surface with a normal. Hierarchical radiosity is now free to subdivide below the level of the input polygons and to "unsubdivide" above this level. This reduces the hitherto inordinate role of the input polygons, permitting hierarchical radiosity to represent light transport at more natural levels of detail, and to operate more efficiently in complex scenes.

The use of multiresolution models improves the accuracy of our representations and permits our algorithm to avoid touching the lowest portions of the hierarchy during iterations. In **Figure 24**c we see how a tree of simplified models is built above the input polygons. The tree nodes below T and above C-J are now face clusters, not volume clusters, while the highest levels of the tree, above connected objects, are volume clusters. Note that the subtree below T can be paged out during simulation, saving time and memory.

3.2.3. Previous Work

The use of simplified models in radiosity has previously been proposed. Rushmeier et al. demonstrated the feasibility of the concept, but their method employed manually-constructed simplified models, making it impractical for

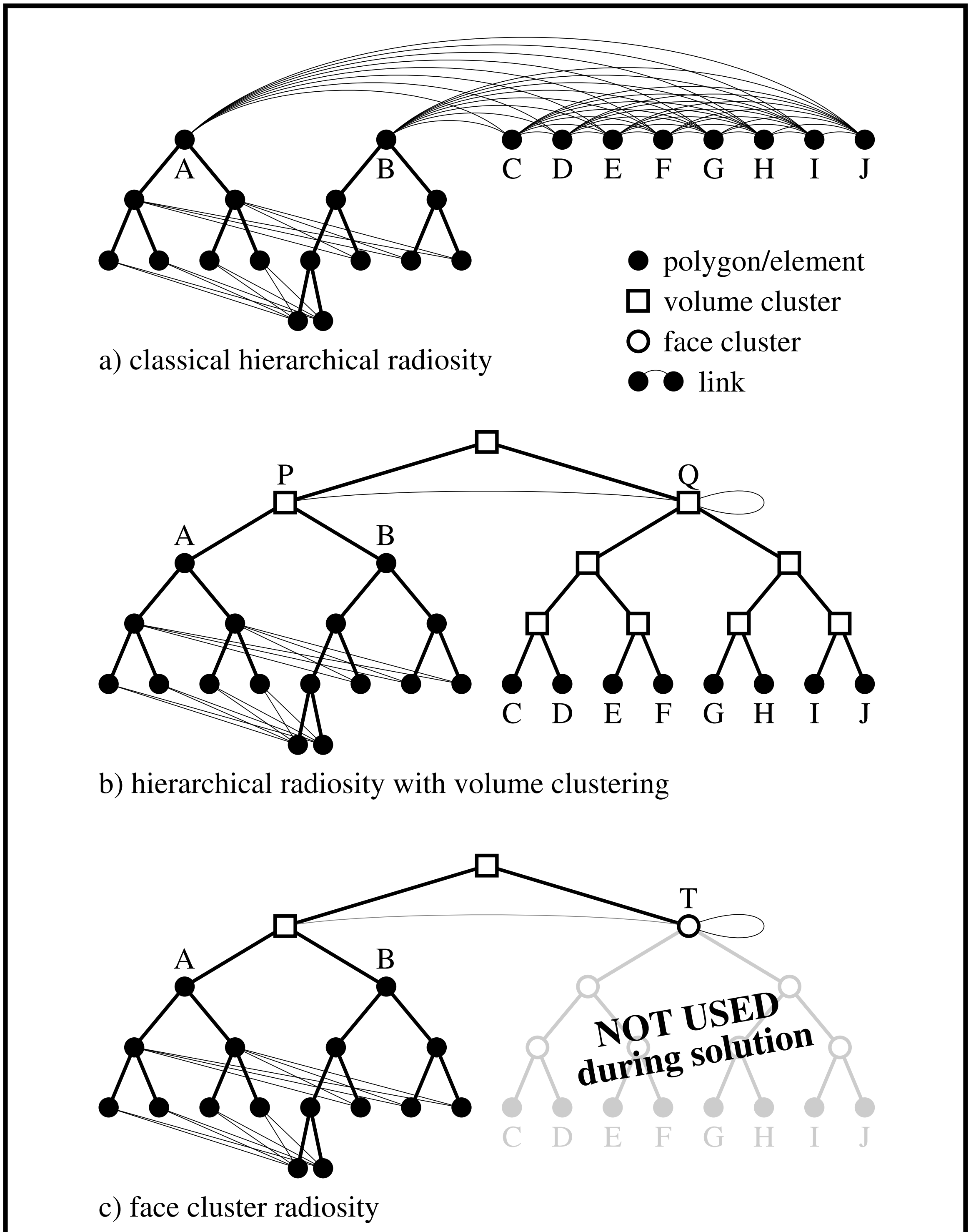


Figure 24: Three approaches to hierarchical radiosity

complex scenes [Rush93]. Greger et al. showed how a radiosity simulation of a simple scene could be applied to a more detailed version of that same scene by using an *irradiance volume* [Greg98]. Both these methods require the user to judge the level of pre-simulation simplification; simplification and simulation are two separate steps, whereas in our algorithm, the level of simplification is driven by the radiosity simulation. That is, the simulation picks the level of simplification appropriate to each transfer of radiosity between solution elements, and multiple levels of simplification exist during the solution.

3.3. Building Surface Hierarchies

3.3.1. Multiresolution Surface Hierarchies

Recent work from the area of surface simplification provided a starting point for our research into radiosity using multiresolution models. Iterative edge contraction, one of the currently popular simplification techniques, can be used to construct a hierarchy of progressively larger vertex neighbourhoods on the surface. Each edge contraction collapses the two vertices at either end of the edge to a single, new vertex. The hierarchy is created by treating the endpoints as the children of this new vertex. These vertex hierarchies have been used primarily for view-dependent refinement of models for real-time rendering [Hopp97, Xia96, Lueb97].

In the original version of the algorithm presented in this dissertation, I used vertex hierarchies directly, as implied by the preceeding section. This sometimes proved problematic, because vertex nodes don't have a well defined area and normal. (These properties are easier to establish for face hierarchies than for vertex hierarchies.)

To address this problem, the initial algorithm treated vertices in the simplification hierarchy as faces on the *dual graph* of the model. That is, conceptually the polygonal mesh was assumed to define the Voronoi diagram of the actual surface used by the radiosity simulation. Both the area and the normal for each vertex were constructed from its surrounding faces; the dual was never explicitly formed.

This approach still had problems, not least of which was that the radiosity solution was being performed on what amounted to a smoothed version of the original model, which made the treatment of creases and sharp edges on the model problematic. Also, without explicitly constructing the dual graph, a step which would have been prohibitive in cost, the vertex nodes did not have a well-defined extent. This made calculating form-factors and good visibility estimates difficult.

At this point the following critical insight presented itself: a more robust approach to the problem would be to use the original faces of the model in the radiosity solution, and instead have the simplification algorithm work on the dual graph of the model.

To meet the need for such an algorithm, Garland developed *face cluster hierarchies* [Garl99], a dual form of the quadric-based simplification algorithm of Garland and Heckbert [Garl97]. (It is also closely related to the "superfaces" simplification algorithm of Kalvin and Taylor [Kalv96].) Rather than iteratively merging pairs of vertices to directly simplify the mesh, the algorithm iteratively merges groups of topologically connected faces, which we refer to as face clusters, thereby partitioning the original mesh (see **Figure 25**). This process does not change the original geometry of the surface in any way; it merely groups surface polygons into progressively larger clusters. These merge operations can be used to create a multiresolution hierarchy in the same manner as edge collapses; an example is shown in **Figure 26**. The resulting *face cluster hierarchy* has the property that any cut across the hierarchy gives a particular partitioning or clustering of the original model. **Figure 27** shows an example hierarchy, and the corresponding clusters on the model for several different cuts.

In Garland and Heckbert's original paper [Garl97], quadric functions of position, $\mathbf{v}^T\mathbf{A}\mathbf{v} + 2\mathbf{b}^T\mathbf{v} + c$, are used to represent the set of planes associated with a vertex node, and can be used to find the best-fit point to those planes. When we are working in the dual space, we instead use quadric functions of the face normal, $\mathbf{n}^T\mathbf{A}\mathbf{n} + 2\mathbf{b}^T\mathbf{n} + c$, to represent the set of vertices in a cluster node, and to find the best-fit plane to those points. Because of this, it can be shown that by applying the quadric error approach to the dual problem, face clusters can be made to preserve planarity where possible, in the same way that vertex simplification tries to preserve shape.

3.3.2. The Face Clustering Algorithm

Much of the material described in the next three sub sections is summarised from Garland's work on surface simplification. Further details can be found in chapter 8 of his dissertation, "Quadric-based Polygonal Surface Simplification" [Garl99]. Further discussion and refinement of the face cluster algorithm can be found in **Chapter 5**.

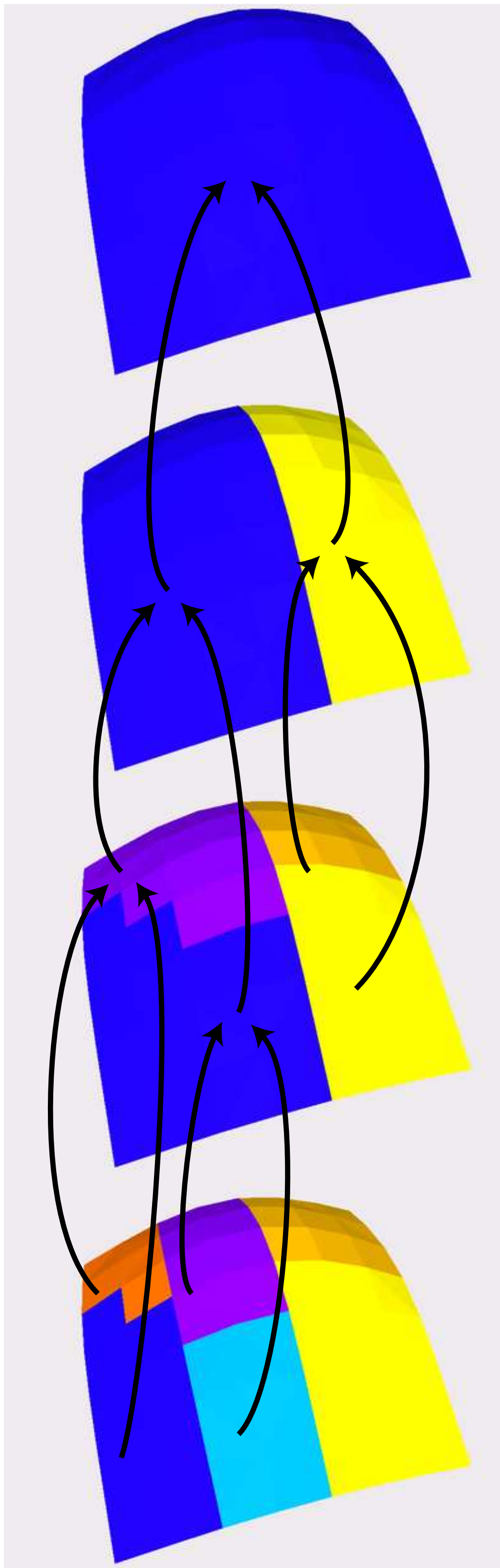


Figure 25: Face clusters. Each face cluster is created by merging two other adjacent face clusters. The several levels of the hierarchy are shown here for a small, curved mesh. At bottom, there are six surface regions, which are iteratively merged pairwise, until we are left with only a single cluster covering the entire surface, at top. This is the root of the cluster hierarchy.

Note how the cluster regions largely follow the curvature of the surface.

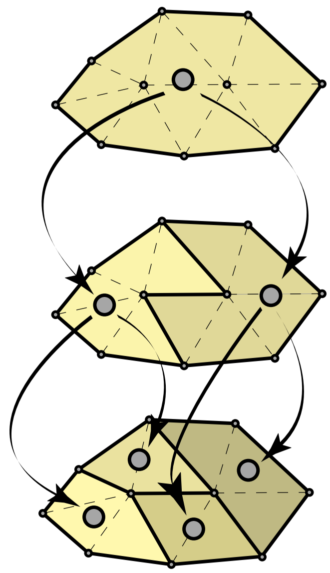


Figure 26: Building a Face Cluster Hierarchy.

Our aim is to automatically construct a hierarchy of surface regions for every connected surface component. Each region must consist of a connected set of faces. We begin by forming the *dual graph* of the input mesh. Every face of the surface is mapped to a node in the dual graph, and two dual nodes are connected by an edge if the corresponding faces are adjacent on the surface. For the moment, we will assume that the surface is a manifold with boundary; in other words, every surface edge has at most 2 adjacent faces. We will also assume that all input polygons have been triangulated.

In the dual graph, an edge collapse corresponds to merging two face clusters together. **Figure 28** illustrates a simple example. On the left is a mesh where each dual node corresponds to a single face; in other words, each face is its own cluster. After collapsing a single dual edge, the two darkened triangles have been merged into a single cluster.

To construct a complete hierarchy, we use a simple greedy procedure to iteratively collapse dual edges. We assign a "cost" to each dual edge and iteratively collapse the edge of least cost. After each collapse, we update the costs of the surrounding edges. At each iteration, we have constructed a partition of the surface into disjoint sets of connected faces. This is in contrast to simplification,

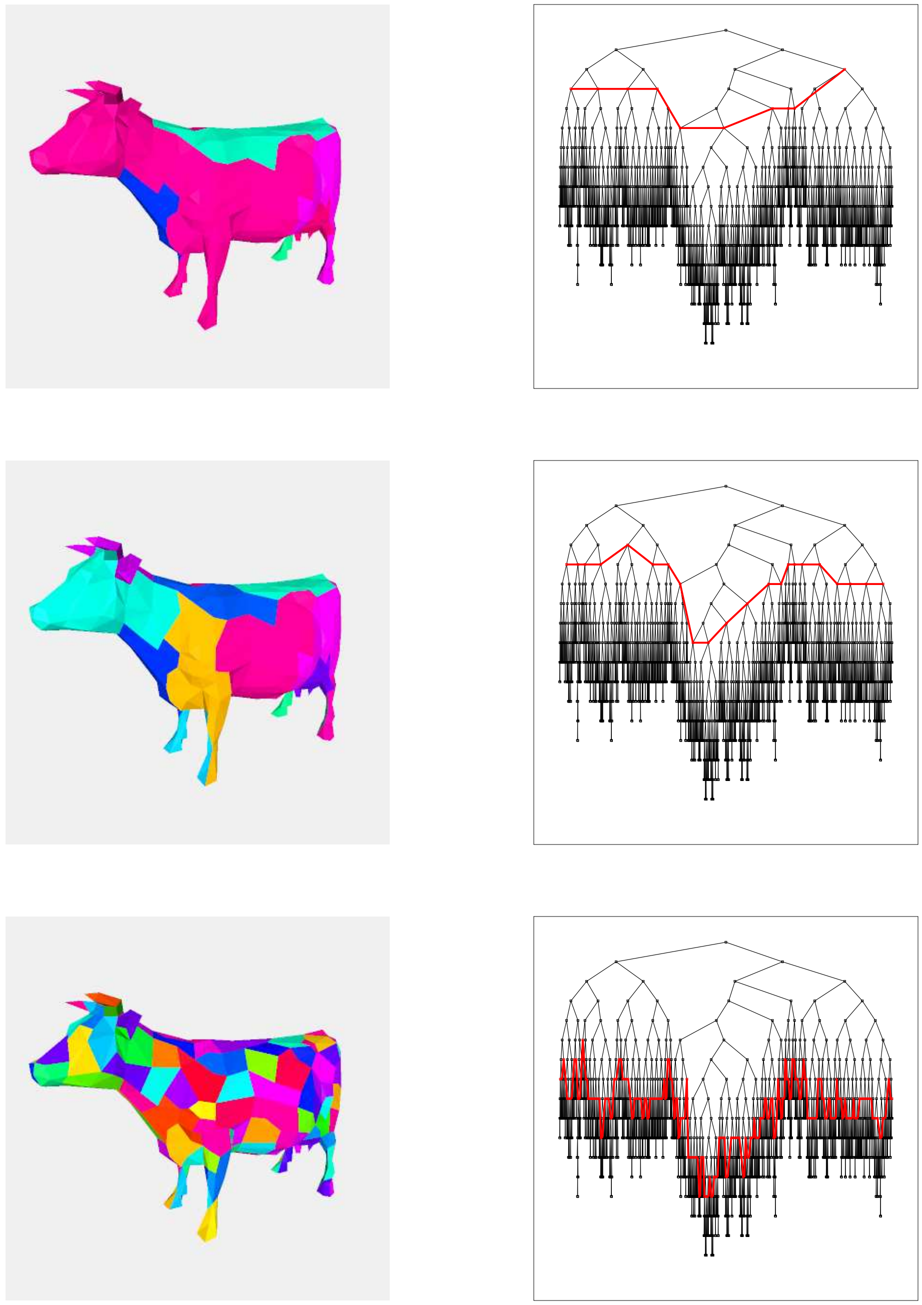


Figure 27: An example face cluster hierarchy. Left: several different clusterings of a cow model. Each cluster has a different colour, and the number of clusters increases from top to bottom. Right: the corresponding cut through the cow's face cluster hierarchy is shown in red.

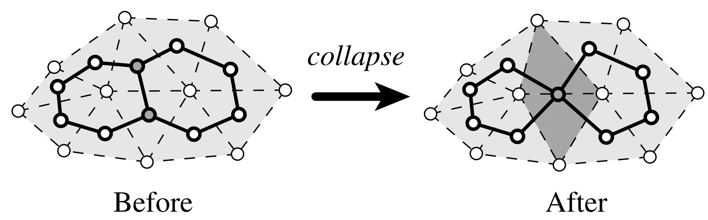


Figure 28: One edge of the dual graph (solid lines) is collapsed. The faces corresponding to its endpoints (darkened) are merged into a single face cluster.

where at each iteration we would have constructed an approximate surface. Face clustering an object with f faces costs $O(f \log f)$ time and $O(f)$ memory.

3.3.3. Dual Edge Cost Metric

Each dual edge corresponds to a set of faces, namely the faces associated with its endpoints, and a set of points $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ determined by the vertices of these faces. We begin by fitting a least squares optimal plane through this set of points using the standard technique of principal component analysis (PCA) [Joll86]. We construct the covariance matrix

$$\mathrm{CV} = \sum_i (\mathbf{v}_i - \bar{\mathbf{v}})(\mathbf{v}_i - \bar{\mathbf{v}})^{\mathrm{T}} \quad \text{where} \quad \bar{\mathbf{v}} = \sum_i \mathbf{v}_i / k \tag{11}$$

The three eigenvectors of the matrix CV determine a local frame with $\bar{\mathbf{v}}$ as the origin. The eigenvector corresponding to the smallest eigenvalue is the normal of the optimal plane, and the remaining eigenvectors define a frame for parameterizing this plane. Given this optimal plane, we assign a cost

$$E = E_{fit} + E_{dir} + E_{shape} \tag{12}$$

to every dual edge. In addition, we use this plane to compute an oriented bounding box for the cluster.

The first, and most important, error term measures the planarity of the cluster, and is defined to be the average squared distance of all the points in the cluster to the optimal plane

$$E_{fit} = \sum_i (\mathbf{n}^{\mathrm{T}}\mathbf{v}_i + d)^2 / k \tag{13}$$

Following the quadric error metric of Garland and Heckbert [Garl98], we can compactly represent this sum using quadrics as

$$E_{fit} = \sum_i P_i(\mathbf{n}, d)/k = \left(\sum_i P_i\right)(\mathbf{n}, d)/k \tag{14}$$

where

$$P_i = (\mathbf{A}_i, \mathbf{b}_i, c_i) = (\mathbf{v}_i\mathbf{v}_i^{\mathrm{T}}, \mathbf{v}_i, 1) \tag{15}$$

$$P(\mathbf{n}, d) = \mathbf{n}^{\mathrm{T}}\mathbf{A}\mathbf{n} + 2\mathbf{b}^{\mathrm{T}}(d\mathbf{n}) + cd^2 \tag{16}$$

Each dual node has an associated fit quadric P which requires ten coefficients to represent the symmetric 3x3 matrix $\mathbf{A}$, the 3-vectors $\mathbf{b}$, and the scalar c. It is important to note that the covariance matrix can be extracted directly from this quadric: $\mathrm{CV} = \mathbf{A} - (\mathbf{b}\mathbf{b}^{\mathrm{T}})/c$.

Minimizing the planarity term E_{fit} will naturally tend to merge clusters which are collectively nearly planar. However, a surface may locally fold back on itself; it will seem nearly planar, but the normal of the plane will not be a good fit for the surface normals in the region. Therefore, we add an additional error term which measures the area-averaged deviation of the plane normal $\mathbf{n}$ from the surface normals

$$E_{dir} = \left(\sum_i A_i(1 - \mathbf{n}^{\mathrm{T}}\mathbf{n}_i)^2\right) / \sum_i A_i \tag{17}$$

where A_i and $\mathbf{n}_i$ are the area and unit normal, respectively, of a face i in the cluster. Again, we can represent this error term using a quadric by rewriting $(1 - \mathbf{n}^{\mathrm{T}}\mathbf{n}_i)^2$ as $R_i(n) = \mathbf{n}^{\mathrm{T}}\mathbf{D}_i\mathbf{n} + 2\mathbf{e}_i^{\mathrm{T}}\mathbf{n} + f_i$ where

$$R_i = (\mathbf{D}_i, \mathbf{e}_i, f_i) = (\mathbf{n}_i\mathbf{n}_i^{\mathrm{T}}, -\mathbf{n}_i, 1) \tag{18}$$

Every dual node has two associated quadrics, a fit quadric and an orientation quadric. Whenever we merge two nodes, we add their corresponding quadrics together to form the quadrics for the resulting node.

3.3.4. Compact Shape Bias

If planarity is our only clustering criterion, then the combination of the error terms E_{fit} and E_{dir} performs well. However, for our radiosity application, we would also like the regions to have a compact shape; a compact region is one which is as nearly circular as possible. Following Kalvin and Taylor [Kalv96], we define the compactness γ of a region to be the ratio of its squared perimeter to its area. Larger values of γ correspond to less compact (more irregular) regions.

For a given dual edge, the clusters associated with its endpoints will have compactness γ_1 and γ_2. If γ is the compactness of the resulting merged cluster, we define the shape penalty as:

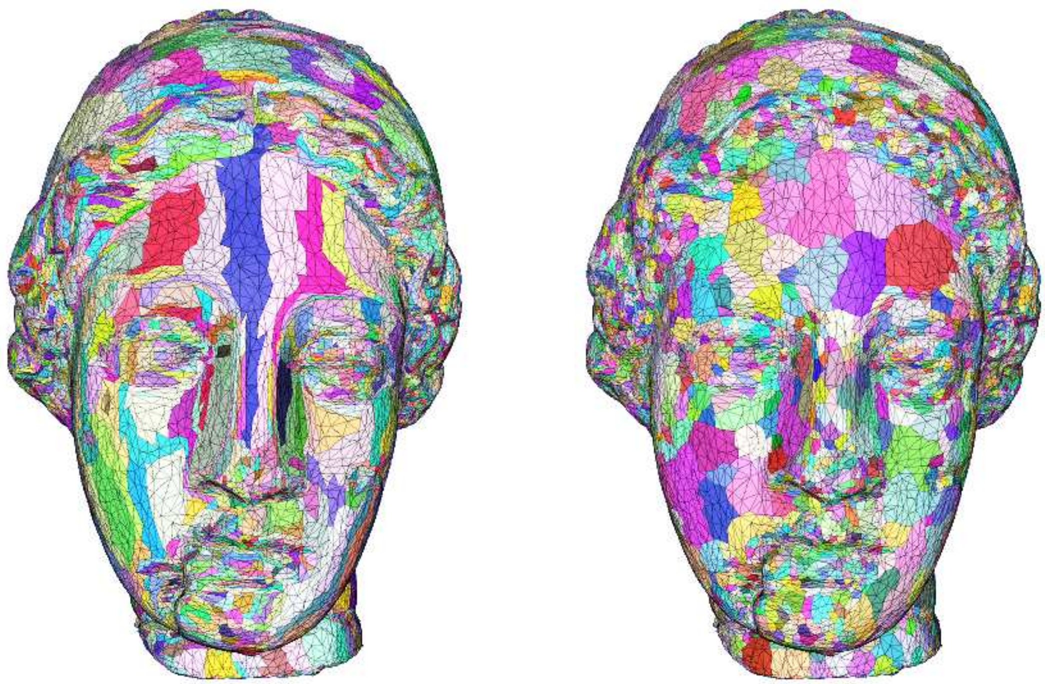
$$E_{shape} = 1 - \max(\gamma_1, \gamma_2)/2\gamma \tag{19}$$

This penalizes clusters where the compactness worsens and rewards clusters where the compactness improves. **Figure 29** illustrates the effect of the shape bias. With the shape bias disabled (a), the clusters tend to follow the features of the model but may have non-compact shapes. The long strips stretching down the forehead and the nose are a good example. In contrast, when using the shape bias (b), the clusters still reflect the form of the surface, but they have a much more compact shape.

3.3.5. A Face Cluster Node

Each node in our face cluster hierarchy is a representation of an approximately coplanar region on the mesh. It acts as a container for a set of connected faces (see **Figure 30**). Once we have constructed the hierarchy, we must address the question of what representative surface properties to store for each node within it, in addition to the pointers to the two child face clusters that partition it. A balance must be struck between the amount of useful information we store, and the amount of memory or disk space the hierarchy will occupy.

As discussed, for each node *i* in the face cluster hierarchy, we calculate an oriented bounding box for the faces it contains using principal component analy-



(a) No shape bias (b) With shape bias

Figure 29: Partitions of the venus head, each with 8000 clusters

sis [Joll86]. For each node, we store this, in addition to the sum of the area-weighted normals of those faces:

$$\mathbf{S} = \sum_j A_j \hat{\mathbf{n}}_j \tag{20}$$

for each face j belonging to the cluster. As we will show in **Section 3.4** and in more detail in **Chapter 4**, this is a useful approximation of the reflective qualities of the faces within the node.

3.4. The Face Cluster Radiosity Algorithm

In this section we describe our algorithm for simulating radiosity on multiresolution models that use face clustering. We show how the standard radiosity equa-

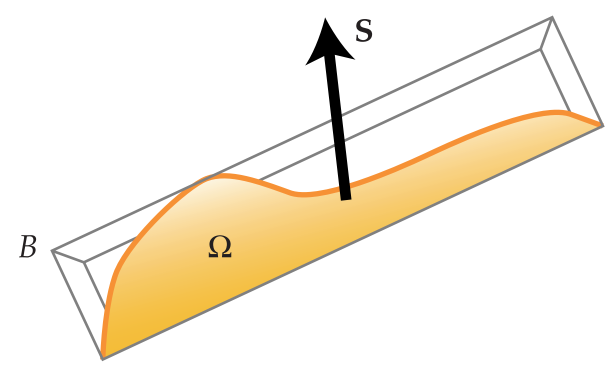


Figure 30: A face cluster. For our purposes, a face cluster is defined by a group of connected faces Ω, with an enclosing oriented bounding box B, and an area-weighted normal, $\mathbf{S}$.

tions can be modified to better suit the use of these hierarchies, starting with the standard formulas governing diffuse interreflection [Cohe93].

3.4.1. Standard Radiosity Derivation

The radiosity b_i of surface i is the outgoing power per unit area due to emission or reflection. It equals emittance e_i plus reflectance ρ_i times the sum of contributions from other surfaces j,

$$b_i = e_i + \rho_i \sum_j F_{ij} b_j . \tag{21}$$

Here F_{ij} is the form factor, the fraction of light leaving surface j that arrives at surface i,

$$F_{ij} = \frac{1}{A_i} \int\int \frac{\cos\theta_i \cos\theta_j}{\pi r_{ij}^2} v_{ij} dA_i dA_j , \tag{22}$$

and A_i is the area of element i, θ_i and θ_j are the polar angles, r_{ij} is the distance between points on elements i and j, and v_{ij} is the visibility between those points: 1 if visible and 0 if occluded.

To solve the above system of equations, our method, like most hierarchical radiosity algorithms, repeatedly gathers light from all other elements *j* to update the radiosity of element *i*. This is equivalent to one iteration of Jacobi's method for solving linear systems. Multiplying the equation above by A_i, we get an equation whose terms have units of power (area times radiosity):

$$A_i b_i \;=\; A_i e_i + \rho_i A_i E_i \tag{23}$$

The *irradiance* E_i is the incident power per unit area. Calculating it exactly is typically intractable, so we approximate it using the point-to-point approximation of the form factor [Cohe93],

$$E_i \;=\; \sum_j \frac{(\hat{\mathbf{n}}_i^{\mathrm{T}} \hat{\mathbf{r}}_{ij})_+ (\hat{\mathbf{r}}_{ji}^{\mathrm{T}} \hat{\mathbf{n}}_j)_+}{\pi r_{ij}^2} v_{ij} A_j b_j \tag{24}$$

where bold symbols denote vectors, $(x)_+ \equiv max(0, x)$, $\hat{\mathbf{n}}_i$ is a unit normal to surface *i*, and $\hat{\mathbf{r}}_{ij}$ is the unit direction vector from *i* to *j*. We have written the dot products in matrix notation because we will shortly be exploiting the associativity of matrix multiplication to rewrite this formula.

3.4.2. Vector-based Radiosity

The classical radiosity method assumes piecewise constant (Haar) basis functions and planar surfaces. In a hierarchy, the children are coplanar with the parent. For the purposes of projecting radiosities up and down the tree, radiosities are scalar quantities. If this method is applied to a multiresolution model, it causes curved or bumpy portions of the model to be shaded a flat colour, leading to a faceted appearance that hides the geometric detail (**Figure 19**c). This is similar to the step-function effect in constant-basis radiosity, but applying a post-process smoothing step at the leaves is no longer sufficient to cover up these discontinuities.

We now adapt the radiosity method to multiresolution models that use face clustering. Consider the light transfer from one cluster to another (**Figure 31**). We let *j* be an element in the source cluster and *i* be an element in the receiver cluster. If we assume that all (*i,j*) pairs are inter-visible and that the sources are close together and far from the receiver, then $\hat{\mathbf{r}}_{ij}$, $\hat{\mathbf{r}}_{ji}$, r_{ij}, and $\bar{v}_{ij}$ are independent of i and j, and we can approximate the irradiance from a single cluster as:

$$E_i \approx \left(\hat{\mathbf{n}}_i^{\mathrm{T}} \left[\frac{-\hat{\mathbf{r}}}{\pi r^2} \left(\hat{\mathbf{r}}^{\mathrm{T}} \sum\nolimits_j \hat{\mathbf{n}}_j A_j b_j \right)_+ \right] \bar{v}_i \right)_+ , \tag{25}$$

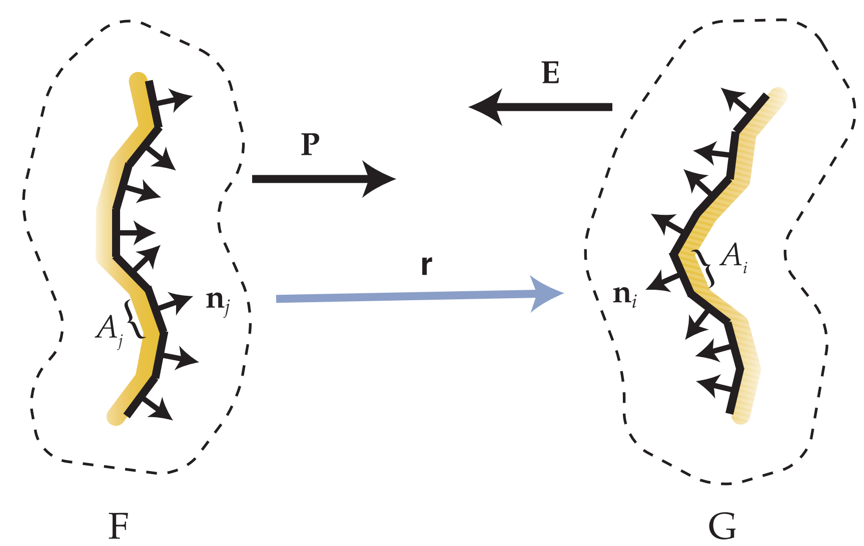


Figure 31: The transfer of radiosity between two clusters of faces. An individual face j in the source cluster F radiates energy towards an individual face i in the receiving cluster G.

where $\mathbf{r}$ is the average distance vector. Using **Equation 19** we can then rewrite the transfer in terms of two vector quantities. We find that:

$$E_i = \hat{\mathbf{n}}_i^{\mathrm{T}} \mathbf{E}\,, \tag{26}$$

where the *irradiance vector* $\mathbf{E}$ is defined as

$$\mathbf{E} = \bar{v}\frac{-\delta(\mathbf{S}_G, \hat{\mathbf{r}})}{\pi r^2}(\hat{\mathbf{r}}^{\mathrm{T}}\mathbf{P})_+\,, \tag{27}$$

with

$$\delta(\mathbf{S}, \mathbf{x}) \equiv \begin{cases} \mathbf{x} & \text{if } \mathbf{x} \cdot \mathbf{S} > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{28}$$

and $\mathbf{S}_G$ being the sum area normal of the receiving cluster. (The irradiance vector has also been employed by Arvo [Arvo94].)

The *power vector* $\mathbf{P}$ is defined as:

$$\mathbf{P} = \sum_j \hat{\mathbf{n}}_j A_j b_j = \sum_j \mathbf{S}_j b_j. \tag{29}$$

If the source cluster has constant radiosity b across its constituent faces, we can simply write:

$$\mathbf{P} = \mathbf{S}_F b, \tag{30}$$

where $\mathbf{S}_F$ is the sum area normal of the source cluster.

The irradiance vector is parallel to $\mathbf{r}$, and is at a maximum when the power vector is also parallel to $\mathbf{r}$; it points towards the source cluster. Recording this information, rather than the scalar irradiance to the average plane of the receiver, allows coarse variations in the irradiance as a function of orientation to be modelled. This eliminates most of the faceting effects of **Figure 19**c, as seen in **Figure 19**d.

We employ the power vector to permit non-planar clusters to approximate their outgoing power compactly and quickly. The magnitude of the vector approximates the total power leaving the cluster, and the direction of the vector indicates the hemisphere toward which most of the energy is directed (**Figure 32**).

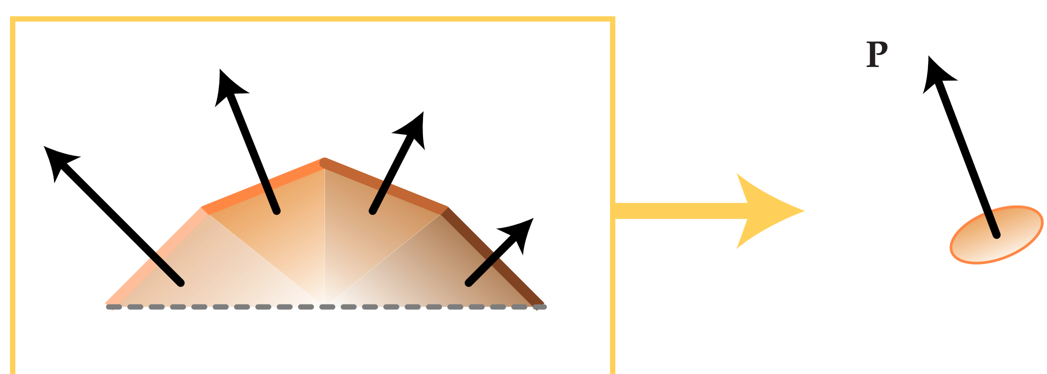


Figure 32: The power vector. The radiosity emitted by the surfaces on the left is approximated by the elemental surface on the right, whose direction is the power-weighted sum normal.

When calculating the illumination of a cluster by one or more other clusters, we transform each power vector into the corresponding irradiance vector via **Equation 27**, and sum the resulting irradiance vectors.

Push Pull

These formulas are generalizations of the standard radiosity equations. In the case of coplanar clusters, they reduce to the familiar hierarchical radiosity push-pull formulas. In the general case, for the push operation, we find that

$$\mathbf{E}_{child} = \delta(\mathbf{S}_{child}, \mathbf{E}_{parent}) + \mathbf{R}_{child}, \tag{31}$$

where $\mathbf{R}_{child}$ is the sum of the irradiance vectors incident directly on the child. For the pull operation, there are two alternatives. We can operate directly on the power vector, so that

$$\mathbf{P}_{parent} = \sum \mathbf{P}_{child}, \tag{32}$$

or we can instead operate on cluster radiosities, with

$$b_{parent} = \frac{\sum b_{child} A_{child}}{A_{parent}}, \tag{33}$$

and reconstruct the power vector with $\mathbf{P}_{parent} = \mathbf{S}_{parent} b_{parent}$, as in **Equation 30**. The former has the advantage of elegance, and the ability to better match the chief direction of radiance of a cluster whose radiosity varies considerably over its surface. The latter has the advantage that it is more amenable to error analysis and the construction of bounds, as we shall see in **Chapter 4**.

Transfer Coefficients

To model the transfer of radiosity, instead of a single transfer coefficient, we store a *transfer vector* $\mathbf{m}$, which allows us to apply **Equation 27**. Where we are working directly with power vectors, as in **Equation 32**, we can write

$$\mathbf{m} = \hat{\mathbf{r}} \sqrt{\frac{\bar{v}}{\pi r^2}}, \tag{34}$$

and we have $\mathbf{E} = \delta(\mathbf{S}_G, -\mathbf{m})(\mathbf{m}^{\mathrm{T}}\mathbf{P})_+$. Where we are working with source radiosities, we can instead use

$$\mathbf{m} = \frac{\bar{v}\delta(\mathbf{S}_G, -\hat{\mathbf{r}})(\hat{\mathbf{r}}^{\mathrm{T}} \cdot \mathbf{S}_F)_+}{\pi r^2}, \tag{35}$$

and $\mathbf{E} = \mathbf{m}b$. Currently, this latter approach is more desirable, as we have a supporting error analysis for it. The former may eventually be preferable if a matching analysis can be found.

Generally, it is easiest to estimate $\mathbf{m}$ by generating n fixed sample points across both the source and receiver clusters, and taking

$$\overline{\mathbf{m}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{m}_i \tag{36}$$

where each $\mathbf{m}_i$ is calculated using $\mathbf{r}_i$, the vector from sample point i on the source to sample point i on the receiver. These samples can also be used for determining bounds on the transfer, and thus an estimate of the error in the transfer, which can be used to drive refinement [Gibs96, Lisc94]. However, I have developed a more accurate and robust approach that relies on conservative bounds for a cluster's projected area in a given direction. An in-depth treatment of this error estimate will be presented in **Section 4.4**.

Leaf Radiosities

Finally, at the leaves of the hierarchy, where we must transform the accumulated irradiance vectors into radiosity, we apply the equation

$$b = \frac{\rho\mathbf{S}^{\mathrm{T}}\mathbf{E}}{A} + e. \tag{37}$$

Again, we can reconstruct a leaf's power vector from this by application of **Equation 30**[1].

This treatment of vector-based radiosity transfer assumes a monochromatic world. It can easily be extended to the familiar RGB colour model; we simply maintain $\mathbf{E}$ and $\mathbf{P}$ or b separately for each colour channel, and operate on each

1. In **Section 4.4.5** we will show how to trivially rewrite **Equation 37** to take account of interreflection within the leaf cluster.

pair independently. Note that the transport vector, $\mathbf{m}$, is still wavelength independent.

3.4.3. Algorithm Description

There are three types of nodes in the hierarchy used by our algorithm. At the top, volume clusters contain all unconnected parts of the scene. In the middle, face clusters contain connected surface meshes. At the bottom, there are polygonal elements, and refinements of those elements. In our implementation, all of these nodes use a common object-oriented interface to communicate with each other. Usually much of each face cluster hierarchy remains unused; only those face clusters at the top of the tree are paged in during the solution phase.

Radiosity using vector-based transfer proceeds in much the same manner as the irradiance/radiosity method first popularised by [Gers94], and outlined in **Figure 33**. In Gershbein's method, irradiance is gathered to each node in the hierarchy, and then pushed down to the leaves, whereupon it is converted to radiosity by the application of reflectance and emittance operators, and pulled back up the hierarchy. In our algorithm, the irradiance vector, rather than scalar, is pushed to the solution leaves, and either the power vector or scalar radiosity is pulled back up the hierarchy. (Here "solution leaves" refers to the lowest level of cluster or face refinement in the hierarchy, not the input polygons.) Below is an outline of the algorithm.

Preprocessing

Face cluster hierarchies are generated for the input models. These hierarchies are dependent only on the geometry of the models, and can be reused over multiple instantiations of each model. This process is done off-line, and typically only once, whenever a new model is acquired.

Initialisation

The scene description is read in. Hierarchical radiosity elements are then created for all root face cluster nodes in the scene (there are typically a small number per model), and these are volume clustered to complete the initial element hierarchy.

Solution

The solver proceeds according to the pseudocode in **Listing 1**. The `refine` procedure follows that outlined in [Cohe93]. When a face cluster element needs to be subdivided, its two child elements are created using position and sum area nor-

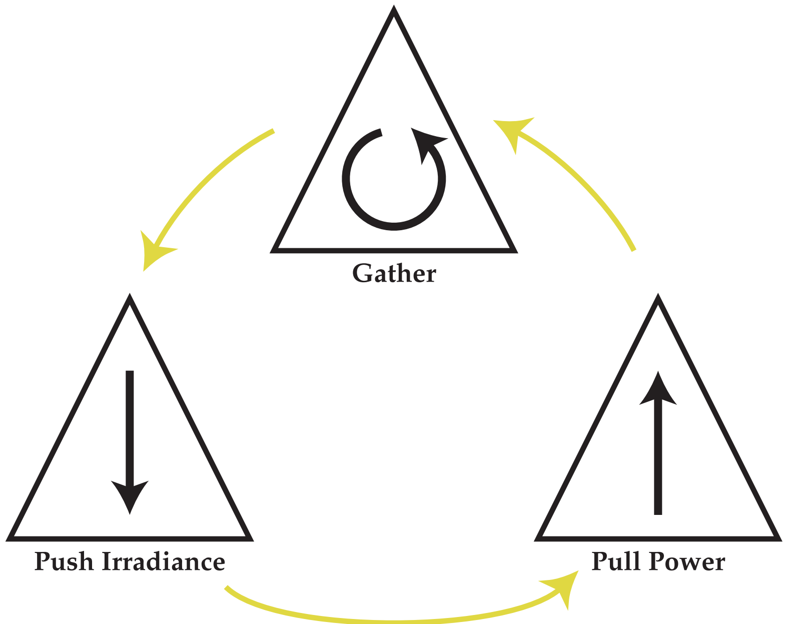


Figure 33: Schematic of the Hierarchical Radiosity Cycle. The current set of element radiosities are *gathered* across transfer links to find the corresponding irradiances at different levels of the solution hierarchy. These irradiances are then *pushed* to the leaves of the hierarchy to find the sum irradiance there, transformed to radiosities by application of the reflection operator at each leaf, and then *pulled* back up the hierarchy to calculate the average radiosity at each level of detail.

mal information retrieved from the cluster file on disk. When the input polygons are reached, the children become Haar elements, and refinement proceeds as in a standard hierarchical radiosity algorithm. Storage for irradiance vectors and other such element information is only required for those face cluster nodes actually being used by the solver. The solver runs in time and space sublinear in k.

Post Processing

After the solution algorithm has terminated, the radiosity solution is propagated to the leaves of the model by applying the irradiance vectors at the leaves of the transport tree to all their descendents (e.g., node T of **Figure 24**c). This final proc-

```
solve(tree root)
     while (not converged)
          gather(root)
          pushpull(root, 0)
          refine(root, ε)

gather(element i)
```

$$\Delta\mathbf{E}_i = -\sum_{\text{links j->i}} \bar{v}_{ij}\mathbf{m}_{ij}\mathbf{m}_{ij}^{\mathrm{T}}\mathbf{P}_j$$

```
     foreach (child c of i)
          gather(c)

pushpull(element i, vector E)
     // E is irradiance on i from parent
```

$$\mathbf{E} = \mathbf{E} + \Delta\mathbf{E}_i$$

```
     if (i is a leaf)
          // convert irradiance to power
```

$$\mathbf{P}_i = \mathbf{S}_i(\mathrm{Max}(\hat{\mathbf{S}}_i^{\mathrm{T}}\mathbf{E}_i, 0)\rho_i + e_i)$$

```
     else
```

$$\mathbf{P}_i = 0$$

```
          // push irradiance, pull power
          foreach (child c of i)
               pushpull(c, E)
```

$$\mathbf{P}_i = \mathbf{P}_i + \mathbf{P}_c$$

Listing 1: Pseudo-code for the Face Cluster Radiosity algorithm

ess is $\Theta(k)$, but is typically insignificant compared to the solution time. The radiosities of the vertices of the models in the scene are then written out to disk.

3.4.4. Examples

Detailed results of the face cluster radiosity algorithm will be presented in **Chapter 7**. In the meantime, here are some informal examples of the algorithm in action. **Figure 34** shows results for a simple scene similar to that of **Figure 14**. In this scene, however, the surface is bumpy, containing 13,000 triangles. Whereas volume clustering takes four minutes to calculate the illumination of the surface, face cluster radiosity takes just over a second. The use of surface-based clusters also helps avoid the artifacts present in the volume clustering solution. These dark patches are caused when the crest of a particular bump belongs to a volume cluster that lies above the cluster containing the rest of the bump. This crest cluster then shadows the cluster underneath, causing the rest of the bump to be underlit. Face clusters explicitly follow the shape of the mesh, avoiding this kind of overlap problem.

Figure 35 shows how a clustered object can also be used as a light source; a tessellated torus is used to illuminate a bumpy surface. Ray tracing methods such as Radiance have particular problems with such large distributed light sources; they must distribute samples over the light source whenever its contribution is being calculated, which requires some knowledge of its geometry. A common approach is to simply sample over the bounding box of the light, which in the case of the torus would result in a large number of samples missing the light source.

