Radiosity for Real-Time Simulations of Highly Tessellated Models

Master's thesis in Computer Science

HENRIK EDSTRÖM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Radiosity for Real-Time Simulations of Highly Tessellated Models

HENRIK EDSTRÖM

Supervisors: TOMAS AKENINE-MÖLLER, JOHNNY WIDERLUND
Examiner: ULF ASSARSSON

Chalmers University of Technology and University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden 2016

Abstract

Radiosity for Real-Time Simulations of Highly Tessellated Models

HENRIK EDSTRÖM

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

The accurate simulation of light is one of the most important aspects of computer graphics. Radiosity is a physically-based method that can generate view-independent solutions of the light scattering in a scene. This thesis describes a radiosity system capable of generating high-quality global illumination solutions suitable for real-time simulations. State-of-the-art algorithms in hierarchical radiosity, such as face clustering and vector-based radiosity, are combined to efficiently handle scenes containing highly tessellated models. A new patch refinement method suitable for meshes with badly shaped triangles is also presented.

A radiosity representation method is proposed, that combines vertex lighting and light mapping to allow real-time simulations of the radiosity solutions. The light map generation is based on the texture atlas approach, using a new segmentation scheme. To allow realistic rendering of materials that are not completely diffuse, the radiosity lighting is combined with specular lighting using OpenGL.

The presented system supports quick visual feedback on how the radiosity calculation progresses. The preprocessing needed before the calculations can begin is very quick and the initial results can be displayed very fast. Even for large models, it is possible get a good idea of how the final result will look like in under a minute.

Keywords: radiosity, hierarchical radiosity global illumination, light mapping, face cluster radiosity, volume cluster radiosity, texture atlas.

Preface

This Master's thesis is part of the Master of Science program in computer science at Chalmers University of Technology and University of Gothenburg. The project was conducted in collaboration with Opticore AB, a software company located in Gothenburg, Sweden.

It should be noted that most of the work presented in this thesis was conducted in the fall of 2001 and early 2002, and for some parts of the discussion, especially those related to computer hardware, the time perspective should be taken into account. However, the algorithms presented in this thesis are not tied to any particular hardware architecture.

The reader is expected to have a basic knowledge of computer graphics and university-level mathematics.

Acknowledgements

First, I would like to thank my thesis supervisor at Chalmers University of Technology, Tomas Akenine-Möller, for his encouragement and helpful discussions during this project. He also pointed me to important papers and publications, especially the face cluster radiosity work of Andrew J. Willmott.

Huge thanks to Johnny Widerlund, my supervisor at Opticore AB, for his invaluable encouragement and support, his proofreading, and for his many insightful comments along the way. Johnny also provided an implementation of the LSCM algorithm used in this thesis.

Thanks to Erik Gustafsson for suggesting the topic for this thesis, and for providing the opportunity to conduct this work in collaboration with Opticore. I would also like to thank all the friendly people at Opticore. Thanks to Fredrik Sandbecker, Markus Jandér, and Johan Stenlund for helpful discussions and comments.

Finally, thanks to Nina for her love and support.

Göteborg, February 2004

Henrik Edström

Contents

1 Introduction

1.1 Background

Over the last three decades the field of computer graphics has evolved at a tremendous rate. In the 1970s, computers capable of displaying digital images were rare and expensive; today, realistic computer-generated images and video sequences can be found everywhere. In recent years, real-time computer graphics has made a huge leap forward and it is now possible to render realistic images at interactive frame rates. This has opened the door for many new applications in fields such as product visualization, virtual reality, and 3D computer games.

The accurate simulation of light is one of the most important aspects in the synthesis of photorealistic images. The physically-based simulation of all light scattering in a synthetic model is called global illumination. This process is quite complex and computationally expensive and most methods are not suitable for real-time applications. The computation times often range from minutes to several hours for large scenes. Ray tracing and its derivatives are inherently view-dependent, meaning that the solution has to be recalculated as soon as the view changes. This makes real-time simulations based on such algorithms difficult. Other methods, such as radiosity, calculate a view-independent solution that can be reused for multiple views. In this thesis we investigate the application of radiosity for real-time simulations of complex scenes.

1.2 Problem Formulation

This work has been conducted in collaboration with Opticore AB, a company that develops real-time visualization software. The task, proposed by Opticore, was to find a radiosity-based algorithm that generates global illumination solutions for scenes with highly tessellated models that can be used in real-time simulations. The lighting calculation is allowed to take some time, but when it is completed the simulation should have about the same real-time performance as before.

1.3 Project Goals

The goal of this thesis is to investigate what methods that can be used to achieve the results mentioned in the previous section. The problem can be divided into two sub-problems. The first one is to generate a global illumination solution for a given scene. The second one is to find a way to display this solution in real time without too much performance impact.

Additional goals, based on Opticore’s requirements, are that the proposed solution is as automatic as possible and that it runs on a wide range of hardware. It is also important that it works well for highly tessellated CAD data, since that is the primary focus area of Opticore’s products.

When the investigation is completed, the algorithms should be implemented and integrated in the Opticore Opus Studio application [Opti01]. The implementation should be based on the OpenGL Optimizer/Cosmo 3D API [Open98, Ecke98].

1.4 Contributions

Our main contribution is the development of a radiosity system capable of generating high-quality global illumination solutions suitable for real-time simulations. We have combined some of the state-of-the-art algorithms in hierarchical radiosity, e.g., face clustering and vector-based radiosity, to be able to efficiently handle highly tessellated models.

Previous work on face cluster radiosity has focused primarily on scenes with a small number of very complex surfaces. Our system combines face cluster radiosity with an efficient volume cluster hierarchy to manage larger sets of disconnected surfaces.

A new patch refinement method is presented, that works well for badly shaped triangles often found in highly tessellated CAD data.

We propose a radiosity representation method that combines vertex lighting and light mapping to allow real-time simulations of the radiosity solutions. This includes a new segmentation scheme for texture atlas creation and a rendering approach to combine radiosity with specular lighting on older graphics hardware.

One advantage of our system is that the user quickly gets visual feedback on how the calculation progresses. The preprocessing needed before the calculations can begin is very quick and the initial results can be displayed very fast. Even for large models, it is possible get a good idea of how the final result will look like in under a minute. This allows the user to quickly determine if some parameters need to be changed, and if so, restart the calculation.

1.5 Thesis Overview

This thesis consists of ten chapters. After this introduction chapter, a brief overview of radiosity and global illumination is given in Chapter 2. Chapter 3 presents the most important previous work on the subject.

Chapters 4-5 describe our proposed solution, i.e., the algorithms that our radiosity system is based on. In Chapter 6, some implementation details are given.

Results and conclusions are presented in Chapters 7-8, and suggestions for future work in Chapter 9. A bibliography can be found in Chapter 10.

2 Radiosity and Global Illumination

2.1 Illumination Models

Simulation of lighting begins with the specification of material properties for the surfaces in the scene and the positions and characteristics of the light sources. The illumination can then be simulated by using a local or a global illumination model.

2.1.1 Local Illumination

A local illumination model considers only the light sources and the surfaces illuminated directly. This means that it does not capture shadows or indirect illumination from other surfaces. Local illumination models are not physically based, and cannot produce accurate simulations of reality. However, because of their simplicity they are commonly used in real-time computer graphics.

2.1.2 Global Illumination

In reality, every surface receives light both directly from the light sources and indirectly from neighboring surfaces. In order to simulate the effects of interreflection, all objects must be considered as potential sources of illumination for all other objects in a scene. This is called a global illumination model. Global illumination methods are generally physically-based and try to capture both indirect illumination and shadows, but all possible light interactions are not necessarily considered. Because of their high complexity, the global illumination methods are often not possible to use in real-time graphics. Figure 2.1 demonstrates the difference between global and local illumination.

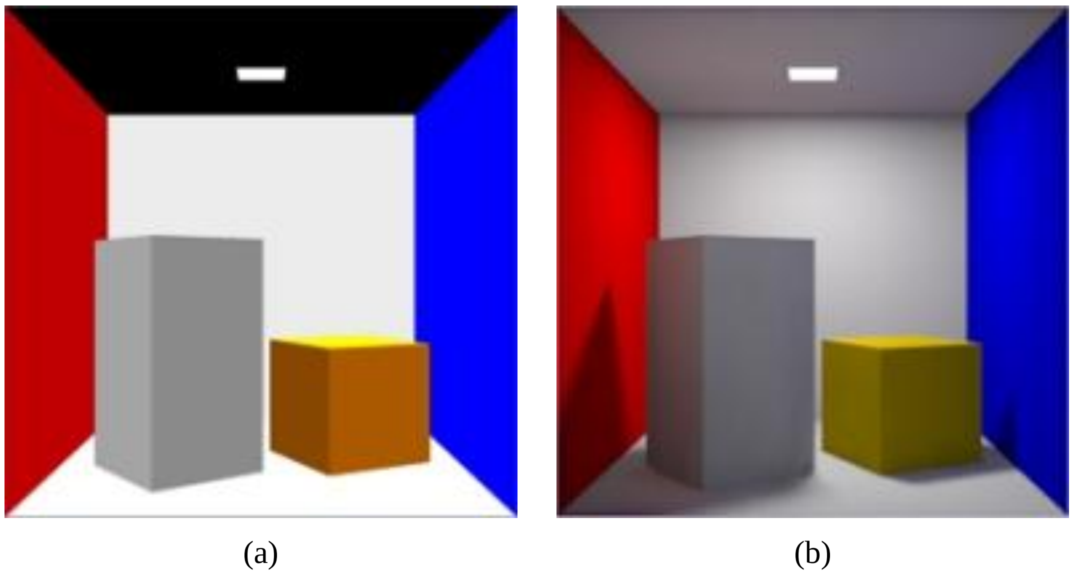


Figure 2.1. Illumination models. (a) Local illumination. (b) Global illumination.

2.2 Terminology and Radiometric Quantities

2.2.1 Radiometry and Photometry

Light is a form of *electromagnetic radiation*. Electromagnetic radiation can exist at any wavelength, and what we see as visible light is only a tiny fraction of the electromagnetic spectrum. Light is studied primarily in the fields of radiometry and photometry. Radiometry is the science of measuring light in any portion of the electromagnetic spectrum. The amount of light at each wavelength can be measured by a spectroradiometer. Photometry, on the other hand, is the psychophysical measurement of the visual sensation produced by the electromagnetic spectrum. The radiometric quantities are measured with respect to a specific wavelength and thus independent of the human visual system, whereas the photometrical quantities are integrated over all possible wavelengths weighted by the response of the human visual system. Our eye is for instance more sensitive to green light than to red or blue light.

Radiometry is more fundamental than photometry, in that photometric quantities may be computed from spectroradiometric measurements. For this reason, it is usually best to use radiometric quantities for computer graphics and image synthesis [Cohe93].

2.2.2 Solid Angle

In order to discuss the radiometric quantities, the concept of solid angle must be introduced. The solid angle describes the area of space occupied by a surface as seen from a point. It is measured by calculating the area of the surface projected onto a unit hemisphere centered at the point. Solid angle is measured in *steradians* (sr), and the solid angle subtended by the entire hemisphere is 2π sr.

Because solid angles are measured with respect to the unit hemisphere, it is often convenient to represent them using spherical coordinates. A position on a sphere can be represented by two angles; the number of degrees from the North Pole or zenith, θ, and the number of degrees about the equator or azimuth, ϕ.

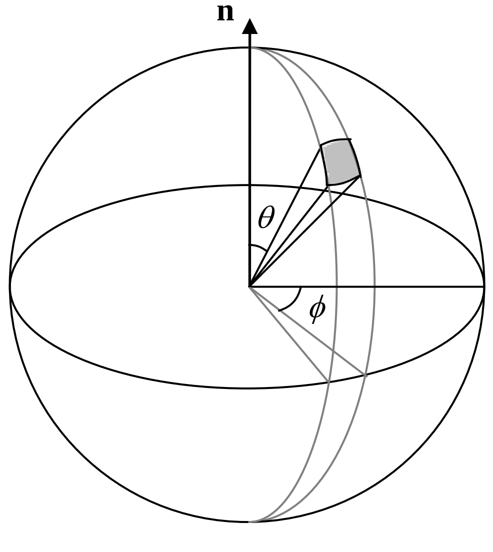


Figure 2.2. Solid angle.

When considering the solid angle subtended by a differential area dA, an approximate value can be obtained by taking the projection of dA onto a plane perpendicular to the direction from the origin of the hemisphere to the differential area. The projection onto the plane has a surface area of $dA\cos\theta$, and the solid angle is obtained by dividing this area by the square of the distance to the origin, to account for the projection onto the unit hemisphere:

$$d\omega \approx \frac{dA\cos\theta}{r^2} \tag{2.1}$$

2.2.3 Radiance

The most important quantity in the physical simulation of light is *radiance*. Radiance is the amount of power radiated from a surface in a particular direction, more precisely defined as the power per unit projected area perpendicular to the ray per unit solid angle in the direction of the ray. The radiance from a point $\boldsymbol{x}$ in direction $\vec{\omega}$ is:

$$L(\boldsymbol{x},\vec{\omega}) = \int p(\boldsymbol{x},\vec{\omega},\lambda)\frac{hc}{\lambda}d\lambda \tag{2.2}$$

where

- $p(\boldsymbol{x},\vec{\omega},\lambda)$ represents the density of light photons at point $\boldsymbol{x}$ travelling in the direction $\vec{\omega}$ with a wavelength λ.
- hc/λ gives the energy of a single photon (h is Planck's constant and c is the speed of light).

Radiance is measured in Watts per unit area per unit solid angle (W/m^{2}sr) and it is denoted by *L*. The corresponding photometric quantity is *luminance*.

2.2.4 Irradiance

Another important quantity is *irradiance* (*illuminance* in photometry), which is the radiant energy per unit area falling on a surface. It is denoted *E* and it is measured in Watts per unit area (W/m^2). The irradiance can be related to the incident, or incoming, radiance (L_i) by integrating over a hemisphere (Ω):

$$d\Phi = \left[\int_\Omega L_i \cos\theta d\omega\right] dA \tag{2.3}$$

to produce the total radiant energy incident on a surface, $d\Phi$. Since irradiance is energy *per unit area*, it is computed as:

$$E \equiv \frac{d\Phi}{dA} \tag{2.4}$$

or equivalently:

$$E = \int_{\Omega} L_i \cos\theta d\omega \tag{2.5}$$

The quantity $\cos\theta d\omega$ is often referred to as the *projected solid angle*. It can be thought of as the projection of a differential area onto a hemisphere projected onto the base of the hemisphere, as shown in Figure 2.3.

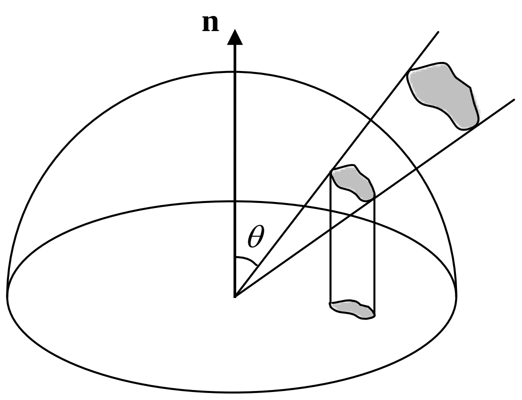


Figure 2.3. Projection of differential area.

2.2.5 Radiosity

Radiosity (denoted *B*) is very similar to irradiance. It is the total energy per unit area that *leaves* a surface. It is computed in a similar fashion by integrating the outgoing radiance over the unit hemisphere and dividing by the area:

$$B = \int_{\Omega} L_o \cos\theta d\omega \tag{2.6}$$

where L_o is the outgoing radiance.

When considering diffusely reflecting surfaces, the outgoing radiance becomes independent of direction and can therefore be brought outside the integral:

$$\begin{aligned} B &= L_o \int_\Omega \cos\theta \, d\omega \\ &= L_o \int_0^\pi \int_0^{2\pi} \cos\theta \sin\theta \, d\theta \, d\phi \\ &= L_o \pi \end{aligned} \tag{2.7}$$

The official term for radiosity is *radiant exitance*, but the term radiosity is commonly used in computer graphics literature. Radiosity is measured in Watts per unit area (W/m^2) and the photometric equivalent is *luminosity*.

2.2.6 Reflectance Functions

The reflecting properties of a material are described by the concept of *reflectance*. The most general expression of the reflectance is the *bidirectional reflectance distribution function*, or BRDF. The BRDF is defined as the ratio of the reflected radiance in the outgoing direction, to the differential irradiance from the incident direction [Cohe93]:

$$\rho(\vec{\omega}_i \to \vec{\omega}_r) \equiv \frac{L_r(\vec{\omega}_r)}{L_i(\vec{\omega}_i)\cos\theta_i \, d\omega_i} \tag{2.8}$$

where

- $L_r(\vec{\omega}_r)$ is the reflected radiance L_r in the outgoing (reflected) direction $\vec{\omega}_r$.
- $L_i(\vec{\omega}_i)\cos\theta_i d\omega_i$ is the irradiance L_i coming from the differential solid angle $d\omega_i$ around the incident direction $\vec{\omega}_i$ with polar angle θ_i.

The directions are often expressed as four polar angles by writing the BRDF as $\rho(\theta_r, \phi_r, \theta_i, \phi_i)$ [1].

The BRDF notation is widely used in computer graphics, but cannot model all physical light interactions. In this presentation all surfaces are assumed to be opaque, the reflected light is assumed to have the same frequency as the incoming light and light is assumed to reflect instantaneously off a surface. We also ignore participating media like smoke, dust etc.

A BRDF is often divided into three components, a *diffuse*, a *glossy*, and a *specular* component. The type of reflection depends mostly on the material and the roughness of the surface. Rough, irregular surfaces scatters light in many different directions and such reflections are called diffuse. An ideal diffuse reflector scatters light uniformly in every direction and is hence characterized by a uniform BRDF that does not depend on the outgoing direction. The reflected radiance is therefore the same in all directions, and the appearance of the surface is independent of the viewing angle. Ideal diffuse surfaces are also called *Lambertian reflectors*.

[1] The standard notation is to specify the reflected direction first in the arguments of the BRDF.

Perfectly smooth surfaces on the other hand, often act as ideal specular surfaces that reflect light only in the mirror direction. The outgoing direction is contained in the plane of incidence, and the outgoing polar angle is equal to the incident polar angle. Ideal specular reflections are also known as mirror reflections.

Real materials are not perfectly diffuse or perfectly specular. To capture the reflectance of real surfaces the BRDF contains an intermediate component for reflections that fall somewhere between a diffuse and a specular reflection. This component is called glossy, semi-specular, rough specular, or directional diffuse. In contrast to the other two components, glossy reflections have a complicated directional dependence.

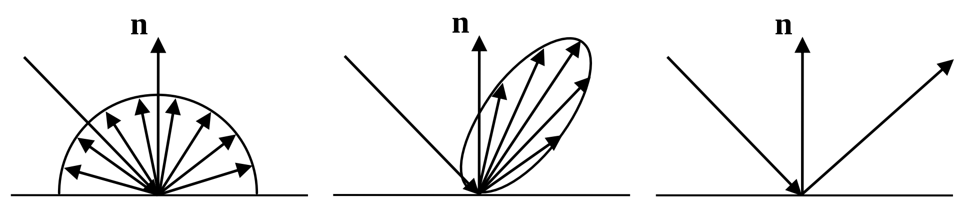


Figure 2.4. Diffuse, glossy and specular reflection.

2.3 Solving the Global Illumination Problem

2.3.1 Historical Perspective

The first global illumination algorithm was introduced in 1980 by Whitted [Whit80]. Whitted applied a ray tracing algorithm recursively to achieve a simple global illumination model that accounted for mirror reflection, refraction, and shadows. The ray tracing approaches were later extended to include glossy reflections and soft shadows using stochastic ray tracing and cone tracing. The rendered images continued to improve, but the models used were not based on physical principles and quantities. There was also no practical way to capture diffuse interreflection with these methods.

Radiosity methods, originally from the field of radiative heat transfer, where applied to the problem of global illumination in 1984 by researchers at Fukuyama and Hiroshima Universities in Japan [Nish85] and at the Program of Computer Graphics at Cornell University in the United States [Gora84]. These methods begin with an energy balance equation to describe the interreflection of light in the scene. This equation is then approximated and solved numerically.

