Vom Fachbereich für Mathematik und Informatik
der Technischen Universität Braunschweig
genehmigte Dissertation
zur Erlangung des Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

Stephan Schäfer

Efficient Object-Based Hierarchical Radiosity Methods

Tag der mündlichen Prüfung: 30.05.2000

1. Referent: Prof. Dr. Dieter W. Fellner
2. Referent: Prof. Dr. Hans-Peter Seidel
eingereicht am: 10.04.2000

Abstract

The efficient generation of photorealistic images is one of the main subjects in the field of computer graphics. In contrast to simple image generation which is directly supported by standard 3D graphics hardware, photorealistic image synthesis strongly adheres to the physics describing the flow of light in a given environment. By simulating the energy flow in a 3D scene global effects like shadows and inter-reflections can be rendered accurately.

The hierarchical radiosity method is one way of computing the global illumination in a scene. Due to its limitation to purely diffuse surfaces solutions computed by this method are view independent and can be examined in real-time walkthroughs. Additionally, the physically based algorithm makes it well suited for lighting design and architectural visualization.

The focus of this thesis is the application of object-oriented methods to the radiosity problem. By consequently keeping and using object information throughout all stages of the algorithms several contributions to the field of radiosity rendering could be made. By introducing a new meshing scheme, it is shown how curved objects can be treated efficiently by hierarchical radiosity algorithms. Using the same paradigm the radiosity computation can be distributed in a network of computers. A parallel implementation is presented that minimizes communication costs while obtaining an efficient speedup.

Radiosity solutions for very large scenes became possible by the use of clustering algorithms. Groups of objects are combined to clusters to simulate the energy exchange on a higher abstraction level. It is shown how the clustering technique can be improved without loss in image quality by applying the same data-structure for both, the visibility computations and the efficient radiosity simulation.

Zusammenfassung

Eines der Schwerpunktthemen in der Computergraphik ist die effiziente Erzeugung von fotorealistischen Bildern. Im Gegensatz zur einfachen Bilderzeugung, die bereits durch gängige 3D-Grafikhardware unterstützt wird, gehorcht die fotorealistische Bildsynthese physikalischen Gesetzen, die die Lichtausbreitung innerhalb einer bestimmten Umgebung beschreiben. Durch die Simulation der Energieausbreitung in einer dreidimensionalen Szene können globale Effekte wie Schatten und mehrfache Reflektionen wirklichkeitstreu dargestellt werden.

Die hierarchische Radiositymethode (*Hierarchical Radiosity*) ist eine Möglichkeit, um die globale Beleuchtung innerhalb einer Szene zu berechnen. Da diese Methode auf die Verwendung von rein diffus reflektierenden Oberflächen beschränkt ist, sind damit errechnete Lösungen blickwinkelunabhängig und lassen sich in Echtzeit am Bildschirm durchwandern. Zudem ist dieser Algorithmus aufgrund der verwendeten physikalischen Grundlagen sehr gut zur Beleuchtungssimulation und Architekturvisualisierung geeignet.

Den Schwerpunkt dieser Doktorarbeit stellt die Anwendung objektbasierter Methoden auf das Radiosityproblem dar. Durch konsequente Ausnutzung von Objektinformationen während aller Berechnungsschritte konnten verschiedene Verbesserungen im Rahmen der hierarchischen Radiositymethode erzielt werden. Gekrümmte Objekte können aufgrund eines neuen Flächenunterteilungsverfahrens nun effizient durch den hierarchischen Radiosityalgorithmus dargestellt werden. Dieses Verfahren ermöglicht ebenso eine effiziente Parallelisierung des hierarchischen Radiosityalgorithmus. Es wird ein parallele Implementierung vorgestellt, die unter Minimierung der Kommunikationskosten eine effiziente Geschwindigkeitssteigerung erzielt.

Radiosityberechnungen für sehr große Szenen sind nur durch Verwendung sogenannter *Clustering*-Algorithmen möglich. Dabei werden Gruppen von Objekten zu Clustern kombiniert um den Energieaustausch zwischen Oberflächen stellvertretend auf einem höheren Abstraktionsniveau durchzuführen. Durch Verwendung derselben Datenstruktur für Sichtbarkeitsberechnungen und für die Steuerung der Radiositysimulation wird gezeigt, wie das Clusteringverfahren ohne Qualitätsverluste verbessert werden kann.

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I would like to thank my supervisor Dieter Fellner for his encouragement and continuing support during the course of my research. His ideas and helpful comments have been invaluable for this thesis.