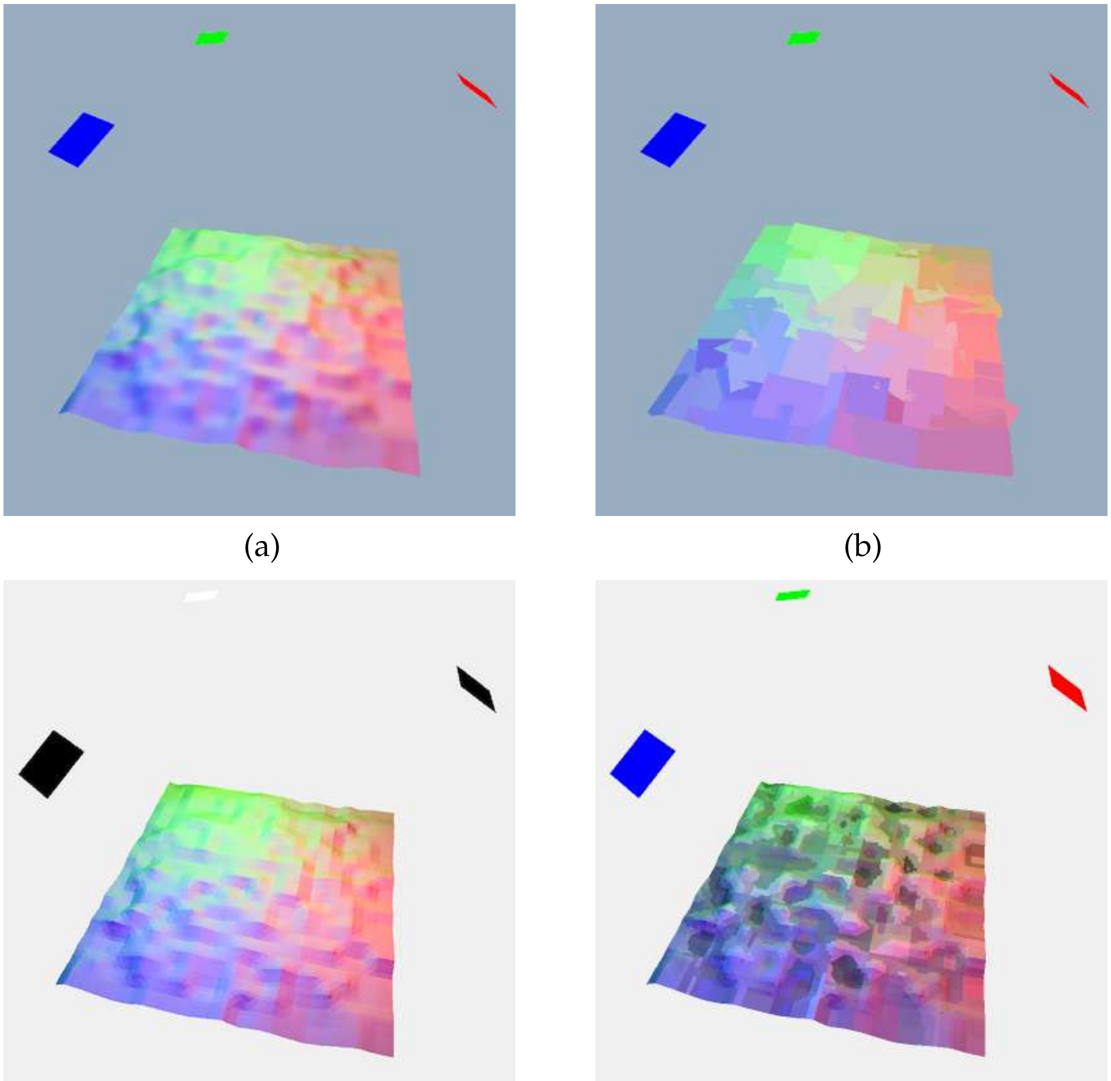
Finally, **Figure 36** shows a scene containing a bust lit from two different directions, along with details of the illumination transfer.

3.5. Discussion

3.5.1. Strengths and Weaknesses

Our algorithm gives the greatest benefit for finely tessellated objects. For more intricate, space-filling objects such as trees, we rely on volume clustering to provide a good simulation. This is because of the limitation in our algorithm that each topologically connected surface has its own face cluster tree. For instance, a pile of pebbles might need a separate tree for each pebble, if those pebbles were modelled separately. While we currently aggregate disconnected components by using volume clustering, better methods for tightly packed components such as the pebble pile are an avenue for future research.

From any given viewpoint, some of the input polygons in a scene containing large models can be invisibly small. Arguably a similar picture from that viewpoint could be generated using a much simpler scene. However, this ignores the possibility of using an output mesh in a walk-through, where we might wish to pass much closer to some of the models in the scene. Also, it is tedious to man-



(a) (b)

(c) (d)

Figure 34: Face-cluster radiosity on a bumpy surface. In (a) face-clustering handles bumpy surfaces as well as a flat mesh. (b) shows the corresponding clusters. This solution took under two seconds to generate. As reference, we have both (c), the results of running Radiance (288s) and (d), volume clustering (260s) on the same scene.

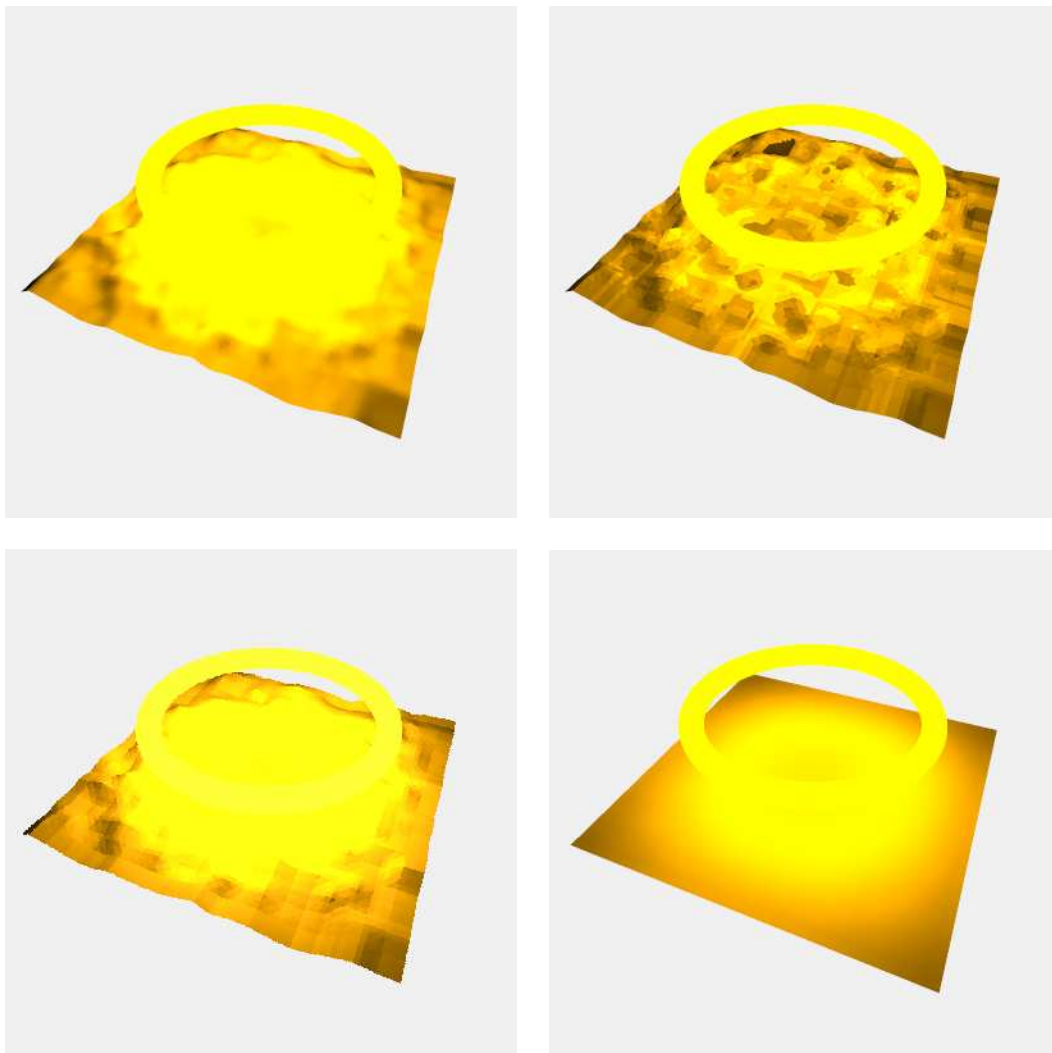


Figure 35: A highly tessellated, curved light source. The light source is a torus containing 8,200 polygons. Top left; face cluster radiosity took 28 seconds to calculate this (view-independent) solution. In contrast, volume clustering (top right) took well over an hour to calculate the corresponding image, and Radiance (bottom-right) took five hours. A solution for a flat surface is shown for comparison purposes, bottom right.

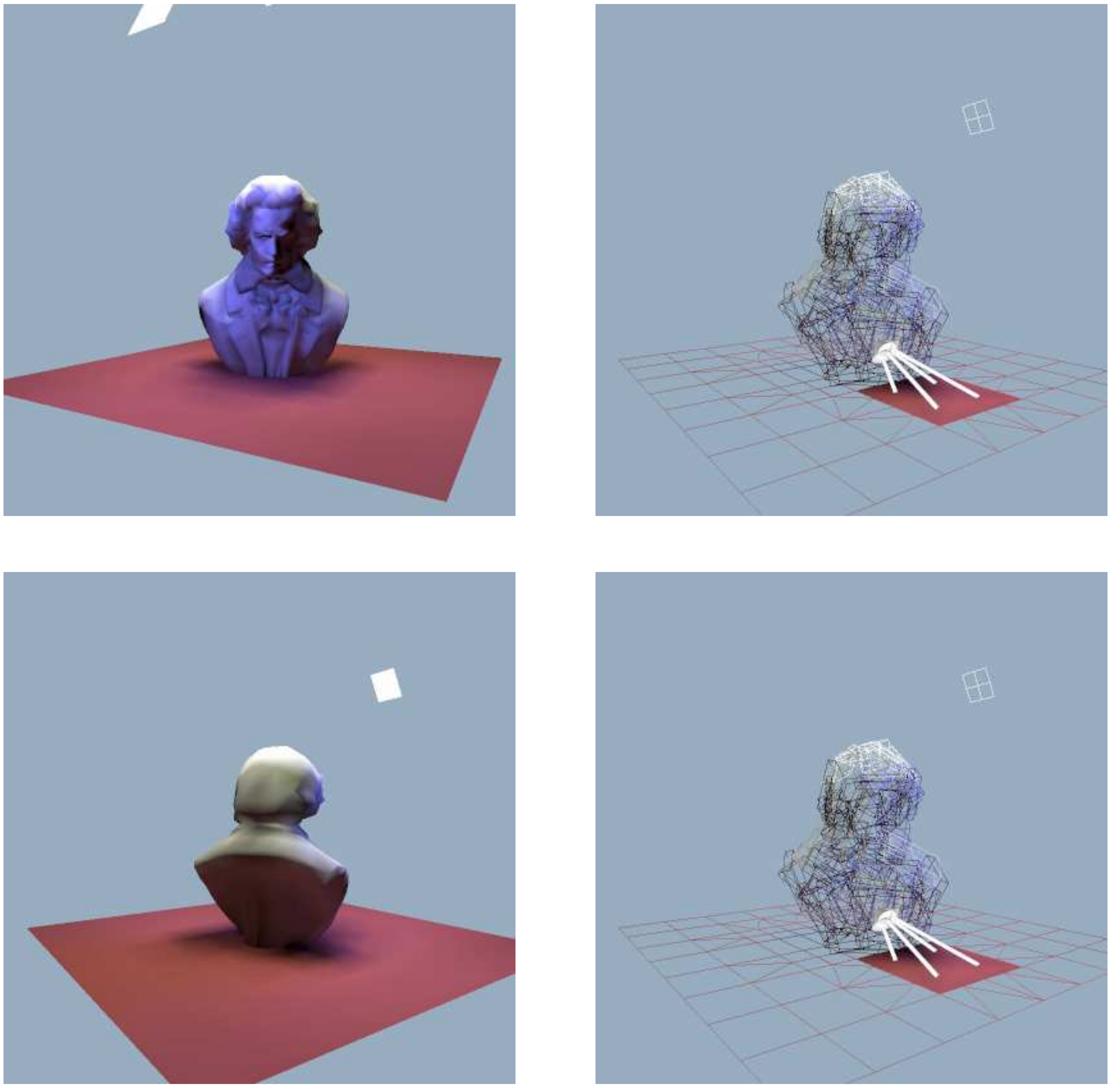


Figure 36: The bust of Beethoven. The bust is lit from overhead by a white light, and from above and to the left by a blue light. Shown are (a) the results of FCR, (b) the corresponding face clusters, (c) reflection off the red carpet onto Ludwig's back, (d) the contribution of nearby patches to one of the back clusters.

ually preprocess a scene in a view-dependent manner to optimise radiosity simulations. Ultimately, we wish the radiosity simulation to be as independent of the model resolution chosen for viewing as possible.

3.5.2. Memory Locality

Radiosity computations are traditionally extremely memory intensive. Because of this, the memory locality of a radiosity algorithm is a critical aspect of its performance, albeit one that is often ignored. For instance, progressive radiosity is well known for its good total memory use; in many cases it is better than that of hierarchical radiosity methods. However, it has very poor locality; each iteration of the algorithm is essentially a linear sweep through several large vectors. Hierarchical radiosity has much better locality, in the sense that each solution iteration, which requires a tree-traversal of the data set, accomplishes much more than the equivalent progressive iteration, resulting in significantly fewer iterations to generate a solution.

This has interesting implications in terms of which algorithm is the speediest. For scenes where both progressive and hierarchical radiosity can fit their computations into physical memory, hierarchical radiosity is quickest. However, if the scene becomes large enough, the hierarchy will no longer fit in physical memory, and progressive radiosity becomes the quicker algorithm. Finally, when the scene is so large that both algorithms must be accommodated by virtual memory, the superior locality of hierarchical radiosity makes it again quicker than the progressive algorithm; in other words, hierarchical radiosity thrashes better than progressive radiosity.

Face cluster radiosity inherits the good data structure locality from the hierarchical radiosity algorithm. To take advantage of this, face cluster files are written in breadth-first order, so we get correspondingly good memory locality. (These files are simply memory-mapped for use by the solver.) FCR also has the added advantage of much smaller active memory use; it is almost always the case that only the first small section of each face cluster file is used during solution.

3.5.3. Surface Material Maps

The visual complexity of a scene is often enhanced by the application of various types of surface maps to the original models. Most of these are handled naturally by our algorithm. Texture maps can be pre-sampled at the leaves and filtered over the vertex or face cluster hierarchy as described in [Gers94]. A similar approach can be taken to emittance maps, although the quantity pre calculated for the hierarchy needs to be an emittance vector, similar to the power vector.

In particular, our approach extends well to the use of displacement maps. Displacement mapping is a useful tool for adding geometric complexity to simple models. For instance in Pharr et al. [Phar97] a highly complex scene of 5 million primitives is constructed from a number of simpler source primitives, each containing on the order of 10,000 polygons, by applying displacement maps. Displacement-mapped objects are characterized by highly tessellated surfaces with a high degree of planar coherence, but widely varying normals. In such cases it is possible to approximate the irradiance of large areas of the model with a single link, applying the irradiance vector to the local surfaces only at the time of the final render. Similarly, having the irradiance vector available makes applying bump maps [Blin78] to surfaces trivial.

3.5.4. Animated Models

The face cluster hierarchies generated by the method outlined here can be reused over different scenes, and even over successive frames of an animation, provided the models they are generated from do not change. Of course, this is not always the case, especially when a model takes the part of a central character in the animation, rather than a part of the static backdrop. It can be shown that rigid body transformations and uniform scaling do not affect the construction of the hierarchy; hierarchies would not have to be rebuilt for model animations featuring only such transformations.

However, deformations [Barr84, Gudu90], commonly used to bend and reshape models on-the-fly in a non-linear manner, can and do affect the hierarchy. In such cases, a new face hierarchy would have to be generated for the model for each frame in the animation in which it is deformed, thus undoing some of the benefit of my algorithm. (It should be noted that standard volume clustering has to rebuild the entire hierarchy for each new frame regardless, although arguably the same hierarchy pre-calculation techniques outlined here could be applied to volume clustering also.) Deformations result in only incremental changes to a model's geometry from scene to scene, though, so we would hope that a local pass could refit the hierarchy at each step of the animation, without the cost of rebuilding it from scratch.

3.6. Summary

We can summarise the steps in the face cluster radiosity algorithm as follows:

- We construct a face cluster hierarchy file for each new model acquired. (Time and space super-linear in k. As we shall see in **Section 5.6**, this is in practice linear in k.)
- We create a scene from these models.
- The radiosity program reads in the scene description, and adds the root face cluster nodes to a volume cluster hierarchy. (Time and space linear in s, where s is the number of disconnected surfaces in the scene.)
- The gather/push-pull/refine solver is run. (Sub-linear in k.)
- The radiosity solution is propagated to the polygonal leaves of all models, and written to disk. (Linear in k.)

As usual, k is the number of polygons in the input scene.

Chapter 4

Analysis

In this chapter we will analyse the vector radiosity approach presented in **Section 3.4** in more detail. We will look more closely at the role the sum-area normal plays in face cluster radiosity, and investigate its relationship to more conventional measures of the projected area of a cluster. We will also develop methods for bounding the transfer of radiosity between two face clusters.

4.1. Sources of Error in Face Cluster Radiosity

I will start by examining possible sources of error in vector radiosity; any approximations made in the method are only acceptable if we have a reliable way of detecting the error in these approximations. As we shall see, it turns out the projected area of a cluster plays a crucial part in such approximations. I will introduce some useful definitions for later sections, and end with an example to illustrate some of the concepts involved.

4.1.1. Recap

In vector radiosity, we are trying to approximate the transfer of radiosity between the individual elements in two clusters of faces, as depicted in **Figure 37**, with a single cluster-to-cluster transfer. We assume the faces within a cluster are all part of one, manifold surface.

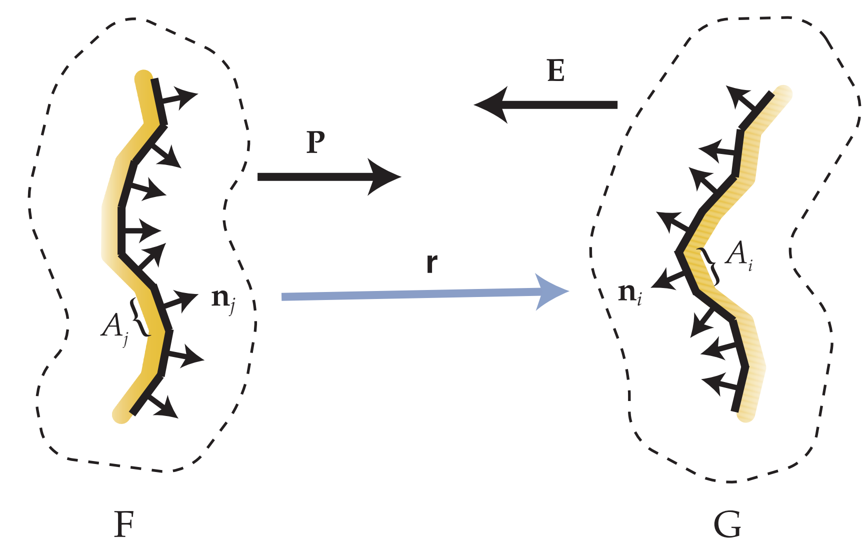


Figure 37: Radiosity transfer between clusters of faces.

As discussed in **Section 3.4.2**, we wish to calculate quickly

$$E_i = \sum_j \frac{(-\hat{\mathbf{n}}_i^{\mathrm{T}} \hat{\mathbf{r}}_{ij})_+ (\hat{\mathbf{r}}_{ji}^{\mathrm{T}} \hat{\mathbf{n}}_j)_+}{\pi r_{ij}^2} v_{ij} A_j b_j . \tag{38}$$

We do this by making the vector radiosity approximation:

$$\begin{aligned} E_i &\approx \left(\hat{\mathbf{n}}_i^{\mathrm{T}} \left[\frac{-\hat{\mathbf{r}}}{\pi r^2} (\hat{\mathbf{r}}^{\mathrm{T}} \mathbf{S}_F)_+ b_F \bar{v} \right] \right)_+ \\ &= (\hat{\mathbf{n}}_i^{\mathrm{T}} \mathbf{E})_+ \end{aligned} \tag{39}$$

For each face cluster we have a bounding box, oriented such that the z axis is the normal of the best-fit plane to the faces within the cluster, and we also store the sum area-weighted normal, $\mathbf{S}$,

$$\mathbf{S} = \sum_i \mathbf{S}_i , \tag{40}$$

where $\mathbf{S}_i = A_i \hat{\mathbf{n}}_i$ is the area-weighted normal of face i in the cluster. The area-weighted average face normal of the cluster is then $\hat{\mathbf{S}}$, and its projected area in that direction is $\|\mathbf{S}\|$. Note that, although $\mathbf{S}$ is roughly perpendicular to the best-fit plane, it is rarely exactly so.

Finally, we also have the total surface area A of the cluster:

$$A = \sum_i A_i . \tag{41}$$

Given this setup, we can state the sources of error in our approximation to the radiosity transfer between face clusters as follows:

- The assumption that the direction vector $\mathbf{r}$ is constant.
- The assumption that $\sum_i (\hat{\mathbf{r}}^{\mathrm{T}}\mathbf{S}_i)_+ = (\hat{\mathbf{r}}^{\mathrm{T}}\mathbf{S})_+$.
- That any self-shadowing within the face clusters is ignored.
- That the occlusion between the two clusters is assumed to be constant.

4.1.2. The Face Cluster Approximation

We are essentially representing the surface within a face cluster by its area-sum normal $\mathbf{S}$, a directed surface element. That is, we are representing a diffusely reflecting curved surface with a single planar Lambertian reflector. If our assumptions about the distance of the cluster from the light sources illuminating it hold up, then when viewed from the direction of $\mathbf{S}$, our approximation is close to perfect. However, from side on, the approximation is not so good. Whereas the apparent size of a single planar element goes to zero as our viewing angle tends to 90 degrees from $\mathbf{S}$, the apparent size of our curved surface does not. In the worst case of a spherical surface, the apparent size will not change no matter where we view it from.

The worst case for this error is generally at the top level cluster, which contains the entire (connected) surface mesh. The planarity metric of the face clustering algorithm takes account of the orientation of faces, and heavily penalises any surface that is curved, so if we descend a few levels in the face cluster hierarchy, the surface will be split into face clusters that are more planar. As long as we detect this source of error, and force subdivision in areas where it is excessive, we can be sure our algorithm will produce a reasonable approximate answer to the global illumination problem.

4.1.3. The Importance of Projected Area

If we multiply the element-to-element radiosity transfer equation, **Equation 38**, by the area of the receiver, we get the received power on element i from element j, which can be written as

$$\frac{(-\hat{\mathbf{r}}^{\mathrm{T}}\mathbf{S}_i)_+(\hat{\mathbf{r}}^{\mathrm{T}}\mathbf{S}_j)_+}{\pi r^2}b_j, \tag{42}$$

where again $\mathbf{S}_i = A_i\hat{\mathbf{n}}_i$. This received power depends on three quantities: the projected areas of the source and destination clusters along $\mathbf{r}$, and a scalar term that accounts for falloff with distance. When we try to use this equation to approximate more complex situations (cluster-to-cluster interactions, for instance), we must be aware of how accurate our approximations to each of these quantities are. Generally this is easy with the distance term, but more difficult to do with projected area terms.

The differences in apparent size between the actual face cluster and our representation of it discussed above can be more accurately viewed as representing an error in our approximation to the actual projected area of the cluster. We are representing the projected area of a face cluster as $(\mathbf{S} \cdot \hat{\mathbf{m}})_+$, and we must be able to decide when this approximation is a bad one in order to force subdivision and improve the approximation. Ideally, this requires bounding the projected area of the cluster in a particular direction $\hat{\mathbf{m}}$. Stamminger et al. have shown how this can be done for simple objects that are capable of providing a cone of normals, but for face clusters we require a more general solution [Stam97c].

4.1.4. Notation

We will be using the mathematical notation shown in **Table 2** for this chapter. There is also a summary of useful physical quantities in Appendix A.

4.1.5. Projected Area Definitions

We define the *visible projected area* (VPA) of a cluster P as

$$\mathrm{VPA}(P, \mathbf{m}) \equiv \int_V \hat{\mathbf{m}} \cdot d\mathbf{S}, \tag{43}$$

where $V \subset P$ is the subset of the surface points in P that are visible from the direction $\hat{\mathbf{m}}$. This is the quantity that we must try to approximate for our transfer calculations.

Symbol	Meaning
$(x)_+$	Maximum of x and 0
$\lceil x \rceil$	Upper bound on x
$\lfloor x \rfloor$	Lower bound on x
$[x]$	Average value of x: $(\lceil x \rceil + \lfloor x \rfloor)/2$
$\langle x \rangle$	The interval of x: $[\lfloor x \rfloor, \lceil x \rceil]$.

Table 2: Mathematical definitions

We also define the *non-negative unoccluded projected area* (NUPA) of the cluster in the direction $\hat{\mathbf{m}}$ as

$$\mathrm{NUPA}(P, \mathbf{m}) \equiv \int_P (\hat{\mathbf{m}} \cdot d\mathbf{S})_+ . \tag{44}$$

This is simply the projected area over all of P, rather than just the visible regions, V. Because occlusion only ever reduces the amount of surface visible in the cluster, $\mathrm{VPA} \leq \mathrm{NUPA}$.

Finally, we define the *signed unoccluded projected area* (SUPA) of a cluster as

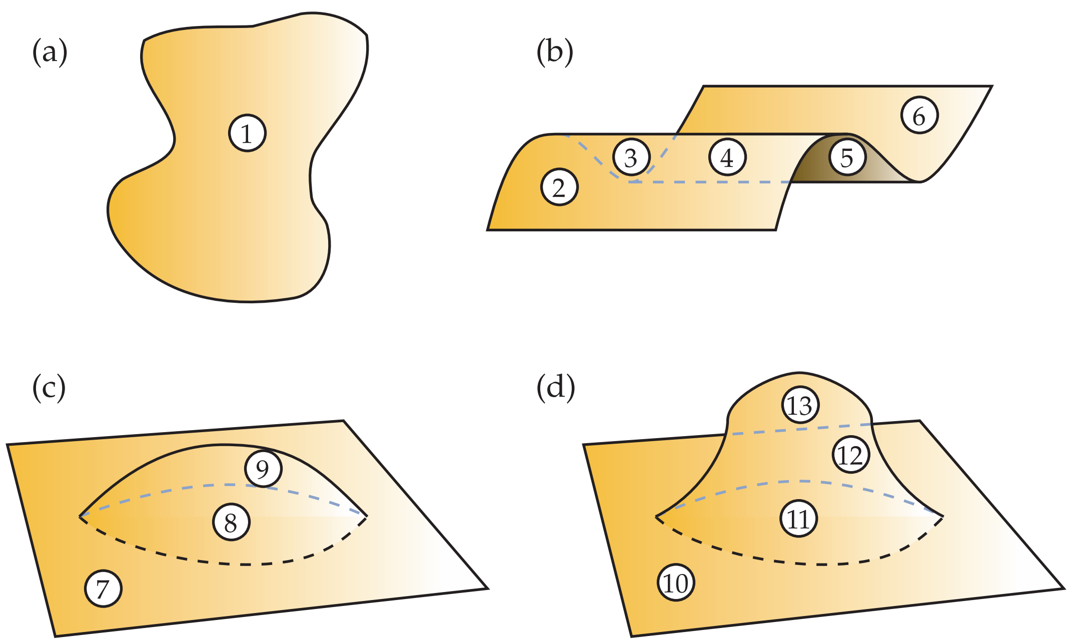
$$\mathrm{SUPA}(P, \mathbf{m}) \equiv ((\hat{\mathbf{m}} \cdot \int_P d\mathbf{S})_+ = (\hat{\mathbf{m}} \cdot \mathbf{S}_P)_+) . \tag{45}$$

For this estimate, we find the sum-area normal vector of the cluster, $\mathbf{S}_P$, and use its clipped dot product with the direction vector $\hat{\mathbf{m}}$ to calculate projected area. This is the equivalent of treating the cluster as a flat surface, and is the primary approximation we employed in **Section 3.4**.

As an illustration of how these methods work, **Figure 38** shows a number of example surfaces split into regions. These projected area of each region may be counted differently by each projected area method. For instance, the VPA will count each region where the surface is facing forwards once, and thus only region 5 is ignored. The NUPA can overcount the forward-facing projected area of some regions, for example region 4, because it ignores visibility. The SUPA can undercount some regions, such as region 11, because the back-facing area in those regions cancels out the forward-facing area.

4.1.6. Visibility Definitions

In our discussions of the role of occlusion in radiosity transfer, it is helpful to establish the following terms:



Surface	Region	VPA	NUPA	SUPA
(a)	1	1	1	1
(b)	2, 6	1	1	1
	3	1	1	0
	4	1	2	1
	5	0	1	0
(c)	7, 8	1	1	1
	9	1	2	1
(d)	10, 11	1	1	1
	12	1	2	1
	13	1	1	0

Figure 38: When calculating the projected area of the above surfaces in the viewing direction, the VPA, NUPA and SUPA count the regions differently, as shown in the table.

- *Tangential visibility*: whether a surface is occluded by its tangential plane. This is usually addressed by clipping the projected area calculation of surface elements to zero.
- *Intra-cluster visibility*: local visibility or shadowing; a surface element is shadowed by other parts of the surface within the face cluster.
- *Inter-cluster visibility*: global visibility or shadowing; the surface element is shadowed by some other surface outside the face cluster.
- *Macro-scale self-shadowing*: the shadowing is on a scale comparable to the surface. For instance, a convex surface where one side shadows the other, or a concave surface where the "lip" of the surface shadows part of the "bowl".
- *Micro-scale self-shadowing*: self-shadowing that occurs on a scale much smaller than the surface. Consider small bumps and notches on the surface. This is the kind of self-shadowing that must be taken into account when computing new BDRFs.

4.1.7. Discussion

An example will illustrate a number of points about dealing with clusters of surfaces. Consider the two-dimensional cluster shown in **Figure 39**, a simple saw-tooth surface. In considering the transfer of radiosity between it and a putative source, we must calculate the visible projected surface area of the cluster as seen from that source. From directly above, i.e., source *a*, this will be $\sqrt{2}nA$, where n is the number of teeth in the surface, and A is the area of one side of a tooth. From side-on, source *c*, it will be zero, as only the right-most face is visible, and it is facing away from *c*. From source *b*, which lies in the same direction as the left slope of each saw "tooth", the visible projected area is simply nA.

Traditional volume clustering algorithms deal with such a situation by summing the projected area of each polygon within the cluster in the direction of $\mathbf{m}$. I.e., they find the NUPA of the cluster, $\sum_i (\mathbf{S}_i \cdot \hat{\mathbf{m}})_+$. This has two drawbacks; it requires traversing every polygon within the cluster, and it ignores occlusion of polygons by each other within the cluster. If this intra-cluster occlusion is ignored, the projected area in direction *c* will be calculated as $nA/\sqrt{2}$, rather than zero. This is a large error, and gets worse as n increases. For a largely flat but crinkled surface, it can lead to badly overestimating the amount of radiosity received or radiated from the cluster horizontally.

Some researchers have proposed accounting for intra-cluster occlusion by calculating an isotropic estimate of it. This is done by a Monte-Carlo sampling of

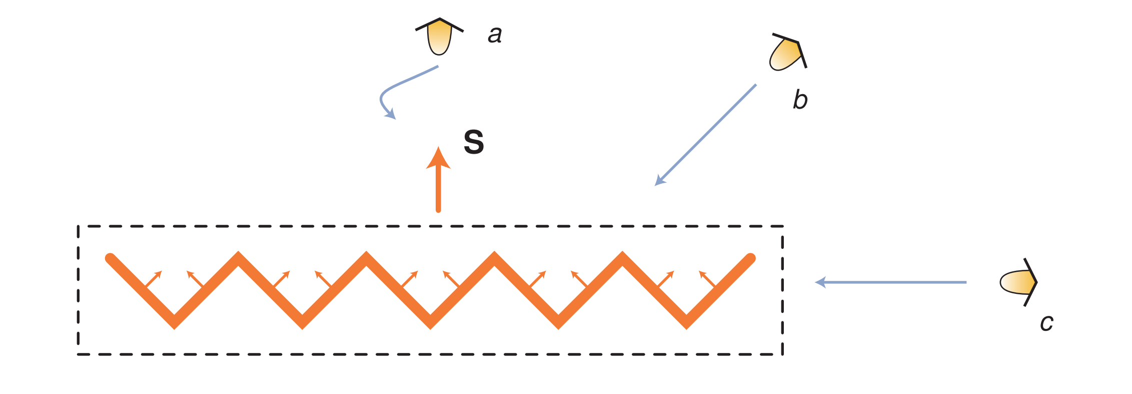


Figure 39: A sawtooth surface. The surface is potentially lit by light sources a, b and c, and has the sum-area normal vector $\mathbf{S}$.

the visibility over different points and directions within the cluster to find an average visibility estimate, $\bar{v}$. This may not help much in the case of directional clusters. In the example in **Figure 39**, such an estimate, being isotropic, would result in underestimating the projected area in direction a, and would still overestimate the projected area in direction c.

It is also possible to calculate an occlusion factor on a per-link basis, in the direction of the link, in the same way that the projected area in the direction of the link is calculated. A number of rays are cast in the direction $\mathbf{m}$ from points evenly distributed through the cluster; the visibility factor $\bar{v}_{\mathbf{m}}$ is taken to be the average visibility value of these rays. While this improves the estimate markedly, it is expensive, and, in the case of surfaces, it still leads to errors. In the sawtooth case, the average visibility in the vertical direction is $\bar{v}_{\mathbf{m}} = 1/2$, even though all of the surface is visible in that direction, because at least half of the samples will fall behind the surface.

I would argue that, for clusters containing surface meshes, the location of a surface and the visibility of points on that surface are too tightly coupled to be separable in a well-behaved way. There is simply too much coherence in the structure of the surface, and alternative techniques are needed. Ideally we should be sampling visibility from points on the surface contained by the cluster, but tracking such points will either be memory or time-intensive.

The vector radiosity approach we introduced in **Section 3.4** approximates the projected area by storing the sum of area-weighted normals of each polygon in the cluster as $\mathbf{S}$, and calculating the projected area of the cluster as $(\mathbf{S} \cdot \mathbf{m})_+$. Thus it uses the SUPA estimate of projected area. This reduces the calculation time to a single dot-product, but seemingly also ignores visibility, and is only an estimate.

So, what is the error in approximating the sawtooth surface with a single sum-area normal vector, $\mathbf{S}$? Obviously, from straight above (source *a*), the calculation has no error. From side-on (source *c*), it calculates the projected area as zero, which in this particular case is correct. Also, from an angle from the vertical smaller than the pitch of the surface's "teeth", there is no self-shadowing taking place, and once again $(\mathbf{S} \cdot \mathbf{m})_+$ returns the correct result. Once the angle increases past this, self-shadowing starts to occur, as in **Figure 40**, and to be accurate we must calculate the visible projected area, the VPA, of the surface, taking into account self-occlusion. We might expect that at this point our sum-area normal approximation would fail.

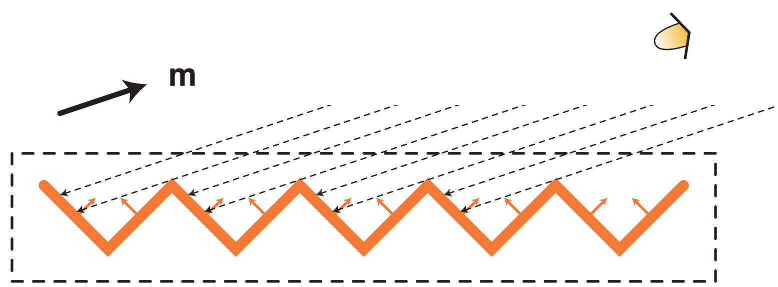


Figure 40: Self-shadowing on the saw-tooth surface. Surprisingly, $(\mathbf{S} \cdot \mathbf{m}_+)$ gives the correct projected area, even taking into account the surface self-shadowing illustrated here.

Surprisingly, this is not the case. It is possible to show with some simple algebra that, regardless of the pitch of the teeth in such a sawtooth surface, its projected area in a direction $\mathbf{m}$, taking into account self-shadowing, is *exactly* $(\mathbf{S} \cdot \mathbf{m})_+$. Thus our estimate of the total projected area of the cluster has turned out to be a much better estimate of the area taking occlusion into account, than the original calculation we were trying to approximate. In this case, the VPA = SUPA.

This result seems almost too good to be true, and unfortunately, in most cases it is. Consider **Figure 41**, which contains an almost identical surface, but one whose side-on projected area is $A/\sqrt{2}$, and not zero as our approximation would predict. It is still the case that the sum-area normal gives a much more accurate estimate of the projected area of the cluster than performing the complete sum, however; the SUPA is a much better approximation to the VPA than the NUPA.

It is also the case that this property does not necessarily hold for weighted area-normal sums. If we instead consider a reflective surface vector, $\mathbf{R} = \sum \rho_i A_i \hat{\mathbf{n}}$, then things work as before if the reflectivity of the surfaces is constant. In **Figure 42** we see a surface where one side of each tooth is considerably brighter than the other. For this surface, $\mathbf{R}$ underestimates the visible projected reflective area from source a, and overestimates it from b.

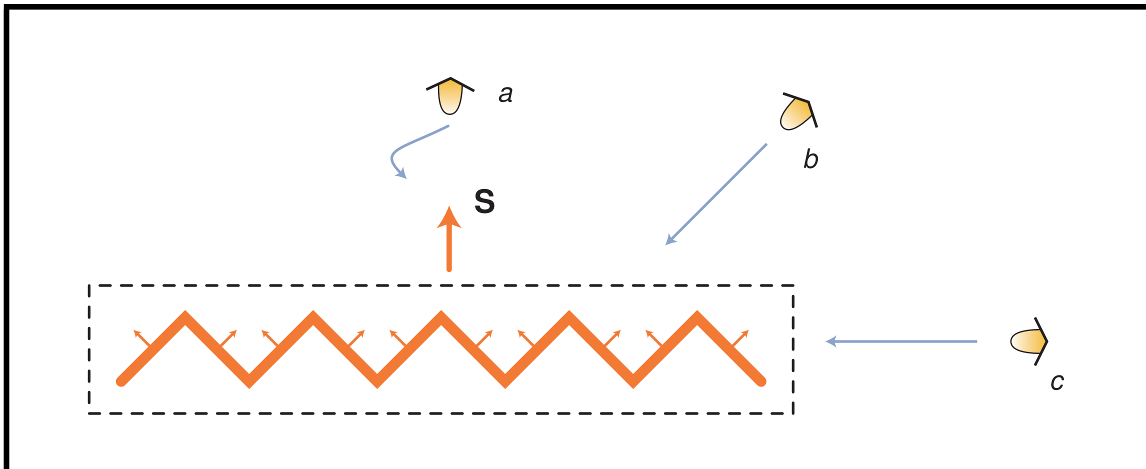


Figure 41: Sawtooth surface with non-zero side area. $\mathbf{S} \cdot \mathbf{m}$ does not give the correct projected area in the direction of c.

Our observation raises some interesting questions; what other surfaces might our estimate do a good job on, and under what conditions? It seems that in some situations the sum-area normal can be used to generate an excellent approximation to the visible projected area of a cluster containing a relatively flat surface. But we must formalize this observation for it to be useful.

In **Section 4.3.2** I will outline the conditions under which this observation holds. I will also show how the error of the approximation can be quantified in **Section 4.2**, firstly by proving that the estimate $(\mathbf{S} \cdot \mathbf{m})_+$ is almost always a lower

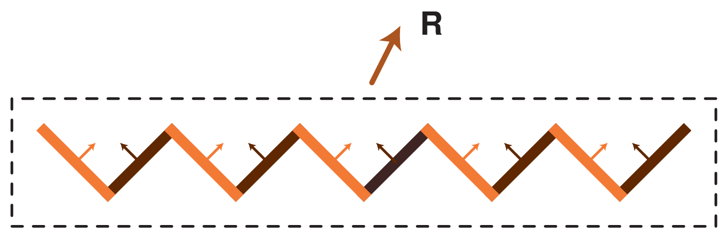


Figure 42: Sawtooth surface with varying reflectance. If we use a reflectance-weighted vector **R**, the estimate fails.

bound on the actual projected area of the cluster, and then by showing how a conservative upper bound can be constructed.

4.1.8. Cancellation

It might seem intuitively that representing a sum of irradiances by a single irradiance vector **E** could lead to cancellation. Consider **Figure 43**, for instance: if we naively sum the irradiance vectors from light sources *b* and *d*, the light from *d* will cancel out the light from *b*, and the point P shown on the surface will be incorrectly illuminated. However this is only a problem if we fail to clip to some consistent plane when calculating **E**. (If we did not have to take into account tangential visibility when calculating the illumination from a light source, then we could indeed just use the vector sum with no error.) To avoid this problem, whenever we gather power across a link, we must clip the resulting irradiance to the corresponding **S**; the same is also true when we push or pull irradiance, as in **Section 3.4.2**.

Even with clipping, it may seem at first glance that light sources can cancel each other out. If we consider light sources *a* and *c* in **Figure 43**, being opposite each other and close to the horizon, it looks as though their horizontal components cancel out when we sum their irradiance vectors. In fact there is no cancellation; this is a standard vector sum. The confusion arises because **E** is implicitly measuring irradiance with respect to the plane of the surface, so the horizontal components of the irradiances from *a* and *c* are irrelevant to the scalar irradiance. Therefore, there is no cancellation problem with our method.

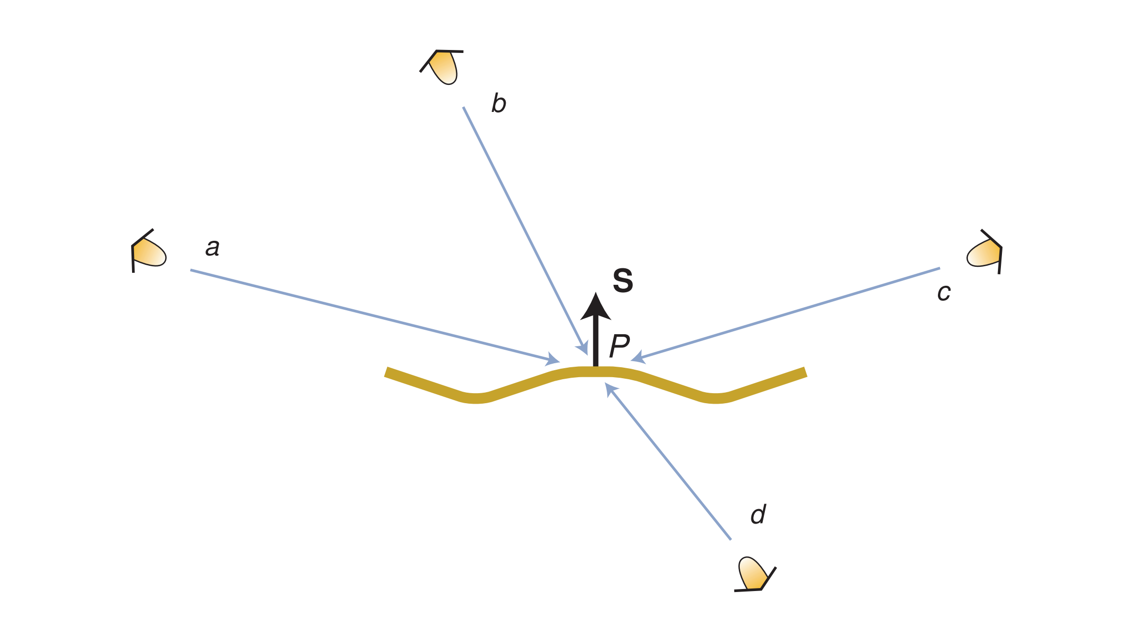


Figure 43: Avoiding cancellation in vector irradiance.

4.2. Bounding the Projected Area of a Cluster

Estimating the actual projected area of a face cluster is a crucial operation in our algorithm. In this section we examine how we can construct bounds on this quantity, such that it can be evaluated in constant time. We start by ignoring visibility, so we are trying to bound the NUPA.

4.2.1. A Lower Bound

Finding a lower bound to the projected area is straightforward. We can use the inequality,

$$(a+b)_+ \le (a)_+ + (b)_+ , \tag{46}$$

to show that, if $\mathbf{S} = \sum_i \mathbf{S}_i$, then

$$(\mathbf{S}\cdot\mathbf{m})_+ \le \sum_i (\mathbf{S}_i \cdot \mathbf{m})_+ . \tag{47}$$

That is, our approximation to the projected area of a cluster is provably a conservative lower bound on the actual projected area: SUPA $\le$ NUPA .

Proof

Obviously given **Equation 46** we have:

$$(\mathbf{S}_0 \cdot \mathbf{m} + \mathbf{S}_1 \cdot \mathbf{m} + \ldots)_+ \le (\mathbf{S}_0 \cdot \mathbf{m})_+ + (\mathbf{S}_1 \cdot \mathbf{m})_+ + \ldots \tag{48}$$

4.2.2. An Upper Bound

Using **Equation 46** it is possible to show that:

if $D_j = \sum_i (\mathbf{S}_i \cdot \mathbf{x}_j)_+$, and $\mathbf{m} = \sum_j c_j \mathbf{x}_j$ where $c_j \ge 0$, then

$$\sum_i (\mathbf{S}_i \cdot \mathbf{m})_+ \le \sum_j c_j D_j . \tag{49}$$

That is, say we sample the projected area of a cluster in a number of different directions $\mathbf{x}_j$. Then, if we can write $\mathbf{m}$ as a linear combination with positive coefficients of some subset of the $\mathbf{x}_j$ vectors, using the same coefficients to interpolate the corresponding projected area samples gives us an upper bound on the projected area in the direction of $\mathbf{m}$. **Figure 44** illustrates how this works in two dimensions.

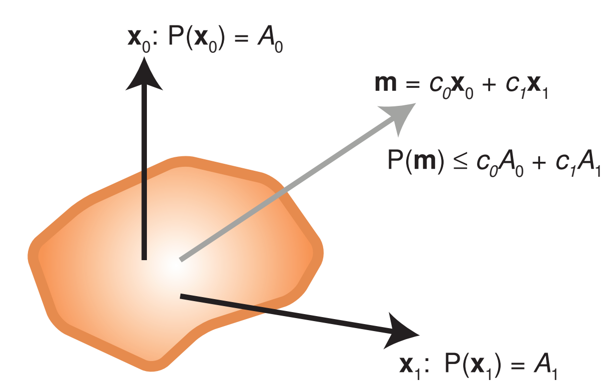


Figure 44: Finding an upper bound from projected area samples.

Neither of these two bounds assume that the surfaces are connected, and thus they can both be applied to volume clusters as well as face clusters.

Proof

We can prove **Equation 49** through the following steps:

$$\begin{aligned}
\sum (\mathbf{S}_i \cdot \mathbf{m})_+ &= \sum_i \Big(\mathbf{S}_i \cdot \sum_j c_j \mathbf{x}_j\Big)_+ \\
&= \sum_i \Big(\sum_j \mathbf{S}_i \cdot c_j \mathbf{x}_j\Big)_+ \\
&\leq \sum_i \sum_j (\mathbf{S}_i \cdot c_j \mathbf{x}_j)_+ \\
&= \sum_i \sum_j c_j (\mathbf{S}_i \cdot \mathbf{x}_j)_+ \quad \textbf{if} \quad c_j \geq 0 \\
&= \sum_j c_j \sum_i (\mathbf{S}_i \cdot \mathbf{x}_j)_+ \\
&= \sum_j c_j D_j
\end{aligned}$$

Implications

If we sample the projected area of a cluster over a number of directions, we can calculate an upper bound on the projected area in an arbitrary direction by interpolating from the three surrounding sampled directions. This estimate will be better the closer to the sampled directions the required direction is, and of course exact if it matches one of the samples[1].

Thus the total projected area of a face cluster can be bounded, as long as we have at least four samples of its projected area in different directions. (The minimal set of samples corresponds to the vertices of the minimal polytope, a tetrahedron.) The fit of the upper bound depends on the sample directions chosen, and the number of samples; obviously the more samples, the better the coverage and resulting bound. Also, if the surface is largely oriented in a particular direction, sampling primarily in that direction will provide better results than sampling in random directions.

One of the simplest collections of directions corresponds to a box, with the projected area sampled in the direction of each of the (six) faces of the box. Inter-

1. I speculate that this technique is approximating the real surface with a polyhedral convex hull.

estingly, if the box in question is a bounding box for the cluster, the six projected-area samples are in most cases bounded by the side areas of the box. There is an important exception; in some cases the fact that we ignore occlusion in the projected area sample can cause it to be larger than the corresponding side area of the box.