2.3.2 The Rendering Equation

If we ignore participating media, the global illumination problem can be summarized by the following integral equation:

$$L(\boldsymbol{x},\vec{\omega}) = L_e(\boldsymbol{x},\vec{\omega}) + \int_\Omega \rho(\boldsymbol{x},\vec{\omega} \to \vec{\omega}_i) L_i(\boldsymbol{x},\vec{\omega}_i) \cos\theta_{\boldsymbol{x}} d\omega_i \qquad (2.9)$$

where

- $L(\boldsymbol{x},\vec{\omega})$ is the total radiance leaving surface point $\boldsymbol{x}$ in the direction $\vec{\omega}$.
- $L_e(\boldsymbol{x},\vec{\omega})$ is the radiance directly emitted from $\boldsymbol{x}$ in the direction $\vec{\omega}$.
- $\rho(\boldsymbol{x},\vec{\omega} \to \vec{\omega}_i)$ is the BRDF describing the fraction of radiance incident from direction $\vec{\omega}_i$ that is reradiated in direction $\vec{\omega}$.
- $L_i(\boldsymbol{x},\vec{\omega}_i)$ is the radiance incident on $\boldsymbol{x}$ from the direction $\vec{\omega}_i$.
- $\theta_{\boldsymbol{x}}$ is the angle between the surface normal at $\boldsymbol{x}$ and $\vec{\omega}_i$.
- Ω is the hemisphere lying above the tangent plane of the surface at $\boldsymbol{x}$.

This equation was first formulated by Kajiya [Kaji86], who named it the *rendering equation*. It basically states that the radiance emitted from a surface point **x** in the direction $\vec{\omega}$ is equal to the radiance the surface itself emits in that direction, plus the integral over the hemisphere of the incoming radiance that is reflected in that direction.

The rendering equation forms the basis for the global illumination algorithms, as it expresses the outgoing radiance for any given point on any surface. A solution to the equation is thus a solution to the global illumination, and all global illumination methods try to solve the rendering equation more or less accurately. The global illumination algorithms can roughly be divided into two groups: Monte Carlo methods and finite element methods [Heck93].

2.3.3 Monte Carlo Methods

The solution of the rendering equation requires numerical evaluation of high-dimensional integrals. Classical, deterministic, integration methods do not capture discontinuities well, and they suffer from *dimensional explosion*, which means that the computational complexity is exponential with regard to the dimension of the domain. The dimensional explosion can be avoided by using Monte Carlo integration methods. Instead of choosing sample points at regular intervals, the Monte Carlo methods pick them at random. The computational complexity of these methods is independent of the dimension of the integral. Many global illumination methods are based on Monte Carlo integration, such as stochastic ray tracing [Cook86], path tracing [Kaji86], and photon mapping [Jens01]. Since the focus of this thesis is on radiosity, these approaches will not be discussed in more depth.

2.3.4 Finite Element Methods

Heckbert and Winget [Heck91] have shown that radiosity is essentially a finite element method. The finite element methods approximate an unknown function by subdividing the domain of the function into smaller pieces called *elements*, across which the function can be approximated using simple functions like polynomials. The unknown function is thus projected into a finite function space, and the resulting system can then be solved numerically. Although the radiosity problem is often viewed as a finite element problem, many different approaches have been taken to solve the transfer of radiosity in an environment, from purely finite element methods to purely stateless Monte Carlo methods.

Finite element methods are typically more suitable to scenes containing diffuse surfaces, whereas the Monte Carlo based methods are better at simulating highly specular surfaces. The Monte Carlo approaches capture discontinuities like shadow boundaries better than the finite element methods. This is because the error in their approximation results in noise, which on the other hand can be highly distracting in areas where the result is expected to be smooth and continuous. Finite element methods often produce the best results for low to medium complexity scenes, whereas the Monte Carlo methods are faster for more complex environments. However, recent hierarchical finite element based radiosity algorithms have shown promising results for very complex scenes, as we shall see later.

2.4 The Radiosity Method

2.4.1 Simulating Diffuse Surfaces

The basic radiosity algorithm is limited to simulating the illumination for scenes with diffuse (Lambertian) surfaces. This limitation also has the advantage that the solution is view independent, which means that once the radiosity calculations are completed the scene can be rendered quickly from any viewpoint without recalculating the illumination. Interactive walkthroughs of the scene can therefore easily be performed.

Specular reflection and other effects can be added to the radiosity solution in a second ray tracing pass which overcomes the diffuse-only limitation to some degree. It is, however, not possible to mix specular and diffuse light bounces [Will00].

2.4.2 The Radiosity Equation

The rendering equation (Equation (2.9)) can be simplified if we assume that all surfaces reflect light diffusely. The outgoing radiance is then the same in all directions, i.e., $L(\mathbf{x},\vec{\omega}) \equiv L(\mathbf{x})$. The BRDF is also independent of the incoming and outgoing directions and can therefore be taken out from under the integral. This gives us the following equation:

$$L(\boldsymbol{x}) = L_e(\boldsymbol{x}) + \rho(\boldsymbol{x})\int_\Omega L_i(\boldsymbol{x},\vec{\omega}_i)\cos\theta_x d\omega_i \qquad (2.10)$$

which, given $d\omega = (\cos\theta dA)/r^2$ (Equation (2.1)), and $B(\boldsymbol{x}) = \pi L(\boldsymbol{x})$ (Equation (2.7)), can be rewritten as the *radiosity equation*:

$$B(\boldsymbol{x}) = E(\boldsymbol{x}) + \rho(\boldsymbol{x})\int_S B(\boldsymbol{x}')G(\boldsymbol{x},\boldsymbol{x}')V(\boldsymbol{x},\boldsymbol{x}')dA' \qquad (2.11)$$

where

- $\mathbf{x}$ and $\mathbf{x}'$ are points on surfaces S in the scene.

- $B(\mathbf{x})$ is the total radiosity leaving point **x**.
- $E(\mathbf{x})$ is the emission function describing the energy per unit area emitted at point **x**.
- $\rho(\mathbf{x})$ is the reflectance function.
- $G(\mathbf{x},\mathbf{x}')$ is the geometry kernel between point **x** and point $\mathbf{x}'$, defined as

$$G(\boldsymbol{x},\boldsymbol{x}') = \frac{\cos\theta_x \cos\theta_{x'}}{\pi \left\| \boldsymbol{x} - \boldsymbol{x}' \right\|^2} \tag{2.12}$$

- $V(\mathbf{x},\mathbf{x}')$ is the binary visibility between point **x** and point $\mathbf{x}'$.

This is the equation that must be solved to find the global illumination for an environment containing only Lambertian diffuse surfaces.

Discretizing Equation (2.11) as a finite element problem gives the *classical radiosity equation* [Cohe93]:

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} \tag{2.13}$$

where F_{ij}, called the *form factor,* is given by

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} G(\boldsymbol{x},\boldsymbol{x}') V(\boldsymbol{x},\boldsymbol{x}') dA_j dA_i \tag{2.14}$$

The form factor represents the fraction of energy that leaves element *i* and arrives directly at element *j*. Each element has a radiosity equation of the form of Equation (2.13) and the system of equations can be expanded into matrix form:

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \tag{2.15}$$

This matrix is known as the *radiosity matrix*. By solving this linear equation system we obtain a radiosity solution which describes the distribution of light between the elements in a scene.

2.4.3 Algorithm Overview

The process of initializing, calculating, and presenting a radiosity solution is sometimes called the *radiosity pipeline* [Yeap97, Niel00]. Figure 2.5 shows this process divided into five different stages.

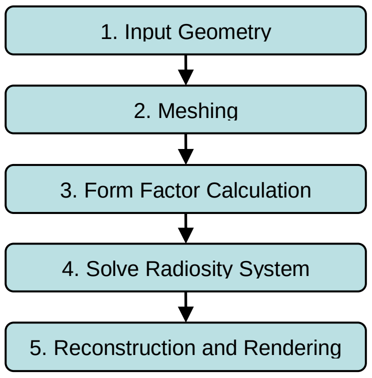

1. Input Geometry
2. Meshing
3. Form Factor Calculation
4. Solve Radiosity System
5. Reconstruction and Rendering

Figure 2.5. The radiosity pipeline.

Stage one defines the input scene geometry. The second stage subdivides the surfaces into elements to be used in the calculation. Stage three calculates the form factors and stage four solves the system of linear equations. The last stage displays the results.

It is important to note that if some parameters are changed it is not necessary to restart at stage one. Only if the geometry of the input scene is changed must the entire process be repeated. If the surface reflectance properties or the lighting parameters are changed, we can restart at stage four. Most importantly, if the viewing parameters are changed we only have to repeat parts of stage five.

2.4.4 Meshing

The meshing stage subdivides the input polygons into elements. It is a very important part of the radiosity process, since both the accuracy and the complexity of the radiosity calculations depend very much on the underlying mesh. The goal is to create a mesh which for a desired accuracy uses as few elements as possible, and distributes the error evenly among those elements.

A simple meshing strategy is uniform subdivision. It performs reasonably well where the radiosity function is smooth but fails where the function changes more rapidly. Increasing the mesh density will improve the accuracy at the cost of higher memory consumption and execution time. The error will however still be unevenly distributed. A better approach is to use a non-uniform subdivision strategy that subdivides to a fine level only where it improves the accuracy. This will distribute the error evenly among the elements. Figure 2.6 illustrates the different meshing strategies for a one-dimensional radiosity function.

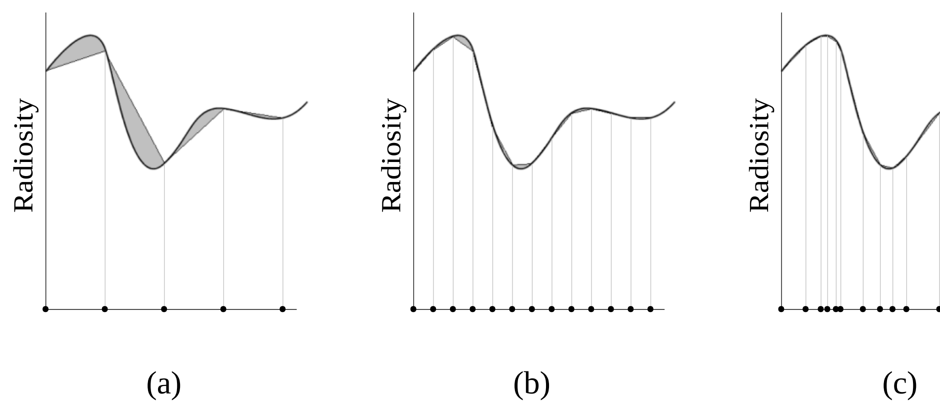


Figure 2.6. Meshing strategies. The shaded area represents the error in the approximation. (a) Uniform subdivision. (b) High density uniform subdivision. (c) Non-uniform subdivision.

2.4.5 Form Factor Calculation

Calculating the form factors is usually the most expensive part of the radiosity method. It can consume up to 90 percent of the execution time of a radiosity program [Piet93]. The form factors represent the geometric relationship between elements, and do not depend on the reflective or emissive characteristics of the surfaces. A form factor between two elements depends solely on the orientation and size of the elements and the distance and visibility between them.

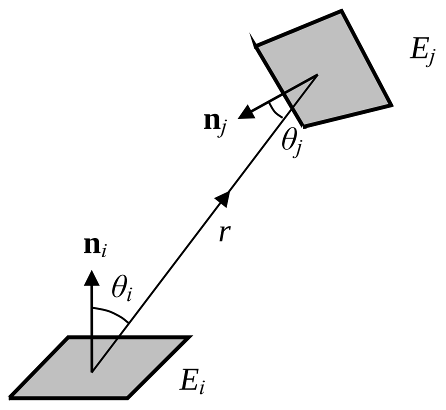


Figure 2.7. Element E_j receiving light energy from element E_i.

Rewriting Equation (2.14) by expanding the geometrical term gives us:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} V_{ij} \, dA_j \, dA_i \qquad (2.16)$$

This equation cannot be solved directly, except for the most trivial cases. For general situations, numerical methods must be used to approximate the form factor.

Equation (2.16) can be simplified by considering the form factor from a differential area element to a finite area element. The emitter is thus treated like a point source, and the equation becomes:

$$F_{ij} \approx F_{dE_i E_j} = \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} V_{ij} \, dA_j \qquad (2.17)$$

This approximation is only justifiable if the distance between the elements is large, since the inner integral then does not change much over the outer integral in Equation (2.16). Ashdown [Ashd94] mentions the *five-times rule* that says that a finite area emitter should be modeled as a point source only when the distance to the receiving surface is greater than five times the maximum projected area of the emitter. If the five-times rule does not hold for any two elements, we can always subdivide the emitter until the rule is satisfied for each subdivided area. However, this fails if the elements share a common edge since the distance between them will then be zero along the edge. In those cases we have to stop the subdivision at an appropriate level.

The visibility between the elements also affects the validity of the above approximation. This problem can be avoided by subdividing the elements until each element is either completely visible or not visible at all.

One way to solve Equation (2.17) is to use *hemisphere sampling*. This method places an imaginary unit hemisphere centered on the differential area, and projects the element radially onto the hemisphere and then orthogonally down onto the base of the hemisphere. The fraction of the base area covered by this projection is equal to the form factor. See Figure 2.8.

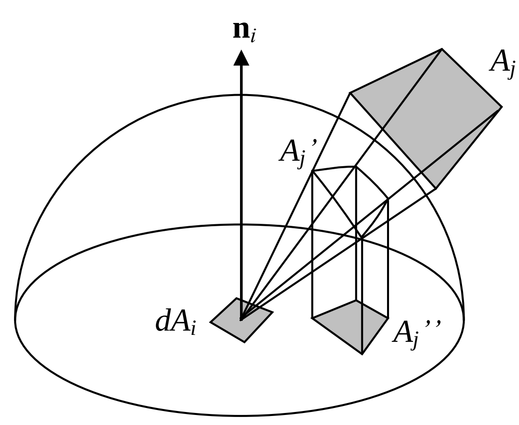


Figure 2.8. Hemisphere sampling.

The differential area-to-element form factor can now be written as:

$$F_{dE_iE_j} = \frac{A_j''}{\pi} \tag{2.18}$$

This geometric solution is known as *Nusselt's analogy*. The area of the projection of the element on the unit hemisphere is, by definition of the solid angle, equal to the solid angle subtended by the element and accounts for the factor cos θ_j / r^2 in Equation (2.17). The projection onto the base accounts for the cos θ_i term, and the π in the denominator is the area of a unit circle, i.e., the base of the hemisphere.

Cohen and Greenberg proposed the *hemicube* form factor algorithm in 1985 [Cohe85]. This method replaces the hemisphere with a hemicube. The faces of the hemicube are divided into a grid of cells, each defining a direction and a solid angle. A *delta form factor*, ΔF, is computed for each cell based on its size and orientation. The delta form factors are pre-computed and stored in a lookup table. The form factor for an element is approximated by projecting the element onto the faces of the hemicube and summing the delta form factors covered by the projection. To determine the visibility of the element the Z-buffer algorithm is used. This simple and efficient approach also has the advantage of wide hardware rendering support. A fast, hardware accelerated hemicube algorithm is described in [Wide99].

Several variations of the hemicube approach have been developed, such as the *cubic tetrahedral method* [Bera91] and the *single plane method* [Sill89].

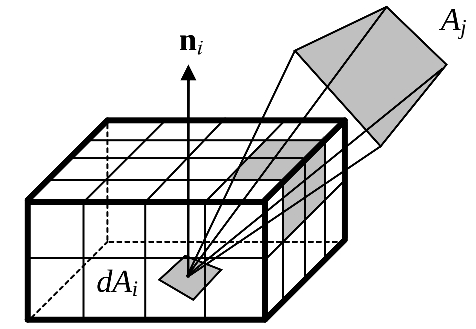


Figure 2.9. The hemicube method.

The hemicube method has several limitations. The resolution is limited which can cause uneven and inadequate sampling. Equally sized elements may be sampled differently, as illustrated in Figure 2.10. Elements may also be missed completely if their projection on the hemicube is small enough to fall between the centers of the hemicube cells. The tetrahedral and single plane methods also suffer from these limitations.

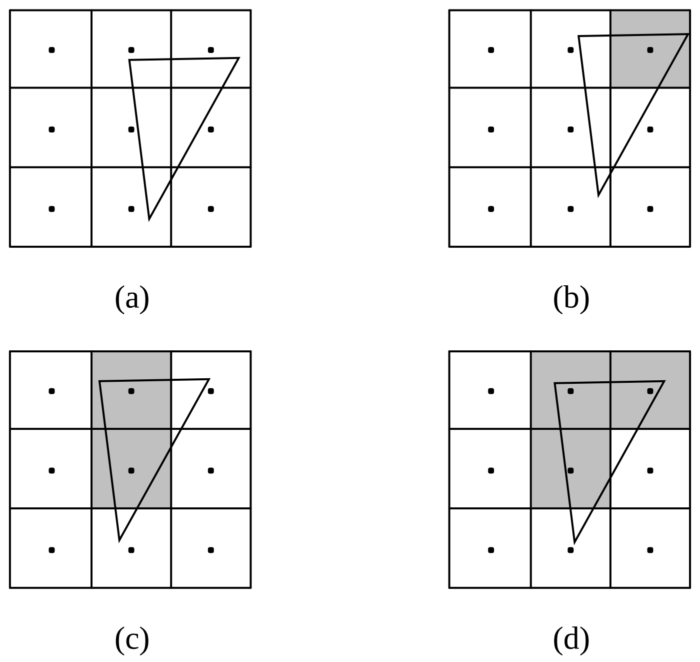


Figure 2.10. Hemicube sampling limitations. Four identical triangles projected onto one of the hemicube sides, each covering a different number of cell centers. This results in a wide variance in the associated form factors.

There are also several ray casting approaches to calculate the form factors. Ray casting offers a flexible way to determine visibility. Any distribution of points on the elements or directions on the hemisphere can be chosen independently and adaptive sampling can be used to distribute the computational effort evenly. Rays can also be distributed stochastically, which can make inadequate sampling less noticeable by converting aliasing to noise [Cohe93].

The ray casting methods also provide an excellent basis for Monte Carlo integration of the form factor equation over the hemisphere. Importance sampling can be used by selecting directions over the hemisphere with a sample density proportional to the cosine. In this way, more effort will be expended where the form factor is largest.

Ray casting is generally more expensive than scan conversion algorithms for form factor calculation, especially when calculating the form factors between one element and all other elements in a scene. Ray casting methods are however often preferred in situations where only a few form factors are needed for a single element. This is often the case with hierarchical radiosity algorithms. The possibility to use importance sampling and the increased flexibility can also lead to better overall performance. There are also many acceleration schemes for ray casting, see [Glas89]. Another advantage is that it can be applied to both planar and curved surfaces.

2.4.6 Solving the Radiosity Equation

When the form factors are known, the linear system in Equation (2.15) can be solved. There are a number of numerical algorithms available to solve linear systems. Direct methods, such as Gaussian elimination, can be applied to the radiosity problem, but they have a computational complexity related to the cube of the number of equations, $O(n^3)$ [Cohe93]. This is far too expensive for large, non-sparse systems like the radiosity system and it is therefore necessary to use iterative solution methods.

Iterative methods begin with a guess for the solution and proceed by repeatedly performing operations that move the guess closer to the actual solution. Gauss-Seidel iteration, also referred to as *gathering* [Cohe93], is one of the more common methods. It traverses all elements and for each element energy is gathered from all other elements. This procedure is repeated until the system converges.

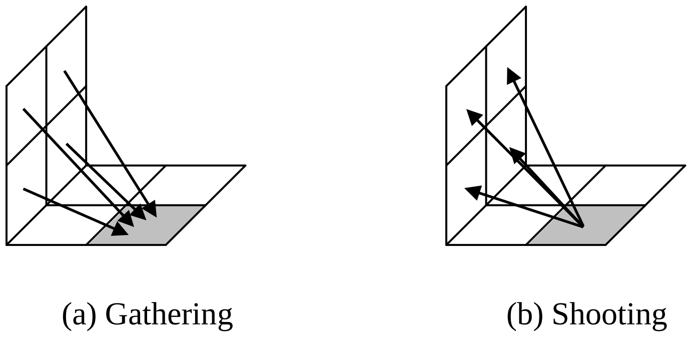


Figure 2.11. Gathering vs. shooting.