I would also like to thank all members of the Graphics Lab in Bonn and in Braunschweig for their contribution to our rendering platform that resembles the foundation for my work. Special thanks to Gordon Müller for his valuable comments on this thesis and for his contribution of the scene structuring algorithm. Discussions with him were always of great value for me. Thanks to Heinzgerd Bendels who implemented the boundary representation data structure and the CSG-code, to Marco Zens for his excellent work on the distributed radiosity algorithm and the scheduling code, and to Frank Büllesfeld for implementing the final gathering, for improving the radiosity code, and for proof reading this thesis. I would also like to acknowledge the *Deutsche Forschungsgemeinschaft* (DFG) for funding this research.

Finally, many thanks go to my parents who always supported me in all aspects of my education and beyond.

CHAPTER 1

Introduction

1.1 Image Synthesis

The computer aided generation of images is still the core aspect of computer graphics research. The application of heuristic methods, approximations, and physically correct simulation methods have led to a variety of rendering algorithms with very different properties. The most important classification of these algorithms is the distinction between local illumination and global illumination. This classification is based on the way how the interaction of light with the environment is simulated in order to compute an image of the given scene.

This chapter gives an overview on digital image synthesis. The rendering pipeline will be introduced to separate and explain all building blocks of a rendering algorithm from reading the scene model upto the final picture. The terms local and global illumination will be discussed and finally two different rendering algorithms will be summarized: ray tracing and radiosity.

1.2 The Rendering Pipeline

Although the lighting calculation is one of the most important steps to actually produce an image it is just one step of a rendering pipeline that fully describes the process of image synthesis [FvDFH90]. Depending on the rendering algorithm described by this pipeline, the building blocks can be arranged in several ways. Figure 1.1 shows an example.

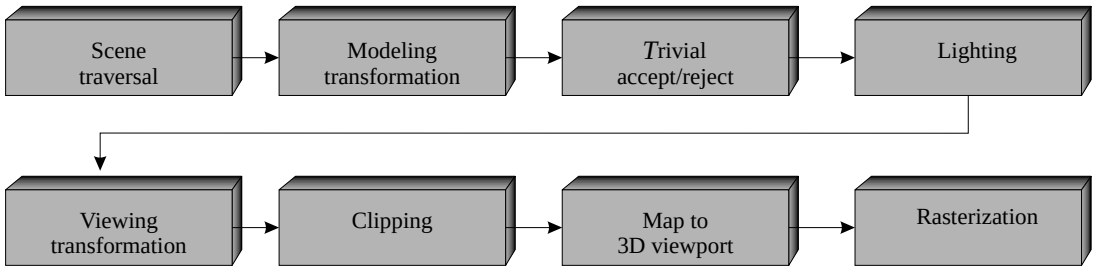

Figure 1.1 Rendering pipeline.

The input to the pipeline is a scene description or scene database that has been generated with a modeling tool. A scene is basically a collection of objects, light sources, and at least

one camera position to describe a view into the environment. Objects are defined by their geometric parameters given as scalar values or three dimensional coordinates and by their material properties. The geometric parameters (position, size, surface normals) can be given explicitly or implicitly. An explicit representation is given when the objects' surfaces are tesselated into a collection of planar polygons consisting of a set of vertices and a set of connecting edges. The resulting mesh thus only approximates the shape of curved objects. An implicit or parametric representation however allows for the exact determination of the required geometric parameters. The material properties of an object describe its basic color and the way how light reflects off or transmits through the object. Colors are typically represented by 3 intensity samples of the visible spectrum at the wavelength of red, green and blue. Finally, there are light sources that can be defined by position and intensity or color but also with extended parameters like a direction, a light cone or some geometric shape.

To render the scene into an image the scene database has to be traversed and transformations that were added during the modeling stage must be resolved first. The modeling transformation includes translation, scaling and rotation to transform each object from its model space or local coordinate system into world space. The world space which is represented by the world coordinate system is the common coordinate space where the lighting calculation and most other parts of the rendering algorithm are performed. Using the camera position a first optimization can be applied. Objects that are behind the observer can easily be ignored in further steps of the pipeline unless the algorithm is able to take their influence on the final image into account (e.g., a mirror might reflect light back into the scene). The lighting step that follows tries to simulate the effect of light illuminating the environment and reaching the eye. The way how this is done greatly influences the quality of the resulting image and the time needed to achieve this result. This will be discussed in more detail in the sections 1.3 and 1.4.

The following steps use the camera parameters to generate an image of the lit environment in screen space. The viewing transformation performs the perspective transformation that maps a 3D point onto a plane parallel to the screen[1]. The following clipping step eliminates vertices that lie outside the viewing frustum, i.e. the volume defined by the eye position, the look-at point and the viewing angles. Finally the resulting vertices have to be transformed onto the window of the viewing device defining the 2D viewport (Figure 1.2).