A good choice for the basis vectors is the set of axes of the tightest-fit oriented bounding box (OBB) enclosing the cluster. The tightest-fit OBB minimises the side areas of the box, and as the side areas are in turn bounds on the projected area, we are in some sense minimising our upper bounds.

4.3. Intra-cluster Visibility

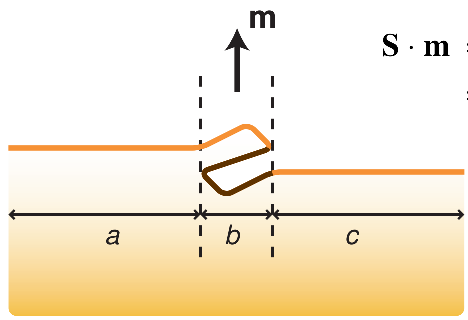
We now consider the visible projected area of a surface, namely, how to calculate the projected area taking into account intra-cluster visibility. We'll start with some simple illustrative examples.

4.3.1. Self-Shadowing

Radiosity papers commonly ignore self-shadowing by the surfaces under consideration. Even the way the familiar form-factor equation accounts for tangential visibility, by the clamping of both cosine factors to zero, is usually implicit. (Consider the standard radiosity equation in **Equation 22**; it is not explicitly stated that we do not let the cosine factors become negative.) This has partly been because the radiosity method has its roots in the consideration of radiosity transfer between two planar polygons, which inherently have no self-shadowing other than tangential. Because face clusters contain curved surfaces which may self-shadow, we must pay closer attention to this phenomenon.

For mostly planar surfaces, self shadowing is restricted to the micro scale: small folds or bumps in the surface that shadow correspondingly minor parts of the surrounding area. **Figure 45** shows an example. There can also be macro-scale self-shadowing: more major silhouette-type shadowing when the surface is curved, as shown in **Figure 46**, where up to half the surface can be shadowed by the other half, even though the surface itself is relatively smooth.

Interestingly, as discussed in **Section 4.1**, our sum-area normal approximation works well where micro-scale self-shadowing is involved; in some cases it can actually help account for small-scale self-shadowing. An example of this can be seen in **Figure 45**. Using the projected area calculated from $\mathbf{S}$ accounts for the self-shadowing in the small fold shown reasonably well. If we clipped on a per element basis when calculating the projected area of the cluster, we would over-



$$\mathbf{S}\cdot\mathbf{m} = a + b - b + b + c$$
$$= a + b + c$$

Figure 45: Micro-scale self shadowing.

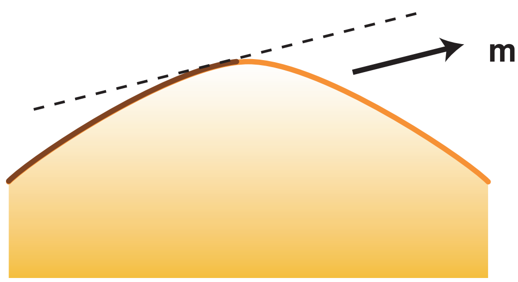


Figure 46: Macro-scale self shadowing.

count contributions from the fold in the surface, and our total projected area in the vertical direction would be $a + 2b + c$.

The way a surface shadows itself is highly correlated with the geometry of the surface itself, and thus its projected area. We need to be able to take it into account, preferably without accessing the detailed geometry of the cluster. (If we don't maintain a constant cost overhead in calculating it, we will lose the efficiency of our algorithm.) Ideally we would like to extend our bounds of a cluster's projected area to bounds on the same area, taking into account self-shadowing. The remainder of this section is thus devoted to the analysis of the visible projected area.

4.3.2. The Visible-Projected-Area Sum Rule

We have observed that in some cases, the sum-area normal approximation to the projected area of *P* is a surprisingly good approximation to the visible projected area. But we need to be clearer about under what conditions this is so. Fortunately, we can construct a rule that allows us to do this. Let us start with the two-dimensional case. The *visible-projected-area rule* for two dimensions is as follows:

> Consider an open path, for which we have defined a top and bottom side; only the top side is considered visible. Consider also the two parallel lines passing through either end point of the path, in direction $\mathbf{m}$. We claim that, for the section of the path that falls between these two end-point lines, the visible projected area in direction $\mathbf{m}$ is exactly equal to the dot product of $\mathbf{m}$ and the sum of area normals $\mathbf{S}$. That is, $\lfloor \mathrm{UPA} \rfloor = \mathrm{VPA}$. Further, we claim that for any remainder of the path lying outside those two lines, the dot product of $\mathbf{m}$ and the sum of area normals is exactly zero.

Figure 47 illustrates some examples of this rule.

Derivation

There are a number of ways to derive this result. The simplest for purposes of explanation is as follows. We divide the path *P* into slabs pointing in direction $\mathbf{m}$, such that each slab has a consistent number of path segments intersecting it. (See, for example, **Figure 47**a-c.) We then connect the two end points of the path with a line *r*, creating a closed path *Q*. We then observe that:

- For any slab, the projected area of a path segment that intersects it is either the positive or negative width of the slab, depending on the orientation of the path segment with respect to $\mathbf{m}$.
- The sum projected area in direction $\mathbf{m}$ of the closed path *Q* within any slab will add to zero. (This is analogous to the famous *winding number* principle used to test for the interior of a closed path.)
- The projected area of the path *P* plus that of the line *r* within each slab is equal to that of the path *Q*, namely zero.
- Within the end-point lines, the projected area of *r* within each slab is the width of the slab; outside it is zero.

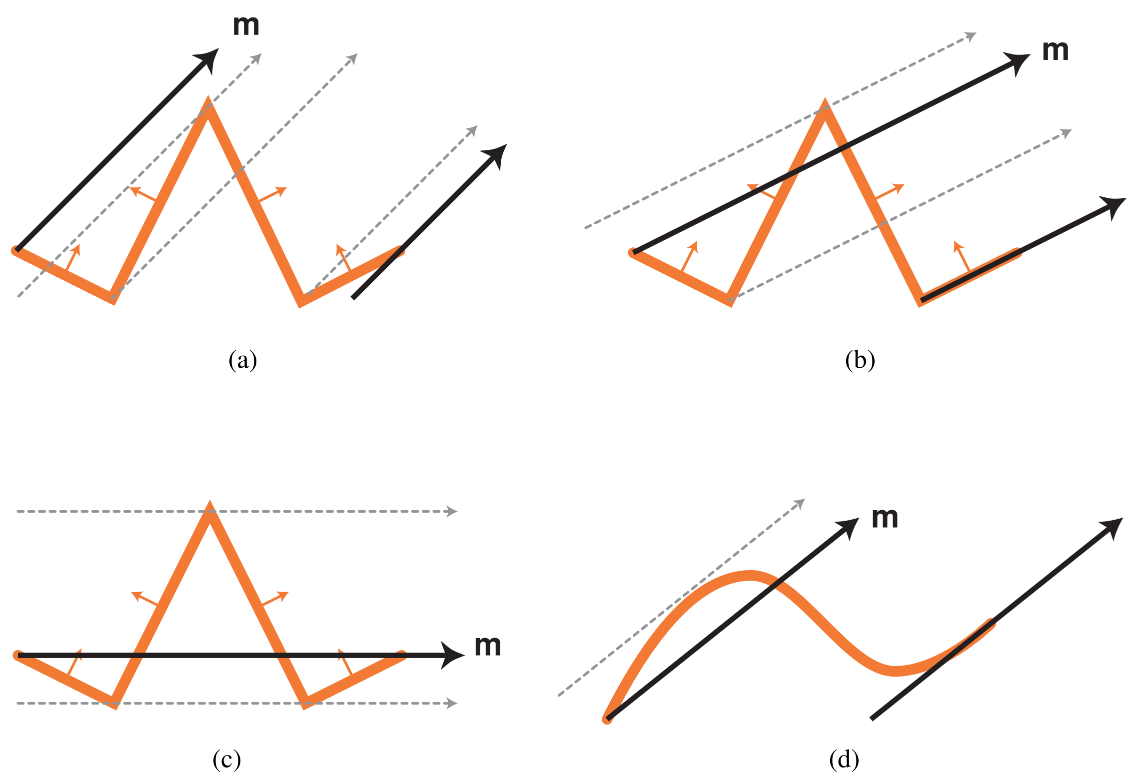


Figure 47: Estimating visible projected areas. In (a) the entire curve falls within the end-point lines, and the estimate $((\sum \mathbf{S}_i) \cdot \mathbf{m})_+$ is correct. In (b), as $\mathbf{m}$ gets more side-on, most of the curve still falls within the lines, but the VPA of the top peak of the curve is incorrectly estimated as zero. (c) Shows the worst case; the end-point lines are coincident, and the visible projected area of the curve is wrongly estimated as zero. (d) Shows how this rule applies to curves, where the estimate is $\int d\mathbf{S}$ rather than a discrete sum.

Thus, for all the slabs that lie between the path end points, the visible projected area of P in direction $\mathbf{m}$ is equal to $(\mathbf{S}_i \cdot \mathbf{m})_+$, where $\mathbf{S}_i \equiv \sum_{\mathbf{S}_j \in slab_i} \mathbf{S}_j$.

For all slabs that lie outside the path end-point lines, this sum is zero. For these slabs, $(\mathbf{S} \cdot \mathbf{m})_+$ obviously underestimates the visible projected area.

A condition on this rule is that the path must not self-intersect. If it does, any path loops cancel themselves out, and visible area can be missed. This is illustrated in **Figure 48**, where using the sum surface vector results in an estimated projected area of 0; instead the loop in the central slab should contribute a positive amount of area.

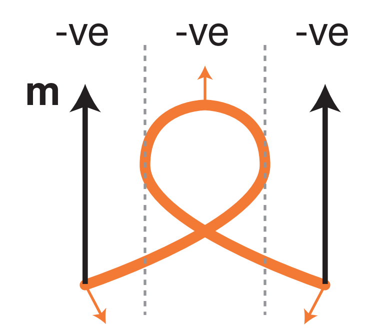


Figure 48: The VPA rule does not hold for paths that self-intersect.

4.3.3. Extension to Three Dimensions

This result can be extended to three dimensions; what was a path becomes a surface, and the end points of the path are now the boundary of that surface. We can state the new rule as follows:

> Consider an oriented manifold surface with a single closed boundary path. Extend the path in the direction $\mathbf{m}$, sweeping out an infinitely-long volume V, and thus sectioning the surface into two parts. Then the visible projected area of the surface inside the volume is *in most cases* correctly calculated as $(\mathbf{S}_V \cdot \mathbf{m})_+$, where $\mathbf{S}_V$ is the sum area normal of the surface lying within the volume. $(\mathbf{S}_\Omega \cdot \mathbf{m})_+$, where $\mathbf{S}_\Omega$ is the sum area normal of the surface lying outside the volume, is always zero.

See **Figure 49** for an example. Again, this usually has the same consequence as the two dimensional rule: $\mathrm{SUPA} \le \mathrm{VPA}$.

There is an important exception to the first part of this rule, hence the qualifiers above. In three dimensions, it is possible for a section of the surface to overlap the rest of the surface without a corresponding back-facing section to cancel out the occluded area. **Figure 50** shows some examples of how this can happen.

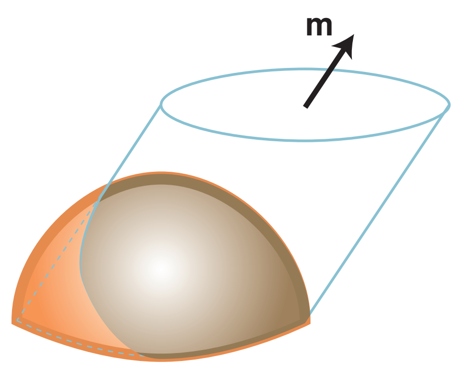


Figure 49: Visible Projected Area Rule in 3D. The surface P shown is a hemisphere. When projected in the direction $\mathbf{m}$, only the blue region of the surface (to the right), which lies within the open boundary of P swept in that direction, is calculated correctly by the SUPA. The external orange region's VPA is incorrectly estimated as zero. As $\mathbf{m}$ tends to the horizontal, the SUPA underestimates the true VPA more and more.

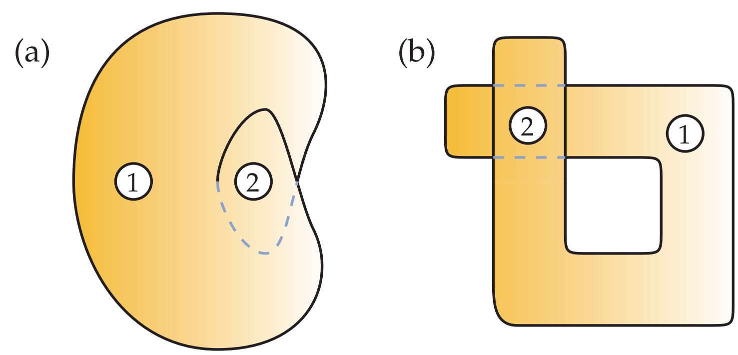


Figure 50: Exceptional cases for the 3D VPA rule. In the examples above the surface overlaps itself in region (2), and the projected surface area in that region will be counted twice, leading to a situation where $SUPA > VPA$. Fortunately such situations aren't common (see text).

These situations seem to occur rarely in practice. For the models and clustering techniques used in the course of this thesis, the SUPA has proven a good lower bound. (See **Section 4.5.5** for empirical data.) When such overcounting situations do occur, they tend to be for only a small part of the surface, and are often balanced out by undercounting elsewhere.

I surmise, but have not yet proved, that such exceptions cannot happen when the boundary contour is convex. Regardless, it almost always seems to be the case that overlaps correspond to severe concavities in the boundary curve. As our cluster cost metric seeks to produce well-shaped clusters by minimizing boundary length (**Section 3.3.4**, **Section 5.4.3**), it in some sense tries to avoid such situations. In general, the clusters in face cluster hierarchies contain surfaces that are compact and largely concave.

The VPA rule can also be extended to a manifold surface with holes; the accurate calculations will take place for those parts of the surface that are both inside the volume swept out by the boundary contour, and outside the volumes swept out by the hole contours.

4.3.4. Winding Numbers and Boundary Paths

We can be more precise about the conditions under which the SUPA fails as a lower bound on the VPA. In three dimensions it is possible for the projected boundary contour of the surface in direction $\hat{\mathbf{m}}$ to have a winding number greater than one for some regions. It turns out that these regions correspond exactly to possible SUPA/VPA bound violations: Note that each of the overcount regions in **Figure 50** has a winding number of two with respect to the boundary path of the surface.

We can explain this observation as follows. Firstly, we define the winding number of a contour γ with respect to a point p as:

$$w(p, \gamma) = \frac{1}{2\pi}\oint_{\gamma} \frac{1}{l - p} dl \tag{50}$$

Figure 51 shows some examples of different curves and their corresponding winding numbers. We observe that for any connected region that does not cross the contour, all points within will have the same winding number. Thus it makes sense to talk of the winding number of such a region with respect to the contour γ.

Consider the silhouette of a surface projected in direction $\mathbf{m}$. The area within the silhouette is precisely the VPA of the surface. Now, we project the sur-

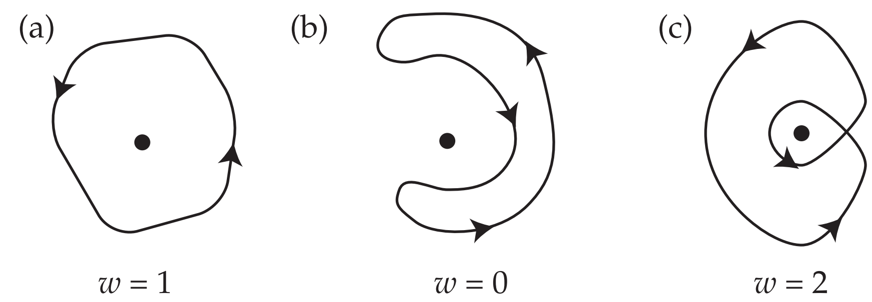


Figure 51: The winding number of a contour. Points inside the projected boundary contour have a winding number of 1, as in (a), and points outside have a winding number of 0, as in (b). When the boundary self-intersects, it is possible to get a winding number greater than 1, as in (c).

face's boundary contour(s) in direction $\mathbf{m}$. This projected boundary path must of course lie within the silhouette.

These projected contours will section the silhouette into a set of regions R_i, each of area A_i, such that $\forall p_i \in R_i, w(p_i, \gamma_S) = w_i$. That is, a set of regions such that all points in region i have the same winding number with respect to the projected boundary curve, γ_S. Then, using a trivial application of Stoke's theorem to convert the area integral of the surface sections within each region to the corresponding path integral, we find that

$$\mathrm{SUPA} = \sum_i A_i w_i . \tag{51}$$

The total area of the silhouette is just $\mathrm{VPA} = \sum A_i$.

As a consequence, if the surface is closed, and there is no boundary path, there is just one region of winding number 0, and $\mathrm{SUPA} = 0$. If $\forall i, w_i \le 1$, then we have $\mathrm{SUPA} \le \mathrm{VPA}$. If there are regions for which $w_i > 1$, which is only possible if the projected boundary path self-intersects, as in **Figure 51**c, it is possible to have $\mathrm{SUPA} > \mathrm{VPA}$.

An interesting way of looking at the SUPA approximation is:

> For a given direction $\mathbf{m}$, we are approximating the *boundary of the projection of the surface* by the *projection of the boundary of the surface*.

That is, we approximate the area within the boundary of the 2D projected surface, its silhouette, by using the area of the projected boundary of the 3D surface[1]. It is possible for areas of the surface to lie outside the boundary path, if it is sufficiently curved; then the SUPA underestimates the VPA. For a flat surface, however, the two are always equal, and $\mathrm{SUPA} = \mathrm{VPA}$.

4.3.5. Implications of the Rule

The visible projected area rule has the following important consequences for face clusters:

- $(\mathbf{S} \cdot \mathbf{m})_+$ is not only a lower bound on the unoccluded projected area of the face cluster, it is *almost always a lower bound on the visible projected area* of the face cluster, except in some uncommon situations where the surface crosses over itself.
- For flat surfaces with mainly micro-level occlusion, using $(\mathbf{S} \cdot \mathbf{m})_+$ to estimate the visible projected area often gives good results.
- The rule means that micro-level self-shadowing is handled well, and macro-level shadowing less well, especially when the surface is highly convex. The worst case comes when the surface is closed, and thus $\mathbf{S} = 0$.

4.3.6. The Bounding Box

Our upper bound from **Section 4.2.1** is a bound on the unoccluded projected area. As occlusion only ever reduces the amount of projected area visible, it does serve as an upper bound on the VPA. However, when there is a lot of concave self-shadowing (as in **Figure 39**), it can fit poorly.

As Stamminger et al. have observed [Stam97c], the projected area of the bounding box of a general cluster is an upper bound on the visible projected area of the entire cluster. To take advantage of its potential superiority in instances of high occlusion, we can combine both bounds. If our directional samples of the projected area correspond with the axes of each cluster's oriented bounding box, we can take

$$D_j^{vis} = min(X_j, D_j), \tag{52}$$

1. Note that the SUPA is in turn just an approximation of this, albeit an exact one when the path does not self-intersect.

where D_j is defined as in **Equation 49**, and X_j is the corresponding side-area of the bounding box. Using D_j^{vis} in **Equation 49** gives us a tighter upper bound on the VPA.

4.3.7. Bounds on the Visible Projected Area

In summary, we have now established bounds on the visible projected area A_P of a cluster P, as:

$$\lfloor \mathrm{VPA}(P, \mathbf{m}) \rfloor \cong (\mathbf{S}_P \cdot \mathbf{m})_+ \tag{53}$$

and

$$\lceil \mathrm{VPA}(P, \mathbf{m}) \rceil = \sum c_j D_j^{vis} \tag{54}$$

where c_j are calculated from $\mathbf{m}$ as in **Section 4.2.2**. We have also shown that usually

$$\lfloor \mathrm{UPA} \rfloor \leq \mathrm{VPA} \leq \mathrm{UPA} \leq \lceil \mathrm{UPA} \rceil, \tag{55}$$

namely, the bounds on the UPA presented in the previous section are often bounds on the VPA.

4.4. Bounding The Radiosity Transfer

A consequence of the hierarchical formulation of the radiosity problem is that for every transfer of radiosity between two parts of the scene that we calculate, we must also calculate an error value that measures the error in the approximation. This error value is used to drive the refinement process.

4.4.1. Choosing an Error Metric

When we evaluate the error in the transfer of radiosity, we must chose some metric with which to do so. Typically we define the norm of a function $f(x)$ as

$$L_n(f(x)) \equiv \left(\int f(x)^n\right)^{1/n}. \tag{56}$$

In a radiosity simulation, we are typically measuring the error between our radiosity estimate for a surface, b_i, and the actual radiosity at each point x on the surface, $b(x)$. The choice of an error metric affects both how we calculate our error measure for b_i, and how we combine the radiosity errors of the child nodes in order to calculate the error of their parent. The relevant expressions for both are shown in **Table 3** for the standard norms (following Lischinski et al. [Lisc94], section 4). In all cases, the error measure is bounded above by an expression of the range of bounds on b_i.

Norm	Error Measure	Parent Error Calculation
L_∞	$\max_{S_i}(\lvert b(x) - b_i\rvert) \le \frac{1}{2}(\lceil b_i \rceil - \lfloor b_i \rfloor)$	The maximum of the children's errors
L_1	$\int_{S_i} \lvert b(x) - b_i\rvert dA \le \frac{A_i(\lceil b_i \rceil - \lfloor b_i \rfloor)}{2}$	Sum the children's errors
L_2	$\sqrt{\int_{S_i} \lvert b(x) - b_i\rvert^2 dA} \le \frac{\sqrt{A_i}(\lceil b_i \rceil - \lfloor b_i \rfloor)}{2}$	Take the root-mean-square of the children's errors

Table 3: Error Norms

Stamminger et al. undertook a study of the results produced by the use of different error metrics in a radiosity algorithm [Stam97b], and concluded that the L_1 and L_2 metrics produced much the same results, and both were better than L_∞. They also found that L_1 and L_2 treat non-conservative bounds better than L_∞. The $\sqrt{A_i}$ factor in the L_2 norm is often taken to be the maximum side length of the polygon being treated; it has been shown that this can be better suited to accounting for the error over long, thin polygons than the straight area-weighted error measure.

Lischinski et al. quote Delves and Mohamed as saying that a cheap, good, occasionally inaccurate error bound often leads to better results than conservative bounds [Lisc94]. This has been borne out in my own experience with bounded radiosity [Will93]. Finding a conservative bound is not enough; it must be shown that as the solution is refined, bounds become more accurate. (We will demonstrate empirically the quality of our bounds in **Section 4.5**.)

The approach to measuring error used in this work is to use the area-weighted irradiance (often referred to as *BFA* in the literature), following the practice of a number of recent hierarchical radiosity papers. Thus the rest of this section will assume the L_1 error norm.

4.4.2. Bounding the Transfer

We can use the bounds we have established on the projected area of a cluster to help bound the transfer of radiosity between two clusters. The bounds give us both an estimate for the radiosity transfer to use in the simulation, and an error term for the transfer, to use in refining the solution.

The derivation is as follows. The total flux received from a source cluster S by a differential element $d\mathbf{S}_d$ on a destination cluster D is (see **Figure 52**):

$$d\Phi_{d \leftarrow S} = \left(d\mathbf{S}_d \cdot \int_S b_s v_{sd} \mathbf{m}_{ds} (\mathbf{m}_{sd} \cdot d\mathbf{S}_s)_+\right)_+ \qquad \text{[Watts]} \tag{57}$$

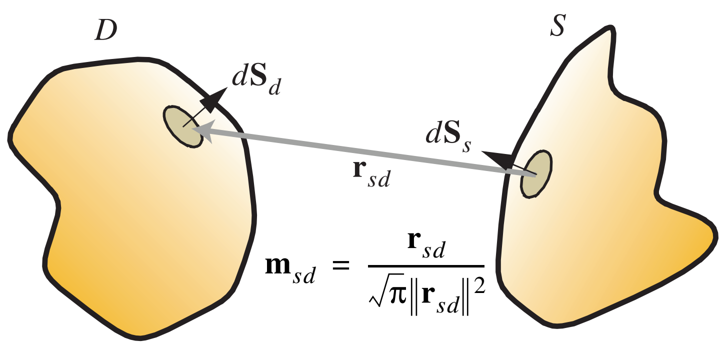


Figure 52: Vector radiosity transfer. Radiosity is transferred between differential surface elements on two face clusters, S and D.

Assume that the source cluster's radiosity is bounded by $\langle b_S \rangle = [\lfloor b_S \rfloor, \lceil b_S \rceil]$ everywhere on its surface. We then wish to formulate an error metric that measures how much error our various approximations are introducing. If we have bounds on the irradiance at every point on the destination cluster, then we can use some L_n norm as a measure of the error in the transfer. As mentioned previously, I use the L_1 norm for this purpose, which corresponds to the *BFA* error metric of traditional hierarchical radiosity, i.e., the error in the total flux Φ (power) received by the destination element. We can then write the flux using interval arithmetic [Moor66, Snyd92, Duff92] as follows:

$$\langle \Phi_{D \leftarrow S} \rangle = \langle G_{D \leftarrow S} \rangle \langle b_S \rangle\,, \tag{58}$$

where $G_{D \leftarrow S}$ is the geometry term between the two clusters,

$$G_{D \leftarrow S} = \int_D \left(d\mathbf{S}_d \cdot \int_S v_{sd} \mathbf{m}_{ds} (\mathbf{m}_{sd} \cdot d\mathbf{S}_s)_+\right)_+ \qquad [\mathrm{m}^2] \tag{59}$$

Thus the total error in the flux can be written as

$$\Delta\Phi_{D \leftarrow S} = \lceil b_S \rceil \lceil G_{D \leftarrow S} \rceil - \lfloor b_S \rfloor \lfloor G_{D \leftarrow S} \rfloor \qquad [\text{Watts}], \tag{60}$$

and the error due to the geometry factor, i.e., the face cluster approximation, as:

$$\begin{aligned} \Delta\Phi^{geom}_{D \leftarrow S} &= [b_S](\lceil G_{D \leftarrow S} \rceil - \lfloor G_{D \leftarrow S} \rfloor) \\ &= [b_S]\Delta G_{D \leftarrow S} \end{aligned} \tag{61}$$

If we were to store bounds on the source cluster's radiosity, we could likewise calculate the error in $\Phi_{D \leftarrow S}$ due to Δb_S, $\Delta\Phi^b_{D \leftarrow S}$, at the cost of added memory use.

We calculate the geometric error $\Delta G_{D \leftarrow S}$ when creating a link, and store it with that link. We then take its product with b_S on each iteration, and use the result to decide whether to refine the link.

We can use interval arithmetic to rewrite $\langle G_{D \leftarrow S} \rangle$ as the product of the bounds on the two projected-area calculations, first rewriting it in discretized form:

$$\begin{aligned} \langle G_{D \leftarrow S} \rangle &= \left\langle \sum_{i \in D} \left(\mathbf{S}_i \cdot \sum_{j \in S} v_{ij} \mathbf{m}_{ij} (\mathbf{m}_{ji} \cdot \mathbf{S}_j)_+ \right)_+ \right\rangle \\ &= \left\langle \sum_{i \in D} v_{ij}^{D \leftarrow} \; (\mathbf{m}_{ij} \cdot \mathbf{S}_i)_+ \right\rangle \langle v_{ij}^{S \leftrightarrow D} \rangle \left\langle \sum_{j \in S} v_{ij}^{S \leftarrow} \; (\mathbf{m}_{ij} \cdot \mathbf{S}_j)_+ \right\rangle \\ &= \langle A_D^{vis}(\langle \mathbf{m}_{DS} \rangle) \rangle \langle v^{S \leftrightarrow D} \rangle \langle A_S^{vis}(\langle \mathbf{m}_{SD} \rangle) \rangle \end{aligned} \tag{62}$$

where

$$\mathbf{m}_{ij} = \int_{i \in D} \int_{j \in S} \mathbf{m}(d_i \leftarrow s_j) \qquad [\text{m}^{-1}], \tag{63}$$

and $A_D^{vis}(\mathbf{m})$ is the visible projected area of cluster D in direction $\mathbf{m}$, and $A_S^{vis}(\mathbf{m})$ that of cluster S in the opposite direction. The term $v_{ij}^{S \leftrightarrow D}$ accounts for all occlusion that takes place between the two clusters, but is not caused directly by either one, $v_{ij}^{D \leftarrow}$ represents any self-occlusion by cluster D, and $v_{ij}^{S \leftarrow}$ does the same for cluster S.

While we have a way of calculating strict bounds on $A_D^{vis}(\mathbf{m})$ and $A_S^{vis}(\mathbf{m})$ for a single direction $\mathbf{m}$ (from **Section 4.3.7**), we do not have an analytical way to do this for a range of directions, $\langle \mathbf{m} \rangle$. (In this case we are interested in the bounds over the range of directions between the two clusters.) We can instead use the standard Monte-Carlo solution to the problem of estimating these bounds by using an appropriate number of samples of $\mathbf{m}_{sd}$ [Lisc94]; this seems to work well.

4.4.3. Refinement

Given that the bounds on the transfer indicate we should subdivide, which of the source and destination clusters should we pick to subdivide? One way to solve this problem is to decide which cluster contributes most to the error in E. If we have a function $F(x, y)$, and it is separable so that $F(x, y) = G(x)H(y)$, the amount of error due to x and y, respectively, is $\Delta G(x)\overline{H(y)}$ and $\overline{G(x)}\Delta H(y)$. If we ignore the error due to m_{ij}, then, and define

$$Q_i = \sum_j (\mathbf{m}_{ij} \cdot \mathbf{S}_j)_+ \tag{64}$$

and vice versa, then the geometric error due to the source is

$$\Delta G_S = (\lceil Q_j \rceil - \lfloor Q_j \rfloor)\frac{\lceil Q_i \rceil + \lfloor Q_i \rfloor}{2} \tag{65}$$

and vice versa. We then choose the cluster with the largest error to subdivide.

This approach is somewhat more accurate than the traditional "choose the patch with the largest projected area". It has the advantage that it is more likely to subdivide a cluster that has a small projected area in the direction of the transfer but a large relative error. In particular, it promotes the subdivision of clusters on the silhouette of an object.

4.4.4. Interreflection

We must consider the special case of a radiosity transfer link between a face cluster and itself, the so-called "self-link". This link measures internal reflection. Traditionally this is calculated by using the same process as a link between two separate clusters. This tends to fail badly for clusters containing any kind of coherent surface. Interreflection is closely tied to self-shadowing, so we can hope to use the bounds on the visible projected area to help.

Most hierarchical radiosity methods ignore micro-scale interreflection. They do this because in its standard form, hierarchical radiosity is a top-down algorithm, in that solution starts at the coarsest representation of the scene, and proceeds to more detailed scene representations as judged necessary; a process known as refinement.

A problem with this approach is that it is hard to estimate the interreflection of a cluster successfully. One approach is to take the amount of form-factor "left over" from 1 after interactions with external nodes are taken into account as an upper bound on the amount of internal interreflection. If the external interactions are small (and this can easily be the case — consider any surface illuminated solely by a small but powerful light source), this can badly overestimate the amount of internal interreflection.

With our visible projected area bounds, we can hope to do better. If we consider a large sphere enclosing a cluster, as in **Figure 53**, then the total form-factor from that cluster to the sphere should account for all possible external interactions the cluster could have. The total internal form-factor is then simply the remainder of this from 1. This is as above, only now we are accounting for all possible external interactions, and not just those that actually exist.

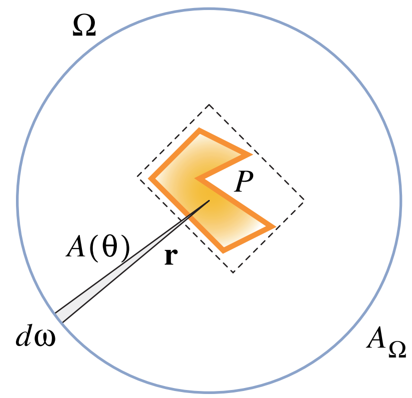


Figure 53: Calculating the external form factor. The projected area of cluster P in direction θ is integrated over the enclosing sphere Ω having surface area A_Ω.

Thus we can write the internal form-factor of cluster P as

$$
\begin{aligned}
F_{P \leftarrow P} &= 1 - F_{P \leftarrow \Omega} \\
&= 1 - \frac{A_\Omega}{A_P} F_{\Omega \leftarrow P}
\end{aligned} \tag{66}
$$

and, if the cluster has projected area $A(\theta)$ in direction θ, we have

$$F_{\Omega \leftarrow P} = \frac{A_P^{vis}}{A_\Omega}, \tag{67}$$

where A_P^{vis} is the *visible external area* of P, and is defined as

$$\begin{aligned} A_P^{vis} &= \int_\Omega \frac{A(\theta)}{\pi r^2} r^2 d\omega \\ &= \frac{1}{\pi}\int_{-\pi}^{\pi}\int_0^{2\pi} A(\theta)\cos\theta d\phi d\theta \end{aligned} \tag{68}$$

It is a straight-forward exercise in integration to use **Equation 67** to show that:

- For a planar cluster, $A(\theta) = A_P(\cos\theta)_+$, and thus $A_P^{vis} = A_P$. This gives us $F_{\Omega \leftarrow P} = A_P / A_\Omega$, and hence $F_{P \rightarrow P} = 0$.
- For a cluster that radiates evenly in all directions (for instance, a sphere), $\mathrm{A}(\theta) = A_X$ (where A_X is the cross-sectional area[1]), $A_P^{vis} = 4A_X = A_P$, and thus again $F_{P \rightarrow P} = 0$.

Both of these are as we expect: neither surface features any self-shadowing and thus interreflection.

We can use our lower bound on the visible projected area, $(\mathbf{S}_P \cdot \mathbf{m})_+$, to establish an upper bound on the internal form-factor. We find that, similar to the planar case, $A(\theta) = (\mathbf{S}_P \cdot \hat{\mathbf{n}}_\theta)_+$, $\lfloor A_P^{vis} \rfloor = \|\mathbf{S}\|$, and thus

$$\lceil F_{P \leftarrow P} \rceil = 1 - \frac{\|\mathbf{S}_P\|}{A_P}. \tag{69}$$

If the cluster P contains a completely flat surface, $\|\mathbf{S}_P\| = A_P$ and $\lceil F_{P \leftarrow P} \rceil = 0$ as we might expect.

For the lower bound, it is possible to show that, for three orthogonal vectors $\mathbf{x}_{0\ldots2}$ spanning an octant of $\Re^3$, and with corresponding projected area samples $D_{0\ldots2}$, the integral of the projected area over that octant is:

$$A_{\text{octant}}^{vis} = \frac{\pi}{4}(D_0 + D_1 + D_2). \tag{70}$$

---

1. The cross-sectional area of a sphere is πr^2; its total surface area is $4\pi r^2$.

Thus, for our standard complete set of six directional samples, the integral over the sphere is

$$\lceil A^{vis} \rceil = \sum_{i=0}^{5} D_i = D_{sum}. \tag{71}$$

If our cluster contains the usual worst-case sphere, **Equation 70** will over-estimate the total surface area as $6D_0$ rather than $4D_0$, i.e., $(3/2)A_\Omega$, leading to a negative lower bound on the internal form-factor; thus we must be careful to clip A_P^{vis} to A_P, so that:

$$\lfloor F_{P \leftarrow P} \rfloor = 1 - \frac{max(A_P^{vis}, A_P)}{A_P}. \tag{72}$$

Using more samples per cluster would help reduce the overestimated visible surface area, but at the cost of added memory. (As our clusters are constructed to be as planar as possible, we think the existing error is acceptable.)

Proof of Equation 70

Assume without loss of generality that our orthogonal sample directions are the x, y and z axes, and that our area samples in those directions are a, b and c respectively. Then

$$\begin{aligned} \lceil A(\hat{\mathbf{v}}) \rceil &= a\hat{\mathbf{v}}_x + b\hat{\mathbf{v}}_y + c\hat{\mathbf{v}}_z \\ \lceil A(\theta, \phi) \rceil &= a\cos\theta\sin\phi + b\cos\theta\cos\phi + c\sin\theta \end{aligned}, \tag{73}$$

and

$$\begin{aligned} A_{\text{octant}}^{vis} &= \int_0^{\frac{\pi}{2}}\int_0^{\frac{\pi}{2}} (a\cos\theta\sin\phi + b\cos\theta\cos\phi + c\sin\theta)\cos\theta \, d\phi \, d\theta \\ &= \int_0^{\frac{\pi}{2}} \left((a+b)\cos^2\theta + \frac{\pi}{2} c\sin\theta\cos\theta \right) d\theta \\ &= \left[\frac{1}{2}(a+b)(\sin\theta\cos\theta + \theta) - \frac{\pi}{4} c\cos^2\theta \right]_0^{\frac{\pi}{2}} \\ &= \frac{\pi}{4}(a+b+c) \end{aligned} \tag{74}$$

4.4.5. Interreflection of Leaf Clusters

In previous radiosity methods, the total element area was used to calculate a leaf element's radiosity from the incident power, in a calculation similar to **Equation 37**. This works well when the leaf solution elements are always planar. However, problems arise when they are not, and internal interreflection occurs. Previous methods always descended to the input polygons, so that this was not a problem, but in face cluster radiosity our leaf solution elements can be face clusters.

Consider the cluster shown in **Figure 54**, which consists of a small externally-visible region, A_{vis}, and a larger internal region. Externally, it seems the cluster should act as a small flat element of area A_{vis}. On the other hand, we can let the total area of the cluster become arbitrarily large without affecting A_{vis}, by increasing the internal area, and thus decreasing our estimate of the cluster's radiosity. This seems counter intuitive, and leads to the question of how we can reconcile such a situation with the constant radiosity assumption that leads us to divide by total area.

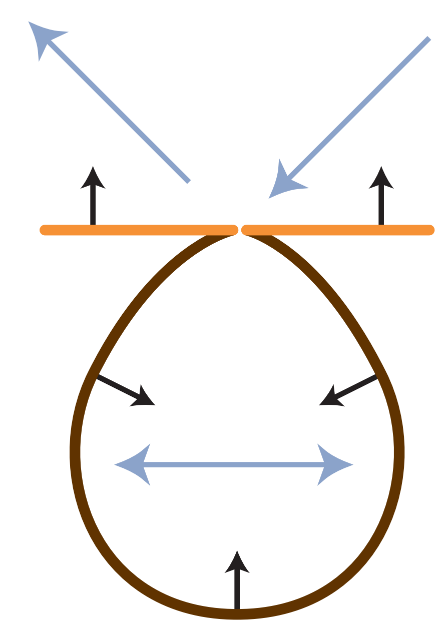


Figure 54: A cluster with a small externally visible region.

The answer is that the difference between our intuition and the calculations we are performing is that we are ignoring potential internal interreflection. To account for this, we must use the internal form-factor of the previous section to write

$$b = F_{P \leftarrow P} b + \frac{\rho \mathbf{S}^{\mathrm{T}} \mathbf{E}}{A}. \tag{75}$$

(We will ignore emittance for the moment.) Solving for b gives us

$$\begin{aligned} b &= \frac{\rho \mathbf{S}^{\mathrm{T}} \mathbf{E} / A}{1 - F_{P \leftarrow P}} \\ &= \frac{\rho \mathbf{S}^{\mathrm{T}} \mathbf{E} / A}{1 - (1 - A_{vis} / A)} \\ &= \frac{\rho \mathbf{S}^{\mathrm{T}} \mathbf{E}}{A_{vis}} \end{aligned} \tag{76}$$

and adding back in emittance gives a new form of **Equation 37**,

$$b = \frac{\rho \mathbf{S}^{\mathrm{T}} \mathbf{E}}{A_{vis}} + e, \tag{77}$$

which satisfies our intuition. The radiosity of the example cluster will be calculated relative to its visible external area, and increasing the internal area of the cluster will have no effect on its radiosity.

Thus, to take into account internal interreflection, we substitute the equivalent cluster external area, A_{vis}, for A. We only do this at the leaf clusters, when calculating the leaf radiosities, because for any other cluster, any interreflection is taken care of by transfer links between its children. Self-links are thus unnecessary for face clusters.

4.5. Empirical Projected Area Results

To test the goodness of fit of our bounds, we ran tests on a number of polygonal models that had been face clustered. Results for two of these objects, a teacup and a pipe fitting (**Figure 55**) are presented here.

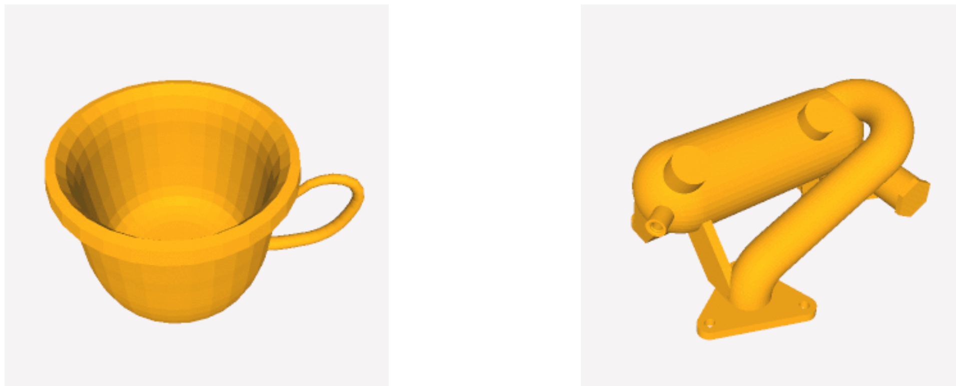


Figure 55: The two objects for which we tested projected area estimations. Left, a teacup, with 1000 polygons. Right, a pipe fitting, with 11,000 polygons.

4.5.1. Hierarchy Bounds Errors

For every cluster in the face cluster hierarchy of a model, a large number of directions distributed evenly over the directional sphere were sampled. For each direction, the actual (unoccluded) projected area, and the upper and lower bounds on the projected area were calculated. I then calculated the root-mean-square error of the bounds over all directions, as a fraction of the total surface area of the cluster. **Figure 56** shows the results plotted for both objects against the node depth. (That is, the depth of the node in the hierarchy.) As we would hope, the error over all directions decreases with depth for both objects. This trend is somewhat noisier in the case of the coarser cup model.

Figure 57 shows the results of plotting the error in the lower bound only (that due to our sum-area normal calculation.) This shows that our clusters are getting flatter and thus our bounds are getting tighter as we descend in the hierarchy. Of the lower and upper bounds, the lower seems to provide a better fit to the actual projected area, which provides some justification for only using the sum-area normal in the actual radiosity calculations. Note that the error shown here is the error between the unoccluded projected area and our lower bound; the error in the bound's representation of the visible projected area is at most this, and possibly smaller.

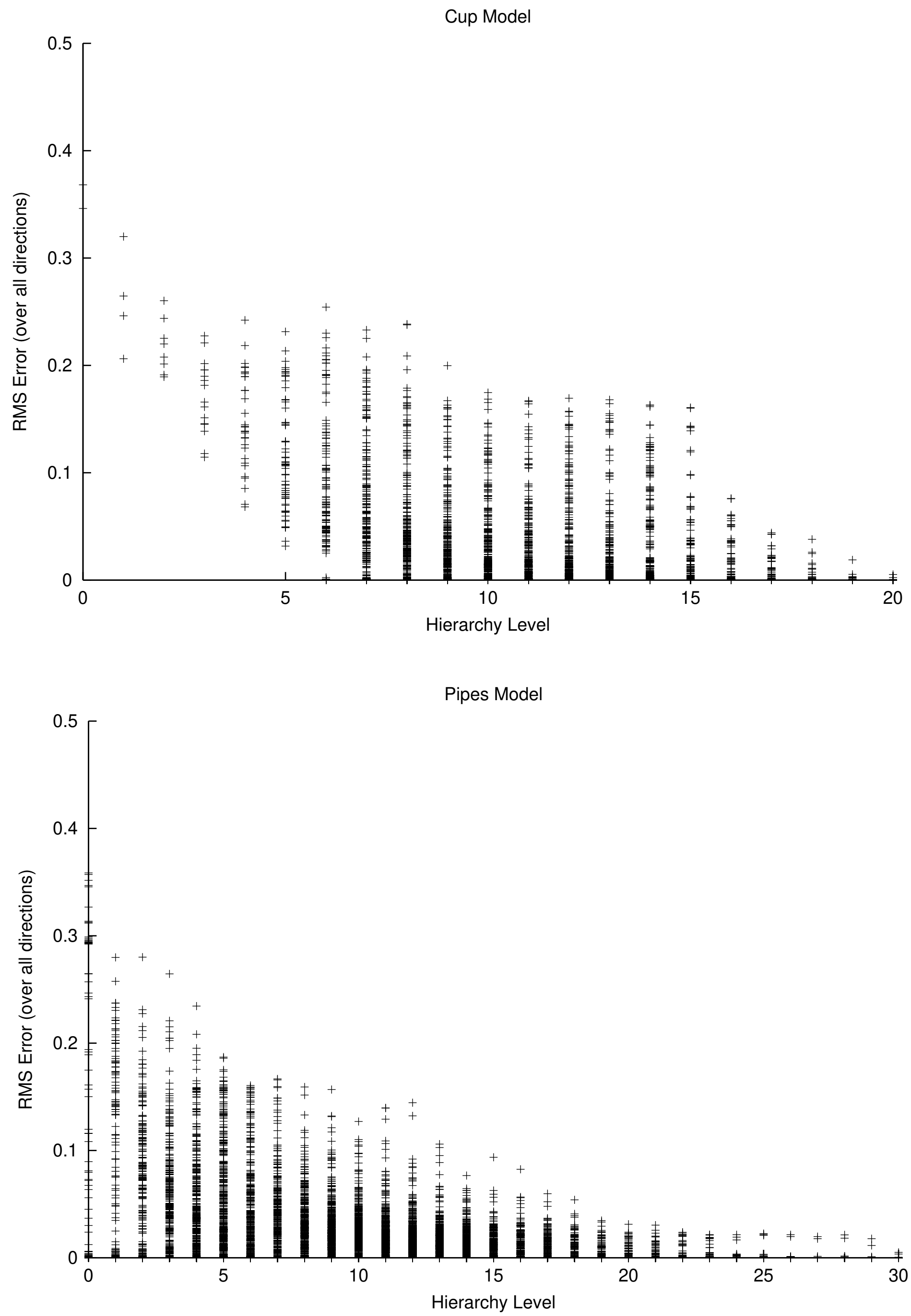


Figure 56: Performance of bounds over the entire face cluster hierarchy. The root cluster is at level zero, and the leaves are at the highest levels.