Another method is called progressive refinement. It performs Southwell and Jacobi iteration on the radiosity system [Cohe93]. In contrast to the gathering method, the element with the most energy is selected and its energy is “shot” to all other elements. Hence, this method is often referred to as *shooting*. The shooting and gathering behavior is shown in Figure 2.11. Both gathering and progressive refinement has a time complexity of $O(n^2)$, but progressive refinement usually converges much faster.

2.4.7 Reconstruction and Rendering

When the radiosity solution has been calculated, the illuminated scene can be rendered. During rendering, the color of each pixel is derived from the radiance of the surface location visible at the pixel. The radiance can be directly determined from the calculated approximation of the radiosity function.

A common way to render the radiosity solution is to calculate the colors at the vertices of the mesh by averaging the colors of the neighboring elements. The mesh can then be rendered using the standard graphics pipeline and linearly interpolated by hardware using Gouraud shading. However, the human visual system is highly sensitive to discontinuities and higher order reconstructions of the radiosity solution may be necessary before rendering.

The mapping[2] from radiance to pixel colors is not trivial. Typically, the display device allows 8-bit integer pixel values for each of the red, green, and blue color channels. In the real world, luminance values can range from 10^{-5} cd/m^2 to 10^5 cd/m^2 [Cohe93]. It is therefore necessary to map the values calculated by the radiosity simulation to the range available on the display device. Another problem is the non-linear relationship between pixel colors and the resulting display device radiance. The adjustments for this non-linear relationship are called *gamma correction*. In the end, the goal of the mapping process from radiosities to pixel colors is to produce a subjective impression of brightness in the image that is equivalent to the perceived brightness in the real environment.

[2] This mapping is often referred to as *tone mapping*, or *tone reproduction*.

3 Previous Work

3.1 Early Radiosity Methods

3.1.1 Matrix Radiosity

Early radiosity approaches calculated the form factor matrix explicitly [Gora84, Nish85, Cohe85], and are thus often referred to as *matrix radiosity*. The matrix radiosity method uses a fixed mesh of elements and computes the form factors between every possible pair of elements. The resulting linear system is then typically solved using an iterative method. This results in time and memory requirements that are quadratic in the number of elements in the mesh, which makes the method useful only for simple scenes.

3.1.2 Progressive Refinement

Progressive refinement, briefly mentioned in the previous chapter, was the first major improvement over matrix radiosity [Cohe88]. The algorithm solves the radiosity matrix iteratively and the goal is to display the results after each iteration. Each element has a radiosity value, which is the radiosity calculated so far for the element, and an unshot radiosity value, which is the portion of that element's radiosity that has yet to be "shot". In the beginning of the process, only the light sources have an unshot radiosity value greater than zero.

During each iteration, the element with the greatest unshot radiosity is chosen and its radiosity is shot through the environment. The other elements may receive some radiosity, which is then added to both their unshot and received radiosity. After the shooting, the unshot radiosity of the shooting element is set to zero. The elements can thereafter be rendered using their received radiosity. The algorithm is usually terminated when the total unshot radiosity falls below a given threshold.

One of the greatest advantages with the progressive refinement method is that the illumination that will have the most impact on the final image is calculated first. The overall time complexity is still $O(n^2)$, but it usually converges very fast. The form factors can also be calculated on-the-fly during the solution stage, which reduces the form factor storage to $O(n)$.

Progressive radiosity is often used in conjunction with substructuring [Cohe86], which introduces a two-level hierarchy; a coarser level of *patches* that shoot light to a finer set of receiving *elements*. This way, the elements can capture fine illumination details without increasing the number of shooting patches.

3.2 Hierarchical Radiosity

3.2.1 Overview

Hierarchical radiosity [Hanr91] is the most significant theoretical modification of the original radiosity algorithm. By using techniques similar to those used in the *n-body algorithm* for gravitational simulations [Appe85], the hierarchical radiosity algorithm reduces the complexity from quadratic to linear in the number of elements used.

The two-level substructuring hierarchy is extended by removing the separation of patches and elements, replacing them with a continuum of hierarchical elements. Radiosity exchange can then be computed between arbitrary levels of the hierarchy. For any pair of surfaces, an appropriate subdivision level for each side is determined and then used to compute the energy exchange. This will in effect divide the form factor matrix into blocks, each of which represents an interaction between groups of elements. The advantage is that the number of blocks in the matrix is $O(n)$ (where n is the total number of elements) compared to $O(n^2)$ entries in the regular form factor matrix [Hanr91].

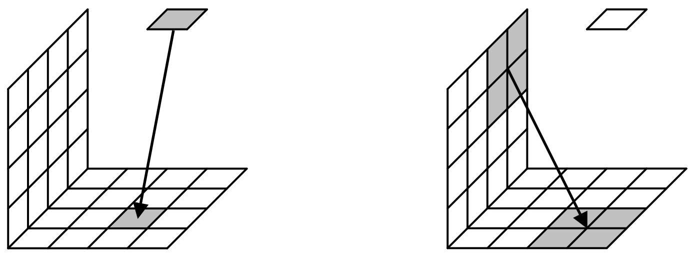


Figure 3.1. Hierarchical radiosity. Direct lighting (a) is calculated at a finer subdivision level than indirect lighting (b). The shaded areas represent pairs of elements exchanging energy and the arrows represent the links between the elements.

For every input polygon in the scene a hierarchical element representation is created. The root node of the hierarchy represents the entire polygon, and deeper nodes represent partitions of it. The interaction of hierarchical elements exchanging energy is represented in the form of *links*, which contain information of the form factor and some estimate of the error in the approximated radiosity transfer. Initially, one link for every pair of input polygons has to be created. Each link is then refined until the estimated error for the link falls below a preset threshold. When a link is refined, either the source element or the receiver element is subdivided. The refined link is replaced by new links pointing to the children of the subdivided element.

Because of the initial linking, the cost is quadratic in the number of input polygons, k, but linear in the number of solution elements, n. The cost is thus actually $O(k^2 + n)$, and the overall linear complexity only holds if $k << n$.

3.2.2 Link Refinement

Link refinement is the first step in the hierarchical radiosity algorithm. All pairs of polygons in the scene are initially connected by a link. Each link is then recursively refined until the energy transfer representation reaches the desired accuracy. The goal is to predict whether a group of form factors can be represented by a single link. Such a prediction method is referred to as a *refinement oracle*.

The refinement oracle is a central part of hierarchical radiosity because it affects both the computation time and the accuracy of the radiosity computation. Early methods used a simple criterion based on the magnitude of the form factor to drive refinement [Hanr91]. More recent work has focused on perceptually-based measures [Gibs97, Prik99, Stam00] or visibility-based measures [Dura99].

When a link between two elements is refined it must also be decided if the source element and/or the destination element should be refined. Typically this decision is made by comparing the relative contribution of the two elements to the total error in the link. This can be done by comparing the projected areas of the elements along the direction of the transport.

3.2.3 Solving the Radiosity System

The link refinement stage results in a hierarchical representation of the form factor matrix which is now used to compute the actual energy transfers. For a given element in the hierarchy, energy is "gathered" through all *incoming* links at that node. This is done by multiplying the form factor stored in the link by the radiosity value for the element at the other end of the link. Since links can be established at all levels in the hierarchy, the radiosity of an element also depends on the links of its ancestors as well as the links of its descendents in the hierarchy. In order to obtain a complete view of the energy transfers in the hierarchy a *push-pull* process is needed. Starting from the root element in each tree, each element's reflected radiosity is added to each of its children's radiosity. This in effect *pushes* radiosity to the leaves of the hierarchy. Since radiosity is defined per unit area, the correct radiosity value for an element is the area average of its children radiosities. Radiosity is therefore *pulled* from the leaves to the root, averaging the radiosity values at each level.

The solution process consists of repeatedly executing the gather and push-pull operations until the system converges.

3.2.4 Interleaving Refinement and Solution

In the original two-stage solution, we first refine the links, and then solve for the radiosities. The refinement oracle can thus only make decisions based on the geometry of the form factor estimates and not on the actual energy transfers in the scene[3]. This is a simple approach, but it ignores the fact that the optimal link refinement is dependent on the final radiosity solution.

An interleaved solution algorithm can make better decisions about link refinement. We then repeatedly execute the refine, gather, and push-pull operations. The refinement oracle can then take the estimated energy transfer across each link into account, leading to a more efficient solution.

A third solution method is to run refinement until all links meet some error criterion, then iterate the gather and push/pull stages until the system converges, and then repeat the whole process for a lower error criterion. This is called a scheduled solution and generally provides the best and most stable results of the various hierarchical radiosity solvers [Will00].

3.3 Clustering Methods for Hierarchical Radiosity

3.3.1 Volume Clustering

We have seen that hierarchical radiosity is linear in the number of solution elements but also quadratic in the number of input polygons. The classical hierarchical radiosity algorithms therefore work well on scenes with a small number of large polygons, but become impractical in time and memory consumption for more complex scenes.

[3] The initial light sources can possibly be accounted for.

To avoid the initial linking of the input polygons in a scene, clustering methods for hierarchical radiosity were developed [Smit94, Sill95a, Sill95b, Gibs96]. These methods create a new *volume cluster* hierarchy above the input polygons with a single root cluster for the entire scene. The volume clusters contain groups of potentially disconnected polygons with varying orientation and reflection properties. With the use of volume clusters, only a single initial link is required and the k^2 term in the complexity is eliminated.

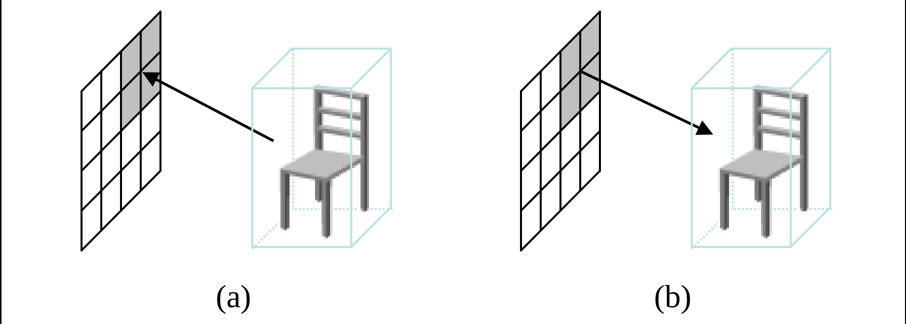


Figure 3.2. Volume Clustering. Groups of polygons, such as the chair above, can both send (a) and receive (b) energy as a whole.

Calculating the light incident on a cluster is not trivial. The simplest approach is to assume the clusters are isotropic, and sum the incoming light from all directions. This method is fast, but often too inaccurate. A more successful alternative is to push the light down to the input polygons when it is gathered across a link, but this can be costly for complex scenes. The clustering methods are still significantly faster than classical hierarchical radiosity.

Another problem with volume clustering is that it is difficult to interpolate the irradiance between clusters. This often leads to blocky results as can be seen in Figure 3.3. One solution is to refine the clusters down to the input polygon level and interpolate over each polygon, but this negates many of the benefits of using clustering. Generally, volume clustering methods are more suitable to handling unorganized sets of polygons than highly tessellated models.

3.3.2 Face Clustering

For groups of polygons that represent connected surfaces, the volume clusters can be replaced by multiresolution models. Such models often provide a much better fit to the original surfaces than volume clusters, and the need to push light to the leaves during iterations can be avoided. Early multiresolution approaches used manually constructed simplified models [Rush93], or applied illumination from a simple scene to a more detailed version of it [Greg98]. A more promising approach, where the simplification is driven by the radiosity algorithm, is called *face cluster radiosity* [Will99, Will00]. The face cluster hierarchy groups together faces that have similar normals, and thus approximates largely planar surfaces well. The rationale for using multiresolution models is that for highly tessellated models, the geometric detail is often much higher than the illumination detail.

Since both face clusters and refined polygons yield a contiguous piece of surface with a normal, they can be treated similarly by the hierarchical radiosity algorithm. The algorithm can thus operate efficiently at any level of detail in the hierarchy. For scenes with highly tessellated surfaces, face cluster radiosity yields sub-linear performance in the number of input polygons [Will99].

Concurrent and independent of our work, the application of face cluster radiosity to highly tessellated models has been explored by Gobbetti *et al*. [Gobb02, Span02, Gobb03].

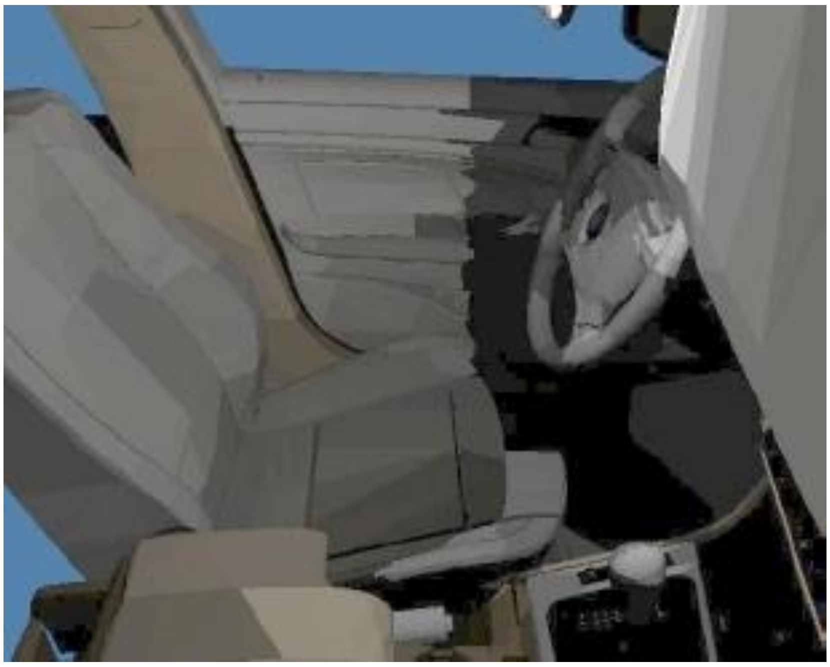


Figure 3.3. Volume clustering artifacts. (From Hasenfratz *et al*. [Hase99]).

3.4 Visibility and Meshing

3.4.1 Visibility Calculation

Evaluating visibility between elements tends to be by far the most expensive part of the form factor calculation. Early radiosity methods, such as the hemicube method, sometimes use scan conversion algorithms to determine visibility, but most methods are based on ray casting. This is especially true for the hierarchical radiosity methods. Ray casting is a common algorithm in computer graphics and most acceleration methods apply to radiosity as well. Overviews of ray casting acceleration techniques can be found in [Glas89] and [Smit98].

To limit the number of visibility queries between elements, methods to determine guaranteed full occlusion and full visibility can be used [Lebl00]. Such methods can improve performance considerably in scenes with large planar polygons. They are however hard to apply to scenes with concave objects and curved surfaces.

3.4.2 Regular Refinement and Discontinuity Meshing

The standard mesh refinement technique used in radiosity is regular refinement. For quadrilateral or triangular elements, each edge is split at its midpoint forming four new elements (see Figure 3.4 a). This results in a *quadtree* hierarchy. Regular refinement has the advantage of simplicity and robustness. Each element can also be refined independently. The disadvantage is that, if an element is poorly shaped or not well aligned to the discontinuities in the illumination, the same will hold true for its subelements.

To combat the problem of capturing discontinuities in the illumination, *discontinuity meshing* algorithms were developed. The early algorithms were based on shadow volumes and tried to determine the boundaries of the umbra and penumbra [Nish85, Camp91]. The polygon mesh was split against these boundaries, thus effectively inserting *discontinuity edges* in the mesh (see Figure 3.4 b). More recent work on discontinuity meshing has focused on critical surfaces and visibility events [Heck92, Lisc92, Dura99].

Discontinuity meshing can create impressive results, especially for relatively sharp shadows. It can however be hard to implement robustly and the number of discontinuities can be overwhelming for complex scenes. It is therefore more suitable for scenes that consist of a small number of large polygons than for scenes with highly tessellated, curved surfaces.

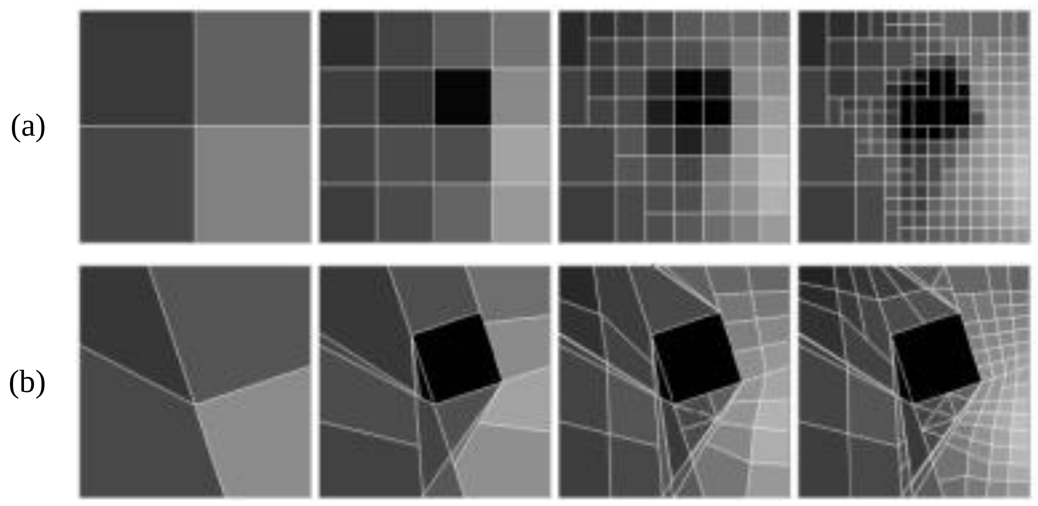


Figure 3.4. Mesh refinement. (a) Regular refinement. (b) Discontinuity meshing. (From Lischinski *et al*. [Lisc93]).

3.5 Output Representations

3.5.1 Gouraud-Shaded Polygons

Gouraud-shaded polygons are the most commonly used output representation for radiosity calculations. The radiosity for each vertex is calculated, and the renderer linearly interpolates these samples over the polygons. This method is also known as vertex lighting.

Since radiosity calculations often result in meshes containing a large number of elements, the number of triangles may be too high for real-time rendering. Various mesh simplification schemes can be applied as a post process to the calculation to reduce the polygon count. Hoppe has shown that radiosity meshes are excellent candidates for polygon simplification [Hopp96].

3.5.2 Textures

Rendering radiosity meshes using vertex lighting can lead to poor performance. An alternative solution is to represent the radiosity using textures. Arvo proposed *illumination maps* to store diffuse illumination [Arvo86], and Heckbert introduced *radiosity textures* [Heck90]. These techniques were suitable for generation of still images but did not exploit hardware accelerated texture mapping[4].

Myszkowski and Kunii [Mysz94] developed a solution where mesh elements are replaced by textures as a post-processing step to the lighting simulation. A combination of vertex lit mesh elements and texture mapping is then used during interactive walkthroughs. A similar approach is used in [Bast96]. Möller [Möll96] describes a method to replace radiosity solutions with textures for NURBS models. This allows the same illumination map to be reused for multiple levels-of-detail.

As texture mapping became available on commodity graphics hardware, computer games started using textures to represent lighting detail. Commodity graphics cards are often heavily optimized for multi-texturing, and the illumination textures can quickly be blended with standard textures to produce the final result, see Figure 3.5. Texture maps used to represent illumination are often called *light maps*. Light mapping for use in games and real-time applications is described in [Zhuk97, Zhuk98].

Storing the illumination using light maps can increase the performance in real-time applications substantially. However, finding a parameterization, or uv-mapping, for arbitrary polygonal surfaces is not straight-forward. Most light map solutions presented so far either map each polygon individually, or a group of adjacent polygons that can be projected onto a single plane without overlapping [Zhuk97], using a simple projection.

The parameterization problem for light maps is more or less identical to the mapping of textures used for 3D painting [Low01]. In our case, we want to paint illumination to the textures. Low identifies three criteria for surface parameterization used in 3D painting:

- *Unique-region criterion*. Each point on the 3D surface must be assigned a unique point in texture space.
- *Adjacency criterion*. Adjacent surface primitives in 3D space should preferably be mapped to adjacent regions in texture space. If this criterion is not met, the texture filtering used during rendering will yield incorrect results near the boundaries of the primitives. Another advantage of meeting this criterion is that texture memory can be used more efficiently.
- *Minimal distortion criterion*. The texture mapping distortion should be minimized to yield a uniform resolution in 3D space and also avoid stretch artifacts during rendering.

These are criteria that a parameterization used for light mapping should try to meet as well. The unique-region criterion must always be upheld, just as for 3D painting, while the others can be more or less relaxed.

[4] Hardware accelerated texture mapping became available in late 1992, the first example being the Silicon Graphics RealityEngine [Baum98].

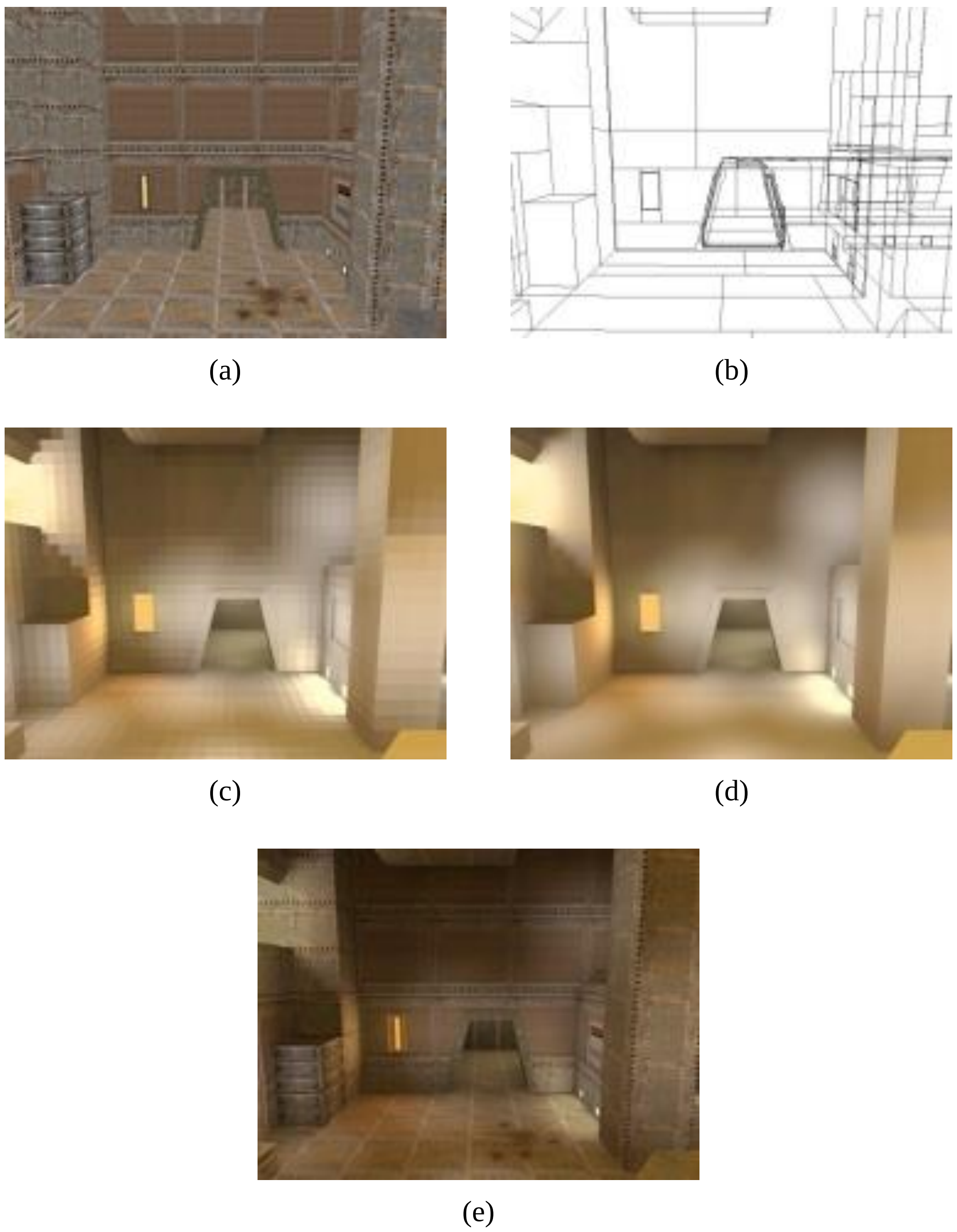


Figure 3.5. Light mapping example from the computer game Quake II, released by id Software in 1997. (a) The original textured scene. (b) Wireframe image showing the low polygon count. (c) Unfiltered light maps calculated with radiosity. Note that the light map textures have relatively low resolution compared to the base textures, but still allow for a much higher light sample density than a vertex lit mesh would have. (d) Filtered version of (c), showing a smooth lighting solution. (e) The base textures from (a) combined with the light maps from (d) results in the final image. Both the filtering step and texture blending step is hardware accelerated on commodity graphics cards.

Most algorithms for parameterization of general polygonal models are based on the *texture atlas* approach [Mail93, Sand01]. The model to be textured is partitioned into *charts* homeomorphic to discs that can be parameterized independently. Various optimization techniques can then be used to minimize the texture distortion for each chart before the charts are finally packed in texture space. Texture atlas construction for trimmed NURBS models is described in [Guth03].

Lévy *et al*. introduced the *least squares conformal maps* (LSCMs) method for chart parameterization [Lévy02]. This robust method has been used to create texture atlases for radiosity purposes in [Ray03a]. Ray and Lévy have also created a hierarchical version of the LSCM method [Ray03b].

4 An Efficient Hierarchical Radiosity Algorithm

4.1 Overview

In this chapter we present a radiosity algorithm that efficiently handles highly tessellated models. The core of the algorithm is based on the face cluster radiosity approach [Will99], which has shown very promising results for such models. Above surface level, we use a volume clustering algorithm based on [Müll99]. Polygon refinement is represented by a BSP tree, using a new splitting approach.

The algorithm consists of a preprocessing phase and a solution phase. During the preprocessing, a hierarchical representation of the scene is constructed. For each group of connected polygons a face cluster hierarchy is built. The face cluster hierarchies are then volume clustered to form a single rooted hierarchy for the entire scene. An initial self-link is created for the root cluster before the solution phase begins. An interleaved solver is used, that repeatedly executes the refine, gather, and push-pull steps described in the previous chapter.

4.2 Face Cluster Hierarchy

4.2.1 Motivation

The face cluster radiosity algorithm has a complexity that is sub-linear in the number of input polygons. When used on scenes with highly tessellated surfaces, it shows a huge performance increase over previous volume clustering algorithms [Will99]. This makes the algorithm perfectly suitable for the kind of data we are trying to handle (see Figure 4.1).

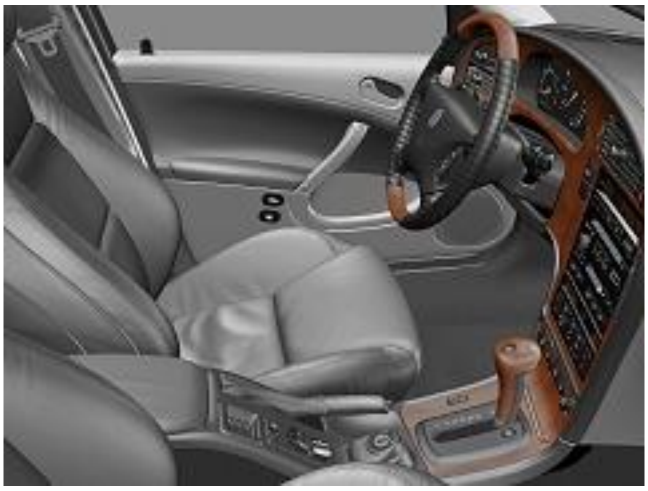


Figure 4.1. Typical input model. This is the kind of data that we are trying to find a suitable algorithm for.

A face cluster hierarchy is basically a multiresolution model. When used in a radiosity calculation it allows some regions to be heavily simplified while other regions are represented in greater detail. The level of detail used is driven by the radiosity calculation; in areas where the illumination varies more, a higher level of detail is used. Often, the input polygons need not to be touched at all. In such cases the complexity of the algorithm is independent of tessellation level and if the number of input polygons in the model is increased by tessellating it further, it will not affect the solution time.

Unlike previous clustering algorithms for hierarchical radiosity, face cluster radiosity does not require the illumination to be pushed down to the input polygons during the solution phase. This is possible because of a combination of face clustering and the use of *vector-based radiosity*. The following sections describe how the face cluster hierarchy is constructed and how vector-based radiosity can be used for representing energy transfers between clusters. Much of the material is summarized from Willmott's and Garland's work on face cluster radiosity and face cluster hierarchies respectively. Further details can be found in [Will00] and [Garl99].

4.2.2 Hierarchy Construction

To build a face cluster hierarchy, we first form the *dual graph* of the model. The dual graph is defined by mapping each face of the model to a node in the dual graph and connecting two dual nodes if the corresponding faces on the model are adjacent. In this thesis we will only consider triangular faces, since our input data is always triangulated.

In the dual graph, each node corresponds to a face cluster; a connected set of triangles that have been grouped together. Initially, each cluster contains only one triangle from the input model. To create the hierarchy, adjacent clusters are merged by collapsing the corresponding dual edges in the graph. This is done by assigning a cost to each dual edge and iteratively collapsing the dual edge with the least cost. After each collapse, the costs of the dual edges pointing to the merged clusters are updated. When the process is complete, each root cluster in the hierarchy will correspond to a connected part of the input model. It is important to note that the algorithm does not change the original geometry in any way; it merely groups triangles into progressively larger clusters. Figure 4.2 shows three levels of a simple cluster hierarchy.

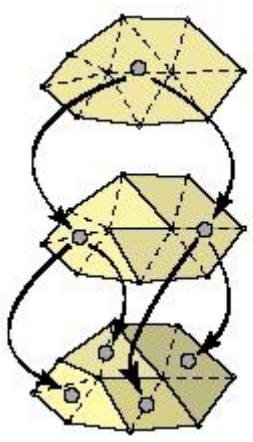


Figure 4.2. Building a face cluster hierarchy. (From Willmott [Will00]).

The cost metric will determine the qualities of the hierarchy. Different applications may need different criteria and for radiosity purposes, planarity is the most important criterion [Will00]. The planarity criterion can be expressed using a dual form of Garland's quadric error metric [Garl99]. Every cluster has an associated set of faces $\{f_1,\ldots,f_n\}$ and a set of points $\{\mathbf{v}_1,\ldots,\mathbf{v}_n\}$ determined by the vertices of these faces. We first need to find the least squares best fit plane to this set of points. For a plane specified by a unit normal **n** and a scalar offset *d*, the distance of a point **v** to the plane is $\mathbf{n}\cdot\mathbf{v}+d$. The *fit error* of a plane $\mathbf{n}\cdot\mathbf{v}+d=0$ is the average squared distance of all the points to the plane:

$$E_{fit}=\frac{1}{n}\sum_{i=1}^{n}(\mathbf{n}\cdot\mathbf{v}_i+d)^2 \tag{4.1}$$

The least squares best fit plane is the plane which minimizes this error. Using Garland's quadric error metric, we can compactly represent this sum as:

$$E_{fit}=\frac{1}{n}\sum_{i=1}^{n}(\mathbf{n}\cdot\mathbf{v}_i+d)^2=\frac{1}{n}\left(\sum_{i=1}^{n}P_i\right)(\mathbf{n},d) \tag{4.2}$$

where

$$P_i=(\mathbf{A}_i,\mathbf{b}_i,c_i)=(\mathbf{v}_i\mathbf{v}_i^{\mathrm{T}},\mathbf{v}_i,1) \tag{4.3}$$

and

$$P(\mathbf{n},d)=\mathbf{n}^{\mathrm{T}}\mathbf{A}\mathbf{n}+2\mathbf{b}^{\mathrm{T}}(d\mathbf{n})+cd^2 \tag{4.4}$$

Each dual node has an associated fit quadric *P* which requires ten coefficients to represent the symmetric 3x3 matrix **A**, the 3-vector **b**, and the scalar *c*.

The standard technique used to find the optimal plane which minimizes *P*(**n**, *d*) is based on principal component analysis (PCA) (see e.g., [Joll86]). We construct the covariance matrix:

$$\mathbf{Z}=\sum_{i=1}^{n}(\mathbf{v}_i-\overline{\mathbf{v}})(\mathbf{v}_i-\overline{\mathbf{v}})^{\mathrm{T}} \tag{4.5}$$

where $\overline{\mathbf{v}}$ is the mean of the vertices:

$$\overline{\mathbf{v}}=\frac{1}{n}\sum_{i=1}^{n}\mathbf{v}_i \tag{4.6}$$

The three eigenvectors of the matrix **Z** determine the local frame with $\overline{\mathbf{v}}$ as the origin. The eigenvector corresponding to the smallest eigenvalue is the normal of the least squares best fit plane through the set of points $\{\mathbf{v}_1,\ldots,\mathbf{v}_n\}$. The covariance matrix can be directly extracted from the fit quadric P:

$$\mathbf{Z} = \mathbf{A} - \frac{\mathbf{b}\mathbf{b}^{\mathrm{T}}}{\mathrm{c}} \tag{4.7}$$

We also use the covariance matrix to calculate an oriented bounding box for each cluster. The bounding box will later be used for visibility calculations.

Minimizing the planarity term E_{fit} will tend to merge clusters which are collectively nearly planar. To ensure that the clusters have compact shape, i.e., are as nearly circular as possible, and that the normals of the triangles in the clusters are consistently oriented, the error metric also has to take other measures into account. Willmott [Will00] has developed a simple and robust error metric for face clustering aimed at radiosity calculations:

$$E = (E_{fit} + k \cdot A)\frac{p^2}{|\mathbf{S}|} \tag{4.8}$$

where

- E_{fit} is the previously described planarity term.
- k controls the importance of balancing the hierarchy at the cost of relaxing the planarity term. A small value will make the clusters more planar, whereas a larger value will create a more balanced hierarchy.
- A is the area of the merged face cluster.
- p is the perimeter of the merged face cluster.
- $\mathbf{S}$ is the sum of the area weighted normals of the triangles in the merged face cluster:

$$\mathbf{S} = \sum_{i=1}^{n} A_i \mathbf{n}_i \tag{4.9}$$

Figure 4.3 shows an example of a hierarchy created using the error metric from Equation (4.8).

4.2.3 Vector-based Radiosity

Classical hierarchical radiosity uses scalar values to represent radiosity. The irradiance E_i of an element can be approximated using the point-to-point form factor approximation [Cohe93]:

$$E_i = \sum_j \frac{(\mathbf{n}_i{}^{\mathrm{T}}\mathbf{r}_{ij})_+ (\mathbf{r}_{ji}{}^{\mathrm{T}}\mathbf{n}_j)_+}{\pi r_{ij}{}^2} \overline{v}_{ij} A_j b_j \tag{4.10}$$

where

- Each element *i* has normal $\mathbf{n}_i$, area A_i, and unknown radiosity b_i
- r_{ij} is the distance between elements *i* and *j*.
- $\mathbf{r}_{ij}$ is the unit vector between elements *i* and *j*.
- $\bar{v}_{ij}$ is the average visibility between two points on elements *i* and *j*: 1 for no occlusion, and 0 for full occlusion.
- $(x)_+$ means that *x* is clipped to zero, i.e., $(x)_+ = \max(0, x)$.
- $\mathbf{a}^{\mathrm{T}}\mathbf{b}$ is the dot product in matrix notation. The associative property of matrix multiplication will be exploited later to rewrite the above formula.

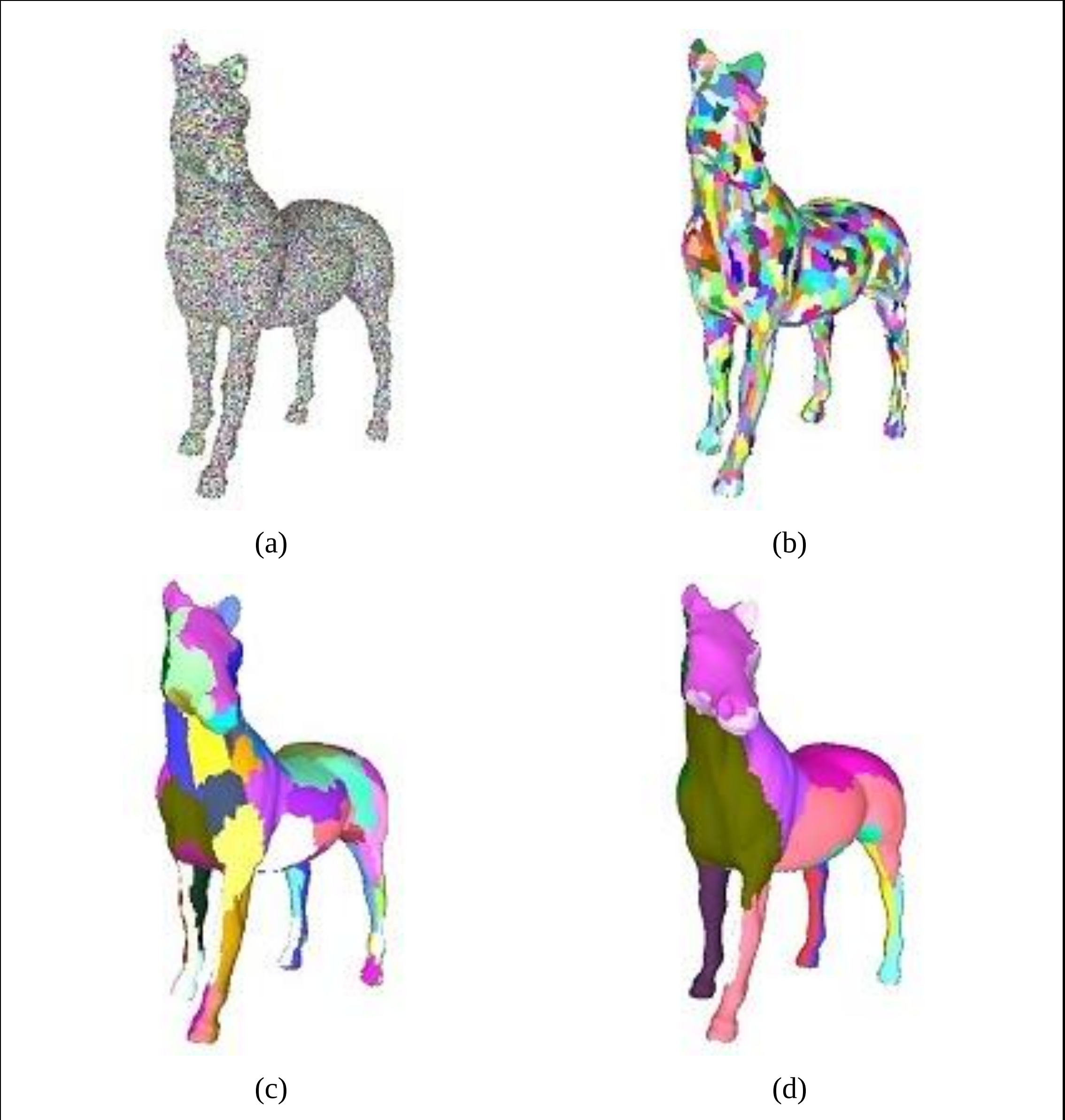


Figure 4.3. A face cluster hierarchy. (a) shows the leaf clusters, i.e., the input polygons, of the model. (b), (c), and (d) show clusters higher up in the hierarchy.

When used in a cluster-based algorithm, a scalar radiosity representation leads to a faceted appearance that hides the geometric detail for clusters containing curved or bumpy surfaces. Willmott *et al*. have adapted the original radiosity method to face clustered models by using a vector-based representation [Will99]. Consider the light interaction between two clusters in Figure 4.4.