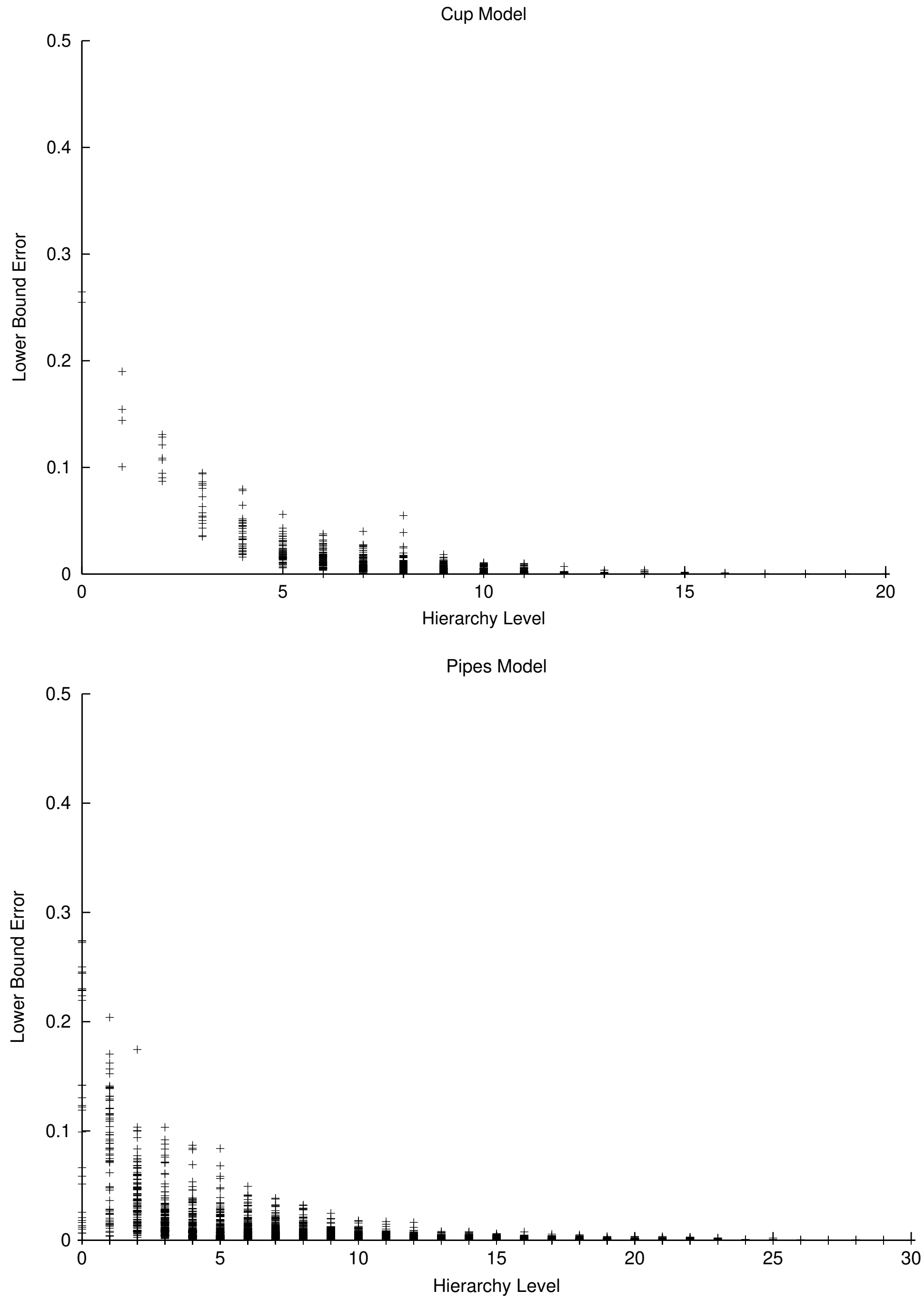


Figure 57: Error in the lower bound on the projected area. Top: for the cup mode the initially high relative error decreases with depth in the hierarchy. Bottom: for the pipes model, even at the highest levels of the hierarchy the lower bound for some clusters is exact.

4.5.2. Cluster Error Bounds

I also looked at the performance of the bounds over several individual cluster nodes, two from each model. **Figure 58** shows the relevant clusters from the cup model, and **Figure 59** shows the clusters from the pipes model. The bounds can be shown relative to the direction from the cluster by plotting them against c, the cosine of the angle between the direction and the "up" direction of the bounding box. Thus at $c = 1$ we are looking at the cluster from directly overhead, at $c = 0$, we are looking at the cluster from side-on, and at $c = -1$ from the rear. Again, we measure the error in the bounds as the difference between our upper and lower bounds, relative to the total surface area of the cluster.

Figure 58: Two clusters from the cup model. The cluster on the left corresponds to the bottom of the cup, and lies three levels deep in the hierarchy. The cluster on the right is the immediate parent of that cluster, and is considerably less planar in character.

Figure 60 shows the results for the cup clusters, and **Figure 61** the results for the pipes clusters. In both cases, while the error curve is large for the less directional clusters, for the smaller clusters it flattens out considerably. We would expect that, as we get deeper in the hierarchy, projected areas for $c \leq 0$ will go to zero, as the clusters get flatter.

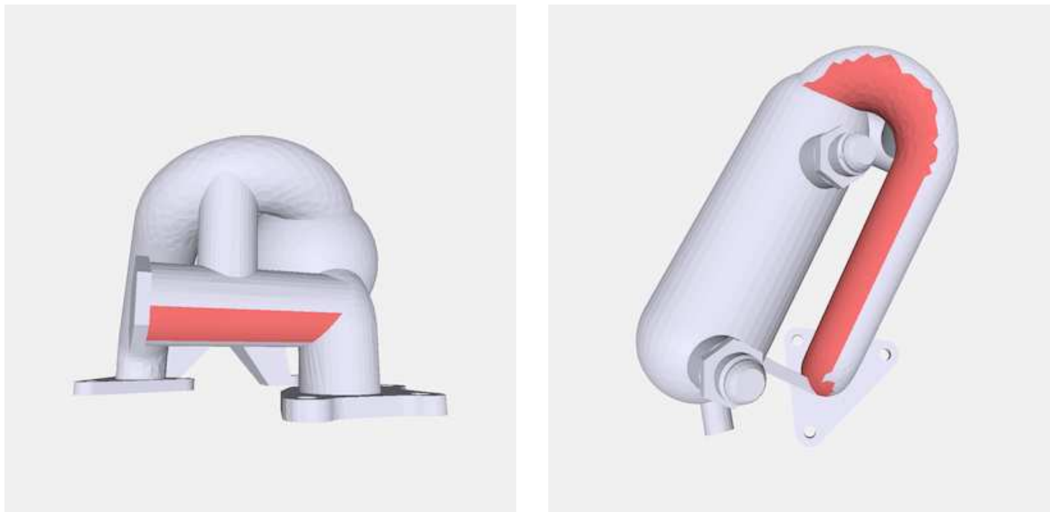


Figure 59: Two clusters from the pipes model. On the left, a smaller, flat cluster from level five of the hierarchy, and on the right, a larger cluster from level three.

Figure 62 shows how the upper and lower bounds behave as the cluster gets flatter. The lower bound tends to be accurate both immediately in front of and behind the cluster, and most inaccurate "side-on" to the cluster, around $c = 0$, where its assumption that the cluster is perfectly flat proves false. The upper bound, on the other hand, has greatest error outside this side-on region, although its error also goes to zero (though not nearly as quickly) at the front, side and back ($c = 1$, $c = 0$ and $c = -1$) of the cluster, because this is aligned with our VPA sample directions. Thus, luckily, the regions of error for both estimates cancel out to some extent.

4.5.3. Projected Area Functions

While graphing these results makes it easier to directly compare the various projected area estimates, for a more intuitive idea of what is happening, we can plot the various estimates in three dimensions as $PA(\hat{\mathbf{n}})$, that is, the projected area over the sphere of all directions. For the "cup bottom" cluster, **Figure 63** shows such plots for the actual projected area of the cluster, as well as our bounds for that area. The actual projected area function extends mainly downwards, with a

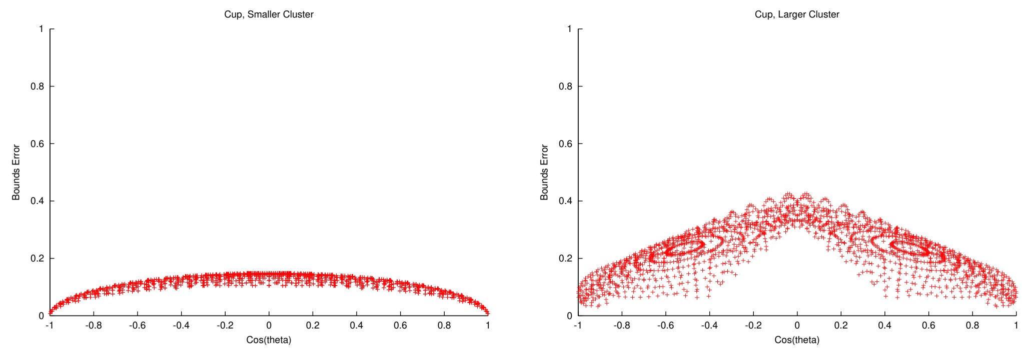


Figure 60: Error in the bounds for the cup clusters.

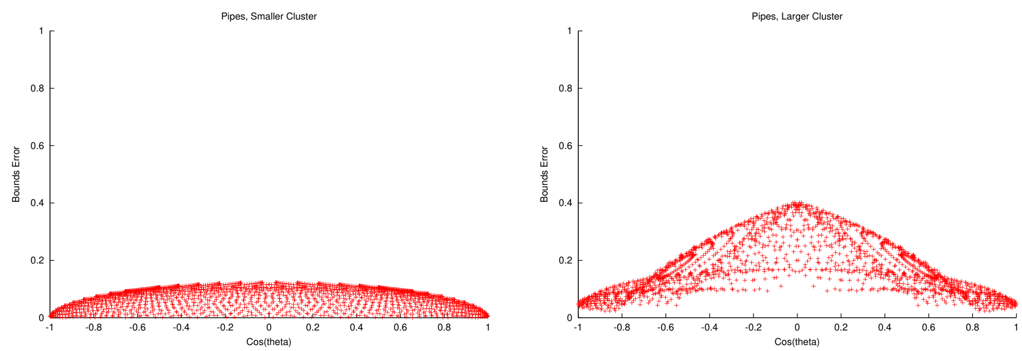


Figure 61: Error in the bounds for the pipes model's clusters.

little ring around the horizontal, and an indentation at the top where the projected area goes to zero as we move directly behind the surface. The lower bound is a simple cosine lobe in the direction of the cluster, while we can see that the upper bound consists of one large lobe in the direction of the cluster, with four much smaller lobes corresponding to the side areas of the cluster. The sixth lobe, corresponding to the sixth area sample, is non-existent, due to the zero projected area behind this cluster. The final image in the sequence shows how closely the two bounds enclose the actual projected area in this case.

Figure 64 shows the same results for the larger, decidedly non-planar cluster which covers part of the side of the cup as well. While the upper bound is much looser for this cluster, indicating the need for further subdivision, note that the lower bound still gives a surprisingly good fit to the function.

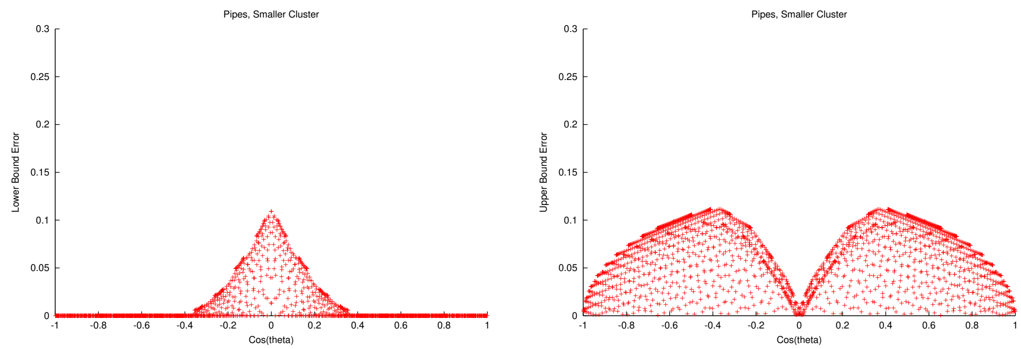


Figure 62: Upper and lower bound error for the small pipes cluster. The errors in the two bounds offset each other for largely flat clusters.

4.5.4. Upper Bound Strategies

I investigated how my preferred strategy of finding an upper bound for a cluster's projected area, by interpolating projected area samples from the axes of its OBB, compared to two other possible strategies. Firstly, we could take samples along the axes of an axis-aligned bounding box; this saves us having to transform the sample direction into the coordinate system of the OBB. Comparing to this strategy is also a good measure of how important picking "good" sample directions is. Secondly, we could follow the approach of previous researchers, and use the projection of the bounding box itself as an upper bound on the projected area of its contents [Stam97c].

Figure 65 compares these approaches for the two clusters from the pipes model. For the large cluster, both sampled approaches clearly outperform the projected bounding box approach, and the oriented sample approach is marginally better than the axis-aligned one. For the small, more planar cluster, the oriented sample strategy does much better than the axis-aligned one; picking good sample directions becomes more important for such a highly directional cluster. The projected bounding box approach deals particularly badly with angles "behind" the small cluster, because it takes no account of face orientation.

4.5.5. The SUPA as a lower bound on the VPA

There were a number of reasons given in section **Section 4.3.3** as to why the signed unoccluded projected area, SUPA, should only rarely overestimate the visible projected area, VPA. However, we would like to show some empirical sup-

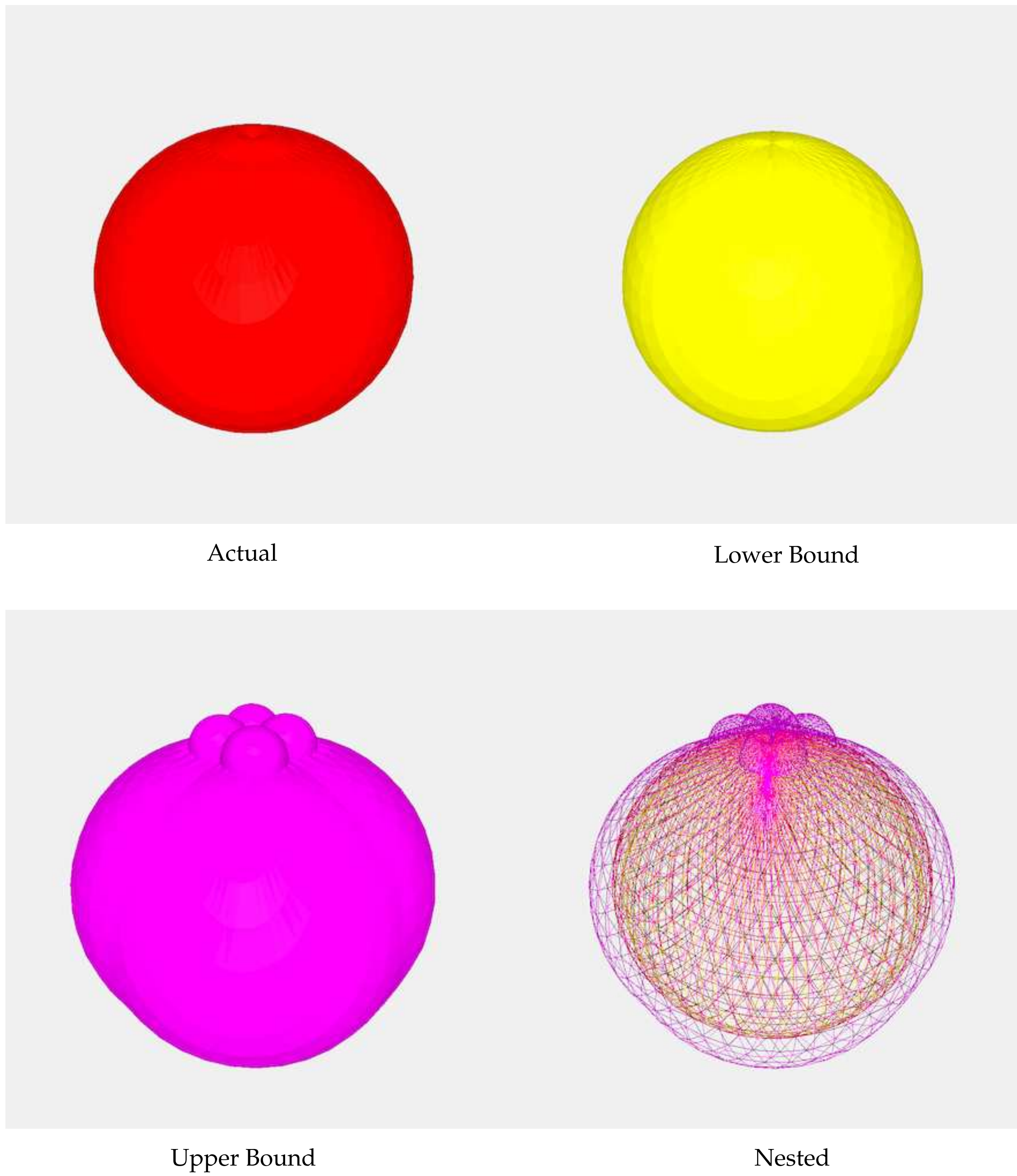


Figure 63: Projected area as a function of direction, cup bottom. From left to right we see the actual projected area of the cluster, the sum-area normal estimate of the area, and the sampled estimate. The last image shows all three functions together, illustrating how they nest correctly.

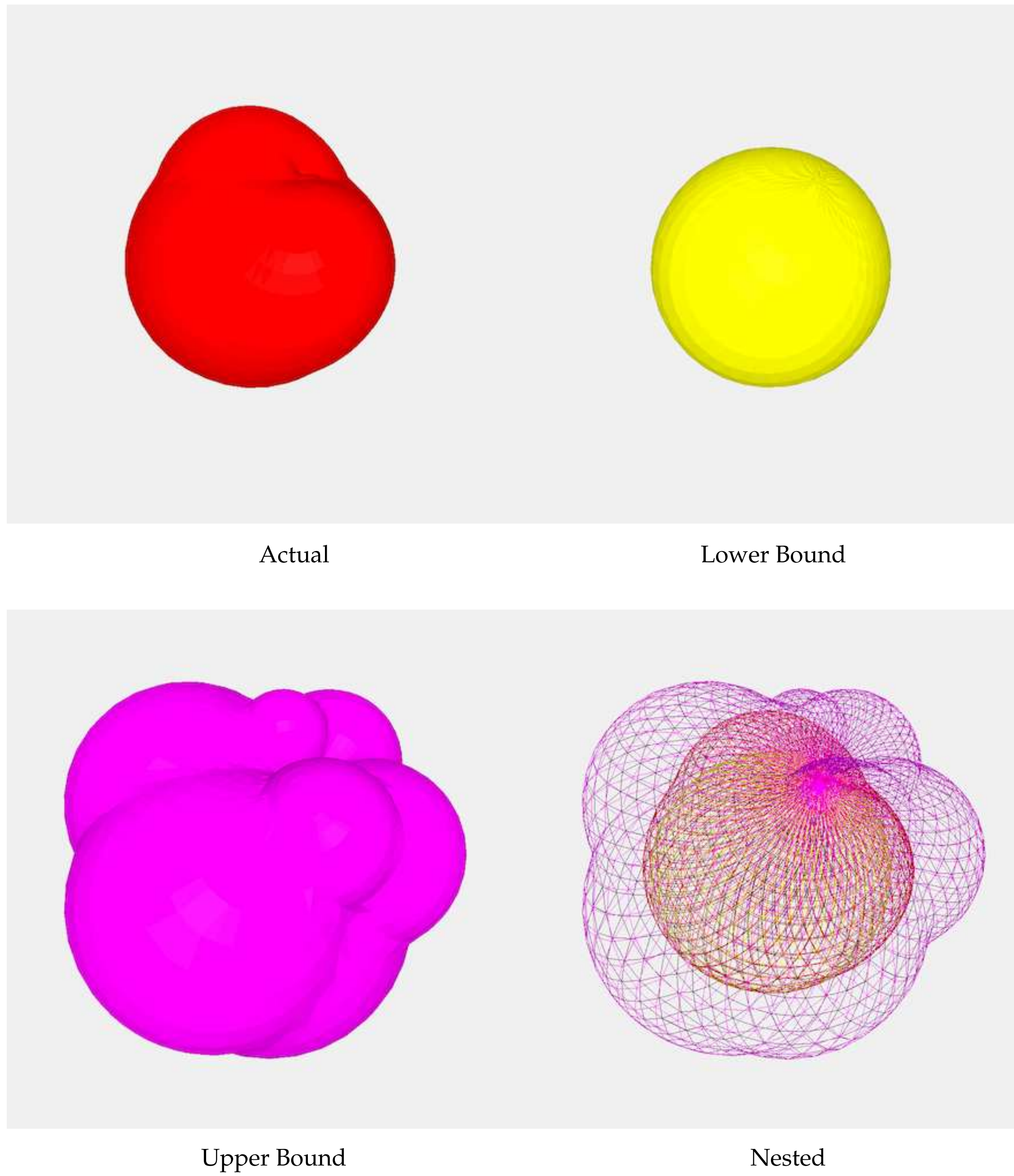


Figure 64: Projected area as a function of direction, cup side. The cup side approximates a flat surface less well, and hence the fit of the lower bound in particular is looser than that for the cup bottom.

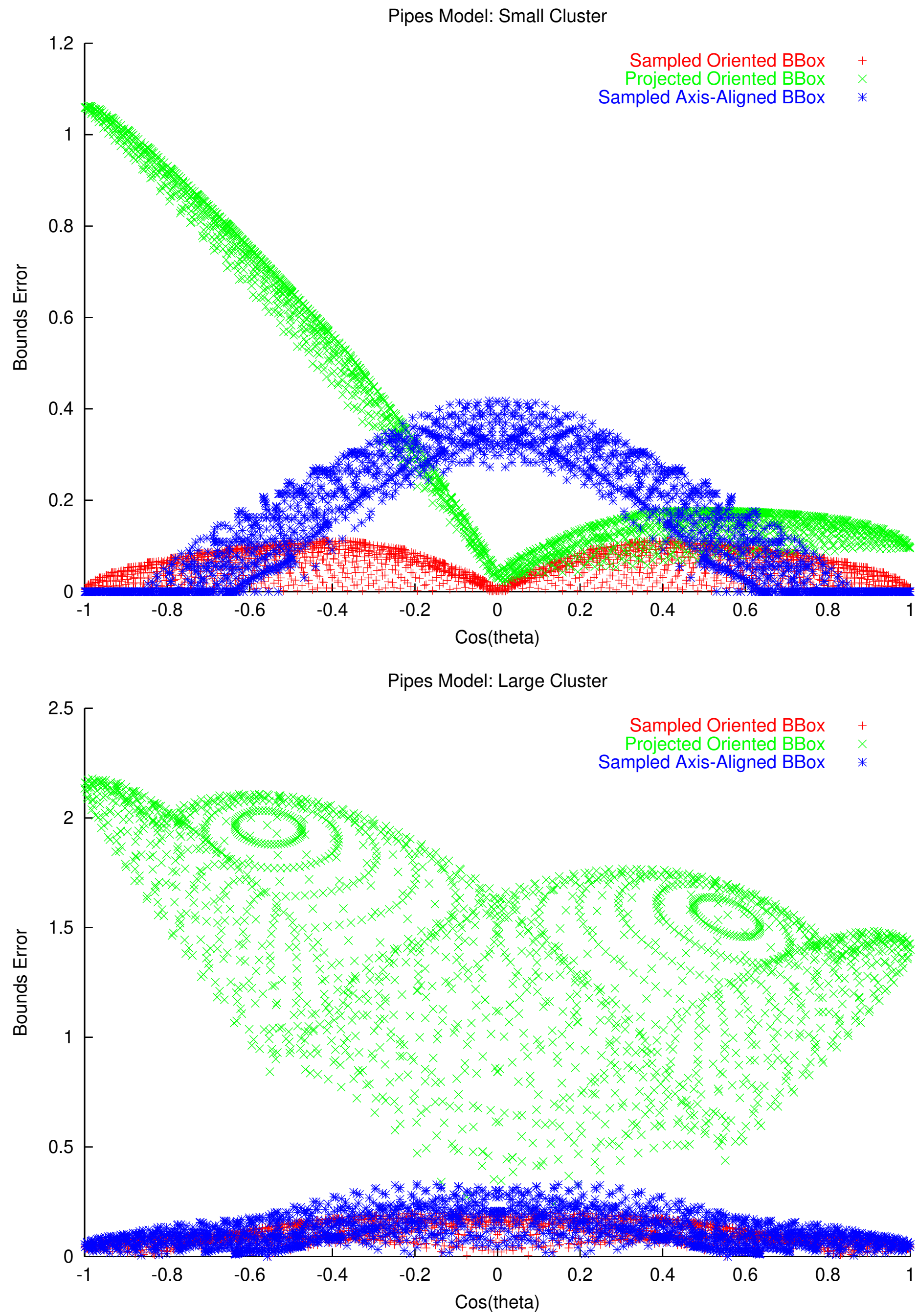


Figure 65: Different approaches to finding the upper bound. Top, errors for the small pipes cluster, and bottom, errors for the large pipes cluster. Using sampled projected areas along the axes of the oriented bounding box of the cluster gives better results than using samples on an axis-aligned bounding box, or the projected area of the oriented box.

port for these observations. To this end, we can test a number of models and their associated cluster hierarchies in the hope of finding the following:

- How often the SUPA fails to be a lower bound on the VPA. (We will refer to this as a bounds exception.)
- By how much it fails. Even if it only happens occasionally, if the SUPA is sometimes wildly wrong, its value as a lower bound estimate is much reduced. If it is close to the VPA even in situations where the bound is violated, however, its value is high.
- What kind of situations it fails in.

Unfortunately, accurately finding the visible projected area is more difficult than for the various other projected area metrics. We could employ analytic methods to calculate the silhouette of the cluster from a particular direction exactly. However, doing this for a large number of direction samples, and for potentially every cluster in a real model with a large number of faces, could take weeks of computation time. We must strike a balance between the accuracy of our VPA calculation, and the ability to test a large number of clusters for potential SUPA bounds violations.

I chose to use sampling to estimate the VPA, as follows. For each face in the cluster, a single ray was traced from the face's centre in direction $\mathbf{m}$. If the ray did not hit another face in that cluster, the face's projected area was added into the total. This approach is not ideal, as the resulting estimate of the VPA can underestimate the actual VPA, due to a face being counted as totally occluded when it is partially unoccluded. Thus when we test the SUPA against this estimate, there is a chance the SUPA will be larger than the VPA due to error in the estimate, rather than because it is actually larger than the VPA. Again, this is a compromise between total running time and accuracy in measuring errors in our lower bound; as it stands, each model test required the casting of over 1.5 billion rays, and a number of hours to run.

Using this technique I measured $\Delta x = (\mathrm{SUPA} - \mathrm{VPA})/A$, namely, the difference between the SUPA and VPA as a fraction of the cluster's total surface area, over a large number of directions[1], for every cluster in a number of models. Whenever this number was positive, a bounds exception was registered, and the direction and cluster involved were logged. Also, the maximum Δx for each cluster, $\Delta x_{\max}$, was logged.

The results for three models, each containing 100,000 polygons, are shown in **Table 4**. (The models can be seen in **Figure 79**.) Because some number of the

1. 484 directions distributed evenly over the sphere.

positive values of Δx will be occurring because of inaccuracies in the VPA estimate, and because we are more concerned with larger violations, exception counts are shown for a number of thresholds. Overall the results are as we might hope: a very small number of exceptions for each model, and none of these particularly large. The Buddha model suffers from the most exceptions, and the Isis model the least. This makes some sense, as the Buddha has a number of holes and areas of sharp curvature, whereas the Isis is reasonably smooth, and has no holes. No model has more than 0.2% exceptions where $\Delta x > 0.01$, that is, where the error is more than one hundredth of the cluster's surface area.

The average maximum exception is also shown, along with the largest such maximum. Clusters with less than 16 faces were ignored for the purposes of calculating the largest maximum, as clusters with so few faces suffer from considerable error in the VPA estimate. These largest maxima tend to be reasonable close to the average. Overall, we can conclude that bounds exceptions, when they do occur, are acceptably small.

Finally, one such bounds exception, from the Buddha model, is shown in **Figure 66**. Almost all exceptions examined where some error is obvious show this general pattern.

Model	Percentage of Bounds Exceptions			Avg. $\Delta x_{\max}$	Max. $\Delta x_{\max}$
	$\Delta x > 0.0$	$\Delta x > 0.001$	$\Delta x > 0.01$		
Buddha	0.52%	0.44%	0.20%	0.0043	0.066
Dragon	0.38%	0.32%	0.14%	0.0034	0.087
Isis	0.24%	0.18%	0.05%	0.0015	0.068

Table 4: Statistics for lower bound exceptions: $\text{SUPA} > \text{VPA}$.

4.6. Summary

In this chapter we have analysed the calculation of radiosity transfer between face clusters, and derived useful bounds and error estimates for our technique. Our bounds are almost always conservative (that is, not violated), and in practice we have found our bounds to be tighter than those previously published. Due to the visible projected area rule, using the sum-area normal vector $\mathbf{S}$ in our radiosity calculations can give surprisingly good results. Recapping:

- A good estimate of a cluster's *visible projected area*, the VPA, is critical to my algorithm.

Figure 66: An example of one of the larger SUPA/VPA bounds violations from the Buddha model. The surface spur marked on the left is obscured in the rotated view shown on the right, but is still counted in the SUPA, leading to an exception. (An orthographic projection was used for both pictures.)

- $(\mathbf{m} \cdot \sum \mathbf{S})_+$, which I call the signed unoccluded projected area, or SUPA, is an approximate lower bound on the VPA.
- In my algorithm, this bound is rarely violated, because of the tendency of face clusters to be flat and well-shaped.
- The lower bound may only be violated when a cluster's projected boundary has regions with a winding number greater than one.
- As an upper bound, we can sample the non-negative unoccluded projected area, the NUPA, in six axial directions corresponding to the bounding box, and interpolate these samples. This bound is conservative.

Both bounds are constant time to evaluate in the number of polygons a cluster contains. They rely on preprocessing of a cluster, which

- for the lower bound is constant time to calculate from its children.

- for the upper bound is linear in the number of polygons in the cluster—no worse than the requirement for the calculation of bounding box extents.

Chapter 5

Improving the Face Cluster Construction Algorithm

5.1. Introduction

In this chapter I explore ways to improve the performance and quality of the original face clustering algorithm presented in **Section 3.3**, and in Garland's dissertation [Garl99]. The face clustering algorithm is a vital part of the radiosity algorithm described in this thesis. For our goal of handling complex scenes on a standard workstation to be achieved, it is imperative that we be able to produce the face cluster hierarchies required by the radiosity algorithm on such a workstation. To achieve this goal, I have reimplemented Garland's original face clustering algorithm, and made a number of changes to improve both general quality and performance, as well as suitability for use with radiosity. The resulting implementation seems similar in speed to the "oconv" octree builder of Radiance [Warda].

I will begin by discussing the various improvements that can be made to the algorithm, present some performance results, and conclude by discussing some other work on clustering strategies for polygonal models.

5.2. Speed

5.2.1. Evaluating Running Time

We start by evaluating the running time of Garland's cluster algorithm. We split this time into two operations, both of which can be carried out independently; construction of the actual hierarchy, and calculation of bounding box extents. In the following sections we shall show how these results can be improved.

Face Cluster Hierarchy Construction

The results of profiling the face cluster algorithm described in **Chapter 3** for a large, 100,000 polygon model, are shown in **Table 5**. Only four routines contribute significant computational cost; all the others contribute less than 1% each to the overall running time.

Time	Pctg.	Routine	Description
9.2s	55%	Jacobi	Find eigenvectors for PCA
4.8s	30%	FindCost	Calculate cost term for a dual edge
1s	6%	MergeEdgeLists	Merge the dual edge lists of two clusters
0.6s	4%	InitFromFace	Create initial leaf cluster
0.7s	5%	-	All other routines
16.3s	**100%**		

Table 5: Profiling Garland's face cluster algorithm

By far the most time spent in the algorithm is on the Jacobi eigensolver routine used to calculate the principle components of each cluster [Pres92]. This routine, the simplest possible for the problem, uses Jacobi transformations to find the eigenvectors of the covariance matrix of the cluster. (As discussed earlier, these components are used both to estimate the flatness of the cluster, and to provide an orientation for the bounding box of the cluster.) Obviously this routine is a natural candidate for optimization.

Bounding Box and Side Area Calculation

After the cluster hierarchy has been constructed, we must also calculate the actual extents of the bounding boxes corresponding to each node in the hierarchy. For each cluster, a simple hierarchy traversal is performed to find the corresponding leaf polygons, and the vertices of these polygons used to calculate its extents. This is an $O(hn)$ operation in time, where h is the height of the hierarchy, and n is the number of cluster nodes. (h is $O(\log n)$ when the hierarchy is balanced, and $O(n^2)$ in the worst case.)

For use with vector radiosity, we must also calculate the projected area of the cluster in each direction corresponding to a face of the box. This can be done at the same time as the iterative bounding operation, and adds an overhead of approximately 30% over just bounding box calculation.

The times taken to calculate the bounding boxes for two cluster hierarchies are shown in **Table 6**. Also shown is the actual height of the hierarchies, and the height of each hierarchy if it were balanced.

Model	Bounding Box Time	Cluster Nodes	Tree Height	Balanced Height
Dragon	17.1s	102,264	342	17
Whale	41.44s	101,812	506	17

Table 6: Garland's hierarchy: results for bounding box and side area calculation for two models.

At first glance, it seems as if the bounding box calculation is a much more significant part than the actual hierarchy calculation of the overall time cost of creating a cluster hierarchy suitable for radiosity. These times show the bounding box calculations taking over twice as long as cluster creation in the case of the whale model. However, this need not be the case. The hierarchies produced by the original clustering algorithm can be extremely unbalanced; in the case of the results above, by balancing the hierarchies we could gain potential speed-ups of 20 and 30 times respectively. This would also have a positive effect on any query-based algorithm that might use face clusters, such as ray-tracing or collision detection, as these also have $O(h)$ complexity.

If the poor balance of these hierarchies were solely a result of the need to constrain clusters to be flat, there is not much we could do to improve the situation without seriously diluting the effect of our cost metric. Luckily, this is not the case. Much of the imbalance is due to highly tessellated flat regions in each model, where the cost metric approaches zero, and provides little guidance as to which clustering operation to perform first. In such situations the heap-based algorithm clusters faces in order, often in the most unbalanced way possible. (This is the reason the much smoother whale model is less balanced than the bumpier dragon model; see **Figure 79** for both models.) As we shall see later, we can exploit this by altering our cost metric to produce more balanced hierarchies, and reduce drastically the time taken to calculate bounding boxes.

5.2.2. Speeding up Principle Component Analysis

A first approach to eliminating the bottleneck of principle component analysis in the cluster creation algorithm is speeding up the eigensolver at its heart. There are a number of more sophisticated algorithms for the eigenvalue/eigenvector problem than the Jacobi eigensolver, which was originally chosen for simplicity of implementation. Techniques based on QR iteration with shifting work well for medium size matrices, and for large matrices, divide-and-conquer works best [Demm97, Golu89].

Unfortunately, these algorithms are more efficient the larger the problem size, and our problem size is extremely small; 3×3 matrices. For such a small problem size, it turns out that brute-force Jacobi iteration is as fast as we can reasonably expect, and has the advantage of being more stable than the other routines [Demm97]. This is illustrated in **Table 7**, which shows the average performance of several eigensolvers over a number of random box-shaped covariance matrices on a Pentium II-class processor. Jacobi takes 60% of the time of the QR shift-based algorithm, where both are hard-coded for 3×3 matrices, and both are an order of magnitude faster than a general SVD routine used for the same purpose.

Clocks	Routine
218,347	Generalised SVD
14,102	QR-Shift
8,395	Jacobi

Table 7: Average performance for various eigenvalue solvers.

As a result, it appears we cannot reasonably expect to speed up the eigensolver routine itself.

5.2.3. Eliminating PCA for Flat Clusters

An alternative to reducing the time taken by the PCA algorithm is to instead reduce the number of such computations needed. The PCA algorithm has two uses in the face cluster algorithm; determining the best fit plane of the cluster, and determining directions for the bounding box. We will examine its application to the bounding box further in **Section 5.3**.

An alternative to calculating the best fit plane of the cluster is to instead use the sum-area normal, $\mathbf{S}$, to determine the orientation of the cluster, and use that in our error metric. This has two advantages. Firstly, it is much easier to calculate, and secondly, it is much more stable than the PCA routine for calculating orientations (see **Figure 67**).

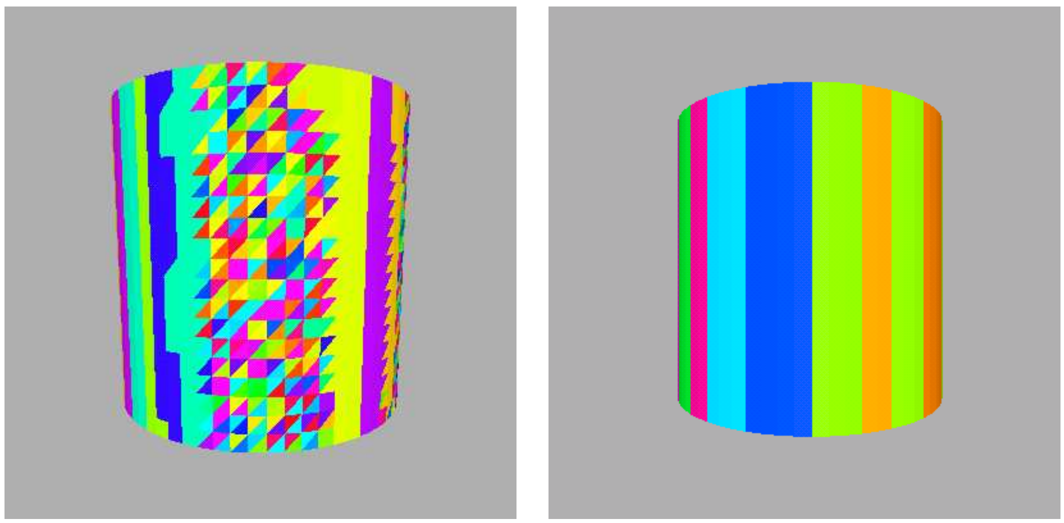


Figure 67: Determining cluster orientation. On the left is a cylinder clustered using PCA to determine the orientation of clusters. On the right are clusters created by using the sum area normal for this purpose. When clusters are reasonably flat, as in this case, doing so leads to more stable results than using PCA.

The disadvantage of using the sum-area normal is that, as clusters become less flat, the quality of the approximation to the best-fit plane falls off, and the estimate of the flatness of the cluster suffers. However, for such clusters we can revert to using the PCA algorithm. A good metric for deciding when to do so is to compare the length of the sum-area normal vector, namely, the projected area of the cluster in that direction, against the total area. This approaches one as the cluster becomes completely flat, and zero as the cluster's surface becomes closed. I have had good results with using

$$\frac{\|\mathbf{S}\|}{A} > 0.5 \tag{78}$$

as a test for whether to use $\mathbf{S}$ instead of the principle component to determine the cluster's orientation. The impact on running time is shown in **Table 8**. The cost of the PCA analysis is much reduced, and in general the performance of the algorithm is at least twice as fast.

Time	**Pctg.**	**Routine**	**Description**
4.5s	58%	FindCost	Calculate cost term for a dual edge
1s	14%	Jacobi	Find eigenvectors for PCA
1s	11%	MergeEdgeLists	Merge the dual edge lists of two clusters
0.6	7%	InitFromFace	Create initial leaf cluster
1s	10%	-	All other routines
7.8s	**100%**		

Table 8: Profiling the new face cluster algorithm

5.2.4. The Impact of Balance on Running Time

At first glance, it would appear that the balance of the hierarchy would not have an effect on the running time of the clustering algorithm. As described by Garland, the clustering and edge-collapse algorithms have a complexity of $O(n\log n)$; there are $O(n)$ merge operations, and each such operation requires updating the heap, an $O(\log n)$ operation [Garl99].

However, on closer examination I have found that balance can have a considerable impact on the running time by affecting the *degree* of clusters in the model. Each face cluster node is connected to its immediate neighbours by dual edges, representing potential clusterings. The degree of a cluster node corresponds to the number of these dual edges. Whenever two clusters are merged, we must merge their dual edge lists (a constant time operation), and visit all of the edges in the new list, updating their corresponding costs in the heap. So the cost of a merge operation is $O(d\log n)$, where d is the number of dual edges in the new cluster.

If the algorithm is producing a balanced hierarchy, the number of dual edges belonging to any given cluster is approximately equal, and constant. Thus

d can be assumed to be constant, and Garland's analysis of $O(\log n)$ for a merge operation holds. In the worst case, however, polygons are iteratively collected into a single large cluster, and this assumption is no longer true. **Figure 68** shows this effect. As a consequence, performance suffers. In fact, for a planar regular grid, the accumulation cluster can have $O(\sqrt{n})$ dual edges incident on it in the worst case. As a result, merge operations are $O(\sqrt{n}\log n)$, and the algorithm as a whole is $O(n^{3/2}\log n)$ rather than $O(n\log n)$. We can expect similar worst-case behaviour for manifold surfaces.

The impact of this on clustering time is illustrated in **Figure 69**, which shows the time taken to cluster flat triangular meshes of varying sizes. (An example mesh is shown in **Figure 70**.) The results for both a constant-cost clustering metric, which results in an almost completely unbalanced hierarchy, and an area-based metric, which results in an almost completely balanced hierarchy, are shown. The reason for the differing performance is clearly illustrated in **Figure 71**, which shows the respective maximum dual edge adjacencies for the two metrics.

(I speculate that this is also the reason for the running-time jump for the Turbine model in figure 6.4 of Garland's dissertation [Garl99]. The turbine contains a number of highly tessellated flat surfaces; when the algorithm starts to simplify them, it seems probable that this will lead to vertices of extremely high degree. I have observed similar effects on smaller flat models.)

This result provides further motivation for altering the cluster algorithm to produce more balanced hierarchies. We will discuss how this can be accomplished in **Section 5.4.1**.

5.3. Calculating Oriented Bounding Boxes

In Garland's original method, the orientation of the bounding box is calculated by performing principal component analysis, and the extents of the box by iterating over each point contained in each cluster node. We shall briefly examine both of these approaches.

5.3.1. The Need for Robust Extent Calculations

Iterating over every face in a cluster for each cluster in order to calculate the extents of its bounding box is costly. In the best case (the hierarchy is balanced) it is $O(n\log n)$; in the worst, $O(n^2)$. A much cheaper approach that has been used elsewhere would be to calculate the extents from just the vertices that define the bounding boxes of its two child clusters, rather than every point in the cluster. This reduces bounding time to a constant per cluster, and overall time to $O(n)$.

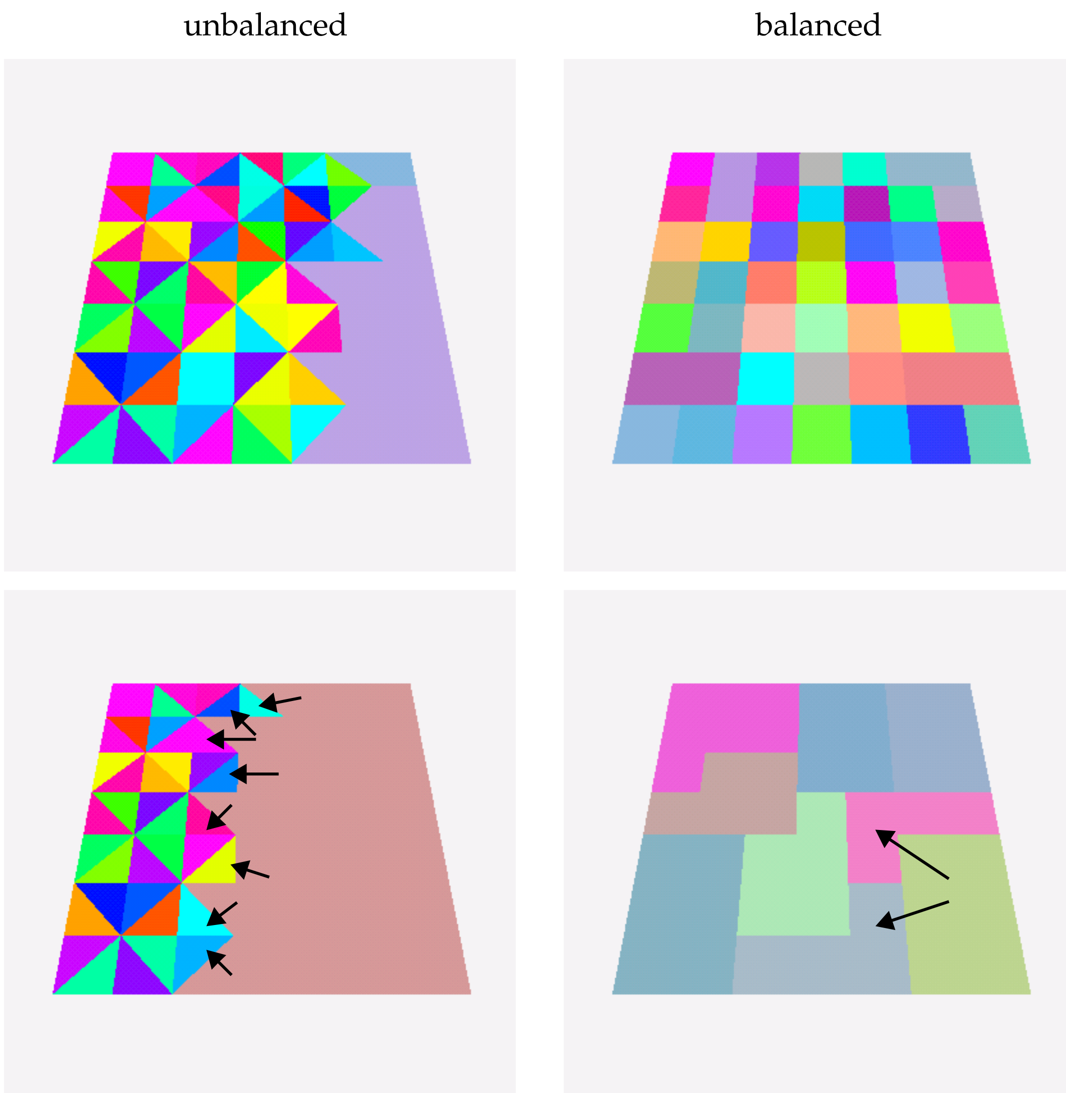


Figure 68: Hierarchy balance affecting dual-edge degree. On the left, in an unbalanced hierarchy, one large cluster slowly absorbs all other clusters (from top to bottom.) On the right, a balanced hierarchy leads to roughly similar-sized clusters as merging proceeds from smaller clusters (top) to larger ones (bottom).

A balanced hierarchy leads to a small, approximately constant number of dual edges per cluster (arrows, bottom-right), while in an unbalanced one, a single large cluster is adjacent to many small ones, leading to a non-constant dual edge count (arrows, bottom-left).

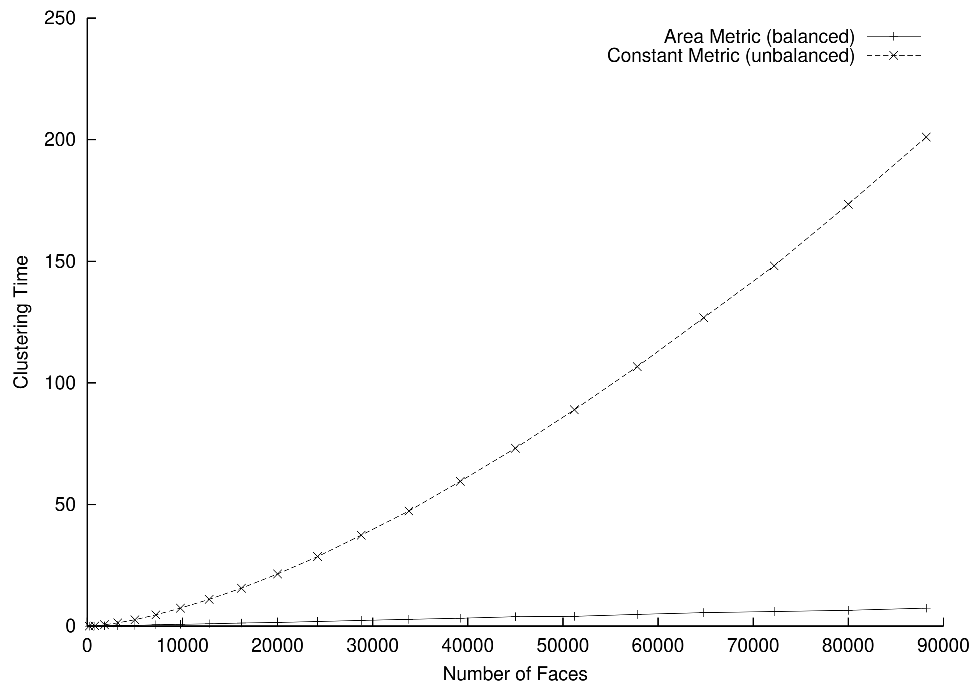


Figure 69: The effect of hierarchy balance on clustering time. An unbalanced hierarchy can make clustering very slow.

Unfortunately, in such an algorithm a little bit of error is potentially introduced in each bounding operation, typically making the parent cluster larger than it really has to be. Worse, the error accumulates up the tree, so that nodes near the root have very sloppy bounds. This effect is especially bad for the large models (and consequently large hierarchies) we must deal with. As the radiosity algorithm is most sensitive to error at the top of the hierarchy, due to its top-down refinement approach, any error, and especially any cumulative error, is unacceptable.

Any approximation algorithm we do choose to speed up bounding box calculation must be carefully designed to avoid cumulative approximation error.

5.3.2. Picking Bounding Box Orientations

A major problem for the use of face cluster hierarchies in radiosity simulations is that principle component analysis can fail to produce a reasonable bounding box

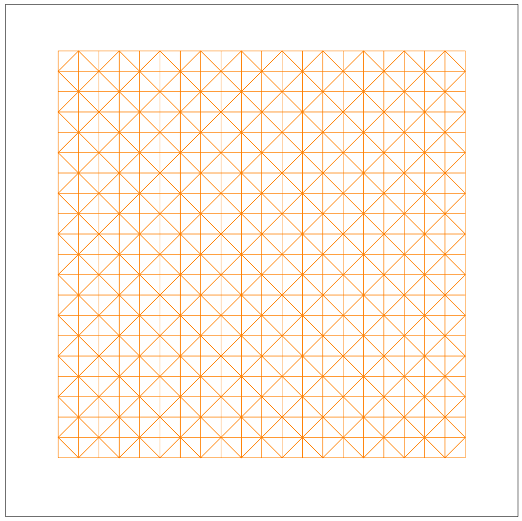


Figure 70: An example flat mesh of 800 faces.

in degenerate cases. While it handles objects with three differently-sized major axes well, if any two of these axes are close to equal, the algorithm becomes unstable. Worse yet, these are cases that occur commonly in radiosity scenes.

The problem with the principle component approach is that it is in essence fitting an ellipsoid to a cloud of points, and then taking that ellipsoid's principal axes as those of the oriented bounding box for the cluster. The worse case for this approach is a cube, where there is no preferred direction at all for the fit ellipsoid (the best fit being a sphere), resulting in an essentially random orientation for the bounding box. Because the algorithm used to calculate the eigenvectors of the covariance matrix is iterative, it is difficult to predict what this orientation will be.

While cubical objects don't occur often in practice, their two-dimensional equivalent, meshes in a rectangular shape, do. This leads to poor bounding boxes, as in **Figure 72**.

In some applications this is not critical. However, in radiosity the quality of our transfer approximations, and in particular the quality of our visibility calculations, is highly dependent on the tightness of the bounding boxes. Consider a floor in a square room; if the bounding box for the entire floor is oriented at 45 degrees to the floor, much of it will lie outside the walls of the room. Visibility tests against this bounding box will show the floor is partially occluded, even

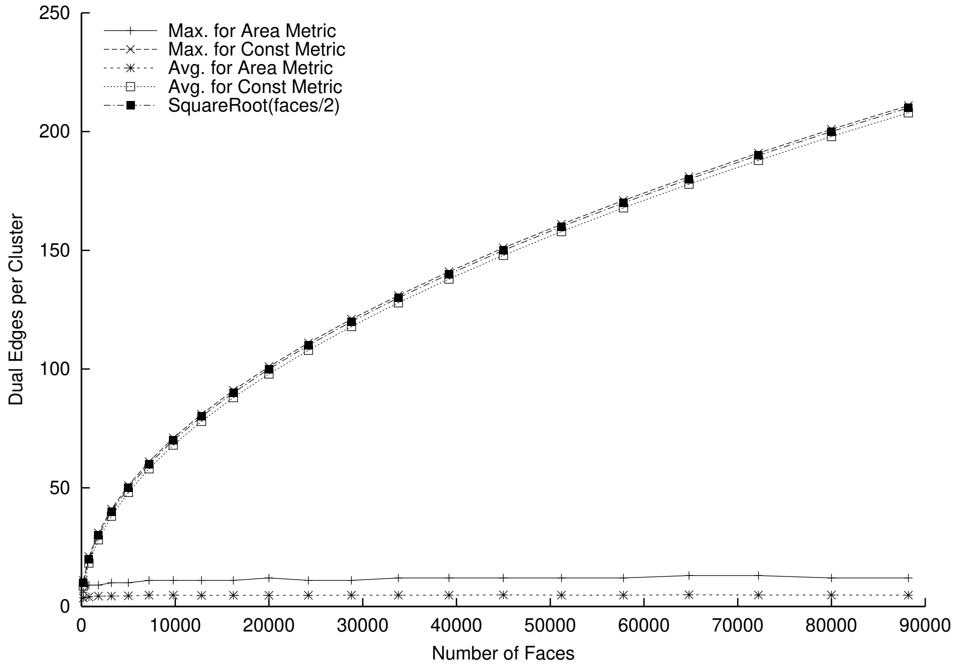


Figure 71: Dual edge degree increases when creating unbalanced hierarchies.

though in reality it is not occluded at all. Such problems lead to poor refinement decisions and unnecessary work on the part of the radiosity algorithm.

It is possible to calculate the optimal minimal-volume bounding box for a cloud of points in 3D, but this takes $O(n^3)$ time [ORo85']. (Principal component analysis is $O(n)$, and of course, we get it largely for free due to needing it also for our cost metric calculations.) This is far too expensive for our situation and geometry size.

A hybrid approach is to use PCA to establish a single principal axis, and then, after projecting all vertices onto a plane perpendicular to that axis, use a simpler algorithm to find the minimum-area enclosing rectangle (MER). The cost of finding the MER is $O(n \log n)$ for finding the two-dimensional convex hull, and $O(n)$ for finding the rectangle using a rotating callipers-type algorithm [Tous83]. (Briefly; it can be shown that a minimum-area enclosing rectangle must have one side coincident with an edge of the hull of the point set being enclosed. We can thus iterate through all such rectangles in linear time, and pick the one with the

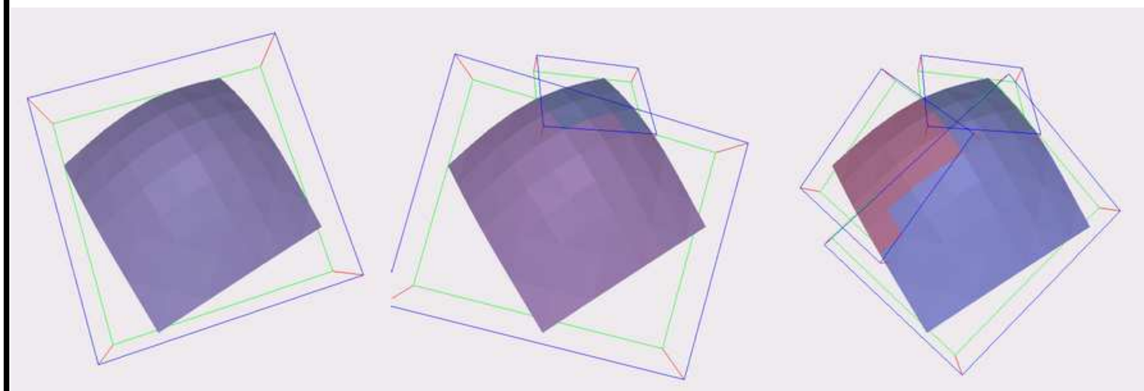


Figure 72: Badly oriented bounding boxes. From left to right we see the first three levels of subdivision of a square but curved surface. Because the clusters are close to symmetrically shaped, their bounding boxes have essentially random directions in the plane.

smallest area.) In practice we can merge hulls as we ascend the hierarchy, rather than creating a new one each time; the cost of merging two hulls is $O(n)$.

This approach works particularly well for flat clusters. For larger, non-flat clusters, the smallest-length component is no longer the most stable orientation feature of the cluster, and just using PCA works as well or better than this hybrid approach. Luckily, this meshes well with the approach to eliminating the need for PCA on flat clusters described in **Section 5.2.3**. For clusters meeting the condition of **Equation 78**, the $\mathbf{S}$ vector is used both to define a fit plane for the cluster, and to define the "flattest" axis of the bounding box, with the minimal-area enclosing rectangle algorithm used to find the remaining two axes. For all other clusters, PCA is used both for the fit plane, and to define the orientation of the bounding box. This approach works well, especially because it limits application of the MER algorithm to those clusters that genuinely need it for stability, and saves on the expense of PCA at the same time.

5.4. The Cost Metric

Garland's original cost metric, as outlined in **Equation 12**, sums three penalty terms to form the total cost of merging two clusters. These per-dual edge costs are then used to decide which two clusters to merge next.

This formulation suffers from a couple of problems. The first is that, because the cost terms are added, they can wind up interfering with each other in some situations. For instance, the original face cluster radiosity paper used weights on the three terms of $w_{fit} = 1.0$, $w_{dir} = 20.0$, and $w_{shape} = 1.0$ for some models, in an attempt to get the directional term to have the desired effect [Will99]. On further investigation, it turned out that the directional and shape costs interfered with each other—turning on the shape metric pretty much destroyed the effect of the direction metric.

Another problem is that the three cost terms have different units. E_{fit} is the variance of the cluster's points in one dimension, so it has units of area. Scaling the model by a factor s will lead to scaling of all associated fit costs by s^2. The E_{dir} term measures the variance of cluster normals from some average normal. This varies from 0, when all normals point in the same direction, to 1, when the normals are distributed randomly over the sphere of directions. Thus it is unitless. Finally, E_{shape} is also unitless, and varies from 0 (when two elongated clusters are merged into a fatter, more rounded one) to 1. Unlike the directional term, however, it tends to usually fall around 0.5, when no improvement to shape is made.

Thus, scaling the model can lead to different ratios of the cost terms, and thus different clustering behaviour, a situation we would like to avoid. We would also like to avoid the situation where the size of one term prevents the others from taking effect. To do this, I propose a modified cost metric:

$$E = E_{fit} \cdot (P_{dir} \cdot P_{shape}) \tag{79}$$

This metric works by taking E_{fit} as the base cost and scaling it according to the penalty terms P_{dir} and P_{shape}, which we will define to be 1 when both terms should have no effect, and to rise above 1 for clusters deemed "bad" by both terms.

5.4.1. Addressing Balance

As discussed previously, Garland's original metric does not produce particularly balanced hierarchies. In the case of flat sections of a mesh, moreover, E_{fit} goes to zero everywhere, and we get extremely unbalanced hierarchies.

There is a simple fix to this problem. If we consider a flat mesh made up of evenly-sized polygons for the moment, a cost metric that would produce a balanced cluster hierarchy for such a mesh is just the surface area of the cluster. Such a metric ensures that clusters are merged evenly in order from small to large, producing a balanced hierarchy.

Of course, this approach only works if all leaf elements are approximately the same size. However, empirically this is the case for most detailed models. Also, where it is not the case, it usually suits the radiosity algorithm better to have approximately equal-area clusters, as this ensures that the depth of hierarchy that must be descended to reach a certain level of accuracy is relatively constant.

We must combine this fix for flat meshes with the usual E_{fit} metric in such a way that the changeover from curved surfaces, where the standard fit metric should predominate, to flat surfaces, where the area-based metric should predominate, is smooth. The approach I use is to combine them as follows:

$$E'_{fit} = E_{fit} + k \cdot area \tag{80}$$

Here the constant k controls the mix of the two metrics. As the variance of the points (roughly speaking, the height of the cluster squared) drops below k times its surface area (again, roughly, the cross-sectional area), balance of the hierarchy starts to take precedence over the flatness of clusters. Note that E_{fit} and $area$ both have units of distance squared. **Figure 73** shows the effects of this new fit term on hierarchy balance.

5.4.2. The Directional Term

The calculation of Garland's E_{dir} term is expensive, especially in terms of memory use, which it doubles. Also, its attempts to minimise variation in normals is not so well suited to our radiosity method. We are generally happy to have large amounts of micro-scale normal variation, as long as the surface is relatively flat, and does not fold over on the macro scale. A cluster can meet both of these conditions, and still be heavily penalised by Garland's variance-of-normals E_{dir} term, because of small bumps on the surface.

An alternative is to use the ratio of the projected area in the direction of the area-weighted normal to the total area of the cluster as a measure of how much it folds over. (This is just $\|\mathbf{S}\|/A$.) In the case of a completely flat cluster this is one; as the cluster folds over and the projected areas on each side of the fold cancel out, it drops, reaching zero in the case of a completely closed surface. We can thus define the penalty term of the previous section as

$$P_{dir} = \frac{area}{projArea}. \tag{81}$$

The major advantage of this is that, while it gives similar results to the original E_{dir}, as seen in **Figure 74**, there is no required additional storage cost, as we must

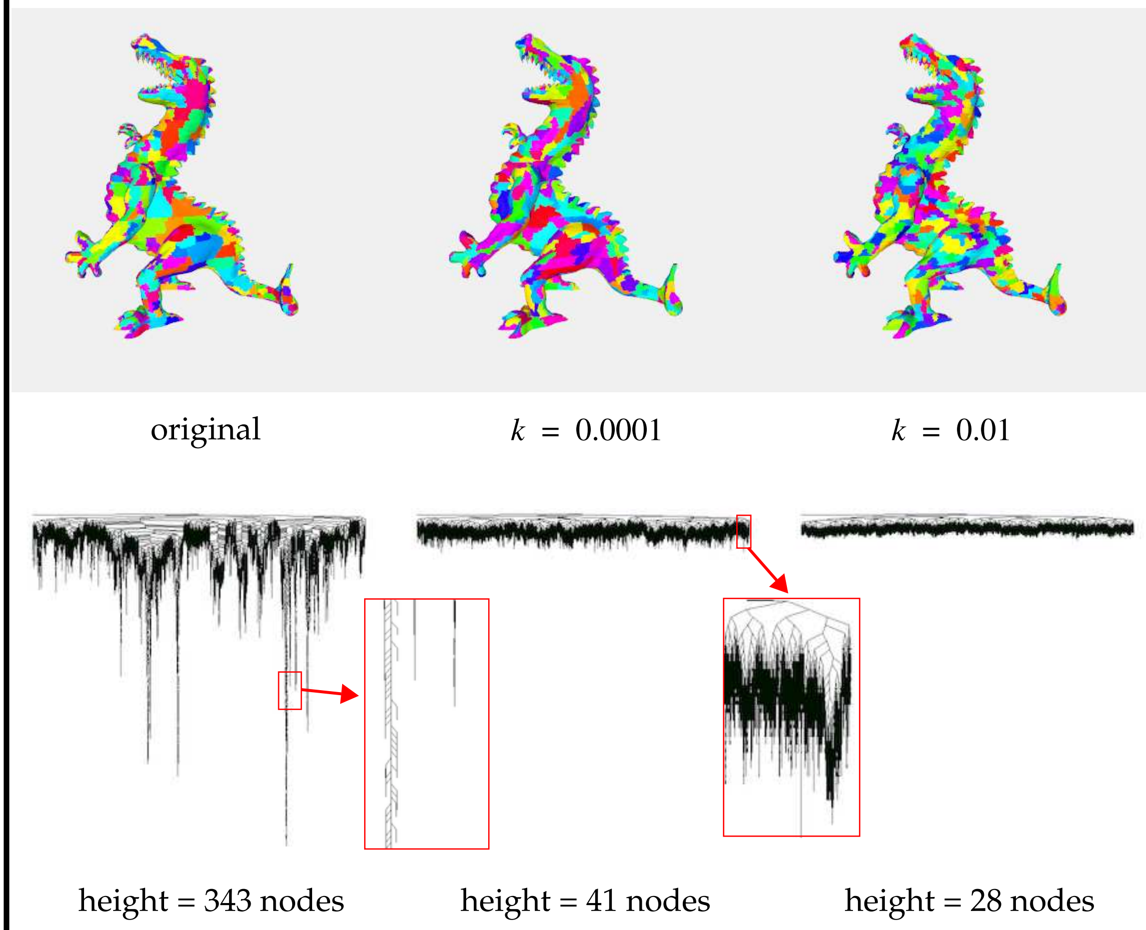


Figure 73: Enforcing hierarchy balance. Top: a dragon model clustered using, from left, the original cost metric, and the modified metric for two values of k. Bottom: the corresponding cluster hierarchies.

already track the area and sum area normal of each cluster. This can lead to space savings of almost 40% over Garland's original method, and can make the difference between being able to cluster the dragon model in under a minute on a 128MB laptop, and having to wait half an hour for results while it thrashes heavily.).

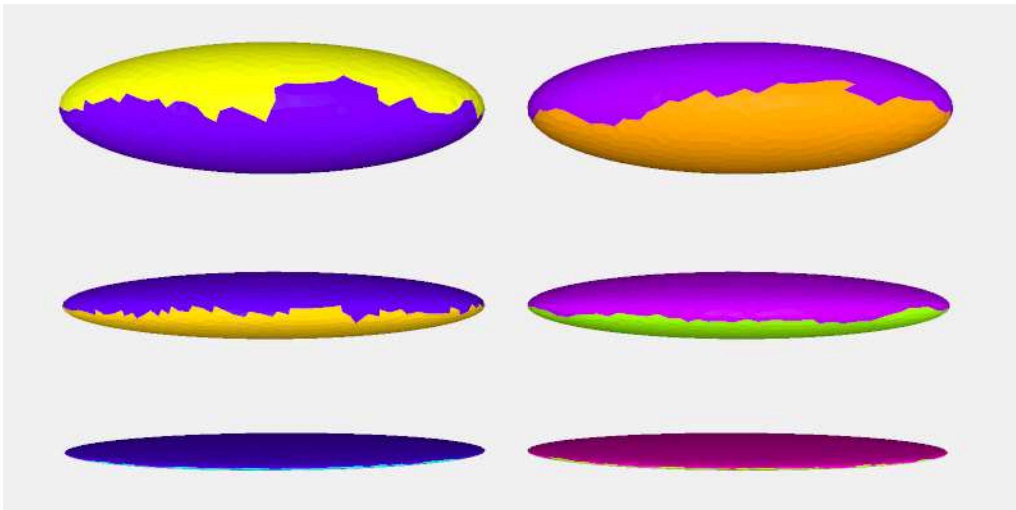


Figure 74: The directional term. Top to bottom: the first level of the cluster hierarchy for successively thinner ellipsoids. Left, the original E_{dir} metric based on normal quadrics. Right, the new metric based solely on projected area.

Ideally, we would like this first split in the hierarchy to divide the ellipsoid cleanly into top and bottom halves. For this particular example, we see that the new metric is no worse (and arguably a little better) than the original metric.

5.4.3. The Shape Term

In tests I found that the difference between using the γ of the cluster (the ratio of squared perimeter to area) and Garland's more complicated E_{shape} term seemed to be minimal; see **Figure 75**, for instance. The γ term works better with the scaling penalty approach outlined above; we can define the shape penalty simply as:

$$P_{shape} = \frac{perim^2}{4\pi \cdot area} \tag{82}$$

Again, this is one for a perfectly circular cluster, the most compact shape possible, and increases as the cluster's shape moves away from that ideal.

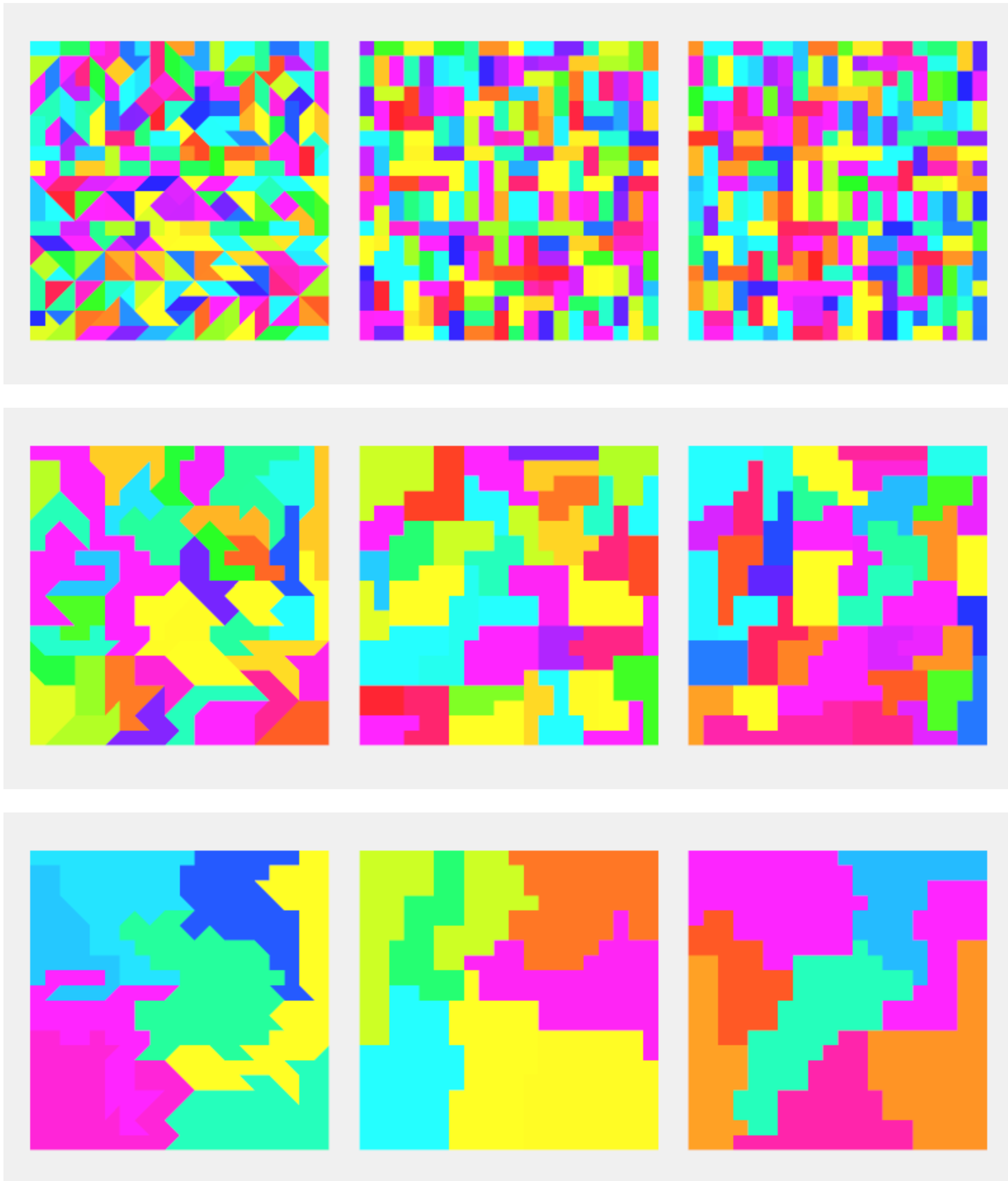


Figure 75: Effect of the shape term. Top to bottom: progressively coarser clusterings of three flat meshes. Left, no shape term; middle, the new term; right, Garland's E_{shape}.

5.4.4. A New Cost Metric

Putting it all together, we can write the complete cost term as:

$$(eFit + k \cdot area)\left(\frac{area}{projArea}\right)\left(\frac{perim^2}{4\pi area}\right) \tag{83}$$

which can be rewritten more simply as

$$\text{cost} = (eFit + k \cdot area)\frac{(perim)^2}{projArea} \tag{84}$$

where as usual, $projArea = \|\mathbf{S}\|$. (We can drop the 4π because the cost term is scale independent.) The k term controls the mix between balancing the hierarchy and obeying the $eFit$ metric; it should be set to some small value (I use 0.0001) for good results.

This metric has proven to be simple and robust, and much less sensitive to the effects of conflicting purposes for the metric. For instance, even if an area-weighted term is added to Garland's original "summed" metric, its effect on balance is easily overwhelmed by the $eDir$ and $eShape$ terms. The approach of scaling the basic cost, $eFit + k \cdot area$, by penalty terms, suffers much less from this.

Finally, while trying to construct a cost metric capable of guiding face cluster creation according to some notion of "goodness", it is tempting to engage in micro management, adding a succession of terms. One must be careful about this; the face cluster algorithm is a greedy algorithm, and trying to get subtle effects via the relatively blunt tool of the cost metric can be a frustrating exercise. Keeping the metric simple and straightforward seems the best approach.

5.5. Time and Space Trade-Offs

The cluster files for complex models can be large. For instance, the geometry of the whale model shown earlier in this chapter requires 3MB of storage (1.1 MB compressed). Its corresponding binary cluster file (which includes the geometry as well as the face clusters) is 15MB (9MB compressed.) While these files can be compressed for storage, they must be uncompressed for use by the radiosity simulation. This is not really a problem for moderate numbers of cluster files given the current cost of hard disk space; approximately US$5/GB for IDE disks. However, while I make an effort to limit the amount of cluster overhead for my particular application, radiosity, it would be nice to have a method for reducing the size of these files in situations where either we wanted to store more information per cluster, or many (different) models must be used. Also, there are some situations in which we must traverse the entire face cluster hierarchy; in such cases, overly large files can lead to long disk read times, and problems with thrashing.

We look at two possible approaches to trading off cluster file size in return for more computation time: increasing the branching factor of the hierarchy, and truncating the hierarchy.

5.5.1. Picking a Branching Factor

The standard face cluster hierarchy is a binary tree; it has a branching factor of $b = 2$. It is relatively simple to recast this hierarchy as one with a larger branching factor; $b = 4$ would give us a quadtree, and $b = 8$ an octree. This can lead to space savings, but we must analyse the corresponding impact on performance.

Let us briefly recap the situation where we wish to build a hierarchy with a branching factor of b above n leaf polygons. The total number of internal nodes (and thus clusters) will be

$$i = \frac{n-1}{b-1}. \tag{85}$$

The height of the corresponding balanced hierarchy is

$$h_{bal} = \left\lceil \frac{\log n}{\log b} \right\rceil + 1, \tag{86}$$

and that of the corresponding unbalanced hierarchy is $h_{unbal} = i + 1$.

Ignoring storage for leaf nodes, which remains constant, the storage required by our hierarchy is $S = (S_{node} + bS_{pointer})i$, where S_{node} is the internal node storage (e.g., a bounding box for a bounding volume hierarchy), and $S_{pointer}$ is the storage size of a pointer. The query time for each node is simply bQ_{node}, where Q_{node} is the time taken to test a single node—we must test each of a node's b children[1]. Thus the worst-case query time is

$$Q_{bad} = bQ_{node}(h_{unbal} - 1) = \frac{b}{b-1}Q_{node}(n-1), \tag{87}$$

1. For a bounding volume hierarchy we must test each child because there is a chance more than one child will match.

and the best-case time is

$$Q_{best} = bQ_{node}(h_{bal} - 1) = \frac{b}{\log b} Q_{node} \log n . \tag{88}$$

In the case of the face cluster hierarchy used by the radiosity algorithm, S_{node} is an order of magnitude larger than $S_{pointer}$, so we approach space savings of $1/(b-1)$ over a binary tree. Thus, a quadtree will take $1/3$ the space of the original binary tree, and an octree $1/7$ the space, ignoring the storage for leaf faces. These are considerable space savings, and they increase with b.

The drawback to a higher branching factor is slower spatial-data structure queries. It is often assumed that a randomly-built hierarchy has height $O(\log n)$, and thus is on average close to being balanced. We will assume that here, especially as we have a specific term to promote balance in our cost equation. (See **Section 5.6** for empirical evidence that our hierarchies are always close to the balanced case.) In such a case, it is Q_{best} that determines query performance. **Figure 76** shows the b-dependent term of Q_{best}. For quadtrees and higher, the cost increases close to linearly with b. Interestingly, quadtrees have the same query cost as binary trees, and a hierarchy with a branching factor of three has a slightly lower cost[1].

This analysis assumes we have built a hierarchy with branching factor b from scratch, guaranteeing that every internal node except one has b children. However, constructing an algorithm that directly produces such a face cluster hierarchy is challenging. We would have track all possible b-way grouping operations on adjacent nodes, rather than using simple bi-directional links. For a node with m neighbours, there would be $m!/((b-1)!(m-b+1)!)$ possible groupings. For example, for $m = 6$ and $b = 4$, we would need to represent (and calculate the cost of) 20 potential groupings rather than 6. Also, if we start from a triangular mesh, for the initial leaf nodes $m \le 3$, and many internal nodes would have only m children, with $m < b$.

In general it is simpler to recast the binary hierarchy produced by the original algorithm into one with a higher branching factor. In doing so, we must maintain the structure of the original hierarchy, so that only groups of adjacent nodes are produced. If this is not done, we could have clusters containing disconnected surface areas, violating the assumptions of the face cluster hierarchy.

1. A three-way hierarchy is only 5% faster in this situation. In general binary trees work better because of the extra cost of three way group-selection or comparison operations. A binary comparison requires fewer gates to implement that a trinary one.

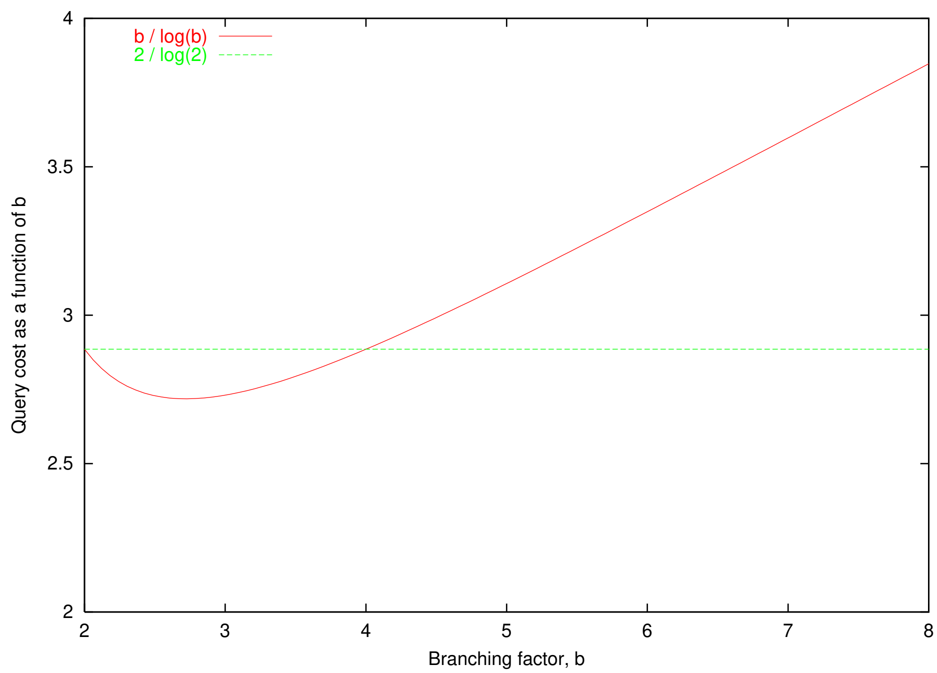


Figure 76: The b-dependent factor in the query cost for a balanced spatial data structure with a branching factor of b.

In practice this leads to the requirement that a node and its children in the transformed tree correspond to the root node and leaf nodes of a valid subtree of the original binary hierarchy. This restriction can produce a number of internal nodes that have less than b children, with two consequences: potentially wasted pointer space on one hand, but fewer nodes to test against on the other.

I will not go into details here, but it can be shown that:

- the smallest possible number of children in such a tree is $b' = b/2 + 1$;
- any completely balanced or completely unbalanced tree can be transformed into one where all but one of the internal nodes have the full set of b children;
- in the worst case, it is possible for all internal nodes in the new hierarchy to have b' children;

- the maximum height of such a worst-case tree is $2(n-1)/b+1$, and the number of internal nodes is $2(n-1)/b$[1].

The consequences of this are that the best-case storage and performance are the same as for our original analysis, but in the worst case performance and pointer storage costs of the transformed tree match that of the original binary tree. However, the total internal node storage is reduced by a factor of $2/b$, so we still save some space with a higher branching factor. For $b = 4$, whereas we were originally guaranteed to reduce our internal node storage to $1/3$ of its original size, with the transformed tree this is just the best case, with the worst case being $1/2$.

In summary, by increasing the branching factor we can decrease the storage needed for face clusters, ignoring pointer storage, to at worst $2/b$ of that required by the original binary hierarchy. This can increase the query time of the face cluster data structure by a factor of up to $b/(\log b)$. However, this is important only if we are using the data structure for visibility queries, or proximity or collision detection. In my current implementation of face cluster radiosity, the data structure is only used for hierarchical radiosity, and a nested grid data structure is used for visibility queries. Thus the only performance consideration is that of the radiosity algorithm, which we will look at next. For face cluster applications which are primarily query-based, there is no performance drawback and considerable space savings to be had by recasting the hierarchy as a quadtree. Higher branching factors are possible, but the space savings become progressively smaller and performance is impacted; finding the ideal branching factor would require further experiment or analysis.

5.5.2. The Impact of Branching Factor on Radiosity

For the face cluster radiosity algorithm, we must examine the impact of the branching factor on the following stages of the radiosity algorithm (see **Section 2.4.2**).

Gather

The cost of the gather operation in radiosity depends solely on the number of transport links currently existing in the simulation. This number depends on the refinement epsilon, ε, as well as the quantization of surfaces—how we discretize them, and how we organize them into a hierarchy. The branching factor affects

1. Briefly, the smallest binary tree corresponding to this worst case contains a left subtree that is unbalanced and has b′ leaf nodes, and a right subtree that has b/2 leaf nodes. Larger such trees can be produced by repeating this base tree.

the set of possible links between hierarchical surface regions. Larger branching factors correspond to a smaller set of possible links, that cover the possible transport space more coarsely. If there is some theoretically optimal number of transport links l_ε that exactly meets the desired refinement threshold, the imposition of any hierarchy leads to approximation error; we still meet the threshold, but it requires l_ε^b links, where $l_\varepsilon^b > l_\varepsilon$. The larger the branching factor, the larger the approximation error, and thus l_ε^b.

Unfortunately, given the wide range of transport schemes, and the dependence of transport error on the current state of the radiosity solution in the popular brightness-weighted schemes, the relation of these quantities is hard to analyse except in trivial, unrealistic cases. We can make the following simplistic analysis, however.

Assume that all nodes have an equal number of links pointing to them (**Section 2.4.1**). We will compare a hierarchy with branching factor $b = 2^m$ to one with branching factor 2, when both have the same n leaf nodes. From **Equation 85** the binary hierarchy must have $(n-1)((b-2)/(b-1))$ more nodes than the hierarchy with higher branching factor. For each of these nodes, the corresponding links must be pushed down to the first descendent node that has a counterpart in the non-binary hierarchy. This must result in the generation of at least one new link, and at most 2^{m-1} new links, for each of the links to be pushed down. Thus we can conclude that, assuming a constant refinement ε and the same scene, the links formed in a hierarchy with branching factor b will have the following relation to the number of links formed in a binary hierarchy:

$$l_\varepsilon^b \geq \left(1 + \frac{b-2}{b-1}\right) l_\varepsilon^{b=2}. \tag{89}$$

This increase in the number of links, and consequently link storage and gather time, quickly approaches a factor of two for branching factors of eight and higher.

Push-Pull

The cost of a push-pull operation for an internal node is linear in the branching factor. (We must push irradiance down to b children, and then average their radiosities during the pull phase.) We can write this cost as $C_P b$. For each leaf there is an additional constant cost in transforming irradiance into radiosity, C_R. This gives us a total cost for a push-pull operation over the solution hierarchy of

$C_P bi + C_R n$. In the balanced and unbalanced cases, where almost all nodes have b children, we have

$$C_{best} = k_P \frac{b}{b-1}(n-1) + k_R n, \tag{90}$$

and in the worst case mentioned above, where all nodes have b' children, we find

$$C_{worst} = k_P \frac{2b'}{b}(n-1) + k_R n = k_P \frac{(b+2)}{b}(n-1) + k_R n. \tag{91}$$

Thus by increasing the branching factor, we can decrease the cost for the internal nodes, by a maximum factor of 2. The cost for a binary tree is $2k_P(n-1) + k_R n$, and for an octree it lies between $1.143k_P(n-1) + k_R n$ and $1.25k_P(n-1) + k_R n$.

We can conclude that for face cluster radiosity, increasing the branching factor would decrease cluster memory use, and speed up the push-pull section of the radiosity algorithm, but likely increase transport link memory use, and slow down the gather section of the algorithm. My feeling is that a low branching factor at the top of the hierarchy is important, as accuracy at the top of the hierarchy is crucial to any top-down algorithm. This hypothesis needs to be tested empirically, however. In the meantime, it seems that increasing the branching factor to at least four is an experiment well worth pursuing, especially considering that previous hierarchical radiosity algorithms typically have branching factors of four or more, and that face cluster storage is typically much greater than link storage. An alternative to increasing the branching factor of the hierarchy everywhere might be to do so at lower levels of the hierarchy only, leaving the higher levels with the original low branching factor.

5.5.3. Truncating the Hierarchy

One of the particular problems with large cluster files for the radiosity application is that typically a large portion of the cluster file goes unused. This is especially the case with models containing large flat areas; a single cluster representing the flat area suffices for the radiosity simulation, and the only use for the rest of the hierarchy in that area is for the final push-to-leaves step.

A useful way to take advantage of this is to truncate the face cluster file, and thus the hierarchy. The cluster file can be organised so that the leaf nodes belonging to any given cluster are numbered consecutively. Thus if we store face ranges with each cluster, it is possible to go directly to the leaf polygons enclosed by that cluster without traversing the hierarchy below it. Because the cluster file is generally stored in a reverse-log format, where clusters occur in reverse order from the order in which they were generated, it is possible to simply eliminate all nodes below a certain cut through the hierarchy by discarding all nodes after a certain point in the file.

The result is a trade-off between memory or disk space and speed. If the radiosity algorithm needs to descend beyond the cut at which the rest of the hierarchy has been removed, it will incur a speed hit as it must switch directly to using the leaf polygons in the simulation. The truncation can also be viewed as a form of lossy compression of the hierarchy. Because the cluster file is written out in order of clustering, the first clusters to be discarded are those most likely to be largely flat, and thus ideal candidates to be discarded.

For technical details on how this truncation can be carried out, see **Section 6.2.3**.

5.5.4. Out of Core Face Clustering

Unfortunately the face cluster algorithm has poor performance when the model being clustered will not fit in main memory. Because the algorithm starts by creating dual edges between all adjacent faces, its memory consumption is greatest during initialisation, when it must access the entire model. Finding adjacency relationships between faces for such large models is also an expensive operation. These problems are similar to those faced by geometric simplification methods based on edge contractions when models grow too large.

One solution that has been presented for geometric simplification is to use a variant of Rossignac and Borrel's vertex decimation [Ross93, Lind00]. In vertex decimation, we place a grid around the model, and then collapse all model vertices in one cell to a single vertex, discarding all triangles that become degenerate. (That is, all triangles that have two or more vertices lying in the same cell.) We can adapt this algorithm to the dual-space face-clustering algorithm, by adding all triangles in the same cell to a single face cluster. The algorithm is outlined in **Listing 2**.

This approach has the advantage that, like the out-of-core simplification algorithm, the memory use is determined completely by the grid size used, and thus can be constrained to match the available in-core memory. Moreover, it generates the appropriate initial set of dual edges as part of the triangle addition step; there is no separate, triangle adjacency-finding algorithm needed.

The main disadvantage of this approach is that it is no longer strictly guaranteed that the surface within a cluster is a manifold connected surface, although

```
OutOfCoreCluster()
{
     Engrid(); // place grid around model

     foreach (cell in grid)
          initialise a cluster quadric

     foreach (triangle in model)
          find cell coordinate of triangle's centre
          add triangle to that cell's cluster, updating quadric

          find cell coordinate for each vertex
          foreach (edge with endpoints in different cells)
               add dual edge between those cells
                    if not already present

     Initialise(dualEdges);  // find error term for each edge
     FaceCluster(dualEdges); // add edges to heap and
                             // cluster as usual.
}
```

Listing 2: Out-of-core face clustering. Each grid cell is treated as a cluster. The process of adding triangles to cells also identifies neighbour relations between those clusters.

it is the usual case. (This is the corresponding problem to the way Rossignac and Borrel-style vertex decimation can produce a non-manifold output surface from manifold input.)

The other disadvantage is that we limit the depth of the face cluster hierarchy. At the leaf clusters corresponding to the original grid cells, we go directly from a cluster to all the faces contained in that cluster. However, this is similar to the approach outlined in **Section 5.5.3**, where we deliberately truncate the hierarchy to save space. In fact, the approach outlined there is needed to handle such a cluster hierarchy, where a cluster can contain more than two leaf faces.

5.6. Performance of Face Clustering

In this section I will give some results for the modified face cluster algorithm presented in this chapter, for the kinds of polygonal models I'm most interested in being able to handle well with radiosity algorithms. **Figure 79** shows sixteen such models, ranging in complexity from a few thousand polygons (the cup and ellipsoid) to over one million (the buddha).

Figure 77 shows log-log plots for both memory and time use by the cluster algorithm. The models in both graphs fall along a line of slope 1, suggesting both are roughly $O(n)$. In the case of memory consumption, this is expected. In the case of time consumption, we expect $O(n \log n)$ best case performance, and get it for this range of models. (For large numbers, $O(n \log n)$ looks much the same as $O(n)$ on any plot.) Crucially, the time results are always close to this best-case, as opposed to the worst-case $O(n^{3/2} \log n)$ performance that would be represented by a line of slope $3/2$.

The reason for this good performance is shown in **Figure 78**, which shows the average and maximum dual-edge degree during the clustering process for each model. The average degree always falls between 4 and 6 edges, which is consistent with the lowest curve of **Figure 71**, and thus the assumption about the dual-edge degree being constant that underlies the theoretical best-case $O(n \log n)$ performance of the algorithm are satisfied. The maximum degree is a measure of the largest cluster boundary. It is particularly high in the case of the buddha; I speculate that this is because of its planar base remaining as one cluster until relatively high in the cluster.

Finally, free source code implementing the face cluster algorithm is available on the internet, at http://www.cs.cmu.edu/~ajw/thesis-code/. As well as its use in radiosity, this code should prove valuable for other spatial data structure-oriented algorithms that need to be able to handle large detailed models well, such as collision detection or ray-tracing.

5.7. Comparison to Other Techniques

5.7.1. Hierarchies of Oriented Bounding Boxes

Hierarchies of oriented bounding boxes have proven useful in other contexts besides radiosity, in particular for query-based operations on spatial data structures, such as collision detection [Ball81]. Barequet et al. present the BOXTREE data structure, which is essentially a hierarchy of oriented bounding boxes, con-

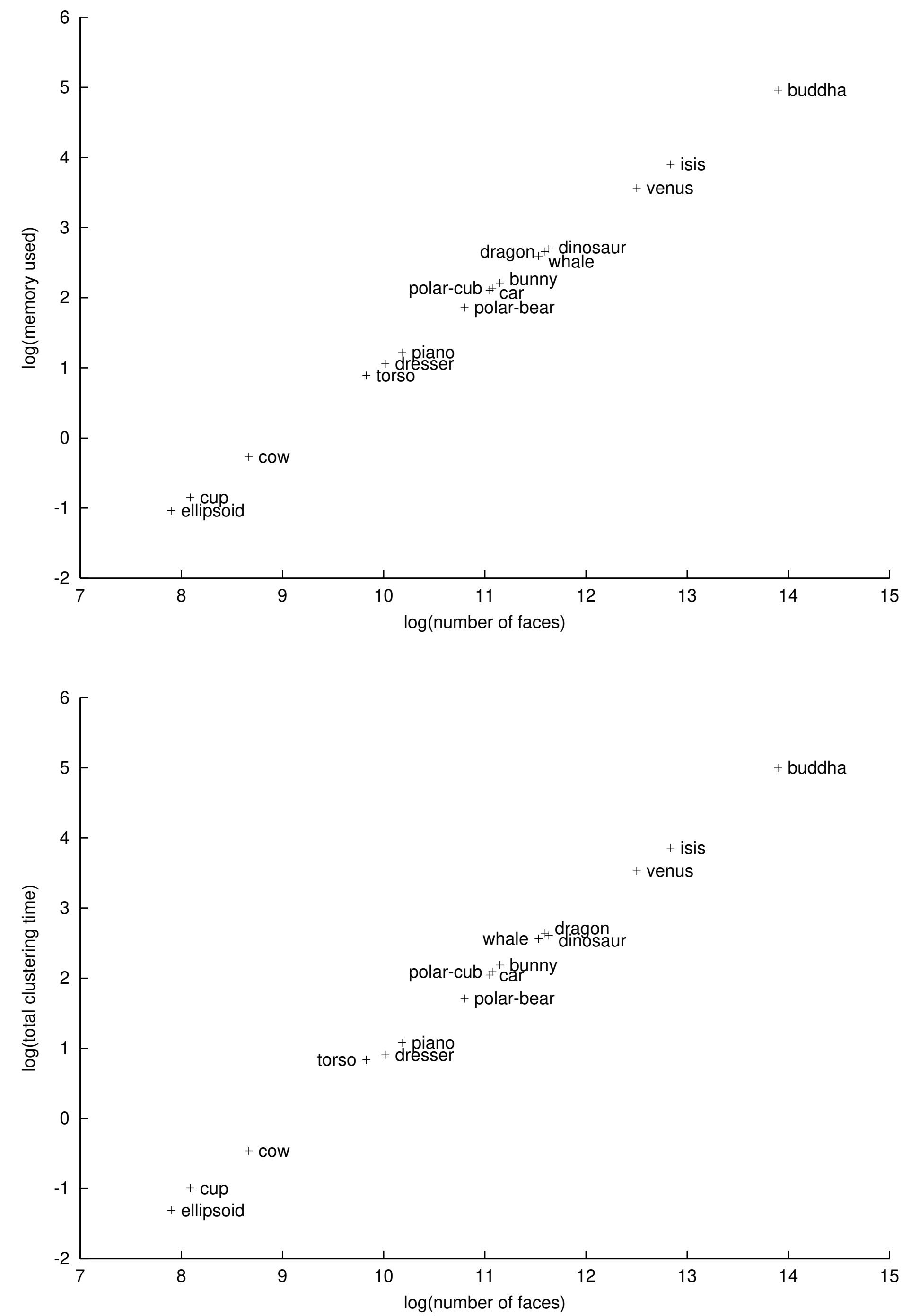


Figure 77: Performance of the new face cluster hierarchy creation algorithm for a variety of models.