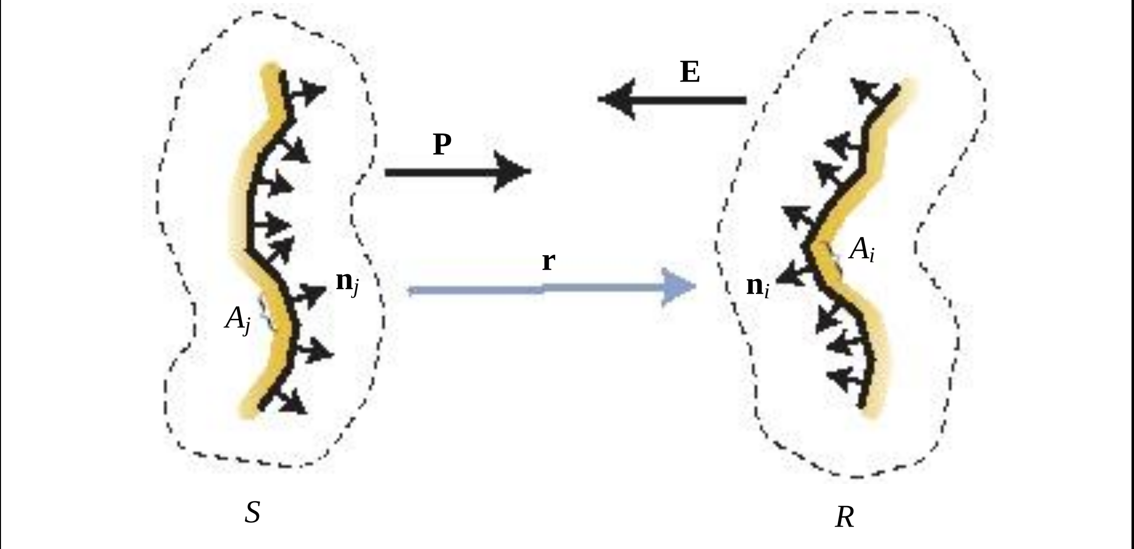


Figure 4.4. Radiosity transfer between two face clusters. Element *j* in the source cluster *S* radiates energy towards element *i* in the receiver cluster *R*. (From Willmott [Will00]).

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 of elements are inter-visible and that the source elements are close together and far from the receiver, then $\mathbf{r}_{ij}$, $\mathbf{r}_{ji}$, r_{ij}, and $\bar{v}_{ij}$ become independent of *j*, and we can approximate the irradiance from a single cluster as [Will99]:

$$E_i \approx \left(\mathbf{n}_i^{\mathrm{T}} \left[\frac{-\mathbf{r}\mathbf{r}^{\mathrm{T}}}{\pi r^2} \sum_j \mathbf{n}_j A_j b_j \right] \bar{v}_i \right)_+ \tag{4.11}$$

The transfer can be rewritten in terms of two vector quantities as:

$$E_i = \mathbf{n}_i^{\mathrm{T}} \mathbf{E} \tag{4.12}$$

where **E** is referred to as the *irradiance vector*, defined as:

$$\mathbf{E} = \bar{v} \frac{\delta(\mathbf{S}_R, -\mathbf{r})}{\pi r^2} (\mathbf{r}^{\mathrm{T}} \mathbf{P})_+ \tag{4.13}$$

where

- $\mathbf{S}_R$ is the sum area normal of the receiving cluster *R*.
- δ clipps against the sum area normal:

$$\delta(\mathbf{S},\mathbf{x}) \equiv \begin{cases} \mathbf{x} & \text{if } \mathbf{x}\cdot\mathbf{S} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

- **P** is the *power vector,* defined as:

$$\mathbf{P} = \sum_j \mathbf{n}_j A_j b_j = \sum_j \mathbf{S}_j b_j \tag{4.15}$$

The irradiance vector is a 3-vector whose components represent the irradiances on the planes normal to the x, y, and z axes at the receiver. Using this representation, rather than a scalar irradiance, allows coarse variations in the irradiance as a function of orientation to be modeled. This eliminates most of the faceting effects that usually appear when using only a single scalar value. An irradiance vector approach has also been employed by Arvo and Glassner [Arvo94].

The power vector, also a 3-vector, allows the outgoing power to have a directional dependency. 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.

These vector quantities can be substituted for the irradiance and radiosity in a standard hierarchical radiosity algorithm by modifying the push-pull and gather operations (see Section 3.2.3). During the push phase, the following equation is used:

$$\mathbf{E}_{child} = \delta(\mathbf{S}_{child}, \mathbf{E}_{parent}) + \mathbf{R}_{child} \tag{4.16}$$

where $\mathbf{R}_{child}$ is the sum of the irradiance vectors incident directly on the child. During the pull operation, the radiosity of the parent is calculated:

$$b_{parent} = \frac{\sum b_{child} A_{child}}{A_{parent}} \tag{4.17}$$

and the power vector reconstructed with:

$$\mathbf{P} = \mathbf{S}b \tag{4.18}$$

When modeling the transfer of radiosity, the standard form factor coefficient is replaced by a *transfer vector*:

$$\mathbf{m} = \frac{\overline{v}\delta(\mathbf{S}_R, -\mathbf{r})(\mathbf{r}^{\mathrm{T}}\mathbf{S}_S)_+}{\pi r^2} \tag{4.19}$$

which allows us to rewrite Equation (4.13) simply as:

$$\mathbf{E} = \mathbf{m}b \tag{4.20}$$

At the leaves of the hierarchy, the accumulated irradiance is transformed into radiosity using the following equation:

$$b = \frac{\rho \mathbf{S}^{\mathrm{T}}\mathbf{E}}{A} + e \tag{4.21}$$

where

- ρ is the reflectance of the leaf element.
- e is the emittance of the leaf element.

The power vector of the leaf can then be reconstructed using Equation (4.18).

In all of the above equations, a monochromatic world is assumed in order to simplify the presentation. In practice we extend them to the familiar RGB color model by storing b and $\mathbf{E}$ for each color channel and operate on them independently. The transfer vector, **m**, is still wavelength independent.

4.3 Volume Cluster Hierarchy

4.3.1 Motivation

Above surface level we use a volume clustering algorithm based on the work of Müller *et al.* [Müll99]. The hierarchy is essentially a binary bounding volume hierarchy of axis-aligned bounding boxes, suitable for both radiosity transfers and visibility queries. This fits in well with the face cluster hierarchy and the radiosity algorithm can operate on both hierarchies in a uniform way.

Based on the results of the analysis of clustering algorithms done by Hasenfratz *et al.* [Hase99], this type of volume hierarchy seems to be the best for our purposes. It has the most predictable behavior of the tested hierarchy types and creates “intuitive” hierarchies for scenes with very different object sizes. The main disadvantages with a bounding volume hierarchy are construction time and cluster overlaps. Both these problem areas are efficiently handled by the algorithm described in [Müll99], resulting in a very promising radiosity clustering algorithm suitable for a wide range of scenes.

4.3.2 Hierarchy Construction

The hierarchy construction algorithm uses a top-down approach to recursively partition the set of root nodes from the face cluster hierarchies. No entities need to be split, and no empty clusters are created. Starting with the entire set of face cluster root nodes, we sort the clusters along the three major coordinate axes, using the center of each cluster as sort key. Based on these sorted lists, we evaluate the potential partitioning positions along each axis by splitting the sorted list of objects in a *left* and *right* part. In contrast to the commonly used median cut scheme, a *cost function* is used to determine the subdivision position. By minimizing this cost function over all partitioning positions, an optimal subdivision is obtained which generates two new subsets containing the *left* clusters and the *right* clusters. The process is recursively applied on these two sets and is terminated when a set contains only one cluster. A small partitioning example is shown in Figure 4.5.

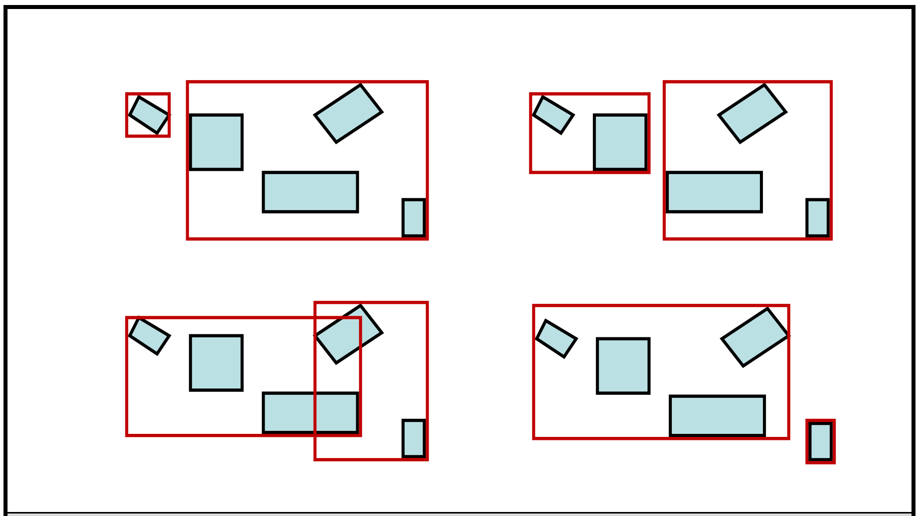


Figure 4.5. Volume cluster hierarchy creation. Four possible partitions along a single axis are shown. The filled rectangles represent the oriented bounding boxes of the root face clusters.

We use the same cost function as in [Müll99], which is similar to the *surface area heuristic* (SAH) introduced by MacDonald and Booth [MacD89]. The cost of a partitioning *H*, with children H_{left} and H_{right}, is given by:

$$C_H(axis) = \frac{S(H_{left})}{S(H)}\left|H_{left}\right| + \frac{S(H_{right})}{S(H)}\left|H_{right}\right| \qquad (4.22)$$

where

- $\left|H\right|$ is the number of clusters in the set H.
- $S(H)$ is the surface area of the axis aligned bounding box around H.
- $axis \in \{X,Y,Z\}$.

This creation algorithm is fast, $O(n \log n)$, while guaranteeing hierarchies of very high quality comparable to much more costly bottom-up methods. In particular, it is efficient for ray acceleration in visibility computations, it minimizes overlaps of bounding volumes, and it adapts well to the distribution of the objects in the scene [Müll99].

4.4 Patch Hierarchy

4.4.1 Motivation

Below the input triangle level, a patch hierarchy is used to represent refinement. The commonly used regular refinement method (see Section 3.4.2) does not work well for highly tessellated geometry. Generally, the tessellation is optimized to represent the curvature of the surfaces, and hence poorly shaped triangles are very common (see Figure 4.6). To overcome this problem, we have developed a novel, more flexible, BSP-based[5] patch hierarchy. Each time a leaf patch is refined, an arbitrary splitting line can be chosen to subdivide the patch into two sub-patches.

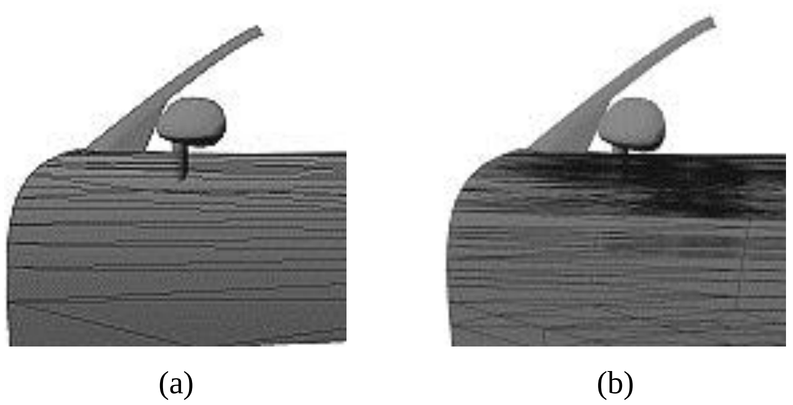


Figure 4.6. Regular refinement for tessellated geometry. (a) The original mesh. (b) Regularly refined mesh. The long, thin triangles make it very hard to capture the shadow from the door mirror.

4.4.2 Splitting Strategy

When a leaf patch needs to be refined, a suitable splitting line has to be selected. As discussed in Chapter 3, such a line can be chosen based on discontinuities in the illumination. This is an interesting area of research, but outside the scope of this thesis. Instead, we try to create patches with regular shape. First, the minimum-area enclosing rectangle (MER) of the patch is calculated. The line perpendicular to the longest side of this rectangle, splitting it at its midpoint, is selected as a candidate splitting line (see Figure 4.7a). To minimize the number of introduced t-vertices we then try to snap the line to the nearest vertex on each side. If a

[5] A *binary space partitioning* (BSP) tree represents a recursive, hierarchical partitioning of space. In two dimensions, nodes are created by splitting larger regions along an arbitrary line.

vertex is within a predefined snap tolerance, the line is moved to intersect the vertex (see Figure 4.7b).

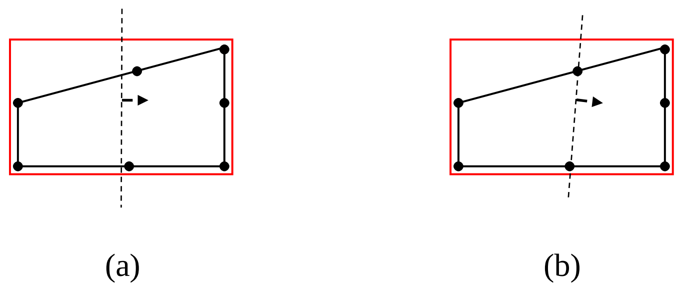


Figure 4.7. Calculating the splitting line for a patch. (a) The line cutting the minimum-area enclosing rectangle of the patch in the middle is chosen as a candidate line. (b) If a vertex is within a predefined snap tolerance, the line is moved to intersect the vertex.

The snapping procedure significantly decreases the number of t-vertices, which leads to fewer triangles when the mesh is later triangulated (see Section 4.4.3). The triangles in the final mesh will therefore generally be more well-shaped, as evident in Figure 4.8.

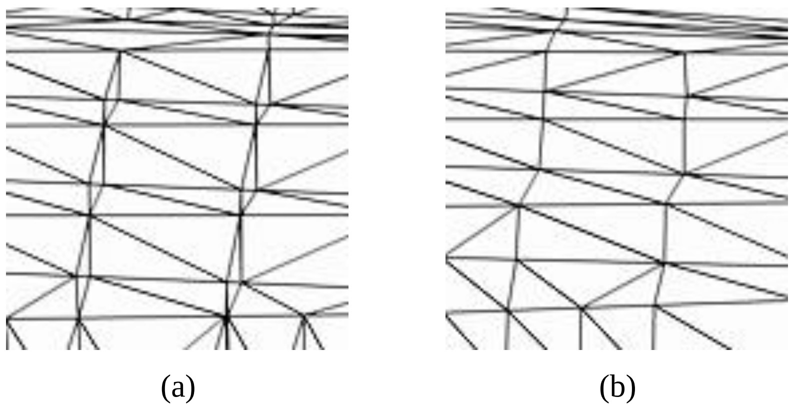


Figure 4.8. (a) A mesh refined without vertex snapping. (b) With vertex snapping.

This splitting scheme has worked well in our experience; creating well-shaped patches even if the original mesh consists of very long and thin triangles (see Figure 4.9).

4.4.3 Eliminating T-vertices

T-vertices are inevitably introduced during patch refinement. If t-vertices are not shared between patches, discontinuities will be visible during rendering since the calculated radiosity at a t-vertex will generally be different from the interpolated value at the same point. To eliminate t-vertices, we use a Delaunay triangulation pass for each patch before rendering.

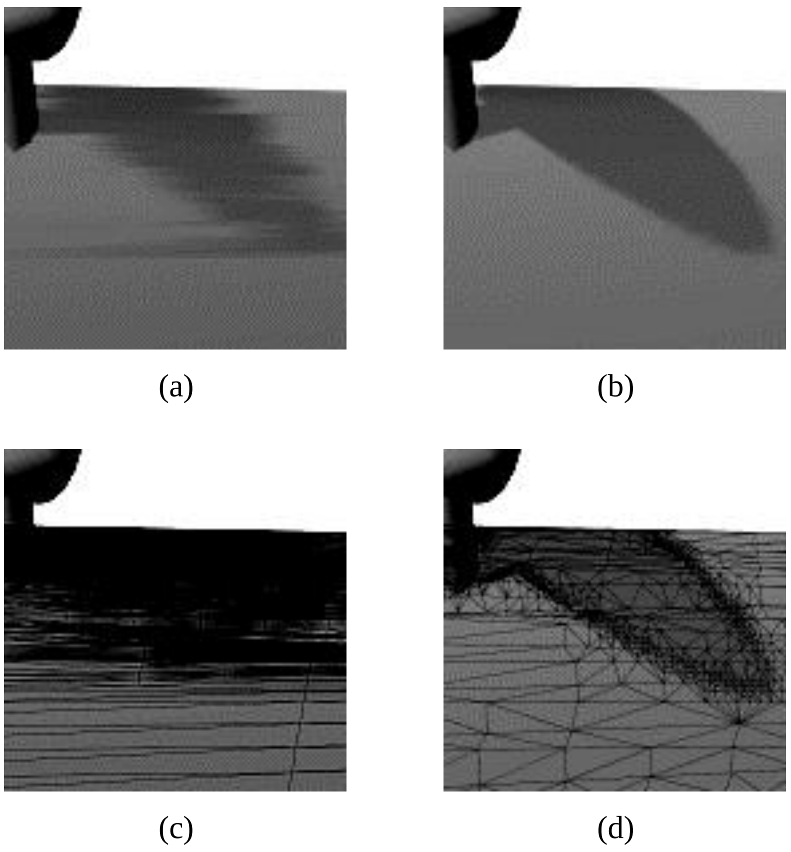


Figure 4.9. Comparison of regular refinement (a) and our novel refinement approach (b). Corresponding wireframe versions are shown in (c) and (d), although most of the triangles in (c) are too thin to be visible in this image. Note that the number of triangles is the same for both methods.

4.5 Algorithm

4.5.1 Preprocess

The preprocess phase first creates a face cluster hierarchy for each object in the scene. This is typically done only once and if the geometry of an object remains unchanged, its hierarchy can be reused for several radiosity calculations. Other model properties like position, orientation, and material attributes can be modified without the need to perform the face clustering again. After the face clustering, a hierarchical radiosity element for each face cluster *root* node is created. Usually much of each face cluster hierarchy remains unused during the radiosity calculation. We therefore begin with just the root nodes and create new elements on demand during refinement. The elements add information such as irradiance vectors, radiosity, radiosity links, and an oriented bounding box for each face cluster. This information is not stored in the original face cluster hierarchies in order to save memory.

The initial face cluster elements are then volume clustered and elements are created for these clusters as well. An initial self-link is finally created for the root node of the entire hierarchy. Non-reflective light sources are volume clustered independently and linked to the root node of the main hierarchy. This speeds up the initial light transfers substantially and also reduces the number of links.

4.5.2 Solver

A standard interleaved solver is used as described in [Cohe93]. The gather, push-pull, and refine stages are repeated until the system converges (see Figure 4.10). When gathering energy for an element, we apply Equation (4.20) for all incoming links and sum the result. During the push-pull phase, we use Equations (4.16) and (4.17). The refine procedure follows that outlined in [Cohe93], using the error estimation described in the next section.

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

Figure 4.10. Solver pseudo-code. The refinement epsilon, ε, is gradually lowered during the solution. Links with an error larger than ε are refined.

4.5.3 Transfer and Error Estimation

To estimate the transfer vector, **m**, we generate a fixed number of sample points on both the source and the receiver and apply Equation (4.19):

$$\bar{\mathbf{m}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{m}_i = \frac{1}{n}\sum_{i=1}^{n}\frac{v_i\delta(\mathbf{S}_R,-\mathbf{r}_i)(\mathbf{r}_i^{\mathrm{T}}\mathbf{S}_S)_+}{\pi r_i^2} \tag{4.23}$$

where v_i, $\mathbf{r}_i$, and r_i is the visibility, direction, and distance between sample point i of n on the source S and the sample point i on the receiver R.

We use the L_1 norm to measure error as described in [Will00]. This corresponds to the BFA-weighted refinement commonly used in hierarchical radiosity.

4.5.4 Visibility

The visibility term in Equation (4.23) is evaluated by ray casting, and a natural choice for acceleration data structure is to reuse the volume cluster and face cluster hierarchies. BVhierarchies are also generally more efficient for visibility testing than grid-based data structures according to [Smith98].

Our volume cluster hierarchy stores an axis aligned bounding box for each cluster, and has already been proven efficient for visibility testing [Müll99]. When the ray caster reaches a leaf in the volume hierarchy, it continues down the corresponding face cluster hierarchy. Since the face clustering algorithm ensures that the face clusters are both planar and well-shaped, the oriented bounding box for each cluster provides a very tight fit to the geometry.