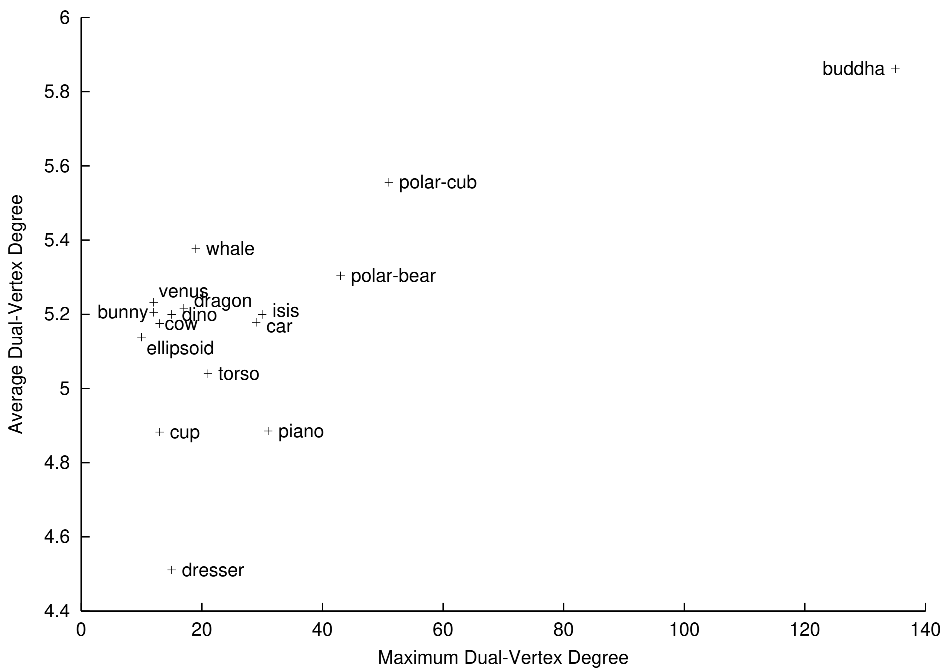


Figure 78: Variance in dual-vertex degree for a variety of models.

structed top-down [Bare96]. They also discuss several methods for both creating the hierarchy and picking appropriate bounding boxes. They conclude that the minimal-component pie-slice works better than oriented bounding boxes for their test models. I suspect that this is because:

- Their box-tree hierarchies calculate bounding boxes for the leaves of the hierarchy, i.e., the original triangles of the model. In this case, the advantage of a triangularly-shaped bounding volume is obvious.
- The triangular shape is more oriented than a box; this could lead to fewer stability problems and more accurate PCA directions for their strategy of using child bounding boxes to form the covariance matrix.

Pie slices also take slightly more storage than oriented bounding boxes (8 floating point numbers plus a rotation, as opposed to 6 floats plus a rotation.)

The BoxTree paper also presents results showing that, as we might expect, the strategy of using PCA to select one direction rather than all directions of the bounding box can lead to better performance.

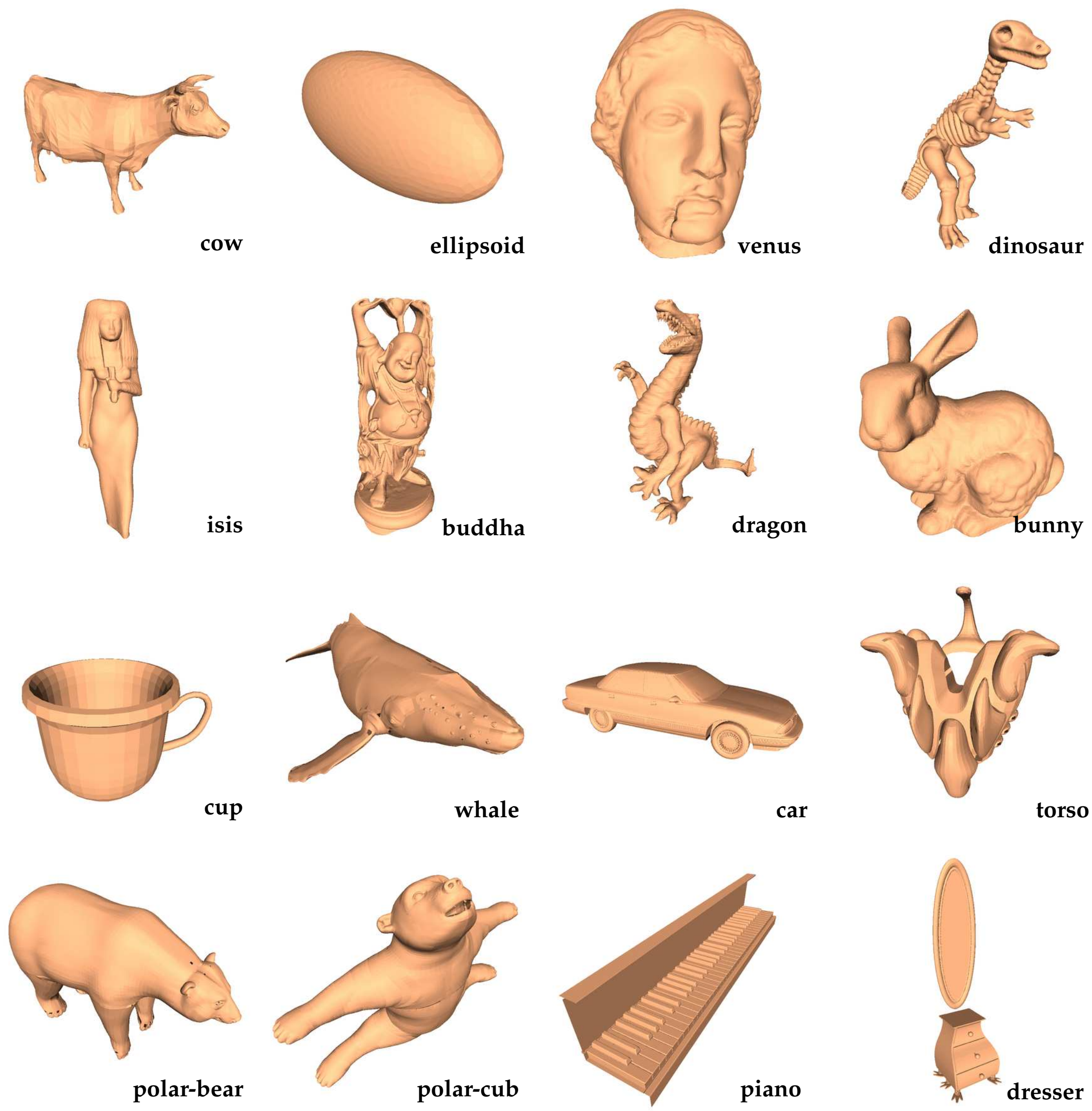


Figure 79: Test models. Shown are a selection of models of varying complexity. The new face cluster algorithm was run on each model and test results collected.

5.7.2. Volume Clustering for Radiosity

There are a number of different approaches to clustering in radiosity. Hasenfratz et al. have studied a number of the data structures and strategies used in volume clustering [Hase99]. They classify clustering approaches by data structure used, and type of construction algorithm. Data structures are classified as either regular structures, such as octrees or k-d trees, that recursively subdivide space, or bounding volume hierarchies, which adapt to object geometry using optimization criteria. Construction algorithms are classified as either top-down (the simplest) or bottom-up. Within this schema, the face clustering algorithm is most closely related to bounding volume hierarchies, as it tries to adapt to object geometry, and it uses a bottom-up construction algorithm.

Face clusters have some of the advantages of both kinds of data-structure. Their speed of construction is similar to regular subdivision-based data structures, such as octrees, however their bounding boxes are of much higher quality. Their adaptiveness to geometry is similar to that of bounding volume hierarchies, but they do not suffer from overlapping clusters or mixing of surfaces in single clusters, because they are constrained to share only connected geometry.

The authors pay some attention to why cluster overlap in bottom-up cluster constructions is a problem. It leads to situations where scattered elements of a surface end up in different clusters, leading to disturbing visual artifacts. The face cluster algorithm, because it always merges topologically adjacent surface clusters, avoids this problem.

The bottom-up construction of face clusters means the algorithm is more complex to implement than simple top-down clustering methods, because adjacency relationships must be tracked. However, the constant-time nature of the quadrics used to evaluate the goodness of potential clusterings, and the use of a greedy iterative clustering routine, means it is much faster than most volume clustering methods that are trying to optimize for some feature of the hierarchy. These methods tend to be quadratic in the number of input surfaces, whereas face clustering is close to linear.

Finally, **Figure 80** compares the time taken for face clustering and volume clustering two of the models from **Figure 79**; the dragon and whale models. The volume clustering method used is one of the simplest (and fastest) possible: the tightest-fit octree. It proceeds by recursively subdividing a subset of the model into eight octants, and placing a cluster around each octant. Polygons in the model that are larger than the size of an octant are kept at the same level as the cluster being processed.

As can be seen, the two algorithms have roughly the same asymptotic complexity. The face cluster algorithm has a larger constant, but as a hierarchy need only be generated once per model, rather than on a per-scene basis, its amortised clustering time may well be smaller in many situations.

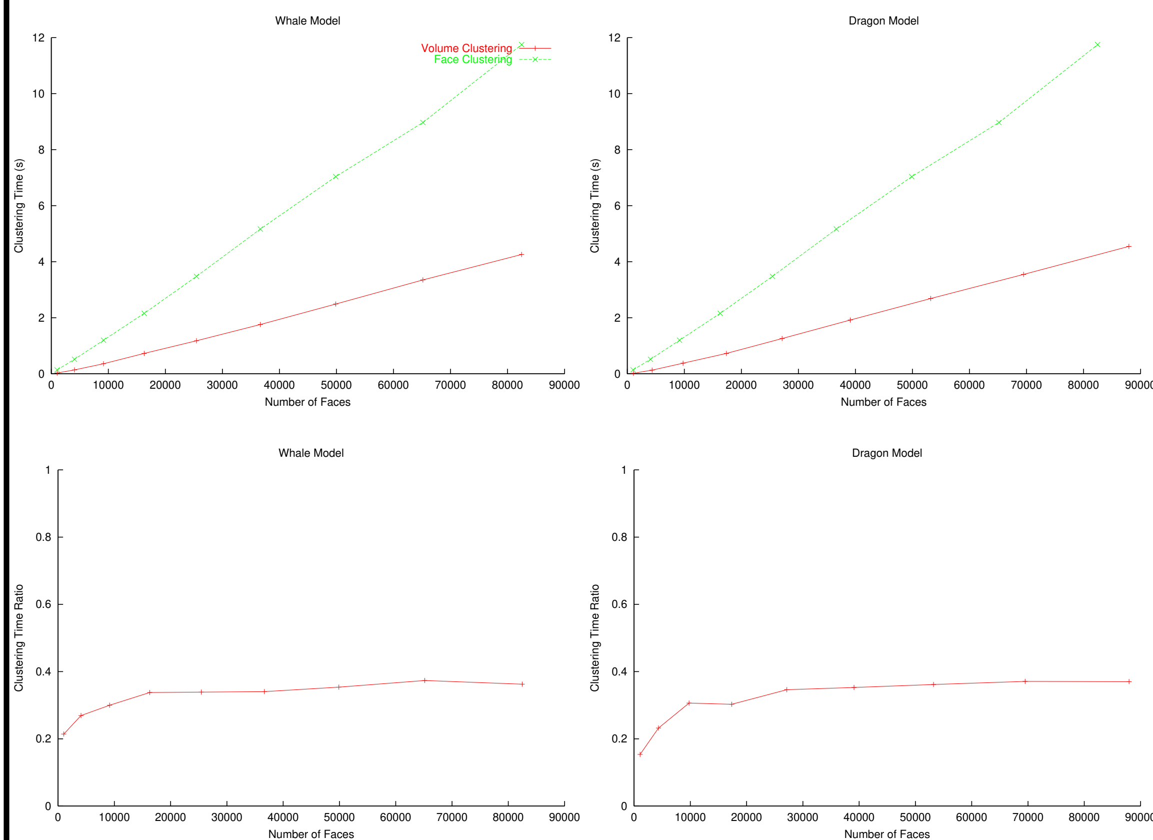


Figure 80: Speed comparison to simple volume clustering. Top: the time taken to volume cluster and face cluster two models, for a number of different resolutions of each model, generated by Garland's simplification algorithm. Bottom: the ratios of these times. The volume-clustering method used is the fastest possible; tightest-fit octree.

Face clustering takes around 2.5 times as long as face clustering, a figure that stays steady with increasing model size. This is a good result considering that the face cluster algorithm is trying to optimize the hierarchy according to flatness and shape criteria, whereas the volume clustering method is simply partitioning space.

5.8. Summary

In this chapter I have analysed the performance of the previous face cluster creation algorithm, in particular showing how hierarchy balance can have an adverse effect on the construction algorithm, and presented a new algorithm that performs considerably better. The algorithm is based on a new cost metric, which

- produces more balanced hierarchies;
- is simpler and cheaper to evaluate, with quality comparable or better than the previous metric.

I have also modified the algorithm, resulting in

- faster bounding box calculations, by avoiding principle component analysis for flat clusters;
- better (tighter) bounding boxes for flat clusters, by using a minimal-area bounding rectangle algorithm.

Finally, I have provided results for the new algorithm for a wide range of models, showing that its running time is close to linear in the number of input polygons, and reasonably independent of model geometry. I have also compared face cluster hierarchies to other hierarchies of oriented bounding boxes, as well as those types of hierarchy commonly used in radiosity.

Chapter 6

Implementation Details

The algorithms described in the previous chapters have been implemented in a radiosity code base I call "radiator". In this section I will discuss some of the more interesting implementation details of my system. We start with a basic overview of the component parts of the system, and then in turn look at details of the face cluster system, and the radiosity system built on top of it.

6.1. System Architecture

The basic architecture of the face-cluster radiosity system is shown in **Figure 81**. All components share a common library, "gcl", which provides rendering and scene-graph services, scene-file handling, and other basic graphics utilities. In particular the library has support for multiresolution models as one of its scene-file types—both the geometry-simplification edge-collapse models, and the geometry-aggregation face-cluster models. Both types of model share the same basic multiresolution data structure. (Indeed, both kinds of model can be merged into a meta model containing basic geometry, a vertex hierarchy, and a cluster hierarchy.)

An original base model file can be processed and an ascii log-format cluster hierarchy output by the "make-fch" utility. (File formats supported include AutoDesk's 3DS, Cyberware's PLY, MGF, and the ascii Wavefront OBJ format.) The gcl library can then translate this file into a convenient memory-mappable

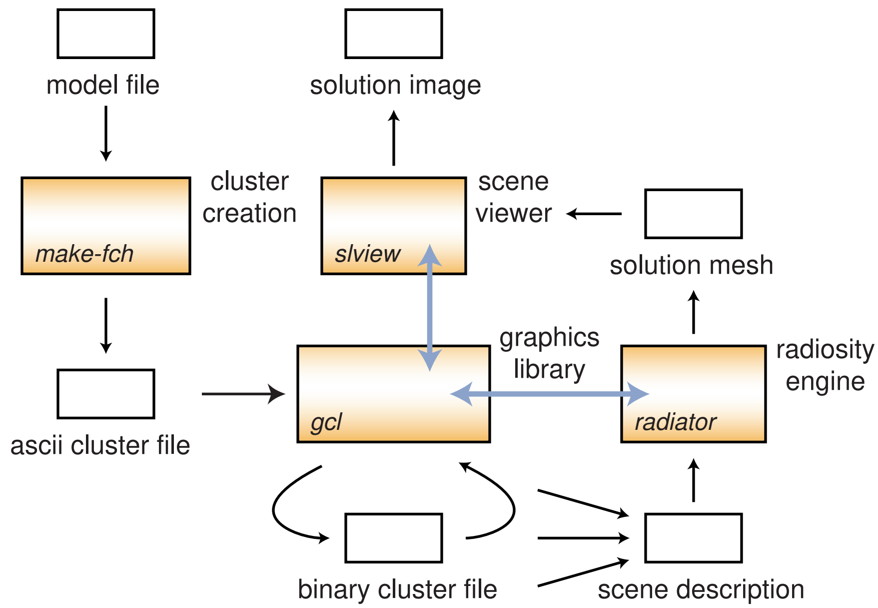


Figure 81: Interaction between various components of the radiosity system.

binary format. (An ascii format is used for the initial cluster file both for flexibility and because it can be moved between machines with different byte orders, whereas the binary format cannot.)

The radiosity system, "radiator", uses gcl to read scene descriptions that reference such models. (These descriptions describe the scene hierarchy with embedded geometric transforms and material properties.) Where necessary, it creates radiosity elements from face clusters on demand. (See **Section 6.3.5**.) Once it has calculated a radiosity solution, it writes out the corresponding solution mesh with vertex colours. If no input polygon refinement has taken place for a particular scene model, only a list of vertex colours needs to be written out; the geometry of the model stays as is.

These view-independent solution meshes can then be viewed with "slview", and images generated from various vantage points. They can also be fed to an animation program for generating fly-throughs. The entire system is available on-line at http://www.cs.cmu.edu/~ajw/thesis-code/.

6.2. Face Clusters: Algorithms and Data Structures

In this section we describe generic face cluster algorithms and data structures. In the next section we will specialise them for radiosity.

6.2.1. A Clustered Model

A clustered model is stored using a number of contiguous arrays, as in **Listing 3**. The base model is stored in the **`points`** and **`faces`** arrays, containing a list of vertices and indexes into that list for each model triangle respectively. Other attributes such as vertex or face colours or texture coordinates can also be stored, but we will consider only geometry here.

```
struct ClusteredModel                                  array length
{
    PointList     points;                                  v
    FaceList      faces;                                   f
    ClusterList   clusters;                                f - r
    IndexList     rootClusters;                            r
    ActiveList    clustersActive;                          2f - r
};
```

Listing 3: A face-clustered model's data structure.

The face clusters themselves are stored in the **`clusters`** array. Only internal nodes are stored; there are no clusters corresponding to the model's faces, in the interest of saving storage space. Each cluster stores information about its bounding box, sum area normal, and two child IDs, as we shall see in the next section.

If there are f total faces in the model, then there are a total possible number of $f - 1$ face clusters, assuming that the model is completely connected and thus there is only one root node. This is not always the case, however, as each separate connected surface component corresponds to a single hierarchy, and a model may contain more than one such component. In the more general case, if there are r root nodes, then there are $f - r$ face clusters. The **`rootClusters`** array holds the IDs of all root clusters in the model.

It is convenient to assign a universal ID to all nodes in the cluster hierarchy, both faces and face clusters, as follows. The faces are numbered 0 to $f - 1$, according to their occurrence in the faces array, and the clusters from f upwards, according to their order of creation by the clustering program. This numbering corresponds to a so-called *log format*; it is the order we get if we start at the model leaves and write each clustering operation to a log file as we successively merge clusters. This numbering has the following advantages:

- The child IDs of any cluster are always smaller than its own ID. This means that by iterating linearly through the IDs we can accomplish a preorder (forwards) or postorder (backwards) traversal of the hierarchy, as illustrated in **Listing 4**[1]. Many common operations on the cluster hierarchy can be implemented in this way.
- We can detect when we are at a leaf by checking the child ID of a particular cluster. If it is less than f, the child is a leaf, and the index of the corresponding face in the faces array is just the child ID.

The resulting set up is shown in **Figure 82**. The face clusters are generally stored in order of their ID in the clusters array, to make it easy to map between the ID and their position in the array. To make truncation of the hierarchy easier, we can store them in reverse order, so that discarding clusters below a certain point in the hierarchy is simply a matter of shortening the cluster array.

Finally, the last field, **`clustersActive`**, is used by visualisation software to track a cut through the cluster hierarchy to be displayed. Any such cut is guaranteed to completely cover the model's surface without redundancy; we are merely varying the number and size of clusters used to do so. The field holds mark bits corresponding to every ID in the cluster hierarchy. The clusters or faces in the current cut have their mark bits set, and the rest are clear. The cut can be moved lower in the hierarchy (and thus the number of active clusters increased) by doing a preorder traversal, and marking the children of any currently active cluster. Likewise, the active cut can be moved higher by doing a postorder traver-

1. These are not strict pre and postorder traversals, as the traversal does not visit child and parent nodes in consecutive order; it is only guaranteed that the parent will be visited before the children and vice-versa. For most purposes this suffices, however.

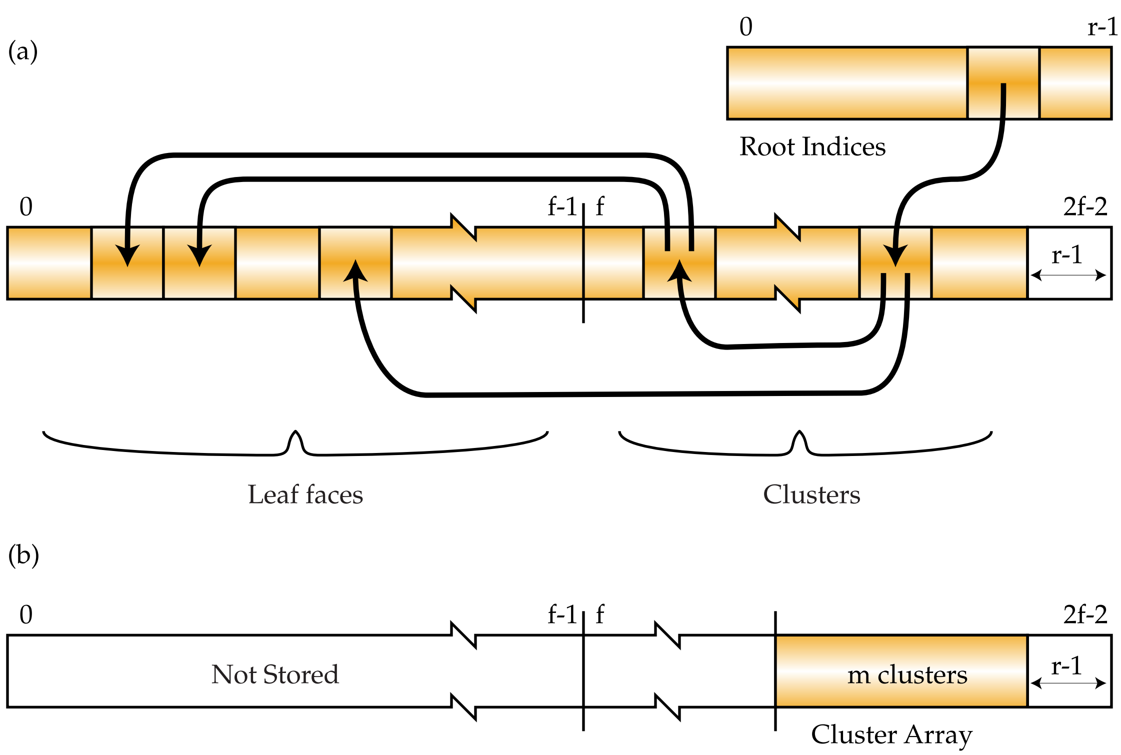


Figure 82: The cluster hierarchy data structure. In (a) we see an array of r root indices, one for each connected submesh. Each root index pointing into the cluster array, which contains the clusters written in log format, from the original model faces to the last cluster. There are f leaf faces, and $2f - r - 2$ total clusters. As shown in (b), we don't store clusters for the leaf faces, and if we want to truncate the hierarchy, we can optionally store only the top m clusters.

sal, and marking any cluster whose children are both currently active[1]. An illustration of the latter is shown in **Listing 4**.

Generally, this field is unused (and unallocated) by the radiosity simulation.

1. This is similar to the approach taken by Hoppe in his method for view-dependent refinement using progressive meshes [Hopp97].

```
// a preorder traversal: from the leaves to the root nodes
for (i = 0; i < maxID; i++)
     VisitNode(i);

// a postorder traversal, checking for cluster type
for (i = maxID - 1; i >= 0; i--)
     if (i < faces.NumItems())
          VisitFaceNode(faces[i]);
     else
          VisitClusterNode(Cluster(i));

// adapting the active hierarchy cut upwards
for (i = 0; (activeClusters > targetClusters) && i < maxID; i++)
{
     if (clustersActive[Cluster(i).child[0]]
          && clustersActive[Cluster(i).child[1]])
     {
          // if both children are active, deactivate them and
          // active the parent

          clustersActive[Cluster(i).child[0]] = 0;
          clustersActive[Cluster(i).child[1]] = 0;
          clustersActive[i] = 1;
          activeClusters--;
     }
}
```

Listing 4: Traversing the face cluster hierarchy. The **`Cluster()`** function maps an ID to the appropriate cluster in the clusters array.

6.2.2. The Face Cluster Data Structure

To represent the oriented bounding box of the cluster we need to store both an orientation, and minimum and maximum bounds of the cluster in each of the three directions of that orientation. The orientation can be represented by an $\Re^3$ rotation, i.e., a 3×3 matrix. We can save space over the nine floats required to store a full rotation matrix (or more correctly, the six floats required to store the symmetric matrix) by storing it as a quaternion, which requires only four floats. (As a comparison, RAPID [Gott96] also uses quaternions, though they store a centroid and the three dimensions of the OBB instead of three min/max pairs). The algorithm for converting the three principle axes to a quaternion and back [Shoe85] has proved to be very stable. Thus it requires a total of 10 floating point numbers to store the oriented bounding box.

Storing the side areas of the cluster, which we need to construct our upper bounds on projected area, requires a further six floats. It is tempting to use instead the side-areas of the bounding box, which are easily available from the OBB min and max fields, and as discussed previously are in turn an upper bound on the cluster side-areas. A compromise, which I use, is to store the more accurate sampled side-areas as 8-bit fixed-point fractions of the side-areas of the bounding boxes. Of course, when converting the fractional area to the 8-bit representation, the fraction must be rounded up so as to maintain our upper bound.

The face cluster data structure is shown in **Listing 5**. Further space savings could be possible by using Deering's normal encoding technique in conjunction with a projected-area scale factor to represent the sum area normal. Such compression must be approached with caution given how heavily we rely on the sum area normal [Deer95].

```
struct FaceCluster                                                bytes
{
     Int            child[2]; // two child cluster IDs      (8)
     Quaternion     axesQuat; // orientation of the OBB     (16)
     Point          min; // min point in OBB coord system   (12)
     Point          max; // max point in OBB coord system   (12)
     Vector         areaNormal;    // sum area normal       (12)
     Float          totArea;       // total surface area    (4)
     Byte           sideArea[6];   // side area fractions   (6)
};
```

Listing 5: The face cluster data structure. The storage in bytes of each field is shown in brackets. The total structure requires 66 bytes.

We can write the total memory use of the face cluster hierarchy data structures as:

```
totalMem = (66 * clusters + 12 * faces + 12 * vertices) bytes
```

If we assume there is only one root cluster, and thus $f - 1$ clusters, and that, as in the limit case for a manifold, there are twice as many faces as vertices, we have:

```
totalMem = (84f - 66) bytes
```

Thus on average we incur an overhead of 84 bytes per face, assuming we store the full hierarchy.

6.2.3. Reordering the Hierarchy

We can reorder the faces in any face cluster model to have the property that, for any cluster in the hierarchy, all its corresponding faces occur in sequential order. This is easily accomplished by a pre-order traversal of the hierarchy in which faces are added to a new face list in the order in which they are encountered. This has the following benefits:

- It improves the locality of face storage, because faces belonging to the same cluster are always guaranteed to be stored contiguously.
- It makes it easier to locate the faces corresponding to a particular cluster. Rather than recursively descending the hierarchy to find these faces, we descend only the left-most and right-most child paths, as in **Listing 6**, and can then perform any operation over the faces via a simple for loop. Indeed, if we are willing to add an overhead of two indices to each cluster, we can get the corresponding range of IDs directly.
- It makes it possible to handle truncation of the hierarchy cleanly. If we wish to modify the hierarchy to remove all clusters below ID ic, we simply iterate through the remaining clusters, and replace all indices smaller than ic with their corresponding left ID or right ID. Again, see **Listing 6**.

To test the benefits of hierarchy reordering and truncation, a simple radiosity simulation was run using different versions of the clustered 1,000,000 triangle "buddha" model from **Figure 79**. All file caches were cleared before each run[1], and the results of five runs averaged together for each version. The results are presented in **Table 9**. Shown are the total size of the face-clustered model, the time taken for radiosity solution, the time for the final post-processing pass (which touches all faces of the model), and the total wall-clock time for the radiosity algorithm. For the original clustered model, the wall-clock time is much longer than the total measured solution and post-processing time (47s compared to 6.5s), indicative of the heavy disk activity taking place. Both the reordered cluster hierarchies do not suffer from this problem; the final post-processing pass becomes a simple ordered

1. Under Linux this is most easily achieved by unmounting and then remounting the partition the cluster file resides on.

```
VisitClusterFaces(Int clusID)
{
     leftFaceID = rightFaceID = clusID;

     while (!IsFace(leftFaceID))
          leftFaceID = Cluster(leftFaceID).child[0];

     while (!IsFace(rightFaceID))
          rightFaceID = Cluster(rightFaceID).child[1];

     for (i = leftFaceID; i <= rightFaceID; i++)
          PerformFaceAction(i);
}

PostOrderAction(Int i, Int truncID)
{
     if (Cluster(i).child[0] < truncID)
          Cluster(i).child[0] =
               Cluster(clus.child[0]).child[0];

     if (clus.child[1] < truncID)
          Cluster(i).child[1] =
               Cluster(clus.child[1]).child[1];
}
```

Listing 6: Visiting the faces in a cluster with a reordered hierarchy, and truncating the hierarchy.

Type of Scene	Size	Soln.	Post.	W/C
Base model	18.5 MB			
Original clustered model	87 MB	4s	2.5s	47s
Reordered	87 MB	4s	1.8s	8.7s
Reordered and truncated	21.4 MB	4s	0.9s	8.0s

Table 9: The effect of reordering face clusters.

step through the faces array. The truncated hierarchy not only takes only 3 MB of extra storage over the 18.5 MB base model, but saves a little on post processing time, because the distance to the end of the cluster hierarchy from the radiosity solution leaf nodes is smaller.

6.3. Radiosity Implementation

In this section we will look at some of the specifics of the face cluster radiosity system built for this thesis.

6.3.1. Volume Clustering

To build volume clusters, I followed the methods described in [Gibs96, Sill95a]. An octree that encloses the scene is created, and scene polygons are placed within that octree according to their size and position. This is the "tightest-fit octree" method referred to by Hasenfratz et al. [Hase99]; see that paper for alternative volume clustering algorithms. In my experience this method provides acceptable quality, and is very fast. Because I don't also use the volume cluster data structure for visibility testing, the advantages of more sophisticated schemes summarized by Hasenfratz et al. are not so important.

6.3.2. Visibility Testing

Visibility is sampled using ray-tracing; the spatial data structure used for acceleration is a nested grid data structure [Fuji86, Jeva89]. Traditionally, of the ray-tracing optimising schemes, grid-traversal is amongst the fastest, if not the fastest algorithms for ray-casting against a scene with many similar-sized primitives. When the scene contains primitives with widely disparate sizes, however, this approach suffers from the so-called "teapot in a stadium" problem. A single uniform-density grid cannot efficiently cover all primitive sizes, leading to many primitives in some cells.

To solve this problem, we can instead create a number of nested grids, each adapted to a particular primitive size. To build such nested grids, within each grid primitives are sorted into cells, and, depending on their local density and size, become candidates for insertion into a subgrid. The grids rely on lazy evaluation—this sorting process only takes place when a ray hits the grid in question. Thus if no rays hit a subgrid, the primitives within it are never processed, saving considerable pre-processing time. However, this optimization is not so important for most radiosity solutions, where the scene tends to be completely and evenly sampled.

Fractional visibility is used during simulation, but the visibility can be resampled at higher resolution during the final push-to-leaves step. This is discussed further in **Section 6.3.7**. The visibility data-structure is usually built from the input polygons only; it ignores subdivided input polygons and face clusters.

6.3.3. Approximate Visibility

While the nested grid scheme is highly efficient, even for large models, for very complex scenes the ray-casting part of the algorithm ends up being carried out at a much higher resolution than the finite-element part of the algorithm. This can result in the occlusion-testing data-structures dominating the time and memory cost of the algorithm.

Ideally, we would like to incorporate some kind of multiresolution visibility algorithm, taking advantage of the face-cluster hierarchies to test visibility against coarser resolutions of the model when our accuracy requirements are not high. This is an interesting area of research, but outside the scope of this thesis.

A simpler approach I currently employ is to use the face cluster hierarchies to construct an approximate occlusion data-structure for the scene. This can be seen as a combination of using the cluster hierarchy to test visibility, as per Sillion [Sill95a], and the voxel-based opacity grids of Christensen and Gibson [Chri95, Gibs95]. It can be summarized as follows:

- adapt the complexity of each cluster hierarchy to a user-specified fraction f_v of the total number of clusters. This is achieved by starting at the root clusters, and expanding the active cluster list until it covers the required number of clusters, as in **Listing 4**;
- add the bounding boxes of these clusters to the nested grid data structure;
- ray-trace against the bounding boxes for the purposes of occlusion testing, optionally using the projected area estimates to determine fractional opacity[1].

There is one minor adjustment needed to make this approach work well. To avoid self-occlusion by clusters, we ignore a cluster for the purpose of occlusion testing when either the start or end point of a shadow test ray falls within it. (Recall that intra-cluster occlusion is subsumed into the visible projected area estimates of **Chapter 4**.)

1. The probability of a ray hitting a surface is proportional to its visible projected area in the direction of the ray [Gold87]. Thus a fractional visibility or albedo estimate for a cluster's bounding box can be obtained by taking the ratio of the bounding box's projected area to that of the surface it contains.

The major disadvantage of this approach is that another parameter must be set for the rendering process. Generally, the smaller epsilon is chosen to be, the larger f_v should be. It is desirable that the clusters used for visibility testing be smaller than those used in the radiosity solution.

This approach can lead to considerable speed and memory savings for large models. See **Section 7.4** for an example.

6.3.4. Transport Calculation and Refinement

In face cluster radiosity we must handle a number of different types of light transfer; radiosity exchange between face clusters, standard planar elements, and volume clusters. This can lead to problems in choosing an error metric that is consistent for all transfers. To handle these in a common framework we use sampling across the receiver to estimate the error in the transfer at the same time as we estimate the transfer itself. We use the L_1 norm to measure error, i.e., the *BFA*-weighted refinement discussed in [Smit94].

The basic "link" data structure representing radiosity transport between two elements is shown in **Listing 7**. The pseudo code shown in **Listing 8** illustrates how the error and the transport vector $\mathbf{m}$ can be calculated for a link, given a number of sample points over each element. For face clusters, the minimum and maximum projected areas are calculated as described in **Chapter 4**. For an input polygon or refinement thereof, the upper and lower bounds are both just the projected area of the polygon. For a volume cluster, the upper bound is established by summing the upper bounds of all the root face clusters or polygons it contains, and the lower bound is taken to be zero.

Given the transport error, we must also decide whether the corresponding transport link should be refined or not by comparing it to a refinement epsilon. For links that are partly occluded, the refinement epsilon is reduced to encourage subdivision at shadow boundaries. (The refinement epsilon controls link subdivision; links with transport error greater than this are split.) Pseudo code for the resulting refinement oracle is given in **Listing 9**.

The algorithm presented chooses which end of the transport link to subdivide by comparing the total surface areas of the elements at either end. This has the advantage of simplicity and robustness, but we can do better with a bit more calculation. **Listing 10** gives pseudo code corresponding to the approach outlined in **Section 4.4.3**. Here, the fraction of the estimated error due to each element is calculated, and the element contributing the most error is subdivided.

```
struct VecLink
{
    Element   from, to;      // can be patches, face or volume
                             // clusters
    Vector    m;             // transport vector m
    Float     error;         // ⟨G_{D←S}⟩ from § 4.4.2
    Float     visibility;    // average visibility between
                             // the two elements
}
```

Listing 7: The vector radiosity transport data structure.

6.3.5. Face Cluster Solution Elements

The data structure for a generic, non radiosity-specific face cluster was presented in **Section 6.2.2**. For those clusters used in the radiosity solution, we must in addition store radiosities and other extra information. Because typically the number of clusters used as radiosity elements is much smaller than the total number of clusters, it makes sense to save time by storing some face cluster attributes transformed into world space coordinates, rather than performing these transformations on the fly. The radiosity element data structure is shown in **Listing 11**.

The use of vector instead of scalar irradiance means that, compared to the storage required for a face in standard hierarchical radiosity, we store 9 real numbers per hierarchical element instead of 3, and 3 reals per link instead of 1. Although the face cluster hierarchical elements are more expensive than standard Haar elements, they are in general more lightweight than volume cluster elements, which require 8 child pointers in our implementation, and much more lightweight than storing a general radiance distribution.

6.3.6. Irradiance Vector Interpolation

One of the great advantages that face clusters have over volume clusters is that they are intrinsically surface-based. This means that it is much easier to interpolate irradiance values across the cluster. In volume-based clusters, there is no

```
VecLink::CalcTransport()
{
     m = 0;
     factor.InitBounds(-infinity, infinity);

     // find samples over the upper face of each cluster's
     // bounding box
     p1 = from.CreateSamples(numSamples);
     p2 = to.CreateSamples(numSamples);

     for (i = 0; i < numSamples; i++)
     {
          r = p1[i] - p2[i];

          rLen = len(r);
          if (rLen > 0.0)
          {
               r /= rLen;
               m2Len = pi * sqr(rLen) + from.area;

               l1 = from.MinProjArea(-r);
               l2 = to.MinProjArea(r);
               factor.UpdateBounds(l1 * l2 / m2Len);

               u1 = from.MaxProjArea(-r);
               u2 = to.MaxProjArea(r);
               factor.UpdateBounds(u1 * u2 / m2Len);

               m += 0.5 * (l1 + u1) * r / m2Len;
          }
     }

     m *= visibility / numSamples;
     error = factor.Range();

     linkViable = (len(m) > 0.0 || error > 0.0);
}
```

Listing 8: Calculating link transport and error. We must calculate an estimate of the transport vector **m**, and the error in the transferred power. Only if both are zero should we ignore this link for the purposes of further refinement (**`linkViable`**).

```
RefChoice VecLink::RefineOracle()
{
     w = abs(1.0 - 2 * visibility) * (1.0 - visEps) + visEps;

     // compare error in power (radiosity * error) to epsilon
     if (error * from.B < w * ε)
          return(SubdivideNone);

     if (to == from)
          return(SubdivideBoth);

     if (to.area > from.area)
          if (to.area < minRefinementArea)
               return(SubdivideNone);
          else
               return(SubdivideTo);
     else
          if (from.area < minRefinementArea)
               return(SubdivideNone);
          else
               return(SubdivideFrom);
}
```

Listing 9: RefineOracle: deciding how to refine cluster links given the transport error and average visibility.

notion of a largely co-planar, contiguous surface, and interpolation leads to problems with visibility, and requires deciding which reference planes to use for sampling irradiance. (Greger and Shirley's "irradiance volume" approach is one solution to this, albeit an expensive one [Greg98].)

A useful strategy in dealing with inter-cluster irradiance discontinuities is as follows:

- On the final solution pass, resample the point irradiance at the corners of each face cluster, rather than once over the entire cluster.
- During the push-to-leaves phase, interpolate the irradiance vectors for the cluster a leaf polygon falls within in order to find the irradiance at its vertices.

```
RefChoice VecLink::FindClusterToSubdivide()
{
     pa1.InitBounds(-infinity, infinity);
     pa2.InitBounds(-infinity, infinity);
     p1 = from.CreateSamples(numSamples);
     p2 = to.CreateSamples(numSamples);

     r = normalised(from.centre - to.centre);
     s1 = from.MaxProjArea(-r);
     s2 = to.MaxProjArea(r);

     for (i = 0; i < numSamples; i++)
     {
          r = to.centre - p1[i];
          rLen = len(r);
          if (rLen > 0.0)
          {
               r *= s2 / (rLen * (pi * sqr(rLen) + to.area));

               pa1.UpdateBounds(from.MinProjArea(r));
               pa2.UpdateBounds(from.MaxProjArea(r));
          }

          r = from.centre - p2[i];
          rLen = len(r);
          if (rLen > 0.0)
          {
               r *= s1 / (rLen * (pi * sqr(rLen) + from.area));

               pa1.UpdateBounds(to.MinProjArea(r));
               pa2.UpdateBounds(to.MaxProjArea(r));
          }
     }

     if (pa1.Range() < pa2.Range())
          return(SubdivideTo);
     else
          return(SubdivideFrom);
}
```

Listing 10: Choosing which cluster to subdivide by estimating the error contributed by each to the transport.

```
struct RadFaceCluster : Element
{
    Int        fci;       // ID of corresponding face cluster
    ModelInfo* info;      // associated model's information

    // copies of face cluster items in world-space coordinates

    Vector     areaNormal;    // sum area normal
    Float      sideArea[6];   // for VPA upper bound
    Float      totArea;       // total surface area
    Float      maxVisArea;    // max externally visible area

    Element    *child[2];     // solution child elements
    Colour     B;             // cluster radiosity
    IrradVec   R;             // vector irradiance: § 3.4.2
};
```

Listing 11: A radiosity face cluster solution element. The **`ModelInfo`** structure contains auxiliary information about the model containing this element, such as a pointer to its cluster data structures (see **`Listing 3`**), and its position and orientation in the scene. Thus a single model can be instantiated at multiple points in the scene. An **`IrradVec`** contains a vector for each of the colour channels represented.

This process is illustrated by **Figure 83**, and the dramatic results it can have in some situations are shown in **Figure 84**.

We can use the midpoint of each of the four edges of the cluster's bounding box that are parallel to the sum area normal to sample vector irradiance. Unfortunately, this is not necessarily the best place to sample visibility. If clusters lie on a concave surface, and their bounding boxes overlap, it is possible for a sample point to lie beneath the adjacent cluster. This problem can be largely overcome by insetting the sample points from the edge of the bounding box by a small amount. However, this leads to a further problem: interpolation overshoot. When we use our irradiance vector samples to reconstruct the vector irradiance at a point on the cluster, we may have to extrapolate if such a point lies outside the samples, which is a possibility now that we have inset them from the edges of the cluster. If the nearest sample is small in magnitude, or zero, this can lead to negative irradiances.

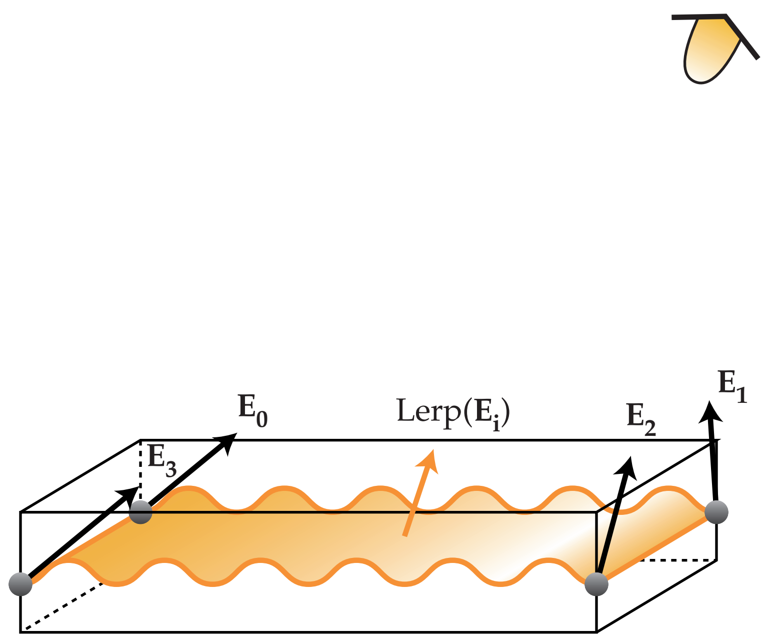


Figure 83: Irradiance vector interpolation. The irradiance vector at any surface point within the cluster is interpolated from samples at the corners of the cluster.

A simple way to overcome this is to only inset the visibility samples, and treat them as if they were collocated with the irradiance samples at the edge of the cluster. This leads to small errors in visibility interpolation, but avoids overshoot.

6.3.7. Post-solution Refinement

It is often the case that some post-solution refinement is carried out after the radiosity solver has finished. This can range from Gouraud-shading the mesh to an image space "final gather" (**Section 2.5.3**) to resampling the visibility more accurately [Gibs95]. The motivation in all cases is that the eye is more sensitive to shadow and shading discontinuities than we can account for with linear radiosity-centred metrics, and thus it makes sense to refine such features in the solution before display. While an image-space final gather has its place, it imposes an expensive cost for each viewpoint generated. Instead, I have adapted some of

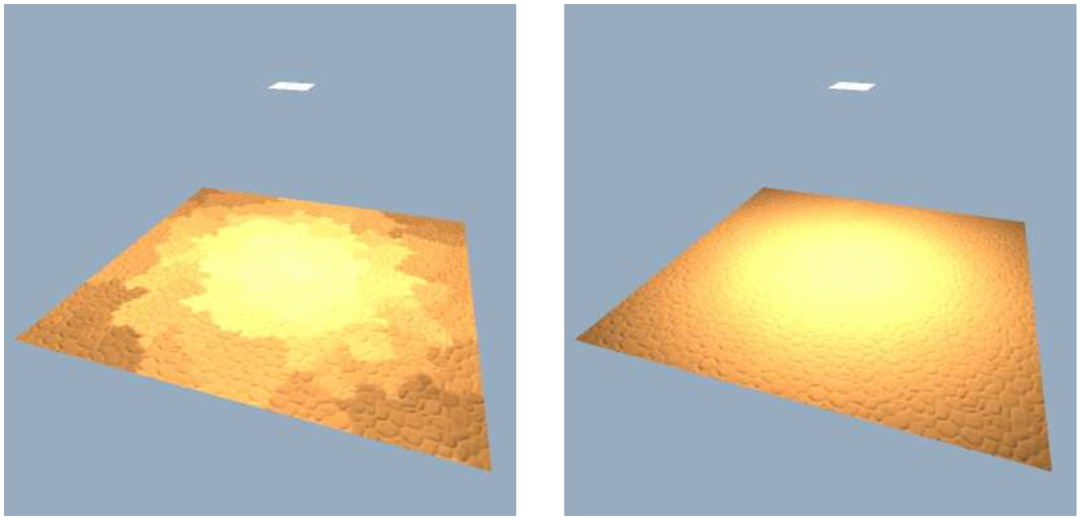


Figure 84: Vector interpolation. On the left, using a constant irradiance vector across each cluster leads to a blocky effect, reminiscent of the Haar basis. On the right, interpolating the irradiance vector across each cluster leads to a smoother result. The level of refinement for both pictures is the same.

Gibson's ideas on post-solution refinement for face cluster radiosity. Whereas Gibson re-evaluates visibility at the input polygon level, for the type of scenes considered in this thesis, this becomes prohibitively expensive.

There are a number of alternative post-processing solutions that can be applied.

- The minimal approach: we use the irradiance vector at each leaf cluster to colour all of its constituent faces. This can work well if visibility does not vary much, and surfaces are rough enough that vector irradiance discontinuities between clusters are masked.
- The irradiance-interpolation approach. We re-evaluate the vector irradiance at all corners of each leaf cluster, and use vector interpolation to colour its constituent faces, as above. This requires reevaluating all transport links at

the leaves, which can be expensive. A link pointing to a node relatively high up in the hierarchy must be re-evaluated at each leaf element it contains. Still, this is a reasonably robust approach.