To further increase the ray casting performance we use the depth-first order array traversal technique described in [Smith98].

4.5.5 Reconstruction

To reconstruct the irradiance for each vertex before rendering, we typically use the irradiance-interpolation-everywhere approach suggested by [Will00]. All links pointing to volume clusters are temporarily pushed down to the root face cluster level. Thereafter we reevaluate the irradiance vectors for each element at the corners of the element and interpolate the result over the element. The irradiance vectors at any given element are calculated by interpolating the irradiance vectors of its parent and adding in the resampled irradiance for the links pointing directly to the element.

This procedure is reasonably fast and can be used for progressive updates of the solution during the radiosity calculation. When the calculation is completed, we provide the user with the option to do a more costly reconstruction where the links are pushed all the way to the vertices of the leaf elements. This is often referred to as a *final gather*. Depending on the complexity of the model, this can be a time consuming process, but it also generates a very smooth solution and depending on the use case, it might be worth the cost. For a further discussion on reconstruction methods, see [Gibs95] and [Will00].

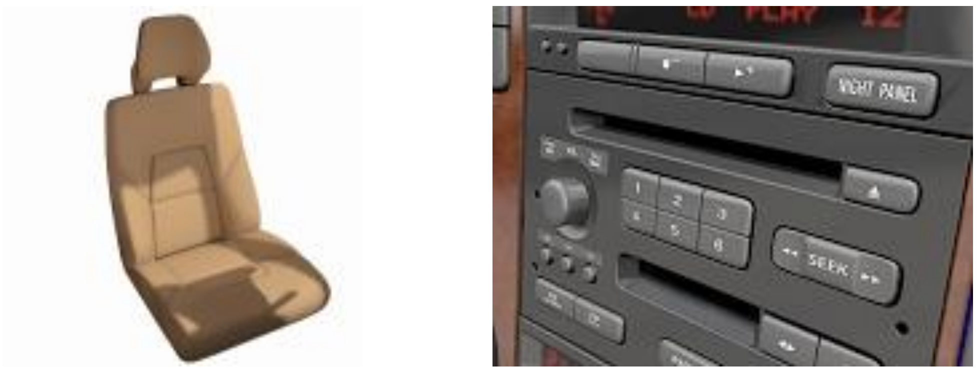
5 Radiosity Representation

5.1 Overview

A radiosity calculation using the algorithm described in the previous chapter results in a mesh that can be rendered directly. However, if the mesh is highly subdivided, e.g., around sharp shadow boundaries, rendering performance can slow down considerably. Preferably, we would like the radiosity mesh to have about the same rendering performance as the original mesh had.

To achieve higher rendering performance, we use textures to represent the illumination in highly subdivided areas to avoid drawing the extra geometry introduced by the radiosity calculation. However, since we work with highly tessellated models, the hierarchical radiosity solution has often not reached the input polygon level. These areas can therefore be represented using vertex colors without affecting performance. It is often preferable to use vertex coloring over textures in such cases in order to save texture memory. Also, when the mesh is highly tessellated, high resolution textures are often needed to represent the shading due to the curvature of the mesh. In areas where the resolution of the texture is lower than the resolution of the mesh, a texture based representation will generally decrease the visual quality.

Our solution is to use vertex lighting for surfaces that are not refined below the input polygon level (see Figure 5.1 b). For surfaces that are more refined, light map textures are used (see Figure 5.1 a). This approach is similar to the one used in [Mysz94].



(a) (b)

Figure 5.1. Radiosity representation. Light maps are used in highly subdivided areas, such as the shadowed areas of the seat in (a). For small details not refined below the input triangles, such as the buttons on the radio panel in (b), vertex lighting is often preferred.

We have chosen to avoid the use of recent programmable graphics hardware in our solution, mainly because we want wide hardware support[6]. However, this is a very interesting area of

[6] When this work was conducted, the latest graphics chip was the NVIDIA NV25 (GeForce4/Quadro4). Systems based on the Silicon Graphics Onyx2 and Onyx 3000 series were still commonly used for high-end, large-scale presentations. Our aim was to find a solution that also worked well on these systems.

research and it makes new radiosity rendering approaches possible. In [Gobb02], a face cluster radiosity solution is rendered directly from the irradiance vectors using extensions to OpenGL. In [Gobb03], this work is extended to include glossy reflections by exploiting more advanced programmable hardware. Investigating such solutions is outside the scope of this thesis.

5.2 Texture Atlas

5.2.1 Motivation and Overview

A texture atlas is an efficient texture representation for general models that allows large connected pieces of geometry to be adjacent in texture space [Lévy02]. More simple methods, like the “polypack” approach proposed by Zhukov *et al*. [Zhuk97], work quite well for architectural scenes but usually generate too many discontinuities in the parameterization of curved geometry. We have therefore chosen a texture atlas approach for our light maps.

The generation of a texture atlas consists of three steps:

- *Segmentation*. The model is partitioned into a set of *charts*, homeomorphic to discs.
- *Parameterization*. Each chart is “unfolded”, i.e., flattened from $\mathbf{R}^3$ to $\mathbf{R}^2$.
- *Packing*. The parameterized charts are packed in texture space.

We use the texture atlas generation pipeline shown in Figure 5.1 for each surface that we want to create an atlas for. A similar approach has been used by [Ray03a].

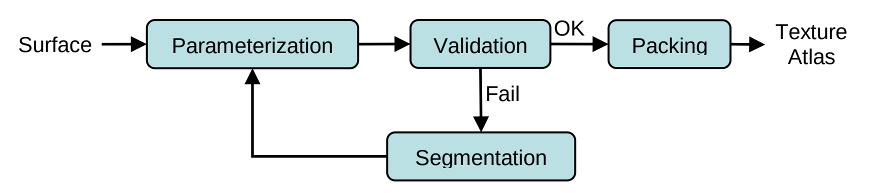


Figure 5.1. Texture atlas generation pipeline.

This means that we first try to parameterize the entire surface. If the resulting parameterization is invalid, e.g., contains overlaps or high area distortions (see [Lévy02]), we subdivide the surface and recursively apply the same procedure on these parts until a valid parameterization is found. The remainder of this section describes the most important steps of our approach.

5.2.2 Parameterization

To parameterize the charts we use the *least squares conformal maps* (LSCMs) approach by Lévy *et al*. [Lévy02]. The LSCM method is both robust and efficient for large, complex models. It uses a *conformal energy criterion* to minimize angle deformations and non-uniform scalings. Ray *et al*. have successfully applied this parameterization method to generate radiosity texture atlases for industrial CAD models [Ray03a].

5.2.3 Segmentation

Since each surface corresponds to a root face cluster, we can directly use our face cluster hierarchy during the segmentation step. If the parameterization of a root face cluster fails, we recursively try to parameterize its children clusters until a valid parameterization is found. The cost function used during the hierarchy creation (see Section 4.2.2) strives to create clusters that are both planar and topologically as close to discs as possible. These are also properties we want the charts to have to make them easy to parameterize.

The segmentation approach based on differential geometry and Morse theory used in the original LSCMs paper did not work well with industrial models [Ray03a]. Ray *et al*. used a simple brute force algorithm to overcome this problem. Our new segmentation method instead uses knowledge of the geometric properties of the mesh, acquired during the face clustering step, to ensure that the charts are well shaped and easy to parameterize.

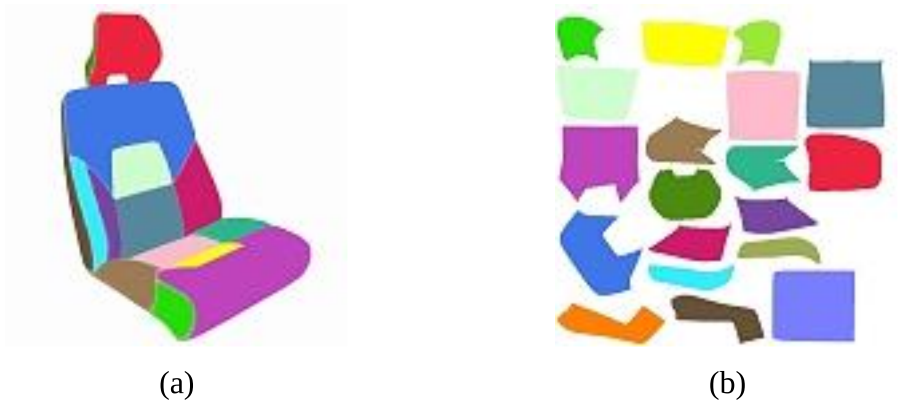


Figure 5.2. Texture atlas. (a) Charts on the model. (b) Corresponding texture atlas. This example contains 18 disconnected surfaces, i.e., face cluster root nodes, and only the surface representing the headrest requires more than one chart.

5.2.4 Packing

Once all charts have been parameterized, they are packed in texture space (see Figure 5.2). It is desirable to minimize the unused texture space in order to save texture memory. Another advantage of packing many charts into a single texture is that it decreases the number of texture switches during rendering.

The packing problem[7] is known to be NP-complete [Mile99]. We want to find a non-overlapping placement of the charts in such a way that the enclosing rectangle is minimized. Approaches based on computational geometry are usually not efficient enough for complex data sets. Instead, several heuristics have been proposed [Zhuk98, Sand01, Lévy02]. We have

[7] The general problem is often referred to as the *bin packing* or *pants packing* problem.

chosen a similar approach to [Sand01] based on packing the bounding rectangles of the charts. More efficient packing of charts with complex borders can be achieved by using the method proposed by [Lévy02]. However, our charts are usually more regularly shaped because of the face cluster construction metric[8].

5.2.5 Light Map Creation

When the texture atlas has been created, we fill the textures with radiosity information. This is done by rendering the radiosity mesh using its uv-coordinates as vertex coordinates to an offscreen buffer. The rasterized pixels are then read back and stored in the texture. Note that the original input triangles of the model are used during parameterization and the refined element mesh during the radiosity texture generation. The uv-coordinates for the vertices introduced by the radiosity calculation are linearly interpolated from the parameterized input triangles.

5.2.6 Avoiding Mip-Mapping Artifacts

In order to avoid aliasing effects when rendering high resolution textures, mip-mapping is used[9] (see e.g., [Möll99]). Schemes that pack multiple charts into a single texture image may give rise to mip-mapping artifacts since coarser mip-map levels will average together spatially disjoint charts. To overcome this problem, we first extrapolate the boundaries of the charts one or a few texels depending on the texture resolution, and then apply a pull-push algorithm [Gort96] to fill in the empty space in the texture (see Figure 5.3).

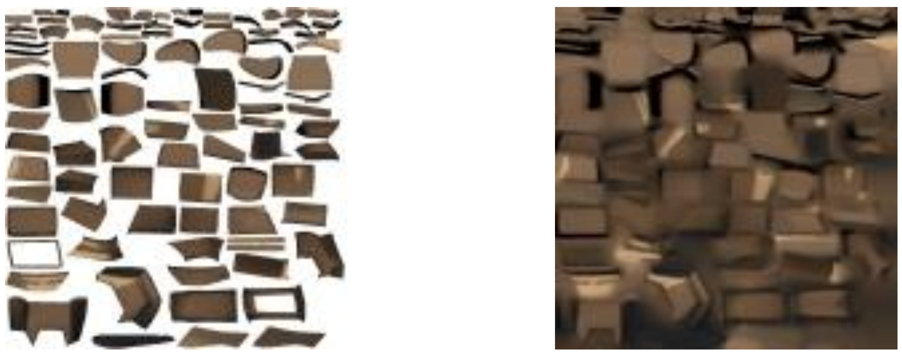


Figure 5.3. Pull-push procedure to fill in the empty space in the texture atlas. (a) A texture atlas before applying the pull-push. (b) After pull-push.

[8] The chart merging method used by Sander *et al*. [Sand01] is also based on [Garl01] and is similar to the face cluster hierarchy creation.
[9] For low resolution light maps, like those in Figure 3.5 or in [Zhuk97], mip-mapping is usually not necessary.

5.3 Selecting Representation

A single representation method is chosen for each surface. This avoids discontinuities between Gouraud-shaded regions and textured regions. Alternatively, both methods could be used for a single surface by using a border for the textured regions and alpha blend it with the underlying Gouraud-shaded mesh to make the transition seamless. We have not implemented this approach yet.

For visual updates during the radiosity calculation it is usually preferable to use a vertex representation to show the full resolution result. When the calculation is completed, highly subdivided surfaces can be replaced by texture representations. However, light maps can decrease the visual quality and sometimes vertex lighting is desirable even if it leads to worse rendering performance. Also, if not enough texture units are available on the particular platform, a light mapped surface might need an extra rendering pass. This will in some cases defeat the performance gain of using light maps. We therefore first use a simple heuristic to select an initial representation for each surface and then allow the user to change it if necessary. A uniform resolution is used, distributing the available texture memory evenly among the light mapped surfaces.

To select an initial representation, we sort the surfaces by the number of introduced triangles during the radiosity calculation in decreasing order. We then traverse the surfaces, converting them to textures, until the total number of introduced triangles falls below a user defined threshold.

6 Implementation Notes

6.1 Overview

The algorithms described in the previous chapters have been implemented in a radiosity module that can be easily integrated with existing applications. Both the Opticore Opus Studio and Opus Realizer software have successfully used this module for global illumination calculations.

All the code is written in C++ and designed in a modular fashion to be portable and reusable. Since Opticore's applications are based on the OpenGL Optimzier/Cosmo3D framework from Silicon Graphics [Open98, Ecke98], we use Cosmo3D for scene graph management and rendering.

An overview of our system is shown in Figure 6.1. The application communicates with a *radiosity engine* that manages all the radiosity functionality. The most important parts are:

- *The preprocessor*. Takes a scene graph as input and creates a hierarchical element representation of the scene.
- *The hierarchical solver*. Iteratively solves for the radiosity in the element hierarchy.
- *The visualization manager*. Updates the scene graph with radiosity information from the element hierarchy.

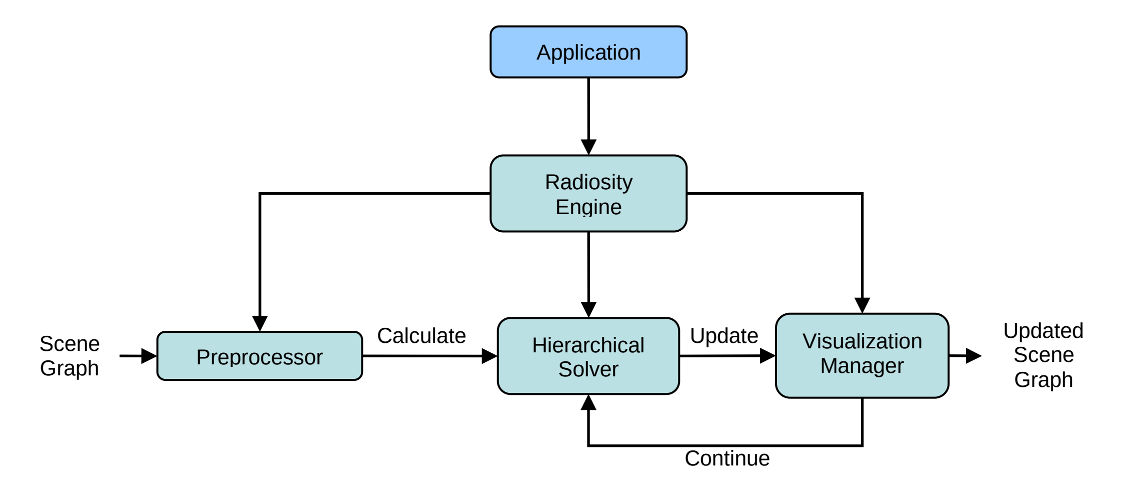


Figure 6.1. System architecture overview.

This chapter briefly discusses how we have implemented various parts of this system.

6.2 Creating the Element Hierarchy

6.2.1 Clustering

First we build a face cluster hierarchy for each object in the scene graph. Some implementation details worth pointing out during the hierarchy creation are:

- For the dual edge collapses (see Section 4.2.2), we use a heap-based priority queue extended to support $O(n \log n)$ updates of sort keys and deletion of elements.
- When calculating the oriented bounding box for relatively planar clusters, the PCA approach becomes somewhat unstable. For those clusters we use the sum area normal as the principle axis, project all vertices onto the plane defined by this axis, and then calculate the minimum-area enclosing rectangle (MER) in this plane to fix the other two axes. We calculate the MER by constructing the convex hull using Graham's scan [ORou98] ($O(n \log n)$) and then finding the rectangle using a rotating calipers algorithm [Tous83] ($O(n)$).

When the hierarchy creation is complete we spilt each hierarchy that contains more than one root, and create a *surface* structure for each root. This surface structure is central in our implementation. Each surface has a single rooted face cluster hierarchy representing a connected group of triangles that also share material properties. It has its own patch data structure representing refined triangles. The output representation method is also selected on a per-surface basis. The surfaces are volume clustered using the algorithm described in section 4.3.2 to form a single rooted hierarchy of the entire scene.

6.2.2 Hierarchical Elements

Hierarchical elements are initially created for all volume clusters and for the root face clusters. We let the elements inherit from an abstract base class that provides the basic algorithm interface (see Figure 6.2.). Many methods are common for all elements and thus implemented in the base class. Generic versions of the refine, gather, and push-pull operations are also provided by the base class.

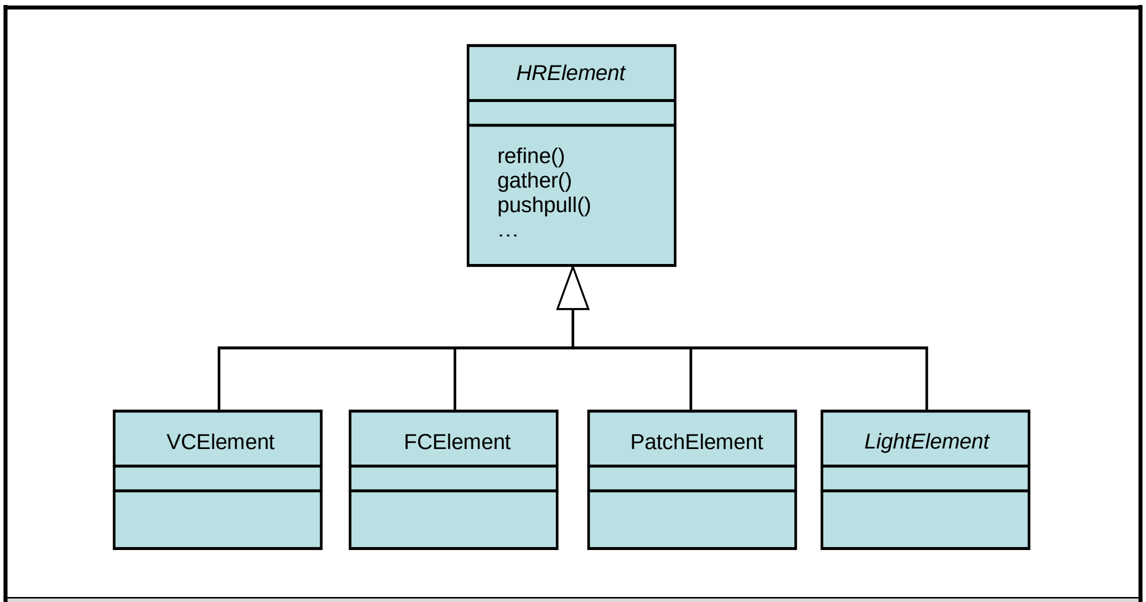


Figure 6.2. UML class diagram for the hierarchical elements. *HRElement* is the common abstract base class for hierarchical elements. *VCElement* and *FCElement* represent volume clusters and face clusters respectively. *LightElement* is an abstract base class for custom light sources not defined by geometry, e.g., point lights, spot lights, directional lights, or sky light.

This framework can easily be extended to include new types of elements. The radiosity solver works only on abstract *HRElements* and needs no knowledge of the implementation of the different element types. Traversal of the hierarchy is handled by the elements themselves; each element type knows how to traverse its children.