- The irradiance-interpolation-everywhere approach. We perform vector irradiance interpolation at all face cluster nodes throughout the hierarchy. Irradiance vector samples at any given node are calculated by interpolating the parent's samples, and adding in resampled vector irradiance for the set of links pointing to the node, similar to the push phase of the familiar push/pull algorithm. This is less robust, but much faster.
- Finally, we can optionally subdivide all solution leaf elements a further m levels before doing our final visibility-resampling pass. This achieves some of the benefits of Gibson's resampling-at-vertices approach, in that shadow resolution is markedly improved, without the cost of descending all the way to the input polygons.

Typically I use the first or third approach. In my experience the vector irradiance interpolation post-process typically takes 50% of the solution time, when it is performed throughout the hierarchy rather than just at the leaves. The extra solution quality makes this cost is almost always worth it.

Chapter 7

Results

I have compared face cluster radiosity to my own implementations of progressive radiosity, and hierarchical radiosity with volume clustering, all of which are handled by a program I call “Radiator”. My implementation of volume clustering follows those of Bekaert and Christensen [Chri95, Beka]. I also compared my results to Greg Ward’s “Radiance” package [Ward94, Warda], a ray-tracing-based renderer which uses diffuse sample caching to perform radiosity calculations quickly. It is the most sophisticated of the various monte carlo-based sample-caching renderers, with respect to diffuse illumination. Finally, to validate my own implementation of hierarchical radiosity, I also used “RenderPark” for some experiments [Beka]. RenderPark is a freely available global-illumination renderer that, amongst other things, performs volume-clustering radiosity with the M2 wavelet basis.

Both the radiosity packages were set to use three gather iterations for hierarchical radiosity, and Radiance was likewise set to calculate to three diffuse bounces of light. The approximate visibility strategy discussed in **Section 6.3.3** was not used, both to provide a fairer comparison to the other methods and so it would not obscure the effects of using face clusters as finite elements. (See, however, **Section 7.4**.) Other settings used for the various renderers, along with the steps necessary to produce consistent output amongst them, are outlined in **Appendix B**.

7.1. Simple Reflectance Tests

I carried out a number of experiments using variations of a simple scene containing three test patches; see **Figure 85**[1]. This was done to test that small-scale interreflection was being handled consistently, and as a validation test amongst the three renderers. We would obviously like to ensure that face-cluster radiosity produces the same results as other global illumination methods.

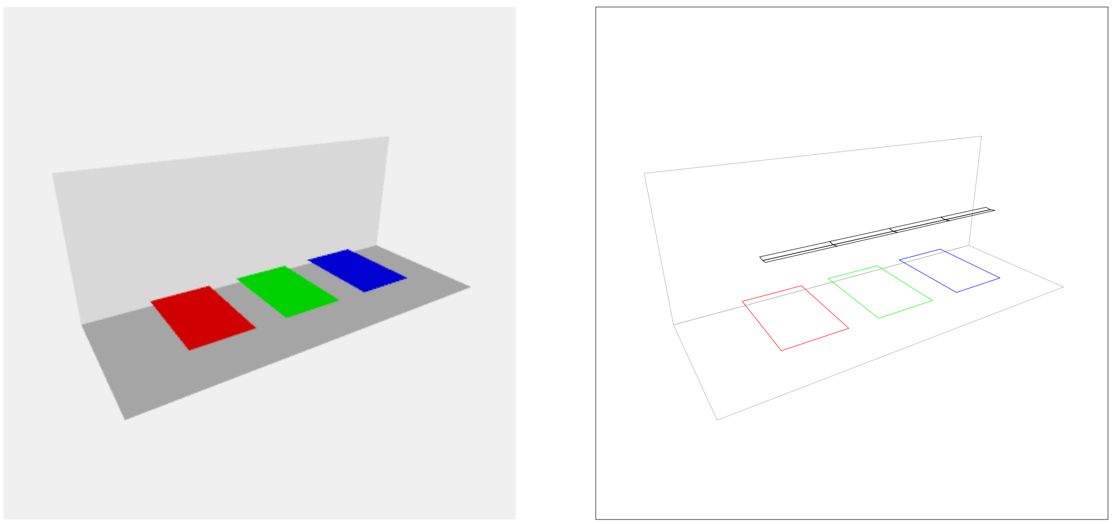


Figure 85: The reflectance test scene. Three patches corresponding to the primary colours reflect light from a long horizontal overhead luminaire onto a grey rear wall.

7.1.1. Flat Test Patches

We start by using a single, flat quadrilateral for the test patch. The main purpose of this test is to show that all three renderers produce consistent results, and that the irradiance vector-based face cluster method produces the same results as ordinary, scalar radiosity. The results are shown in **Figure 86**. As we can see, all images look nearly identical. The Radiance picture suffers from noise artifacts,

1. All results were collected on a Dell Precision Workstation 610 with a 450MHz PIII processor and 256MB of main memory, running RedHat Linux 6.1.

giving the rear wall a dappled look, especially near where the plane of the test patches intersects the rear wall. However, it has high-quality shadows.The two Radiator methods produce good strong diffuse reflections, although the shadows cast by the patch are a little softer than the Radiance solution. The RenderPark picture is probably the cleanest of the three, its M2 basis being well suited to this scene. We can conclude that all methods produce consistent results for this simple test.

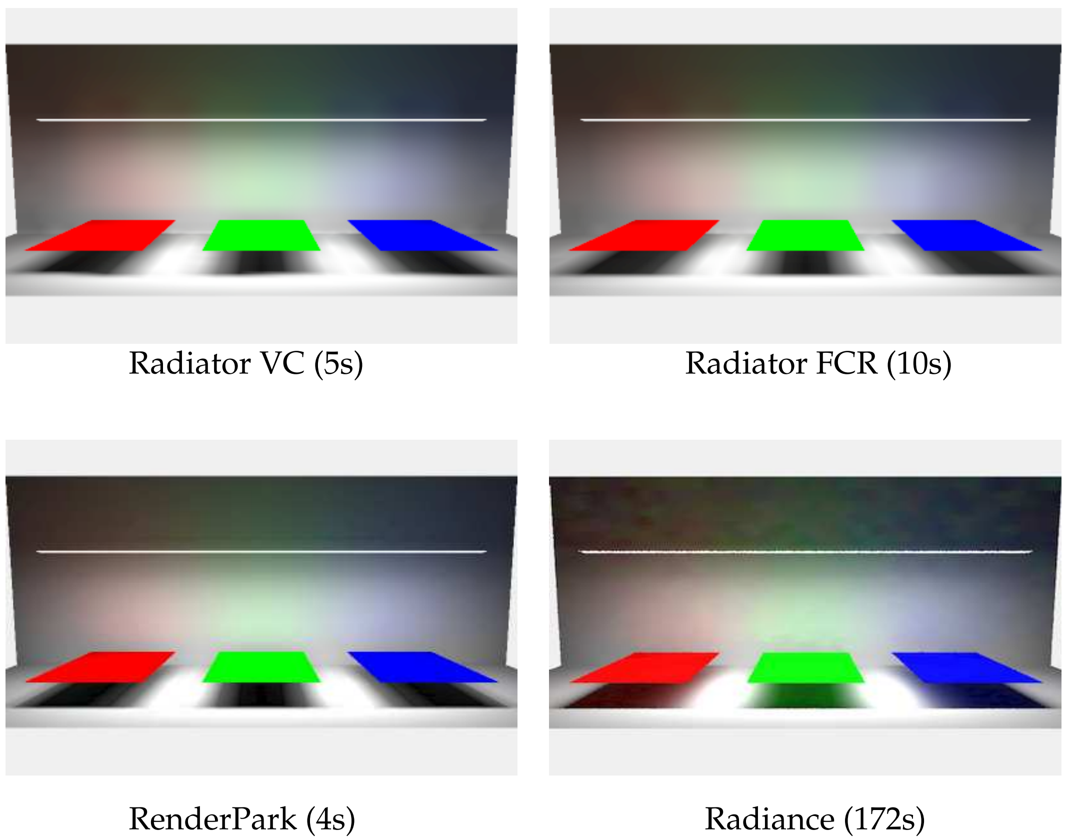


Figure 86: The flat patch scene. RenderPark does the best in terms of quality and speed. The face cluster method takes twice as long as the volume clustering methods due to the extra overhead of the irradiance vector. Radiance does the worst in terms of time and quality.

Radiance takes by far the longest of the methods, but this is partly because it has to ray-trace the final image, and it has the advantage that, as we shall see, its rendering time doesn't rise much with added scene complexity. Face-cluster radiosity takes twice as long as the two volume-clustering methods, because of the extra overhead of using irradiance vectors.

7.1.2. Bumpy Test Patches

The second test exchanged the flat patches for displacement-mapped sawtooth surfaces; these are the three-dimensional equivalent of the surfaces in **Figure 39**; an example is shown in **Figure 87**. Each bumpy surface contains 8,000 polygons. For diffuse light transfer, these patches are practically equivalent to the previous flat patches; solving the radiosity problem for this scene should ideally be no more complicated than the flat-patch scene. The results are shown in **Figure 88**.

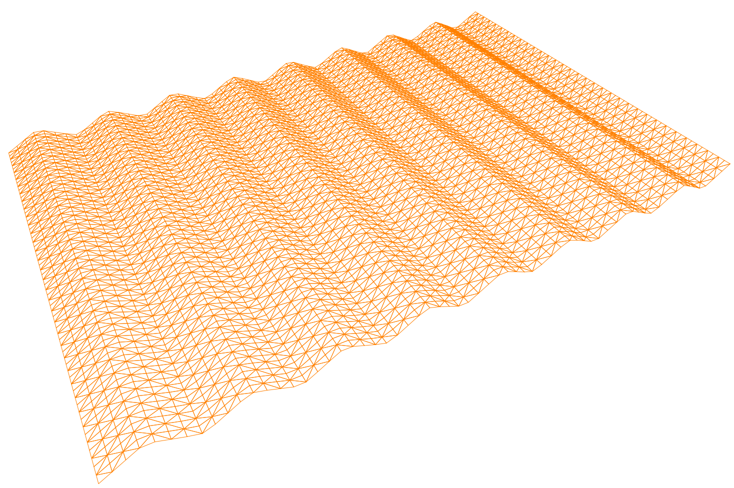


Figure 87: The sawtooth surface used in the "bumpy" test scene.

The most interesting result here is that the two volume clustering methods are very slow on this seemingly innocuous scene. As we shall see in the next test, their poor times are not due to the polygon count. The reason is the geometry of the bumpy patch. Because the volume clustering methods use the total projected

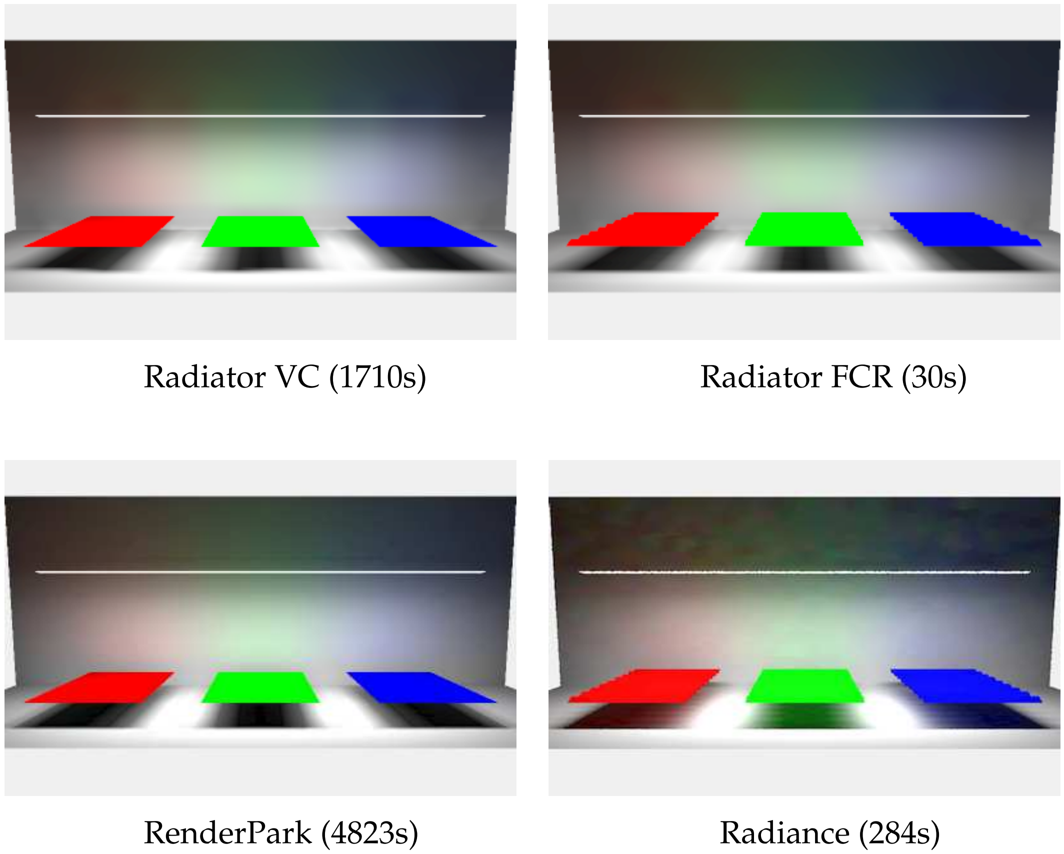


Figure 88: The bumpy patch scene. The volume clustering methods are very slow on this scene. Radiance is fast, but the rear-wall reflection is somewhat splotchy. Face cluster radiosity is even faster.

area (NUPA) in a particular direction to estimate radiosity transfer, rather than the total *visible* projected area (VPA), they can heavily overestimate the transfer parallel to the patch. (In effect they count each bump in a cluster as contributing energy to adjacent clusters, rather than just the bumps at either end.)

This results in over subdivision of the patch in an attempt to find the amount of radiosity it reflects to itself, and the consequent poor rendering time performance. On the other hand, the bounded visible projected area estimates introduced in **Section 4.3** prevents face cluster radiosity from suffering from this kind of problem; they correctly estimate the side area of the bumpy patch clusters, requiring fewer interreflection links.

For these images, each program was run such that approximately equal quality resulted. If we had normalised on speed instead, HRVC and Radiance would have looked much worse.

We conclude from this test that the higher quality approximations being made by the face cluster radiosity algorithm permit it to represent light transport using a shallower tree than HRVC, saving significant time and memory.

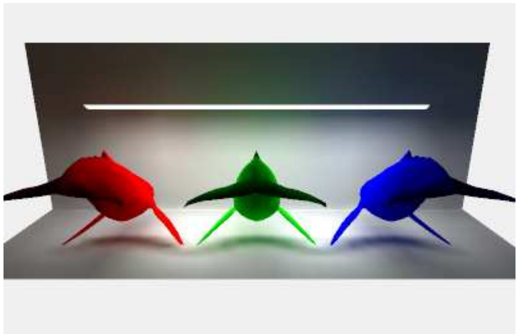
7.1.3. Whale Test Patches

As a final stress test, the medium-resolution displacement-mapped patches were replaced with high-resolution whale models, each containing 100,000 polygons. The whales provide more realistic objects, being very much the kind of model used in a high quality scene, while the test setup still allows us to check that colour bleeding is still occurring properly, and for consistent results between renderers. **Figure 89** shows the results for both a simplified, 1,000 polygon of the original model, and the full, detailed model. **Figure 90** shows a close-up of the whales of the face cluster radiosity solution.

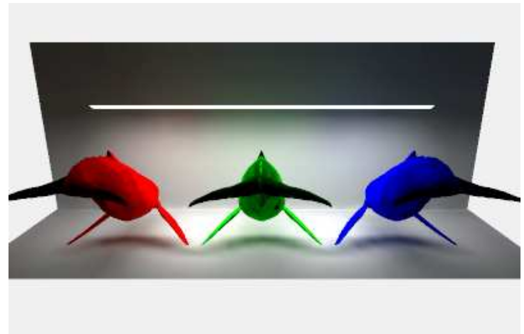
While the simulation times for face cluster radiosity and the two hierarchical radiosity with volume clustering methods are similar for the simplified, low-detail whales, the times diverge dramatically for the more complex scene, even though both were run with the same refinement epsilon as for the simpler scene, and both produce similar results. Face cluster radiosity takes hardly any more time at all to deal with the complex model, while RenderPark in particular suffers from a twenty times increase in solution time. (Most of the increase in the rendering time for face cluster radiosity can be attributed to the greater expense of visibility queries for such a large scene.

Radiance does well with the full resolution whales compared both to the simplified whale scene and the previous bumpy-patch scene; its rendering time increases by about half. Its under-fin shadows are the best of the methods, although the softer shadows start to get a little noisy. The linear basis functions of RenderPark have some problems with the shadows in this scene; inter-patch discontinuities are obvious in several places.

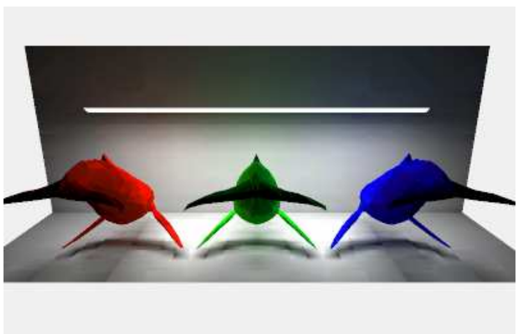
The statistics for the renderers are shown below in **Table 10**, including the preprocessing time for the Radiance and FCR methods. (With Radiance you must build an on-disk octree of the scene before running the actual renderer, using the program *oconv*.) Interestingly, this preprocessing time is roughly the same for both



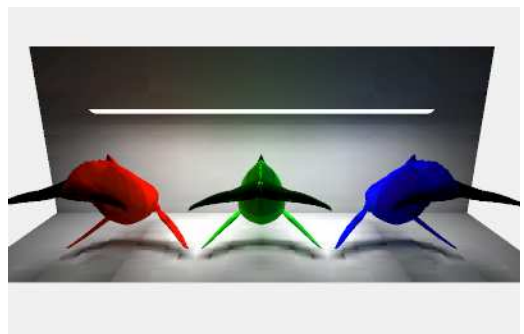
FCR (113s)



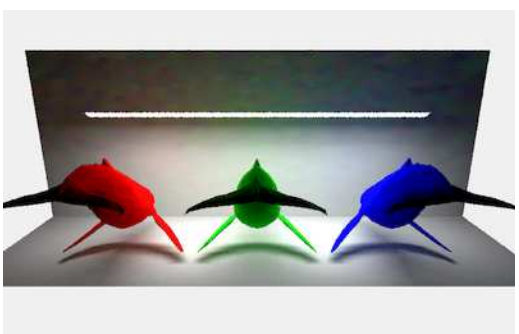
FCR (127s)



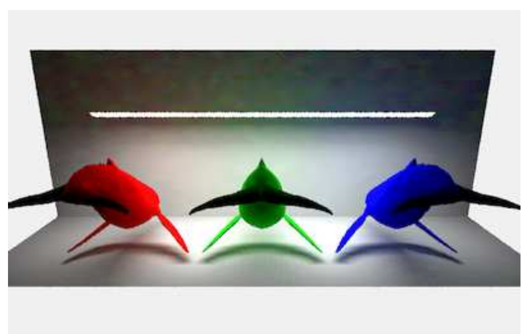
RenderPark (125s)



RenderPark (2702s)



Radiance (248s)



Radiance (378s)

Figure 89: The test patch whales. These magnificent creatures of the deep gracefully reflect light onto both the rear wall and the floor. Left: Results using a simplified 1,000 polygon whale model. Right: Results for the full scene; each whale contains 100,000 polygons.

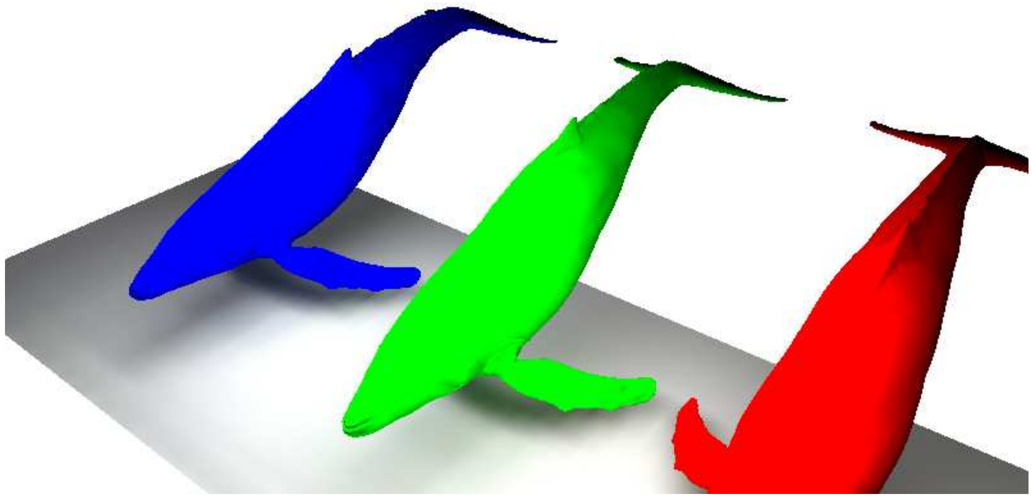


Figure 90: A whale close-up from the FCR solution.

methods; though the face-clustering time is shown as one third of the octree-building time, this is because the whale's cluster hierarchy is reused across the three instances of the whale in the scene.

	Simple whale		**Complex whale**	
Renderer	**Rendering**	**Preproc.**	**Rendering**	**Preproc.**
Radiator Vol. Clustering	111s		953s	
RenderPark	125s		2702s	
Radiance	248s	1.5s	378s	76s
Face Cluster Radiosity	113s	0.2s	127s	25s

Table 10: Rendering times for the Test Whale scene.

7.2. The Museum Scene

For a more general test of renderer performance, I designed an indoor scene more typical of those seen in the radiosity literature. The scene contains a number of high resolution scanned models, a polygonized implicit surface (the podium), and a displacement-mapped surface (the floor). These models range in complexity from 4,140 polygons to 100,000 polygons. For the lighting design, there are three overhead spotlights illuminating in turn, the Isis statue, the Buddha statue, and the table. The Venus head is illuminated by a floor-mounted red spotlight.

The scene is available in the MGF format [Wardb] in a number of different resolutions from the web page http://www.cs.cmu.edu/~ajw/thesis/museum-scene/.

7.3. Empirical Complexity

Using the quadric-based surface simplification method [Garl97], I applied all renderers to versions of the museum scene with varying polygon counts, to demonstrate the effect of using ever more detailed models on these algorithms[1]. Because the Radiator algorithms share the same code base, we can be reasonably sure that any differences in results are not due to any variance in code optimization or approaches to, for example, visibility testing or geometrical representation. I also performed the same tests using Radiance and Renderpark, to verify my results against well-known external code bases.

While previous experiments have investigated the effect of increasing the lighting or geometric complexity of a scene on radiosity algorithms [Will97b], this experiment shows the effect of increasing model complexity in a scene with fixed geometric layout. While exact error comparisons between the solutions are not available due to the difficulty of generating a reference solution, I took care to use similar parameter settings for all three methods. The most complex version of the scene in this experiment was generated by simplifying all large models to 100,000 polygons, and creating multiresolution models from the results. Ten "complexities" of the scene were then generated by simplifying each model to have s^2k polygons, where k was the number of polygons in the highest resolution of the model, and scene n had $s = n/10$. This produced ten scenes with polygon counts ranging from 7,400 polygons to 546,000 polygons. **Table 11** shows the time it took to generate the associated face cluster hierarchies for the highest complexity level, and the size of each file.

Figure 91 shows the results of running the two volume clustering methods on this scene. Interestingly, they both have very poor, super-linear performance in time; this is much worse than the $k \log k$ performance we would hope to see. This occurs for similar reasons as their poor performance in the "bumpy" test patch scene; the flat but bumpy stone floor in the original scene leads to over refinement

1. All results were collected on a Dell Precision Workstation 610 with a 450MHz PIII processor and 512MB of main memory, running RedHat Linux 6.1.

Model	**Polygons**	**Clustering Time**
Buddha	100,000	25.3s
Isis	100,000	25.5s
Dragon	100,000	25s
Venus	100,000	25s
Pedestal (under Venus)	24,000	5.7s
Candlestick x 2	4,150	1s
Floor	100,000	18s
Picture x 2	7,400	1.63s
Total	**547,100**	**127.13**

Table 11: Face Cluster Hierarchy Generation

due to overestimates of the side areas of floor clusters, especially with the red spot-light spilling directly onto it. **Figure 92** shows images of the resulting scenes.

To give in some senses a fairer comparison of these methods with face cluster radiosity and Radiance, I reran them on a version of the scene without the detailed stone floor. It was instead replaced with a flat mesh, having the same number of polygons as the original floor in each scene. This gave performance results much closer to those expected for hierarchical radiosity with volume clustering, but it should be noted that in the following results both face cluster radiosity and Radiance are solving a more difficult scene.

To give an idea of the image quality produced by the various methods, **Figure 93** and **Figure 94** show images for the lowest and highest resolution scenes respectively.

Figure 95 shows a graph of solution time for Radiance, volume clustering, and face clustering. (RenderPark was omitted because its solution times were so much larger; on the order of hours.) Notably, the time cost of the face cluster radiosity algorithm stays approximately constant, while the other methods have costs that are linear (volume clustering) and sub-linear (Radiance) in the number of input polygons. As suggested in **Figure 96**, this is because after a certain point, fine levels of detail of the face cluster hierarchy are never accessed, in spite of the increasing polygon count. .

The most important conclusions we can draw from this test are

- unlike previous finite-element methods, face cluster radiosity is sublinear in the number of input polygons;

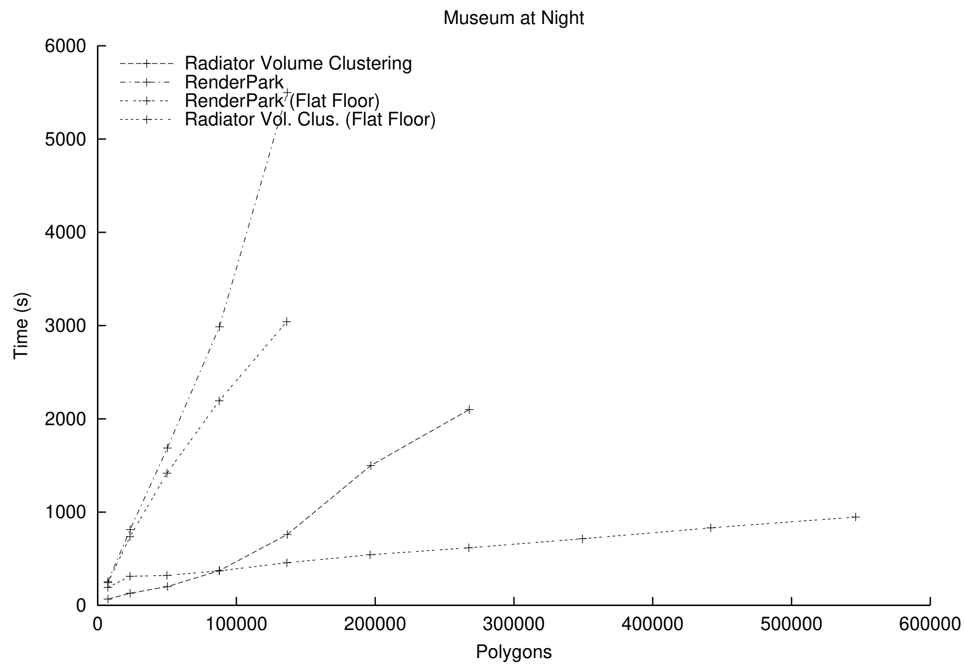
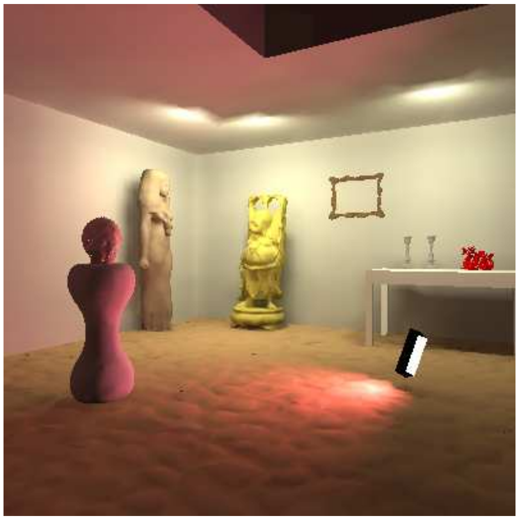
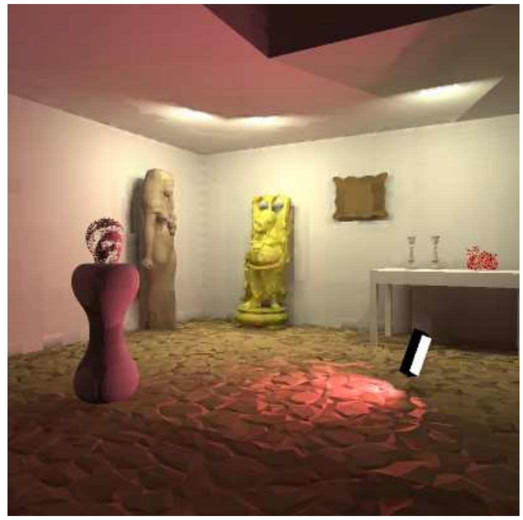


Figure 91: Volume clustering on the museum scene. The bad performance of volume clustering methods on the stone floor is illustrated by the much better performance of the methods when the floor is replaced by a single, flat polygon. In general, RenderPark takes much longer to render the scene than Radiator's Haar-based volume clustering, even though its results are not as good. (See **Figure 92**.)

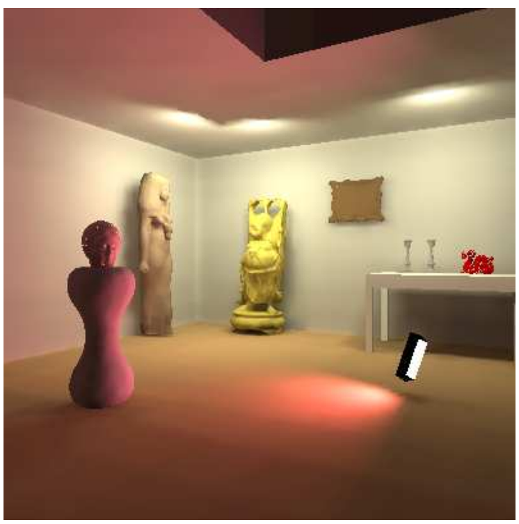
- beyond a certain point (200,000 polygons in this case) its solution time is independent of the number of input polygons;
- volume clustering methods can do poorly on flat but bumpy surfaces, due to over-refinement;
- Radiance is also sublinear in the number of input polygons, although it does not level off so quickly, and has a higher constant than face cluster radiosity.



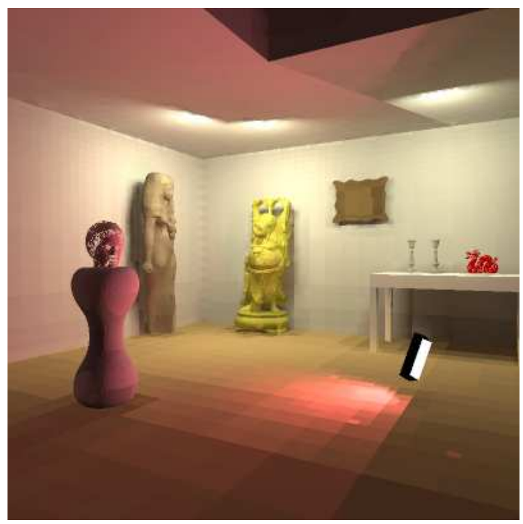
Radiator, Haar (760s)



RenderPark, M2 basis (5500s)



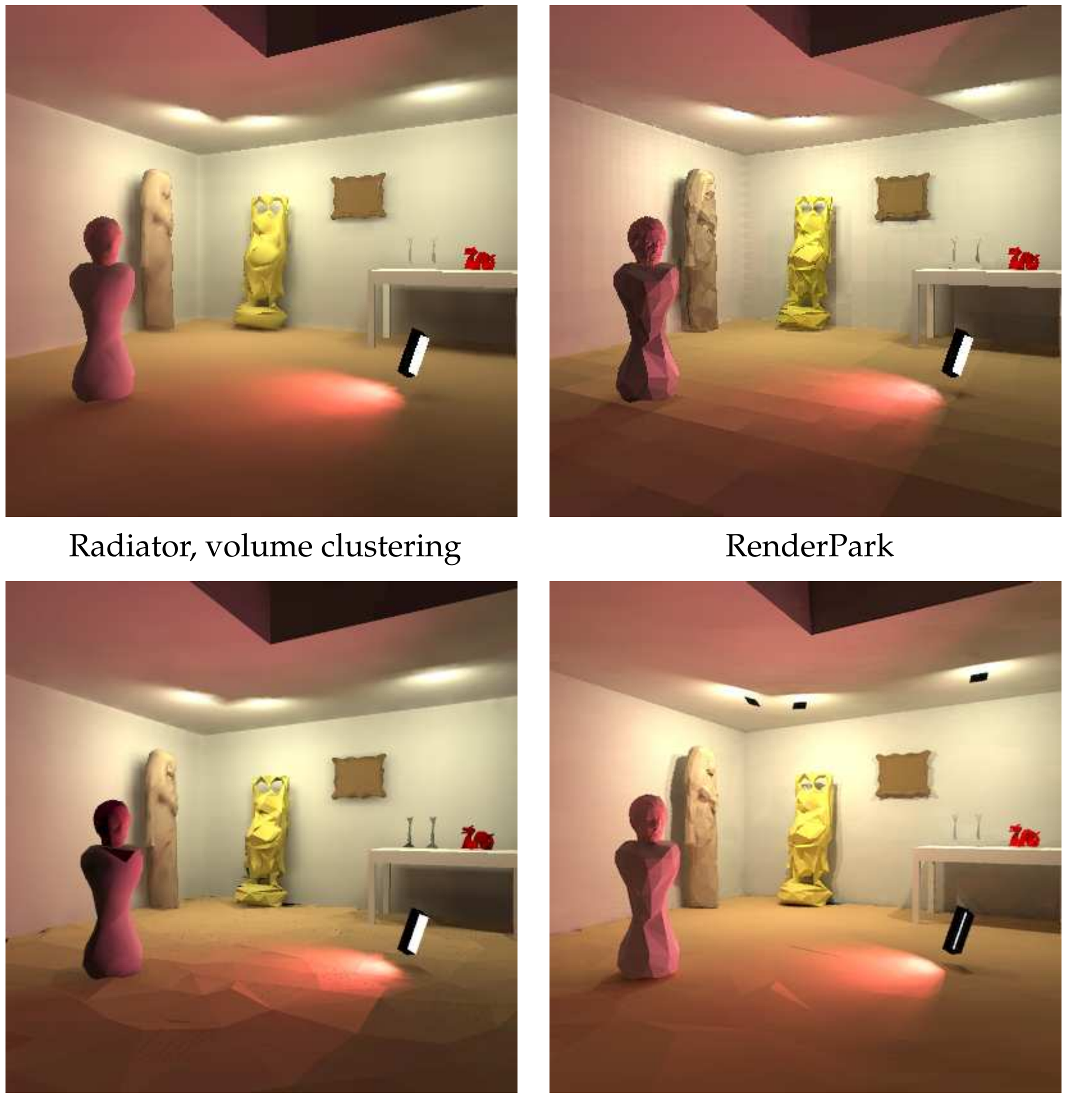
Radiator, Haar (207s)



RenderPark, M2 basis (3042s)

Figure 92: Images for the volume clustering methods, medium scene complexity (140,000 polygons). Rendering times are drastically reduced by replacing the detailed stone floor (top) with a flat one (bottom).

Note that overall Radiator gives better results than RenderPark for this scene. It is possible to get higher quality output from RenderPark, but doing would raise the already large rendering time.



Radiator, volume clustering

RenderPark

Radiator, face clustering

Radiance

Figure 93: Images for the lowest scene complexity level for the four methods. This scene contains 7,400 polygons.

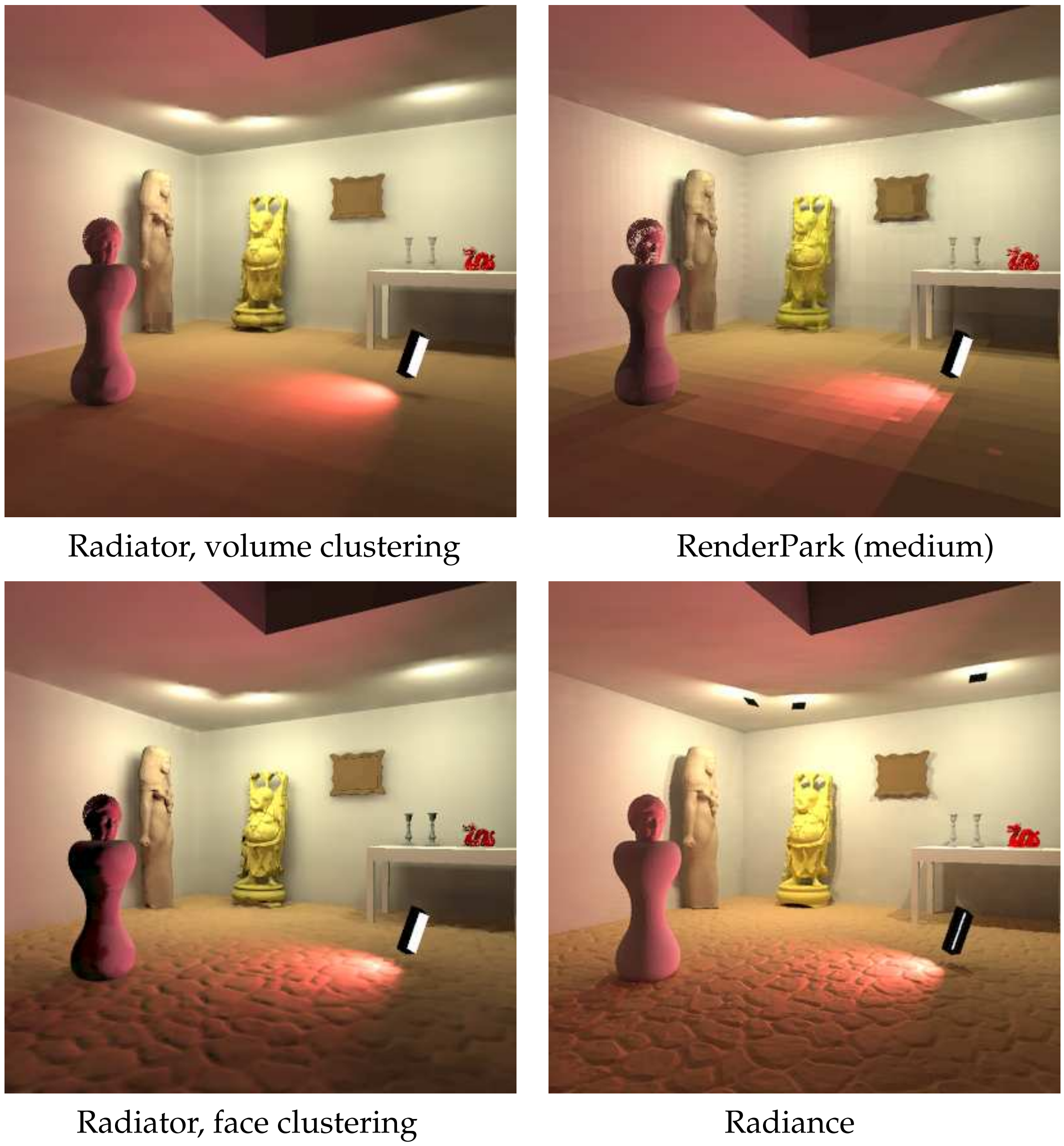


Figure 94: Images for the highest scene complexity level. This scene contains 546,000 polygons. Face clustering and Radiance both produce the best results.

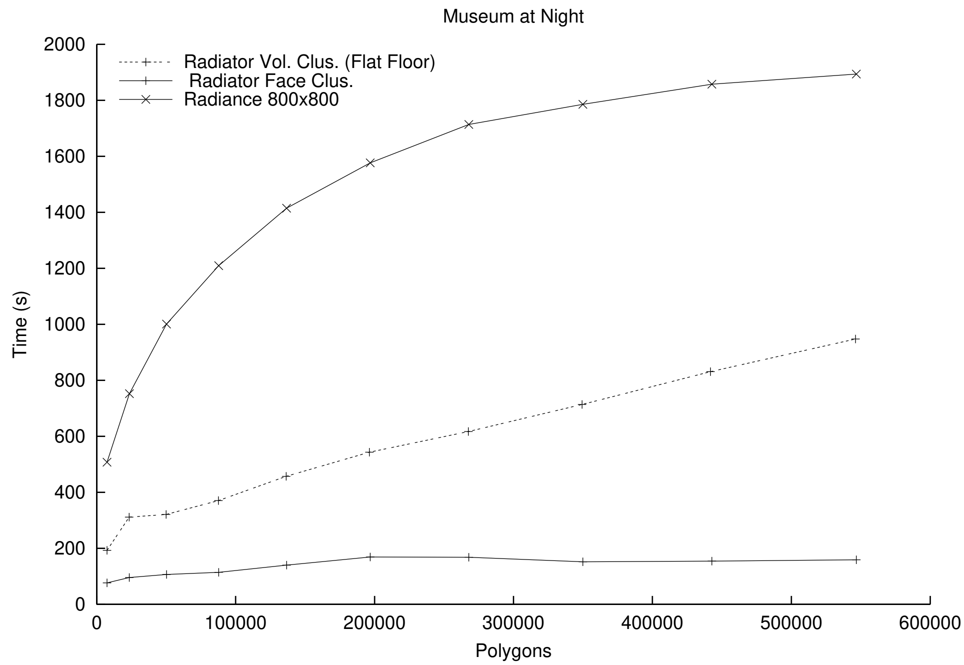


Figure 95: Rendering times for Radiance, volume clustering, and face clustering. While Radiance is initially expensive, its sub-linear performance means it can handle large scenes as well as smaller ones. Volume clustering is rendering a simpler scene than the other two methods, because of its quadratic behaviour with a non-flat floor. Its performance is linear, and thus eventually it will be more expensive than radiance. Face clustering, on the other hand, has close to the ideal flat performance after 200,000 polygons, and is much faster than the other two methods. It produces better results than volume clustering, and comparable results to Radiance.

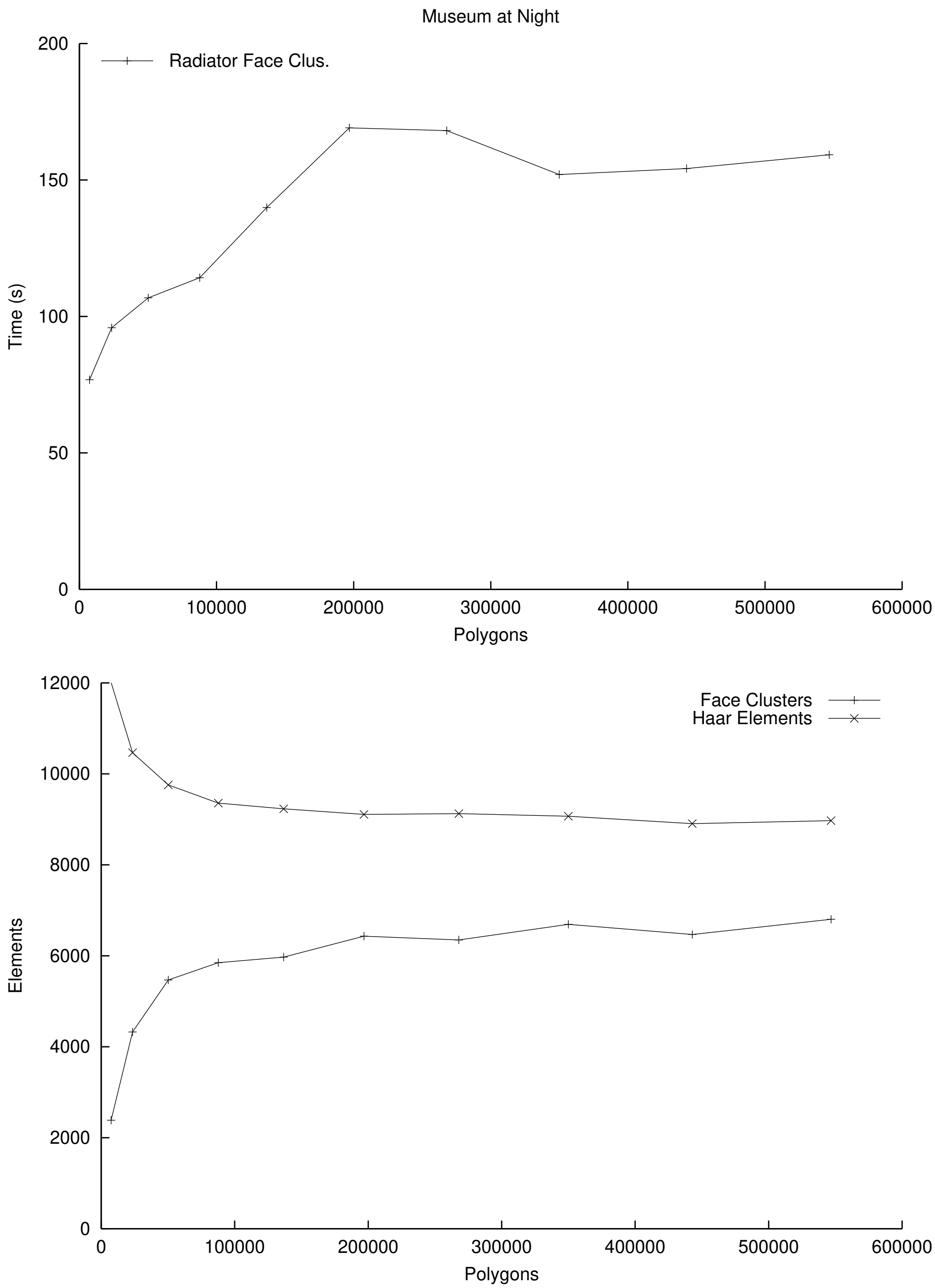


Figure 96: Rendering time for face cluster radiosity. Top: The rendering time increases until around 200,000 polygons, and then levels out. Bottom: initially Haar elements predominate in the final solution. These are gradually replaced by face clusters until a steady state is reached around 200,000 polygons.