6.2.3 Leaf Mesh Data Structure

When patches are refined, the minimum-area enclosing rectangle is calculated to help determine the splitting line (see Section 4.4.2). Again, we use a rotating caliper method for this as described in Section 6.2.1. In order to maintain connectivity information during patch refinement, a half-edge data structure is used for the leaves of the hierarchy. Each time a leaf patch element is refined, the half-edge mesh is updated. This also allows us to quickly remove t-vertices and render a triangulated mesh representing the radiosity solution at any time during the calculation. Initially we used the classic winged-edge data structure for this purpose (see e.g., [Glas91]), but later found the half-edge data structure easier to work with [Mänt88].

We have implemented a general half-edge based data structure similar to those used in OpenMesh [Bots02] and CGAL [Fabr00]. This allows us to easily reuse it for other purposes, e.g., in the parameterization step. We also had in mind the possibility of using a post-process simplification step of the mesh when choosing this representation.

6.3 Radiosity Calculation

6.3.1 Hierarchical Solver

The solver simply calls the refine, gather, and push-pull methods of the root element node, and each element recursively calls its children. The solver can be temporarily halted by the radiosity engine to allow updates of the scene graph in order to display the current state of the solution.

6.3.2 Visibility

Since the visibility sampling is the most costly step of the algorithm, we try to avoid some unnecessary calculations. If a light transfer between two planar, or nearly planar, elements has a sampled visibility of 1.0, we try to accurately determine if it is completely visible. This is done by forming a shaft between the two elements [Hain00]. If no geometry intersects the shaft, we can flag the transfer as completely visible. Further refinements between the elements can then skip the visibility sampling step.

This approach can only be used for planar elements, otherwise important self-shadows can be missed. Patch elements are inherently planar, but we also use it for face cluster elements that are nearly planar using a small epsilon value. In order to avoid counting the elements themselves or geometry behind the elements as occluders, we cap the shaft with the planes of the elements (see Figure 6.3).

Applying a similar scheme to determine if two elements are fully occluded from each other is usually much more complicated, especially for scenes with concave surfaces. Generally, convex occluders are required to conservatively determine full occlusion (see [Lebl00]). We have used the method presented in [Lebl00], but since our geometry often is highly tessellated only very few transfers can benefit from this approach.

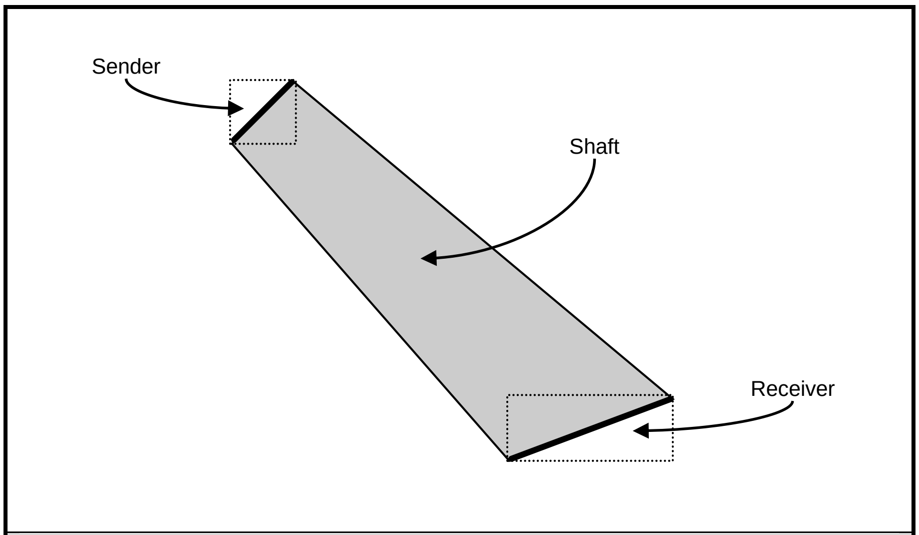


Figure 6.3. Shaft culling to accurately determine complete visibility. This figure shows a simplified 2D-version of the shaft to illustrate the principle. In practice the shaft has a volume that encloses the axis-aligned bounding boxes of the elements (see [Hain00] for further details).

For all shaft intersection tests, we use the same bounding volume hierarchy as we do for ray casting to quickly process the entire scene.

6.4 Scene Graph Updates

6.4.1 Calculating Vertex Colors

At any time during the solution phase the scene graph can be updated to show the current state of the solution. This task is handled by the visualization manager that first pushes irradiance down to the vertices of the mesh. For leaf elements at face cluster level, the irradiance is interpolated from the corners of the clusters. The irradiance is then converted to radiosity yielding the final radiosity mesh.

Before we can render the results of the calculation, the radiosity values stored at the vertices have to be mapped to displayable colors. To achieve this, we use a tone mapper similar to the one described in [Rein02]. This gives us a standard RGB color for each vertex. For surfaces using vertex lighting the vertices are simply copied to the geometry nodes in the scene graph.

6.4.2 Light Maps

For surfaces using light maps, we first generate a parameterization using the input triangles. This parameterization is stored in the surface to avoid recalculating it for future updates. We then render the radiosity mesh to a texture as described in Section 5.2.5. The scene graph is finally updated to use the generated radiosity light map and to draw the original vertices.

6.5 Rendering

6.5.1 Adding Specular Lighting

Since the radiosity calculation only generates diffuse lighting, it is often desirable to add specular lighting. We do this using the standard specular lighting in OpenGL. This will obviously not result in a physically correct solution, but it usually gives a much more realistic result than excluding specular lighting completely.

To render the radiosity solution plus specular lighting in a single rendering pass using only standard OpenGL 1.1[10] we use the following setup:

- Diffuse and ambient material is set to black color. Thus, the colors calculated in the lighting equation only represent specular lighting.
- Emissive lighting is set to the color values stored per vertex[11]. This way our diffuse radiosity lighting is automatically added to the specular color calculated by OpenGL.
- For surfaces using light maps, the emissive material is set to black and the texture blending mode to ADD. This gives us identical lighting equations for both vertex lighting and light maps.
- When combining vertex lighting and light maps, we use a black 1 x 1 light map for the surfaces using vertex lighting and black vertex colors for the light-mapped surfaces.

We have chosen this approach to avoid changing the structure of our Cosmo3D scene graph. Cosmo3D sets the rendering state on a per-shape level (see [Ecke98]) and we might have several surfaces using different representations under a shape node. By using black textures for vertex lit surfaces, black color for light mapped surfaces, and additive texture blending we can use the same OpenGL state for multiple surfaces using different representations. Otherwise, we would have to restructure the scene graph to have one shape node for the vertex lit surfaces and one for the texture mapped surfaces.

Much more sophisticated solutions are possible using more recent programmable hardware extensions.

6.5.2 Textures and Environment Mapping

The procedure described above can be seen as a replacement for the standard OpenGL lighting. Other effects, such as regular textures or environment maps, can be added as usual. Depending on the number of available texture units on the particular hardware, an extra rendering pass may be necessary for those surfaces that use light maps.

Figure 6.4 shows an example OpenGL setup for a light-mapped surface using both a base texture and an environment map. If three texture units are available, this can be rendered in a single pass. Otherwise, multiple passes and frame buffer blending is necessary.

[10] We use the texture environment mode ADD, which is not part of the OpenGL 1.1 specification. However, it is commonly available and since it is used in the Cosmo3D API it is supported on all our target platforms.
[11] This is achieved using the call `glColorMaterial(GL_FRONT, GL_EMISSION)`.

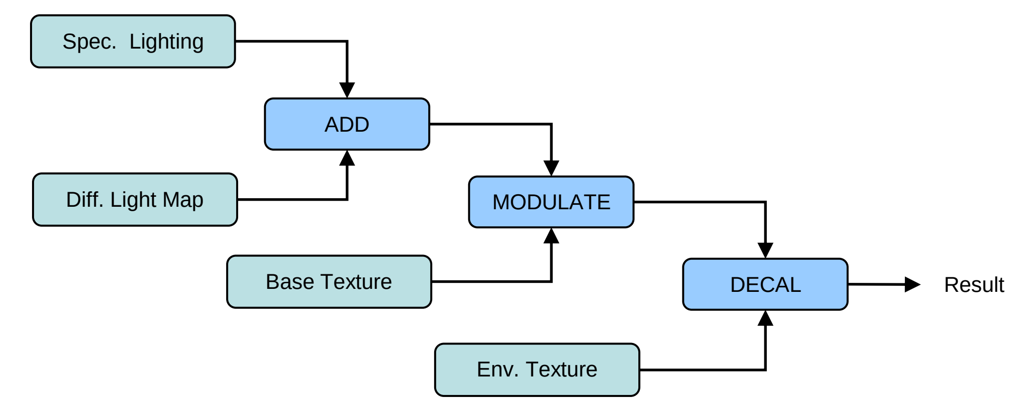


Figure 6.4. Example OpenGL rendering setup. The specular lighting calculated by OpenGL is first added with our radiosity light map. A base texture is then modulated with the result and finally an environment map is added using the decal texture mode.

7 Results

7.1 Overview

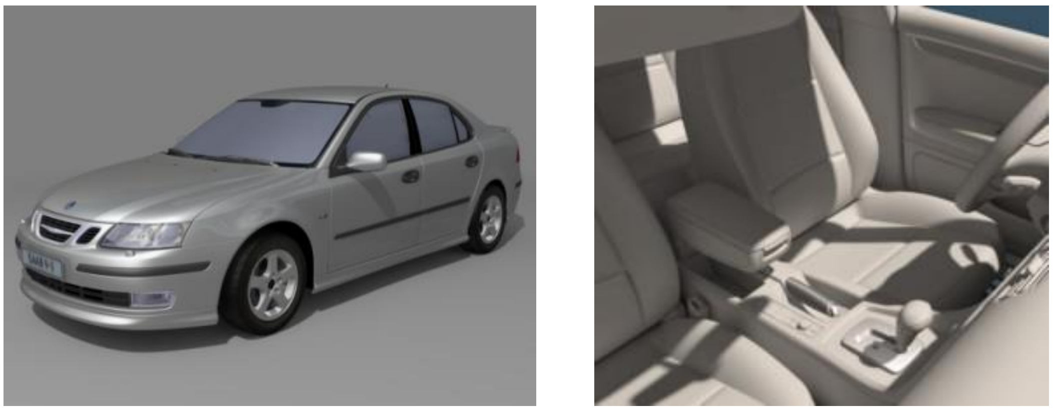
In this chapter we present some results of our work. First, the performance of the algorithm is analyzed in terms of time and memory consumption. Then, some examples of the visual quality will be shown.

7.2 Test System

All performance tests were performed on an AMD Athlon 1.4 GHz, running Windows 2000. The computer was equipped with 1 GB of RAM and an Nvidia GeForce3 graphics card.

7.3 Test Scenes

Two scenes have been selected to test our radiosity system (see Figure 7.1). The models are from the automotive industry and representative for the type of data used in Opticore's applications. This will demonstrate how the algorithms perform under the circumstances they were designed for.



(a) Car exterior (b) Car interior

Figure 7.1. Test scenes.

7.4 Preprocessing

The first step in our algorithm is the preprocessing that prepares the scene for radiosity calculations. The main parts of the preprocessing are the face clustering, the volume clustering, and the creation of the mesh data structure. Table 7.1 shows the total preprocessing time and memory consumption for the test scenes.

Scene	**Polygons**	**Time**	**Memory**
Car exterior	242 764	15.23 s	110.7 MB
Car interior	521 699	24.66 s	190.5 MB

Table 7.1. Preprocessing time and memory consumptions for the test scenes.

To show the contribution of the different stages to the total preprocessing time, we have profiled the face clustering, volume clustering, and mesh creation stages. These are the most interesting algorithmic parts of the preprocessing. Table 7.2 shows the results.

Scene	**Stage**	**Time**	**%**
Car exterior	Face clustering	10.40 s	68.3 %
	Volume clustering	0.05 s	0.3 %
	Mesh creation	3.18 s	20.9 %
	Other	1.60 s	10.5 %
	Total	**15.23 s**	**100 %**
Car interior	Face clustering	16.71 s	67.8 %
	Volume clustering	0.21 s	0.9 %
	Mesh creation	5.83 s	23.6 %
	Other	1.91 s	7.7 %
	Total	**24.66 s**	**100 %**

Table 7.2. Breakdown of preprocessing time for the two test scenes.

7.5 Radiosity Calculation

Since the algorithm is based on a progressive hierarchical approach, the calculation time is determined by the desired accuracy and visual quality. The following figures illustrate the time and memory required in order to achieve a level of visual quality sufficient for typical use cases.

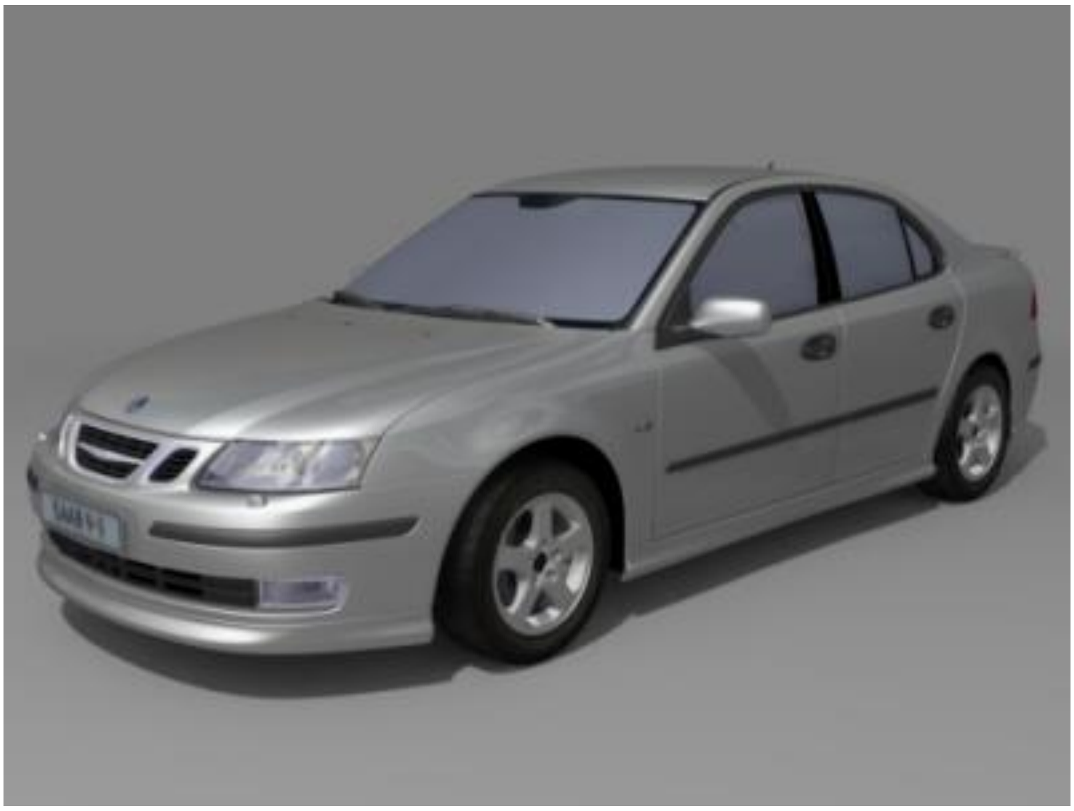


Figure 7.2. Car exterior example (calculated in 25 minutes using 411 MB of peak memory).

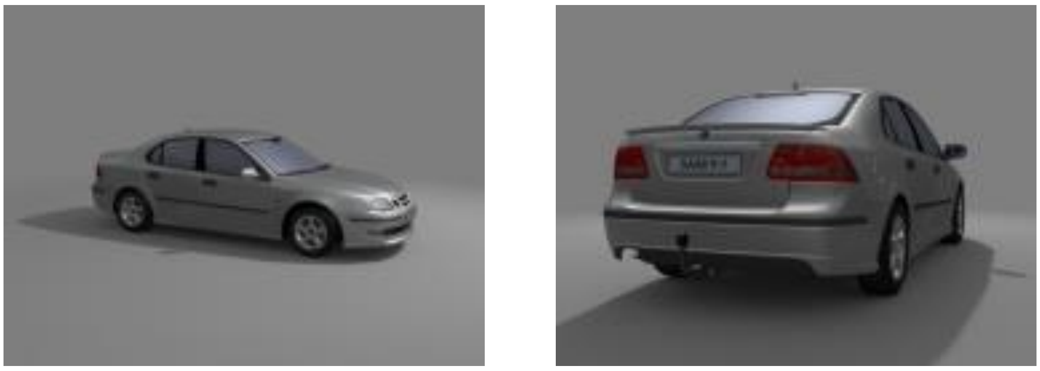


Figure 7.3. The same scene as in Figure 7.2 from different angles. Notice how the shadow gets darker under the car, because less indirect lighting reaches those areas.

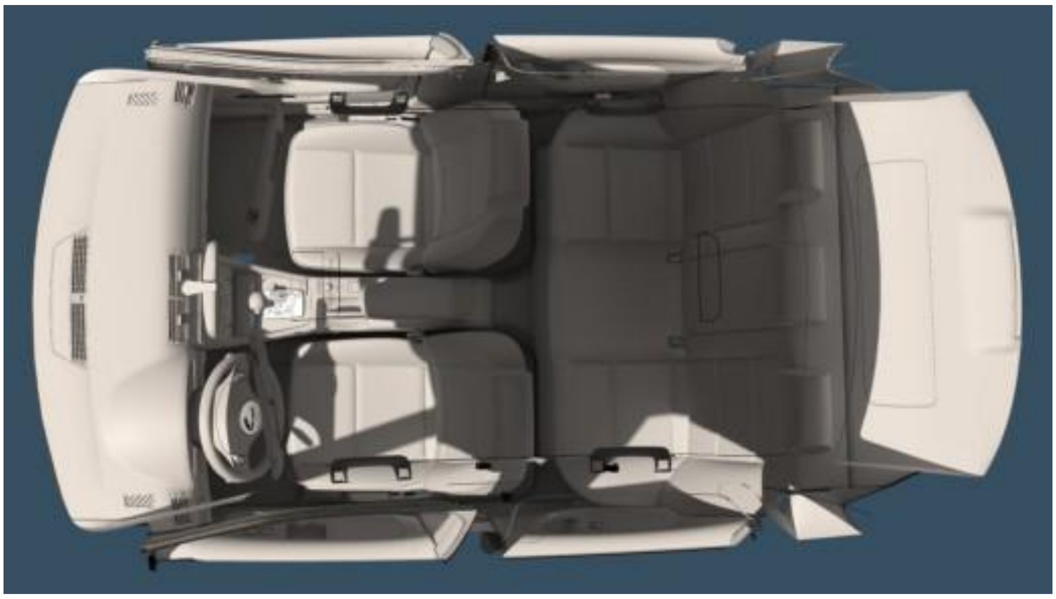


Figure 7.4. Car interior example (calculated in 4 hours 30 minutes using 762 MB of peak memory).

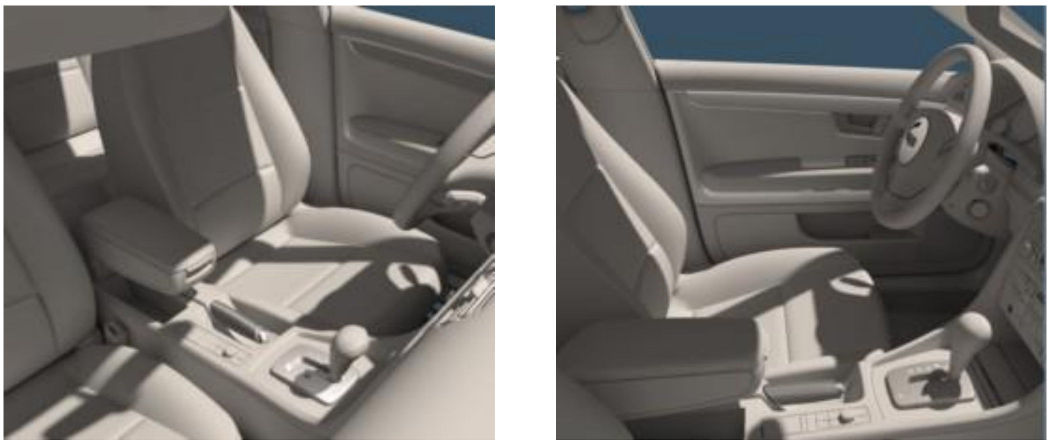


Figure 7.5. The same scene as in Figure 7.4 from different angles.

7.6 Rendering

Rendering performance is related to the number of extra triangles introduced during the calculation. As long as no extra triangles are created, the performance will generally not be affected. Often, light maps can be used for most surfaces refined below the input triangle level and hence the rendering performance will be nearly the same as before the radiosity calculation. Sometimes it might be desirable to use vertex lighting for some refined surfaces, e.g., when the light map resolution is inadequate. In those cases a tradeoff between visual quality and rendering performance has to be made. In the examples shown in this chapter, the car exterior has a rendering overhead of about 10 % compared to the input model, whereas the interior uses more vertices and has an overhead of about 35 %.

8 Conclusions

In this thesis we have presented a radiosity system for global illumination simulations of highly tessellated models like those found in computer aided design (CAD). To manage the complexity of such models we use a hierarchical algorithm based on the face cluster radiosity approach developed by Willmott *et al*. We have shown that the face clustering approach is very suitable for this type of data.

Previous work on face cluster radiosity has focused on scenes with a small number of very complex surfaces [Will00, Gobb03]. In our case we have to deal with a far larger number of disconnected surfaces, increasing the need for a good hierarchy above surface level. We use a relatively simple bounding volume hierarchy for this purpose. It performs reasonably well in our experience, especially for ray casting acceleration. However, it does not exploit the orientation of the clustered elements like the face cluster hierarchy do. This makes the approximation of light transfers between volume clusters worse than between face clusters. In most cases, this is not a problem since the most important light transfers are refined below the volume clusters.

When dealing with poorly shaped triangles, our novel patch refinement scheme can yield significant improvements over the regular refinement method commonly used in hierarchical radiosity. We have also experimented with a few grid-based refinement approaches, but unsatisfied with the results we developed our own method instead.

We have shown that our radiosity system can handle scenes with several hundred thousand polygons relatively well. The radiosity solution is often a huge improvement in visual quality and realism compared to the standard local lighting models commonly used in real-time computer graphics.

In order to display the radiosity solutions in real-time, we use a light map representation for highly subdivided areas. This makes it possible to eliminate most of the extra geometry generated during the calculations. The parameterization scheme has shown good results and light mapped areas are often indistinguishable from highly subdivided vertex lit meshes.

One of the main advantages with our system is its user-friendliness. The preprocessing needed before the calculations can begin is very quick and the initial results can be displayed very fast. Even for large models, the user can often get a good idea of how the final result will look like in under a minute. This allows the user to quickly determine if the setup is correct or if the light settings or other properties need to be changed. Also, during the calculation, the user can quickly get feedback on how the solution progresses.

Many improvements of the algorithms are possible and some of these will be discussed in the next chapter.

9 Future Work

9.1 Overview

In this chapter we discuss some future improvements of our work. We have selected a few areas that we find particularly interesting and important to investigate further.

9.2 Improving the Accuracy of the Algorithm

9.2.1 Discontinuity Meshing

Our current patch refinement approach tries to create regularly shaped elements. This generally works well, but hard shadow boundaries can still be difficult to capture with sufficient accuracy. Choosing the splitting lines based on discontinuities in the illumination could lead to improvements in both efficiency and quality in areas where sharp shadow boundaries are present. As several authors have pointed out though, discontinuity meshing can be difficult to apply to highly complex scenes, both in terms of robustness and efficiency.

9.2.2 Improved Refinement Oracle

For refinement decisions, we have chosen the commonly used BFA approach that measures the error using area-weighted irradiance. This error does not always correspond to the visual error in the final solution though, when irradiance has been mapped to display color. In particular, the importance of relatively weak indirect lighting is often underestimated. A perceptually-based refinement method could lead to more efficient refinement decisions.

Another problem is the difficulty to estimate the error due to visibility variation. When using sampling for visibility determination, it is always possible to miss partially occluded transfers. This has to be taken into account by the error criteria. For transfers that are guaranteed to be completely visible, a higher refinement threshold could be used. Likewise, transfers that are guaranteed to be occluded could be eliminated completely. Thus, more efficient ways of determining exact visibility could greatly improve the refinement process.

9.2.3 Volume Hierarchy Construction

We would like to improve the cost function for the volume hierarchy creation and take the orientation of the clustered elements into account. This would allow us to more accurately approximate the transfers between volume clusters. Also, since the number of clustered elements is relatively low in this case, we can afford a slightly more costly construction approach.

9.3 Improving Visualization

9.3.1 Parameterization

The hierarchical extension to the LSCM method could be used to speed up the parameterization [Ray03b]. Also, as noted in [Ray03a], the stability of the parameterization could be improved by removing badly shaped triangles from the conformal energy. This would reduce the number of necessary charts and also lessen distortion. Several other improvements are also possible, e.g., selecting the light map resolution based on the variation in the illumination.

9.3.2 Programmable Graphics Hardware

It would be interesting to take advantage of recent programmable graphics hardware, as in [Gobb03]. This would also allow us to incorporate specular lighting and other effects in a more correct way.

9.3.3 Simplification and Level-of-Detail

When texture memory is limited or vertex lighting is preferred for other reasons, it would be beneficial to include some kind of simplification post-process or level-of-detail scheme. As mentioned in Chapter 3, radiosity meshes have been shown to be excellent candidates for simplification algorithms [Hopp96].

10 Bibliography

[Appe85] Andrew W. Appel. An Efficient Program for Many-body Simulation. In *SIAM Journal on Scientific and Statistical Computing*, 6(1):85-103. January 1985.

[Ashd94] Ian Ashdown. *Radiosity: A Programmers Perspective*. John Wiley & Sons, New York, NY, 1994.

[Arvo86] James Arvo. Backward Ray Tracing. In *Developments in Ray Tracing*, ACM SIGGRAPH '86 Course Notes, Volume 12, August 1986.

[Arvo94] James Arvo and Andrew Glassner. The Irradiance Jacobian for Partially Occluded Polyhedral Sources. In *Proceedings of SIGGRAPH '94*, pages 343-350, July 1994.

[Bast96] Rui Bastos, Michael Goslin, and Hansong Zhang. *Efficient Radiosity Rendering using Textures and Bicubic Reconstruction*. Technical Report TR96-025, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, May 1996.

[Baum98] Dan Baum. 3D Graphics Hardware: Where We Have Been, Where We Are Now, and Where We Are Going. In *SIGGRAPH Computer Graphics Newsletter*, Vol. 32 No. 1, February 1998.

[Bera91] Jeffrey C. Beran-Koehn and Mark J. Pavicic. A Cubic Tetrahedral Adaptation of the Hemi-Cube Algorithm. In *Graphics Gems II*, pages 299-302, Academic Press Professional, Boston, MA, 1991.

[Bots02] Mario Botsch, Stephan Steinberg, Stephan Bischoff, and Leif Kobbelt. OpenMesh – A Generic and Efficient Polygon Mesh Data Structure. In *OpenSG Symposium 2002*, 2002.

[Camp91] Alvin T. Campbell, III. *Modeling Global Diffuse Illumination for Image Synthesis*. PhD thesis, Dept. of Computer Sciences, University of Texas at Austin, December 1991.

[Cohe85] Michael F. Cohen and Donald P. Greenberg. The Hemi-cube: A Radiosity Solution for Complex Environments. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, **19**:3, pages 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. In *IEEE Computer Graphics and 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. In *Computer Graphics (SIGGRAPH '88 Proceedings),* **22**:4, pages 75-84, August 1988.

[Cohe93] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press International, Boston, MA, 1993.

[Cook86] Robert L. Cook. Stochastic Sampling in Computer Graphics. In *ACM Transactions on Graphics*, Vol. 5, No. 1, January 1986.

[Dura99] Frédo Durand, George Drettakis, and Claude Puech. Fast and Accurate Hierarchical Radiosity Using Global Visibility. In *ACM Transactions on Graphics*, April 1999.

[Ecke98] George Eckel. *Cosmo 3D Programmer’s Guide*. Document Number 007-3445-002. Silicon Graphics, Inc., 1998.

[Fabr00] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. On the Design of CGAL, the Computational Geometry Algorithms Library. In *Software - Practice and Experience 30*, pages 1167-1202, 2000.

[Fole96] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice, 2nd Edition in C*. Addison-Wesley, Reading, Massachusetts, 1996.

[Garl97] Michael Garland and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH '97*, pages 209-216, August 1997.

[Garl99] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, 1999.

[Garl01] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical Face Clustering on Polygonal Surfaces. In *ACM Symposium on Interactive 3D Graphics*, March 2001.

[Gers94] Reid Gershbein, Peter Schröder, and Pat Hanrahan. *Textures and Radiosity: Controlling Emission and Reflection from Texture Maps*. Technical Report CS-TR-449-94, Department of Computer Science, Princeton University, Princeton, NJ, March 1994.

[Gibs95] Simon Gibson. *Efficient Radiosity for Complex Environments*. Master’s thesis, Manchester, UK, 1995.

[Gibs96] Simon Gibson and Roger Hubbold. Efficient Refinement and Clustering for Radiosity in Complex Environments. In *Computer Graphics Forum*, 15(5):297-310, December 1996.

[Gibs97] S. Gibson and R. J. Hubbold. Perceptually-Driven Radiosity. In *Computer Graphics Forum*, 16(2):129-140, 1997.

[Glas89] Andrew S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press Limited. 1989.

[Glas91] Andrew S. Glassner. Maintaining Winged-Edge Models. In *Graphics Gems II,* pages 191-201, Academic Press Professional, Boston, MA, 1991.

[Gobb02] Enrico Gobbetti, Leonardo Spanò, and Marco Agus. *Hierarchical Higher Order Face Cluster Radiosity*. Technical Report CRS4 TR/. CRS4, Center for Advanced Studies, Research, and Development in Sardinia. Cagliari, Italy, March 2002.

[Gobb03] Enrico Gobbetti, Leonardo Spanò, and Marco Agus. Hierarchical Higher Order Face Cluster Radiosity for Global Illumination Walkthroughs of Complex Non-Diffuse Environments. In *Computer Graphics Forum*, 22(3), September 2003.

[Gora84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the Interaction of Light Between Diffuse Surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, **18**:3, pages 212-222, July 1984.

[Gort96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The Lumigraph. In *Proceedings of SIGGRAPH '96*, August 1996.

[Gott96] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A Hierarchical Structure for Rapid Interference Detection. In *Proceedings of SIGGRAPH '96*, pages 171-180, August 1996.

[Gott00] Stefan Gottschalk. *Collision Queries using Oriented Bounding Boxes*. PhD thesis. Department of Computer Science, UNC Chapel Hill, 2000.

[Greg98] Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. The irradiance volume. In *IEEE Computer Graphics & Applications*, (2):32–43, March-April 1998.

[Guth03] Michael Guthe and Reinhard Klein. Automatic Texture Atlas Generation from Trimmed NURBS Models. In *Proceedings of Eurographics 2003*, September 2003.

[Hain00] Eric Haines. A Shaft Culling Tool. In *Journal of Graphics Tools*, 5(1):23-26, 2000.

[Hanr91] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, **25**:4, pages 197-206, July 1991.

[Hase99] Jean-Marc Hasenfratz, Cyrille Damez, François Sillion, and George Drettakis. A Practical Analysis of Clustering Strategies for Hierarchical Radiosity. In *Computer Graphics Forum*, 18(3):221-232, September 1999.

[Heck90] Paul S. Heckbert. Adaptive Radiosity Textures for Bidirectional Ray Tracing. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, **24**:4, pages 145-154, August 1990.

[Heck91] Paul S. Heckbert and James M. Winget. *Finite Element Methods for Global Illumination*. Tech. Rep. UCP/CSD, Computer Science Division (EECS), University of California, Berkeley, July 1991.

[Heck92] Paul S. Heckbert. Discontinuity Meshing for Radiosity. In *Third Eurographics Workshop on Rendering*. Bristol, UK, May 1992.

[Heck93] Paul S. Heckbert. Finite Element Methods for Radiosity. In *Global Illumination Course Notes (SIGGRAPH '93)*, Anaheim, August 1993.

[Hink98] André Hinkenjann and Georg Pietrek. Reconstructing Radiosity by Scattered Data Interpolation. In *Proceedings of WSCG'98 - Central European Conference on Computer Graphics and Visualization '98*, pages 133-140. Plzen, Czech Republic, 1998.

[Hopp96] Hugues Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH '96*, **30**:4, pages 99-108, August 1996.

[Jens01] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A K Peters, Natick, Massachusetts, 2001.

[Joll86] Ian T. Jolliffe. *Principal Component Analysis*. Springer Verlag, New York, 1986.

[Kaji86] James T. Kajiya. The Rendering Equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, **20**:4, pages 143-150, August 1986.

[Lebl00] Luc Leblanc and Pierre Poulin. Guaranteed Occlusion and Visibility in Cluster Hierarchical Radiosity. In *Eurographics Workshop on Rendering 2000*, pages 89-100, June 2000.

[Lévy02] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least Squares Conformal Maps for Automatic Texture Atlas Generation. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, pages 362-371, July 2002.

[Lisc92] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg, Discontinuity Meshing for Accurate Radiosity. In *IEEE Computer Graphics and Applications*, 12(6): 25-39, Nov. 1992.

[Lisc93] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg, Combining Hierarchical Radiosity and Discontinuity Meshing. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 199-209, August 1993.

[Low01] Kok-Lim Low. *Simulated 3D Painting*. Technical report, Department of

Computer Science, University of North Carolina at Chapel Hill, 2001.

[MacD89] J. David MacDonald and Kellogg S. Booth. Heuristics for Ray Tracing using Space Subdivision. In *Proceedings of Graphics Interface ’89*, pages 152-63, Toronto, Ontario, June 1989. Canadian Information Processing Society.

[Mail93] Jerôme Maillot, Hussein Yahia, and Anne Verroust. Interactive Texture Mapping. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 27-34, August 1993.

[Mile99] Victor J. Milenkovic. Rotational Polygon Containment and Minimum Enclosure Using Only Robust 2D Constructions. In *Computational Geometry*, 13(1):3-19, 1999.

[Müll99] Gordon Müller, Stephan Schäfer, and Dieter Fellner. Automatic Creation of Object Hierarchies for Radiosity Clustering. In *Proceedings of Pacific Graphics '99 (Seventh Pacific Conference on Computer Graphics and Applications)*, pages 21-29, IEEE Computer Society Press, Los Alamitos, CA, October 1999.

[Mysz94] Karol Myszkowski and Tosiyasu L. Kunii. Texture Mapping as an Alternative for Meshing During Walkthrough Animation. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 375-388, Germany, June 1994.

[Mänt88] Martti Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, MD, 1988.

[Möll96] Tomas Möller. Radiosity Techniques for Virtual Reality – Faster Reconstruction and Support for Levels Of Detail. In *WSCS ’96*, Plzen, Czech Republic, 1996.

[Möll98] Tomas Möller. *Real-Time Algorithms and Intersection Test Methods for Computer Graphics*. PhD thesis. Department of Computer Engineering, Chalmers University of Technology, Sweden, 1998.

[Möll99] Tomas Möller and Eric Haines. *Real-Time Rendering*. A K Peters, Natick, Massachusetts, 1999.

[Niel00] Kasper Høj Nielsen. *Real-Time Hardware-Based Photorealistic Rendering*. Master's thesis, Department of Mathematical Modelling, IMM, The Technical University of Denmark, Lyngby, Denmark, 2000.

[Nish85] T. Nishita and E. Nakamae. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, **19**:3, pages 23-30, July 1985.

[Open98] *OpenGL Optimizer Programmer’s Guide: An Open API for Large-Model Visualization*. Document Number 007-2852-002. Silicon Graphics, Inc., 1998.

[Opti01] *Opticore Opus Studio Users Guide*. Opticore AB, Sweden, 2001.

[ORou98] Joseph O'Rourke. *Computational Geometry in C (Second Edition)*. Cambridge University Press, 1998.

[Piet93] Georg Pietrek. *Fast Calculation of Accurate Formfactors*. Technical Report No. 483, University of Dortmund, Germany, May 1993.

[Prik99] Jan Prikryl and Werner Purgathofer. *Overview of Perceptually-Driven Radiosity Methods*. Technical Report TR-186-2-99-26, 1999.

[Ray03a] Nicolas Ray, Jean-Christophe Ulysse, Bruno Lévy, and Xavier Cavin. Generation of Radiosity Texture Atlas for Realistic Real-Time Rendering. In *Proceedings of Eurographics 2003*, September 2003.

[Ray03b] Nicolas Ray and Bruno Lévy. Hierarchical Least Squares Conformal Map. In *11th Pacific Conference on Computer Graphics and Applications (PG'03)*, October 2003.

[Rein02] Erik Reinhard, Mike Stark, Peter Shirley, and Jim Ferwerda. Photographic Tone Reproduction for Digital Images. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, pages 267-276, July 2002.

[Roge98] David F. Rogers. *Procedural Elements for Computer Graphics, 2nd Edition*. McGraw-Hill, Boston, MA, 1998.

[Rush93] Holly Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. In *Graphics Interface '93*, pages 227–236, May 1993.

[Sand01] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture Mapping Progressive Meshes. In *Proceedings of SIGGRAPH 2001*, pages 409-416, August 2001.

[Sill89] François X. Sillion and Claude Puech. A General Two-Pass Method Integrating Specular and Diffuse Reflection. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, **23**:3, pages 335-344, August 1989.

[Sill91] François X. Sillion. Detection of Shadow Boundaries for Adaptive Meshing in Radiosity. In *Graphics Gems II*, pages 311-315, Academic Press Professional, Boston, MA, 1991.

[Sill94] François X. Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufman, San Mateo, CA, 1994.

[Sill95a] François X. Sillion, George Drettakis, and Cyril Soler. A Clustering Algorithm for Radiance Calculation in General Environments. In *Rendering*

Techniques '95, pages 196-205, Eurographics, June 1995.

[Sill95b] François X. Sillion. A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters. In *IEEE Transactions on Visualization and Computer Graphics,* 1(3):240–254, September 1995.

[Smit94] Brian Smiths, James Arvo, and Donald Greenberg. A Clustering Algorithm for Radiosity in Complex Environments. In *Proceedings of SIGGRAPH '94*, pages 435-442, July 1994.

[Smith98] Brian Smiths. Efficiency Issues for Ray Tracing. In *Journal of Graphics Tools*, 3(2):1-14, February 1998.

[Span02] Leonardo Spanò and Enrico Gobbetti. Radiosity for Highly Tessellated Models. In *SIMAI 2002 Symposium on Adaptive Techniques in Numerical Simulation and Data Processing*, May 2002.

[Stam00] Mark Stamminger, Annette Scheel, and Hans-Peter Seidel. Hierarchical Radiosity with Global Refinement. In *Vision, Modeling, and Visualization 2000 (Conference Proceedings)*, pp. 263-271, 2000.

[Tous83] Godfried Toussaint. Solving Geometric Problems with the Rotating Calipers. In *Proceedings of IEEE MELECON'83*, Athens, Greece, May 1983.

[Whit80] Turner Whitted. An Improved Illumination Model for Shaded Display. In *Communications of the ACM*, **23**:6, pages 343-349, 1980.

[Wide99] Johnny Widerlund. *Increasing Realism in Real-Time Visual Simulations Using Hardware Accelerated Progressive Radiosity*. Master's thesis, Chalmers University of Technology, Sweden, 1999.

[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.

[Will00] Andrew J. Willmott. *Hierarchical Radiosity with Multiresolution Meshes*. PhD thesis, Carnegie Mellon University, 2000.

[Woo99] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide*, Third Edition. Addison-Wesley, Reading, Massachusetts, 1999.

[Yeap97] Tralvex Yeap. *Radiosity for Virtual Reality Systems*. Master's thesis, University of Leeds, Leeds, United Kingdom, August 1997.

[Zhuk97] S. Zhukov, A. Iones, G. Kronin. On a Practical Use of Light Maps in Real-Time Applications. In *Proceedings of SCCG'97 vol. 2,* pages 7-14. Slovakia, Bratislava, 1997.

[Zhuk98] S. Zhukov, A. Iones, G. Kronin. Using Light Maps to Create Realistic Lighting in Real-Time Applications. In *Proceedings of WSCG'98 - Central European Conference on Computer Graphics and Visualization '98,* pages 464-471. Plzen, Czech Republic, 1998.