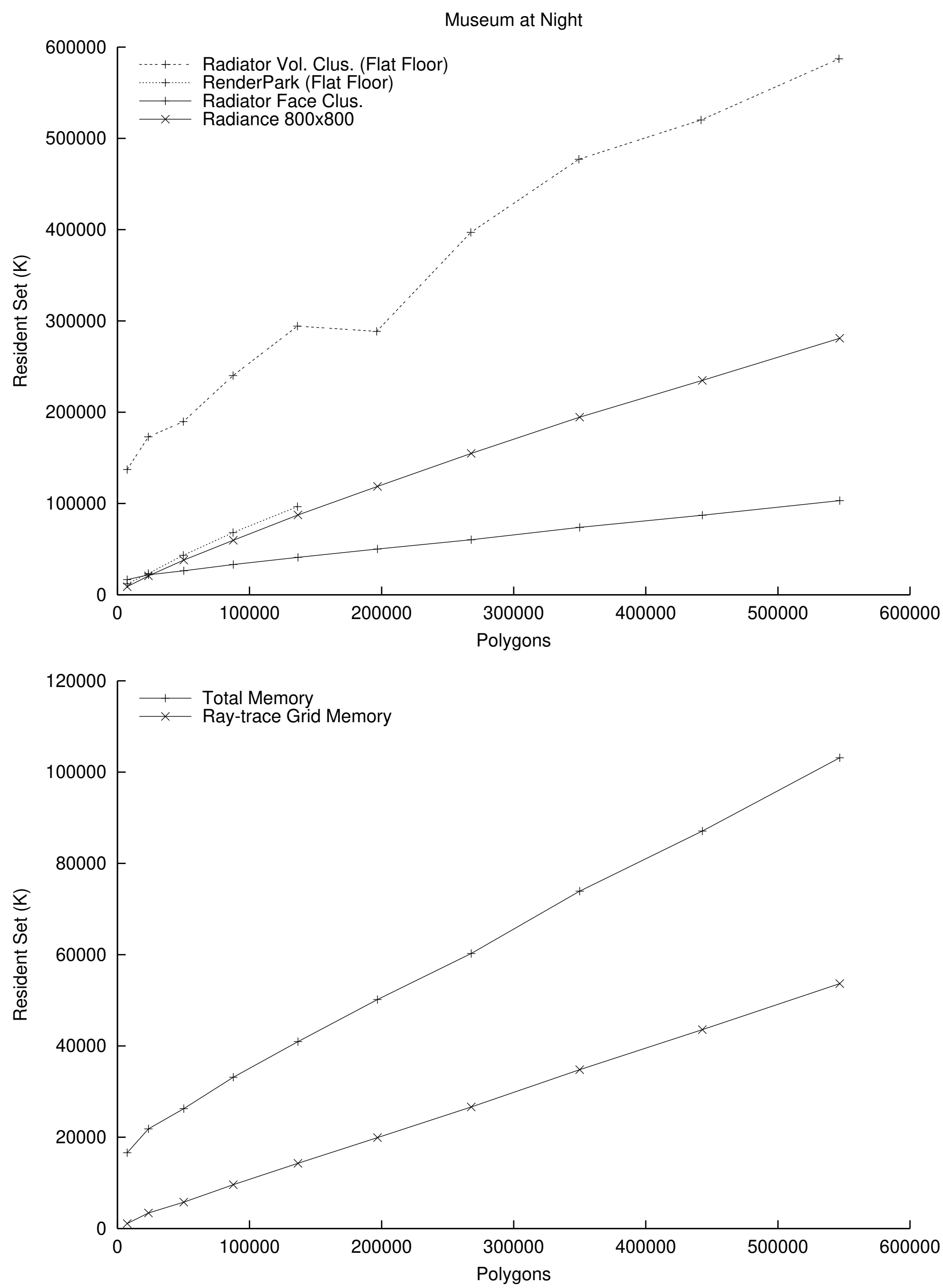


Figure 97: Memory results. All methods demonstrate sub-linear memory complexity. The volume clustering methods and radiance have very similar memory consumption. Bottom: Much of the memory use in face cluster radiosity is attributable to the data-structures necessary for occlusion testing via ray-tracing

7.4. A Highly Complex Museum Scene

As a final stress test of the radiosity, I ran face cluster radiosity on the museum with the full resolution version of each model. The full scene contains over 3.3 million polygons. The resulting illuminated mesh is shown in **Figure 98**, and some statistics for the corresponding radiosity run given in **Table 12**.

For this scene, the approximate visibility strategy outlined in **Section 6.3.3** was used, primarily as a memory-saving device. Without its use, the ray-tracing data structures alone took up some 330 MB of memory. By using the face cluster hierarchy to provide approximate visibility, this was reduced to under 40 MB[1].

In some senses this is an unrealistic scene; the models it contains are far more detailed than needed for most viewpoints, except for extreme close-ups. However, it does illustrate the freedom face cluster radiosity gives us with respect to model size. No other finite-element radiosity method is capable of rendering this scene without requiring an unrealistically large amount of CPU time and especially memory. Previously, only parallelised radiosity methods have been able to deal with scenes this large.

The extra detail in this scene can be important. **Figure 99** shows two close-ups of the Buddha statue in the scene—one from the complexity scene discussed in the last section, and one from the full-size scene. The full-size statue clearly shows extra detail over the simplified one, which looks like a smoothed version of the original. In some cases, and for some camera angles, this detail can be important.

Figure 100 shows a close-up of the Dragon model in the full-size scene. This picture is useful because it illustrates some of the limitations in the method; the model is small enough that its lighting is relatively coarse compared to the larger models. In particular, there are obvious discontinuities at shadow boundaries, most particularly on the hump of the dragon. The worst artifact is the outer claw on the hind leg, where the cluster has been incorrectly treated as entirely in shadow. In general, incorrect shadow resolution is most often responsible for any artifacts in output from the face cluster radiosity algorithm.

These artifacts can be eliminated by lowering the refinement epsilon to a suitable level, at the expense of increased rendering time. However, this will also increase the accuracy of all other parts of the solution scene, even if this accuracy is unnecessary—this is a limitation of having one global epsilon value controlling the accuracy throughout the scene. For a particular animation containing both longshots and extreme close-ups, using the camera path to set epsilon differently

1. The complexity level used was 0.1.

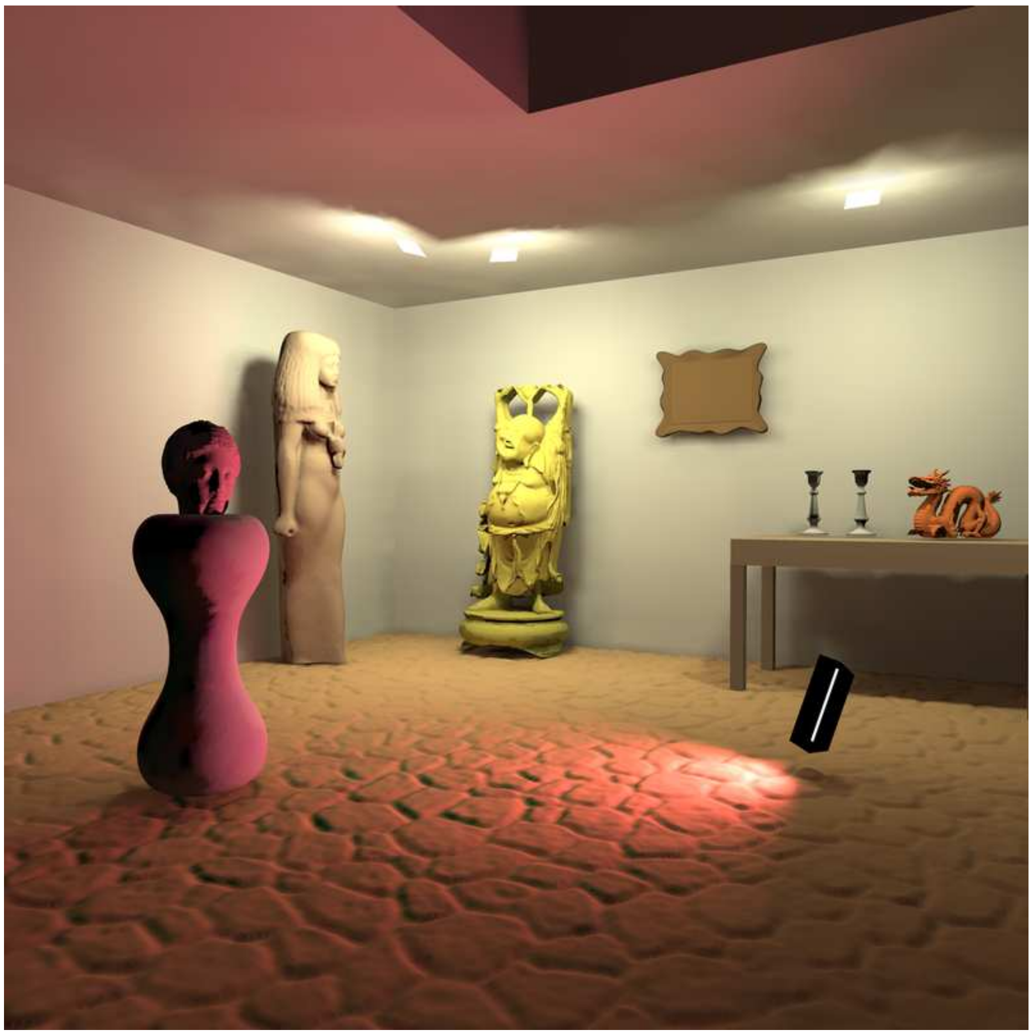


Figure 98: The highly complex museum scene. This scene contains 3.3 million polygons. The solution took 216 seconds. A final pass with vector irradiance interpolation took a further 105 seconds. (See **Table 12**.)

After	Time	Polygons	Face Clusters	Transport links	Memory
Initial linking	0 s	142	92	1	32 MB
Iteration 1	120 s	11658	8478	74510	39.2 MB
Solution	216 s	11,670	8,666	120,345	41 MB
Mesh Colouring, including VRI	321 s	11,670	8,666	120,345	41 MB

Table 12: Statistics for the big scene. The memory use includes ray-tracing data structures (a constant 32MB) and radiosity data structures only, ignoring geometry overhead. There were 33 volume clusters in this scene.

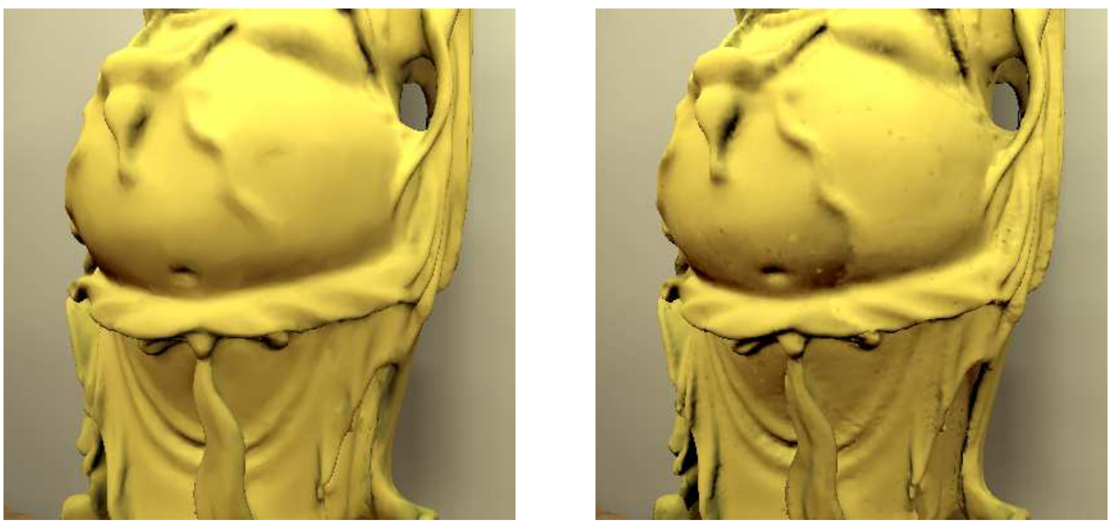


Figure 99: Close-ups of the Buddha for a 100,000 polygon model (left), and the full 1,000,000 polygon model (right).

in different parts of the scene, or the application of importance-based radiosity concepts [Smit92], may help reduce solution times significantly.

The results shown here use a final pass in which the irradiance vector to each cluster is resampled at four points, and then interpolated between those points (**Section 6.3.6**). While this process typically adds 50% to the total rendering time, it can be worthwhile where illumination is changing rapidly, especially on a

Figure 100: A close-up of the Dragon model from the museum scene.

smooth surface. **Figure 101** shows the effects of this process on the front of the Isis model, where it makes a noticeable improvement.

The polygon counts and face clustering times for each of these models are shown in **Table 13**. The preprocess of building face cluster hierarchies for each model in the Museum scene took a total of 436 seconds, and no model took more than 150MB of memory to process (and that only for the million-polygon model.) As pointed out previously, this cost can be heavily amortized; the same models can be reused over multiple scenes and radiosity simulations.

Finally, **Figure 102** illustrates the difference global illumination makes to this scene. The floor from the full museum scene radiosity solution is shown, together with another solution with all objects removed except for the table and the lights, for comparison purposes. The latter essentially shows the floor with direct illumination only. Notice how in the full solution the reflected light from the walls "fills in" the unlit areas of the floor, and under the table. Also, there is a yellowish area in the upper left of the full solution where light spills off the buddha statue onto the floor.

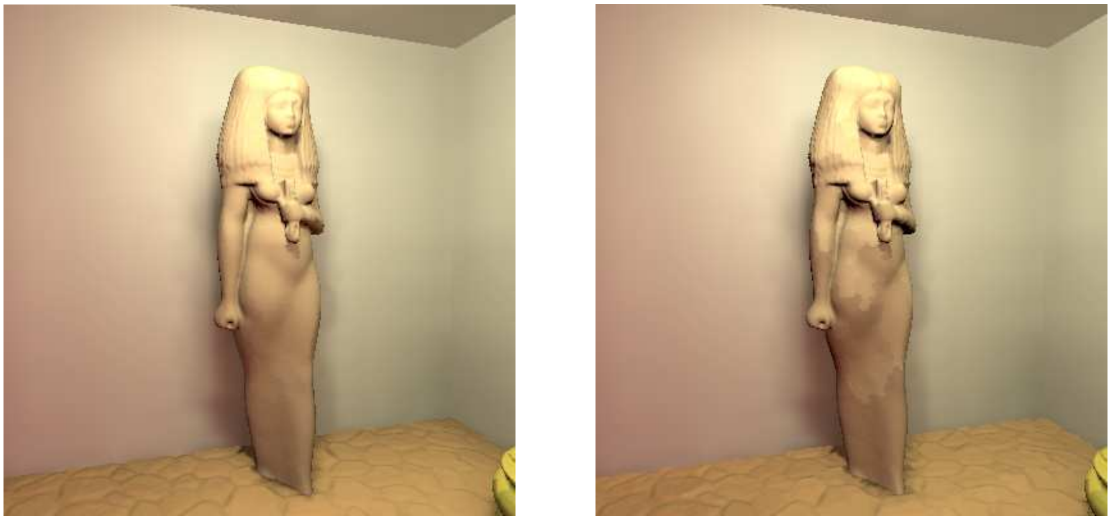


Figure 101: The effect of vector irradiance interpolation on the museum scene. Shown is a close-up of the Isis statue both with (left) and without (right) interpolation.

Model	Polygons	Clustering Time	Disk Use	Disk Use (Truncated)
Buddha	1,085,634	148s	158 MB	30 MB
Isis	375,736	47s	55 MB	11 MB
Dragon	869,928	109s	127 MB	24 MB
Venus	268,714	34s	39 MB	8.2 MB
Pedestal	24,000	5.7s	3.4 MB	744 kB
Candlestick x 2	4,150	1s	621 kB	132 kB
Floor	717,602	87s	104 MB	22 MB
Picture x 2	7,400	1.6s	1.1 MB	246 kB
Total	**3,357,310**	**436s**	**488.2 MB**	**96.7 MB**

Table 13: Face cluster generation for the full scene

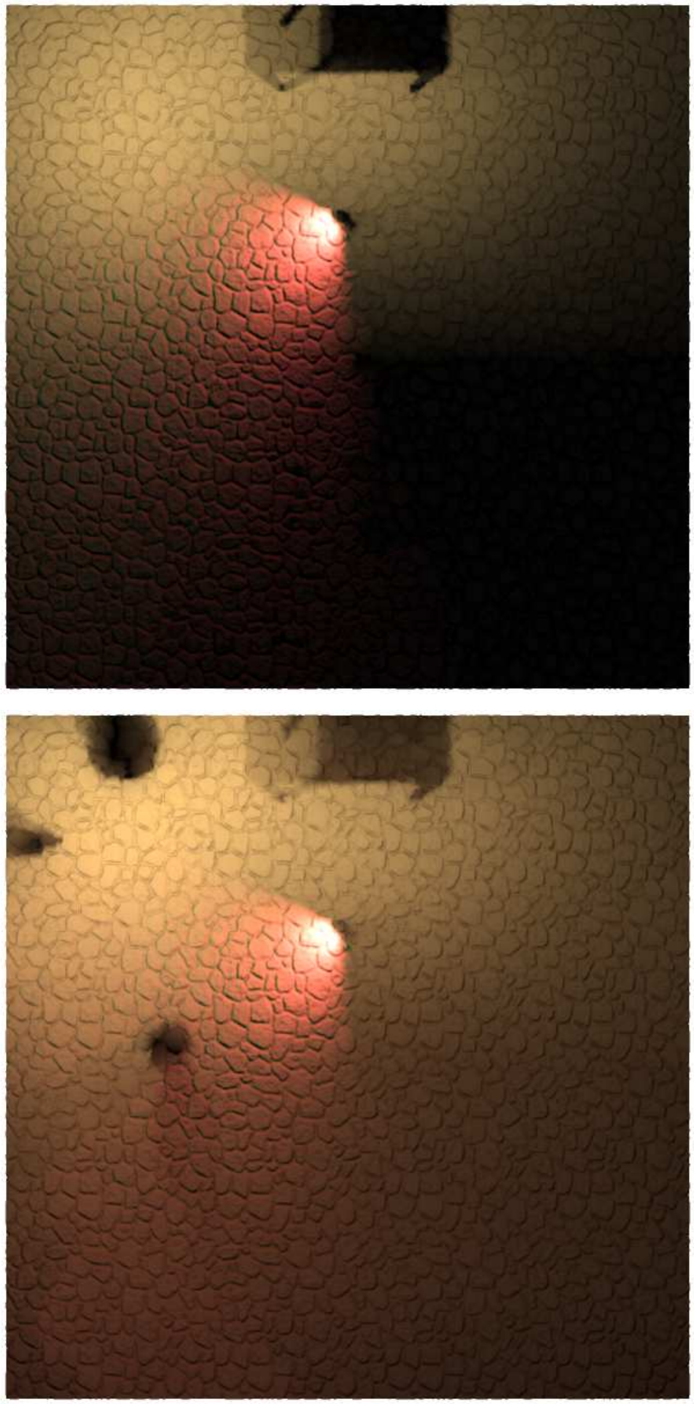


Figure 102: The effects of including diffuse global illumination. Shown are the stone floor from the museum scene with direct lighting only, top, and the complete radiosity solution for the floor, bottom.

Chapter 8

Conclusions

The simulation of global illumination is an important tool in generating realistic images. It is challenging because the illumination in an environment is interdependent; the light reflected by any object in the scene potentially depends on the light reflected by all other objects in the scene.

Finite element-based radiosity methods solve for the global illumination of a scene containing diffuse (matte-like) surfaces. They are useful because their output is view-independent illuminated geometry, which can be re-rendered from any viewpoint, without repeating the global illumination calculations.

8.1. Contributions

In this dissertation, I have described a method for performing radiosity calculations on scenes containing large, detailed models. I call this method *face cluster radiosity*. It can result in large performance gains when model detail is greater than the natural element size needed to solve for scene radiosities. Because it uses surface-based clusters, it avoids a number of potential artifacts possible with the previously-used volume-based clustering algorithm. In particular, irradiance can be interpolated across the cluster in a natural way. Thus solution quality is also improved.

My algorithm is capable of performing the most time-consuming part of the radiosity simulation, solving for element radiosities, in a time sub-linear in the number of input polygons. This compares favourably to the $k\log k$ complexity of the previous best algorithm, hierarchical radiosity with volume clustering. In particular, after a certain point, arbitrary tessellation of the input scene does not affect the solution time. That is, in the limit, the complexity of the solution phase of the new algorithm is constant time.

The algorithm does rely on a preprocessing phase which is $O(k\log k)$, but this cost is a per-model one that can be amortized over a number of different model instantiations and scenes. The complexity of the post processing phase during which mesh vertices are actually coloured must of course be $O(k)$, but the constant is quite small.

My algorithm has three essential components.

- **The use of a surface-oriented *face cluster hierarchy*.** These hierarchies can be pre-calculated on a per-model basis. The hierarchies can be built with a variant of Garland & Heckbert's quadric-based simplification algorithm, one that works on the geometric dual of the model in question. This has the advantage that it optimizes for the production of flat clusters.
- **Radiosity calculations based on *vector irradiance*, rather than scalar irradiance.** This allows us to avoid the push-to-leaves phase normally associated with radiosity clusters, while still modelling the effect of the variation in surface normal during the final model-colouring stage. This saves considerably on memory use and time, as unnecessary model detail need never be paged in.
- **Bounds on the projected area of a cluster of connected polygonal faces.** I have established constant-time methods for finding relatively tight upper and lower bounds for the projected area of a face cluster. This is a crucial step in accounting for the error introduced by the vector radiosity approximation, and thus driving refinement of clusters to the point where this error is acceptable.

Together, they make it possible to calculate radiosity solutions for scenes that were not tractable with previous methods, on a modest-sized personal computer. They also make radiosity on such scenes much faster than the best existing ray-tracing approach, which uses diffuse-sample caching. It is possible, for example, to compute a radiosity solution for a scene of three million polygons in four minutes on a 450 MHz Pentium processor with 160 MB of memory. The improvements to the radiosity algorithm presented here will help make it more usable in industry for applications such as architectural CAD, movie special effects, and video games, especially as the number of polygons used to represent model and scene detail continues to climb.

Finally, I have also

- Modified Garland's original face cluster creation algorithm to be robust in the presence of flat mesh regions, produce more balanced hierarchies, and run in essentially linear time. The new implementation, *make-fch*, can cluster a million polygon model in under three minutes.
- Implemented the face cluster radiosity algorithm within the context of my radiosity program *Radiator*. This program also handles most previous radiosity algorithms, and has a visual front-end for demonstration purposes.

Together these programs make up more than 80,000 lines of C++ code. The code and associated documentation is freely available for research or production use from the web site http://www.cs.cmu.edu/~ajw/thesis-code/.

8.2. Future Directions

There are a number of ways to extend the method presented in this dissertation. The following avenues of research show particular promise.

Higher-order irradiance vector representations

The irradiance vector plays a prominent part in face cluster radiosity. However, it is to vector radiosity what the Haar basis is to scalar radiosity; a constant quantity averaged over the finite element in question. Thus there exists the opportunity to exploit smoothness in the radiosity transport kernel by generalising vector irradiance to tensor representations of various ranks. This could also make interpolating vector irradiance unnecessary, though experience with wavelet bases in the scalar domain indicates this may be unlikely.

Glossy Illumination

Any work on radiosity methods is almost compelled to mention extensions to handle specular or more realistically glossy reflection, although this is rarely carried out. In our case, however, the work described in **Chapter 4** holds some promise. In the face cluster radiosity algorithm, we are essentially modelling a collection of small, connected diffuse surface elements with a single larger element. If our lower and upper bounds could be generalised to surfaces with more general radiance distributions, there exists the possibility that the same could be done with non-diffuse surface elements. The results could be invaluable in either controlling a glossy global illumination algorithm, or determining "good" sampling directions for a Monte Carlo-based algorithm.

Better Visibility Approximations

Currently fractional visibility is used during the radiosity simulation, and (when interpolation is used) area-to-point visibility is interpolated for the final output pass. This relatively coarse approach to sampling visibility could be usefully refined by employing higher-order approximations.

Also, in this dissertation, visibility sampling has been carried out by using a nested grid representation of the original scene. This can result in large amounts of memory being used to store the grids. Although we can employ a form of approximate visibility to reduce memory and time requirements, we must pick a single approximation level, and this must be more complex than the radiosity solution, to avoid self-shadowing problems. Thus the resolution of our visibility queries is often much greater than strictly required, especially when sampling visibility over the larger finite elements in the scene.

An interesting alternative currently under investigation is to use the face cluster hierarchy to enable multiresolution visibility queries. Queries would have some notion of the region over which visibility was being sampled, and we would only descend as far down the cluster hierarchy, which acts as a bounding box hierarchy, as needed to properly cover this region. Moreover, the projected area of a face cluster, which I have investigated heavily in the course of this thesis, is intimately related to the probability of a ray hitting the surface within that cluster.

Previous algorithms proposed to speed up visibility sampling for radiosity have used density grids to avoid ray-casting against the original scene geometry, but such visibility estimates are isotropic. For instance, a long flat surface will take on the width of the grid's cell size. By taking advantage of the various projected visible area estimates we have for face clusters, we can avoid this isotropy.

Handling disconnected topology

The algorithm presented in this dissertation performs best on connected surfaces. It attempts to exploit the coherence such surfaces present. Groups of disconnected surfaces are handled by using the volume clustering approach much used by previous hierarchical radiosity algorithms. More could be done to bridge the gap between these two representations. When we have a number of disconnected surfaces which are similarly oriented, and do not overlap much, we could use the same approach as for face clusters with little loss of accuracy. When we have a number of randomly oriented surfaces, with significant shadowing occurring, we should fall back to a more volume-oriented approach, but it is possible we can still produce constant-time projected-area bounds by using a sample interpolation method similar to that presented in **Chapter 4**. Identifying these situations and exploiting them is an avenue for further research.

Visual Masking of Bumpy Surfaces

While working with face cluster radiosity, I have observed that when a surface contains many bumps and ridges, and consequently the scalar irradiance of the surface is changing rapidly, errors in the radiosity field are much less noticeable. It seems plausible that the visual masking techniques of Ferwerda et al. could be used to add a "masking" weight to each cluster [Ferw97]. Clusters with small weights could then be refined less than smoother clusters.

Illumination Maps

One of the most exciting possible further directions of this research is the adaption of the algorithm to produce an *irradiance vector map*. The output of the solution algorithm, before the final post-processing stage, essentially partitions the geometry into a set of regions, each with a number of (possibly interpolated) irradiance vectors representing the illumination of that region. We can refer to this set of (disjoint) regions as the irradiance vector map.

This map could then be used to render illuminated geometry quickly and efficiently, as the viewpoint changes, taking into account specular reflection. Rendering each frame would only require evaluating the local illumination, and only for a single direction. It has the great advantage that this re-rendering would be independent of the number of light sources in the scene. This kind of vector-based shading also lends itself well to some of the more recent graphics hardware [Heid99, Kaut99].

This would not result in a complete global illumination solution, as specular reflections are limited to the final stage of any illumination path. Moreover, approximating the incoming light as uni-directional may be unrealistic where specular BRDFs are involved; a small number of the most significant transport links may have to be used instead. However, such an approach may be able to match the quality of the radiosity/ray-traced post-process algorithms commonly in use for architectural global illumination solutions.

Indirect output primitives

Currently, the focus of the algorithm has been the output of illuminated geometry: a (possibly refined) version of the input scene, together with vertex colours. Interesting alternatives include the generation of shadow maps and environment maps or virtual light sources. Commodity graphics hardware is starting to support these primitives, and at some point it will become reasonable to essentially perform the final pass of the radiosity algorithm in hardware.

Bibliography

[Appe85] Andrew W. Appel. An efficient program for many-body simulation. *SIAM Journal on Scientific and Statistical Computing*, 6(1):85–103, January 1985.

[Arvo94] James Arvo and Andrew Glassner. The irradiance jacobian for partially occluded polyhedral sources. *Proceedings of SIGGRAPH 94*, pages 343–350, July 1994.

[Auto] Autodesk. Lightscape. http://www.lightscape.com.

[Ball81] Dana H. Ballard. Strip trees: A hierarchical representation for curves. *Communications of the ACM*, 24(5):310–321, 1981.

[Bare96] Gill Barequet, Bernard Chazelle, Leonidas J. Guibas, Joseph S. B. Mitchell, and Ayellet Tal. Boxtree: A hierarchical representation for surfaces in 3d. *Computer Graphics Forum*, 15(3):387–396, August 1996.

[Barn86] J. E. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324(6270):446–449, 1986.

[Barr84] Alan H. Barr. Global and local deformations of solid primitives. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):21–30, July 1984.

[Barr94] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. 1994.

[Baum91] Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James M. Winget. Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):51–60, July 1991.

[Beka] Philippe Bekaert, Frank Suykens de Laet, and Philip Dutre. Renderpark, a photorealistic rendering tool. http://www.cs.kuleuven.ac.be/cwis/research/graphics/RENDERPARK/.

[Blin78] James F. Blinn. Simulation of wrinkled surfaces. *Computer Graphics (Proceedings of SIGGRAPH 78)*, 12(3):286–292, August 1978.

[Chri95] Per Henrik Christensen. *Hierarchical Techniques for Glossy Global Illumination*. Ph.D. thesis, Technical Report, Seattle, Washington, 1995.

[Chri96] Per H. Christensen, Eric J. Stollnitz, David H. Salesin, and Tony D. DeRose. Global illumination of glossy environments using wavelets and importance. *ACM Transactions on Graphics*, 15(1):37–71, January 1996.

[Chri97] Per H. Christensen, Dani Lischinski, Eric J. Stollnitz, and David H. Salesin. Clustering for glossy global illumination. *ACM Transactions on Graphics*, 16(1):3–33, January 1997.

[Cohe85] Michael F. Cohen and Donald P. Greenberg. The hemi-cube: A radiosity solution for complex environments. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):31–40, July 1985.

[Cohe86] Michael Cohen, Donald P. Greenberg, Dave S. Immel, and Philip J. Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics & Applications*, 6(3):26–35, March 1986.

[Cohe88] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):75–84, Aug. 1988.

[Cohe93] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston, MA, 1993.

[Cuny00] Francois Cuny, Laurent Alonso, and Nicolas Holzschuch. A novel approach makes higher order wavelets really efficient for radiosity. volume 19, 2000.

[Daub97] Katja Daubert, Hartmut Schirmacher, François X. Sillion, and George Drettakis. Hierarchical lighting simulation for outdoor scenes. *Eurographics Rendering Workshop 1997*, pages 229–238, June 1997.

[Deer95] Michael Deering. Geometry compression. *Computer Graphics*, pages 13–20, 1995.

[Demm97] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Norfolk, VA, 1997.

[Duff92] Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):131–138, July 1992.

[Ferw97] James A. Ferwerda, Sumanta N. Pattanaik, Peter Shirley, and Donald P. Greenberg. A model of visual masking for computer graphics. *Proceedings of SIGGRAPH 97*, pages 143–152, August 1997.

[Fuji86] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, pages 16–26, April 1986.

[Garl97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of SIGGRAPH 97*, pages 209–216, August 1997.

[Garl98] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization 98 Conference Proceedings*, pages 263–269,542, October 1998. http://www.cs.cmu.edu/~garland/quadrics/.

[Garl99] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, 1999. http://www.cs.cmu.edu/~garland/thesis.

[Gers94] Reid Gershbein, Peter Schröder, and Pat Hanrahan. Textures and radiosity: Controlling emission and reflection with texture maps. In Andrew Glassner, editor, *Computer Graphics (Proceedings of SIGGRAPH '94)*, pages 51–58, July 1994. http://www-graphics.stanford.edu/papers/texture.

[Gibs95] Simon Gibson. Efficient Radiosity for Complex Environments. M.Sc. thesis, Manchester, UK, 1995.

[Gibs96] S. Gibson and R. J. Hubbold. Efficient hierarchical refinement and clustering for radiosity in complex environments. *Computer Graphics Forum*, 15(5):297–310, 1996.

[Glas94] A.S. Glassner. Principles of digital image synthesis, morgan kaufmann, san francisco. *CA*, 94:1994, 1994.

[Gold87] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics & Applications*, 7(5):14–20, May 1987.

[Golu89] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 1989.

[Gora84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):213–222, July 1984.

[Gort93a] Steven J. Gortler, Michael F. Cohen, and Phillip Slusallek. Radiosity and relaxation methods: Progressive refinement is southwell relaxation. February 1993.

[Gort93b] Steven J. Gortler, Peter Schröder, Michael F. Cohen, and Pat Hanrahan. Wavelet radiosity. *Proceedings of SIGGRAPH 93*, pages 221–230, August 1993.

[Gott96] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. Obb-tree: A hierarchical structure for rapid interference detection. *Proceedings of SIGGRAPH 96*, pages 171–180, August 1996.

[Gree87] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, August 1987.

[Greg98] Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. The irradiance volume. *IEEE Computer Graphics & Applications*, 18(2):32–43, March-April 1998.

[Gudu90] Ugur Gudukbay and Bulent Özguç. Free-form solid modeling using deformations. *Computers & Graphics*, 14(3/4):491–500, 1990.

[Hain91] Eric Haines. Efficiency improvements for hierarchy traversal in ray tracing. *Graphics Gems II*, pages 267–272, 1991.

[Hanr91] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):197–206, July 1991.

[Hase99] Jean-Marc Hasenfratz, Cyrille Damez, FranÁois Sillion, and George Drettakis. A practical analysis of clustering strategies for hierarchical radiosity. *Computer Graphics Forum*, 18(3):221–232, September 1999.

[Heck] Paul S. Heckbert and Michael Garland. A survey of polygonal surface simplification algorithms. ftp://ftp.cs.cmu.edu/afs/cs/project/anim/ph/paper/multi97/release/heckbert/s% imp.pdf.

[Heck90] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):145–154, August 1990.

[Heid99] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. *Proceedings of SIGGRAPH 99*, pages 171–178, August 1999.

[Hopp97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Proc.*, pages 189–198, August 1997. http://research.microsoft.com/~hoppe/.

[Jens96] Henrik Wann Jensen. Global illumination using photon maps. *Eurographics Rendering Workshop 1996*, pages 21–30, June 1996.

[Jeva89] David Jevans and Brian Wyvill. Adaptive voxel subdivision for ray tracing. *Graphics Interface '89*, pages 164–172, June 1989.

[Joll86] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[Kaji86] James T. Kajiya. The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH 86)*, 20(4):143–150, August 1986.

[Kalv96] Alan D. Kalvin and Russell H. Taylor. Superfaces:polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Appl.*, 16(3), May 1996. http://www.computer.org/pubs/cg&a/articles/g30064.pdf.

[Kaut99] Jan Kautz and Michael D. McCool. Interactive rendering with arbitrary brdfs using separable approximations. *Eurographics Rendering Workshop 1999*, June 1999.

[Kell97] Alexander Keller. Instant radiosity. *Proceedings of SIGGRAPH 97*, pages 49–56, August 1997.

[Lang94] Brigitta Lange. The simulation of radiant light transfer with stochastic ray-tracing. *Second Eurographics Workshop on Rendering (Photorealistic Rendering in Computer Graphics)*, pages 30–44, 1994.

[Lind00] Peter Lindstrom. Out-of-core simplification of large polygonal models. *Proceedings of SIGGRAPH 2000*, pages 259–262, July 2000.

[Lisc93] Daniel Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. *Proceedings of SIGGRAPH 93*, pages 199–208, August 1993.

[Lisc94] Dani Lischinski, Brian Smits, and Donald P. Greenberg. Bounds and error estimates for radiosity. *Proceedings of SIGGRAPH 94*, pages 67–74, July 1994.

[Lueb97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Proc.*, pages 199–208, August 1997.

[Moor66] Ramon E. Moore. Interval analysis. 1966.

[Neum95] László Neumann, W. Purgathofer, R. Tobler, A. Neumann, P. Elias, M. Feda, and X. Pueyo. The stochastic ray method for radiosity. *Eurographics Rendering Workshop 1995*, pages 206–218, June 1995.

[Nish85] Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of 3-D objects taking account of shadows and interreflection. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):23–30, July 1985.

[O'Ro85] Joseph O'Rourke. Finding minimal enclosing boxes. *International J. Comput. Inform. Sci*, 14:183–199, June 1985.

[Patt92] S. N. Pattanaik and S. P. Mudur. Computation of global illumination by monte carlo simulation of the particle model of light. *Third Eurographics Workshop on Rendering*, pages 71–83, May 1992.

[Phar97] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. *Proceedings of SIGGRAPH 97*, pages 101–108, August 1997.

[Pres92] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge (UK) and New York, 2nd edition, 1992.

[Ross93] J. Rossignac and P. Borrel. Multi-resolution 3d approximation for rendering complex scenes. *Second Conference on Geometric Modelling in Computer Graphics*, pages 453–465, June 1993.

[Rush88] Holly E. Rushmeier. Realistic image synthesis for scenes with radiatively participating media. 1988.

[Rush93] Holly Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. *Graphics Interface '93*, pages 227–236, May 1993.

[Schr96] Peter Schroder. Wavelet Radiosity: Wavelet Methods for Integral Equations. In *ACM SIGGRAPH '96 Course Notes - Wavelets in Computer Graphics*, pages 143–165. 1996.

[Shaw53] F. S. Shaw. *An Introduction to Relaxation Methods*. Dover, 1953.

[Shir91] Peter Shirley. Radiosity via ray tracing. *Graphics Gems II*, pages 306–310, 1991.

[Shir92] Peter Shirley. Time complexity of monte carlo radiosity. *Computers & Graphics*, 16(1):117–120, 1992.

[Shir95] Peter Shirley, Bretton Wade, Philip Hubbard, David Zareski, Bruce Walter, and Donald P. Greenberg. Global illumination via density estimation. *Eurographics Rendering Workshop 1995*, pages 219–231, June 1995.

[Shoe85] Ken Shoemake. Animating rotation with quaternion curves. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3):245–254, July 1985.

[Sill94a] Francois Sillion. Clustering and Volume Scattering for Hierarchical Radiosity Calculations. In *Fifth Eurographics Workshop on Rendering*, pages 105–117, Darmstadt, Germany, June 1994.

[Sill94b] François X. Sillion and Claude Puech. Radiosity and global illumination. 1994.

[Sill95a] François X. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):240–254, September 1995.

[Sill95b] François X. Sillion, G. Drettakis, and Cyril Soler. A clustering algorithm for radiance calculation in general environments. *Eurographics Rendering Workshop 1995*, pages 196–205, June 1995.

[Slus94] P. Slusallek, M. Schroder, M. Stamminger, and H.-P. Seidel. Smart limks and efficient reconstruction for wavelet radiosity. Technical Report Technical Report 14/1994, Computer Science Department, University of Erlangen-Nuremberg, 1994.

[Smit92] Brian E. Smits, James R. Arvo, and David H. Salesin. An importance-driven radiosity algorithm. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):273–282, July 1992.

[Smit94] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. *Proceedings of SIGGRAPH 94*, pages 435–442, July 1994.

[Snyd92] John M. Snyder. Interval analysis for computer graphics. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):121–130, July 1992.

[Stam97a] M. Stamminger, W. Nitsch, and Ph. Slusallek. Isotropic clustering for hierarchical radiosity – implementation and experiences. In *Proc. Fifth International Conference in Central Europe on Computer Graphics and Visualization '97*, 1997.

[Stam97b] M. Stamminger, P. Slusallek, and H.-P. Seidel. A comparison of different refiners for hierarchical radiosity. In B. Girod, H. Niemann, and H.-P. Seidel, editors, *3D Image Analysis and Synthesis '97*, 1997.

[Stam97c] Marc Stamminger, Philipp Slusallek, and Hans-Peter Seidel. Bounded clustering - finding good bounds on clustered light transport. Technical Report Technical Report 2/1997, Computer Science Department, University of Erlangen-Nuremberg, 1997.

[Stam98] M. Stamminger, H. Schirmacher, P. Slusallek, and Hans-Peter Seidel. Getting rid of links in hierarchical radiosity. *Computer Graphics Forum*, 17(3):165–174, 1998.

[Stol96] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, San Francisco, CA, 1996. http://www.amath.washington.edu/~stoll/pub.html.

[Stra86] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, Box 157, Wellesley MA 02181, 1986.

[Tobl97] Robert F. Tobler, Alexander Wilkie, Martin Feda, and Werner Purgathofer. A hierarchical subdivision algorithm for stochastic radiosity methods. *Eurographics Rendering Workshop 1997*, pages 193–204, June 1997.

[Tous83] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON '83*, pages A10.02/1–4, 1983.

[Tumb99] Jack E. Tumblin. *Three methods of Detail-Preserving Contrast Reduction for Displayed Images*. PhD thesis, Georgia Institute of Technology, 1999. http://www.cc.gatech.edu/gvu/people/jack.tumblin/.

[Veac94] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. *Fifth Eurographics Workshop on Rendering*, pages 147–162, June 1994.

[Veac95] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. *Proceedings of SIGGRAPH 95*, pages 419–428, August 1995.

[Walt97a] Bruce Walter, Gün Alppay, Eric P. F. Lafortune, Sebastian Fernandez, and Donald P. Greenberg. Fitting virtual lights for non-diffuse walkthroughs. *Proceedings of SIGGRAPH 97*, pages 45–48, August 1997.

[Walt97b] Bruce Walter, Philip M. Hubbard, Peter Shirley, and Donald F. Greenberg. Global illumination using local linear density estimation. *ACM Transactions on Graphics*, 16(3):217–259, July 1997.

[Warda] Greg Ward. The radiance synthetic imaging system. http://radsite.lbl.gov/radiance/.

[Wardb] Greg Ward, Rob Shakespeare, Ian Ashdown, and Holly Rushmeier. The materials and geometry format 2.0. http://radsite.lbl.gov/mgf/HOME.html.

[Ward88] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):85–92, Aug. 1988.

[Ward94] Gregory J. Ward. The radiance lighting simulation and rendering system. *Proceedings of SIGGRAPH 94*, pages 459–472, July 1994.

[Will93] Andrew James Harman Willmott. Fuzzy rendering for high quality image generation. M.Sc. thesis, Department of Computer Science, University of Auckland, 1993.

[Will97a] Andrew Willmott and Paul Heckbert. An empirical comparison of progressive and wavelet radiosity. In Julie Dorsey and Phillip Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 175–186, New York, NY, 1997. Springer Wien.

[Will97b] Andrew J. Willmott and Paul S. Heckbert. An empirical comparison of radiosity algorithms. Technical Report CMU-CS-97-115, School of Computer Science, Carnegie Mellon University, April 1997. http://www.cs.cmu.edu/~radiosity/emprad-tr.html.

[Will99] Andrew J. Willmott, Paul S. Heckbert, and Michael Garland. Face cluster radiosity. In D. Lischinski and G. W. Larson, editors, *Rendering Techniques '99 (Proceedings of the Tenth Eurographics Workshop on Rendering)*, pages 293–304. Springer-Verlag/Wien, 1999. http://www.cs.cmu.edu/~ajw/paper/fcr-eg99/.

[Wysz82] G. Wyszecki and W. S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley, 1982.

[Xia96] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of Visualization '96*, pages 327–334, October 1996.

[Zatz93] Harold R. Zatz. Galerkin radiosity: A higher-order solution method for global illumination. *Computer Graphics (SIGGRAPH '93 Proceedings)*, August 1993.

Appendix A

Definitions

Symbol	Meaning	Units
b	Radiosity	$\mathrm{Watt/m^2}$
e	Emittance	$\mathrm{Watt/m^2}$
E	Irradiance	$\mathrm{Watt/m^2}$
Q	Radiant Energy	Joule
P, Φ	Power, flux	Watt
L	Radiance: power incident on a unit surface area per solid angle.	$\mathrm{Watt/m^2 sr}$
I	Radiant Intensity: watts per solid angle. Most often used to deal with point sources.	$\mathrm{Watt/sr}$
$L(x \to y)$	Radiance leaving surface point x in the direction of point y.	
$L(x \leftarrow y)$	Radiance incident at surface point x from point y.	
$\mathbf{S}$	Area-weighted normal: $\int_S \mathbf{n} dA$	m^2

Table 14: Physical Quantities

Symbol	**Meaning**
$(x)_+$	Maximum of x and 0
$\lceil x \rceil$	Upper bound on x
$\lfloor x \rfloor$	Lower bound on x
$[x]$	Average value of x: $(\lceil x \rceil + \lfloor x \rfloor)/2$
$\langle x \rangle$	The interval of x: $[\lfloor x \rfloor, \lceil x \rceil]$.

Table 15: Mathematical definitions

Appendix B

Renderer Settings

B.1. Renderer Settings

One of the greatest difficulties in comparing the results from the three pieces of rendering software used was ensuring consistent units between the three. My own radiosity code is focused purely on the physical simulation of radiosity propagation. Thus both input light sources and output are specified in terms of radiosity ($\mathrm{Watt}/\mathrm{m}^2$).

To convert from radiometric quantities to photometric quantities, we must generally convert from Watts to candela. The candela is defined as being 1/683 of a Watt for the flux emitted in a certain direction by a line source at the wavelength of 555nm. Generally, to convert a tristimulus colour from watts to candela, we must multiply by the candela value corresponding to the reference white point we've chosen for our colour system. For the colour system adopted by the MGF standard, this is CIE illuminant E (which has a chromaticity of $(1/3, 1/3)$), and a luminance value of 179.0 $\mathrm{Candela}/\mathrm{Watt}$ [Wysz82].

To convert from irradiance to radiance, we divide by π; this is the effective solid angle a lambertian source radiates over.

B.1.1. Radiance

Somewhat unsurprisingly, radiance does its calculations in terms of radiance (Watt/m^2sr). Thus, when we convert our scene files to radiance format, we simply divide the light source emittances by π.

To achieve good results for the simulation of diffuse interreflection by radiance, I used the following options:

```
rpict ...
   -ps 1 -pt .04 -dp 2048 -ar 32 -ms 0.025
   -ds .1 -dj .65 -dt .05 -dc .75 -dr 3 -sj 1 -st .01
   -ab 2 -aa .15 -ad 256 -as 0 -av 0 0 0
   -lr 12 -lw .0005
```

The most pertinent settings are **-ab** *a*, which sets the maximum number of ambient bounces *a* performed, and **-av** *r g b*, which sets the “ambient” light factor for surfaces after those *a* bounces. The rest of the settings follow the defaults produced by the “rad” program front-end to radiance with a quality setting of high.

Radiance outputs picture files with pixel values also in radiance. To produce a image comparable to the other two renderers, I used the following command line:

```
pfilt -1 -e 3.14159 $1:r.pic | ra_ppm -g 1 > $1:r.ppm
```

This converts the radiance values to irradiance, and avoids Radiance’s perceptual algorithms (-1), and gamma correction (-g 1).

A modification to the Radiance code base is necessary to accommodate the large size of the “test whale” and “museum” scenes; the constant MAXOBJBLK in src/common/object.h must be increased to at least 10,000. I also modified the code to track memory usage.

B.1.2. RenderPark

As input, renderpark takes scene files in the MGF format [Wardb]. This measures the intensity of diffuse light sources in terms of luminosity (lm/m^2), the photometric correspondent to radiosity. To produce such files from our internal scene format, we must convert our RGB colours to the CIE (x,y,Y) colour space, and multiply our emittances by $179.0/\pi$.

Internally, RenderPark carries out its calculations in units of luminance. When it creates its output meshes, it remaps the intensity of colours using the CIE definition of lightness, L^*, which is proportional to the cube root of its luminance. I modified the RenderPark code (color.c:RadianceToRGB) to skip this stage, and instead map the luminance to emittance by multiplying by $\pi/179.0$.

The command line arguments used for RenderPark were as follows:

```
-radiance-method Galerkin
-gr-no-lazy-linking
-gr-clustering
-gr-hierarchical
-gr-no-importance
-gr-no-ambient
-gr-link-error-threshold 5e-5
-iterations 3
```

B.1.3. Radiator

For both volume clustering and face cluster radiosity, the refinement epsilon was set to $\varepsilon = 3e-5$, via the **`-ferr`** flag. Face cluster radiosity is selected via the **`-fcr`** flag, and hierarchical radiosity with volume clustering with **`-hier`**. Running radiator with no arguments gives a full list of options.

The complexity experiments did not use the approximate visibility method discussed in **Section 6.3.3**, to avoid confusing the performance gains due to the radiosity method with any gains made by the use of approximate visibility. Radiator can be forced to use the approximate visibility method by adding the flag **`-flrtComp`** s, where $s = \sqrt{f_v}$.