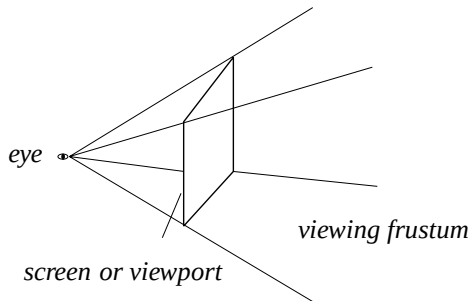

Figure 1.2 The camera model.

[1]The first occurrence of perspective transformations was found in the fifteenth century and greatly influenced renaissance artists [Wat89].

The rasterization step now determines each device pixel covered by the transformed primitive (typically a triangle) and fills it with the appropriate color. The color can be a result from the lighting step or for example a color value retrieved from a texture map, i.e., a bitmap image that was attached to the scene object.

Currently available graphics accelerators implement parts or all of the rendering pipeline in hardware. Accelerators that only implement a part of the pipeline typically leave all transformations and clipping to the system processor but provide very efficient rasterizers. Highend graphics workstations often provide a full implementation in hardware. With the rapid growing of floating point power in modern microprocessors however, special chips supporting the 3D calculation are questionable.

1.3 Local Illumination

The term *local* illumination refers to the simplifying assumption, that the illumination of a point or a surface element only depends on the local material and surface properties and the incident light sources. All that has to be calculated is the total amount of light reflected into the direction of the viewer. Given the surface normal **N** and the direction to a (point) light source **L** the *diffuse reflection* can be computed according to Lambert's law. The amount of reflected light is proportional to the cosine of the angle θ between the surface normal and the vector to the light source (Figure 1.3). Note that the direction to the viewer is unimportant.

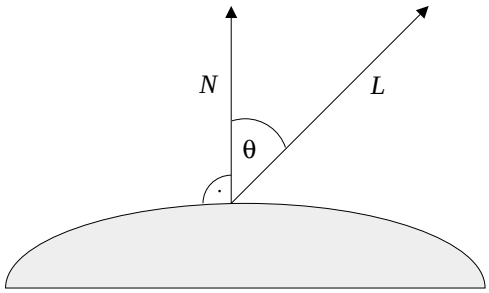

Figure 1.3 Lambert's law.

An additional ambient amount of light avoids black objects if the angle exceeds 90 degree (negative cosines are treated as zero). To obtain more realism the *specular reflection* which is dependent on the viewing direction and is responsible for highlights should be added. Depending on the material properties a highlight with varying intensities can be noticed in a cone surrounding the reflected light vector **R** (Figure 1.4). For perfect mirrors this cone reduces to a single line.

The intensity cone can be calculated by an approximation developed by Phong [Pho75]. With the highest intensity of the highlight at the center a rapid falloff is determined by a $\cos^n \alpha$ term, where α is the cone's angle and n is a material property specifying the sharpness of the highlight. Thus, summing over all light sources we get the following formula describing the intensity of a surface point when seen from a given point:

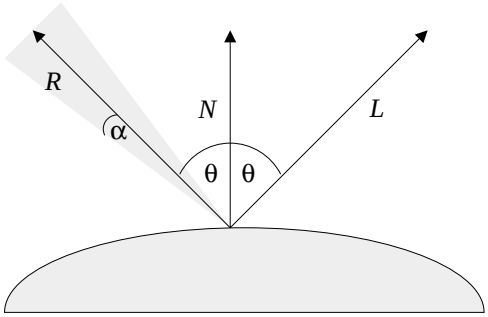

Figure 1.4 Specular reflection.

$$I = I_a k_a + \sum_{i=1}^{n} I_i (k_d \cos\theta_i + k_s \cos^n \alpha_i) \tag{1.1}$$

The coefficients k_a, k_d and k_s specify the ambient, diffuse and specular material properties respectively. I_i are the intensities of the light sources and I_a that of the ambient light. As mentioned above the ambient term is responsible for all illumination that is not directly coming from a light source. This constant term is a very rough approximation of all possible interreflections of light that might occur in the environment. Due to its simplicity formula 1.1 is also available in hardware graphics accelerators.

1.4 Global Illumination

To achieve more realism a global illumination model must be used that accurately accounts for interreflections of light between objects. In contrast to a local reflection model where only the current surface point and the direction (without occlusion) of light sources is considered potentially all scene objects have to be taken into account. In 1980 Whitted [Whi80] implemented a first global illumination algorithm using the technique of *ray tracing*.

1.4.1 Ray Tracing

To simulate optical effects that can not be achieved with the simple rendering pipeline (shadow, global reflection, transparency), ray tracing reverses the photographic process.

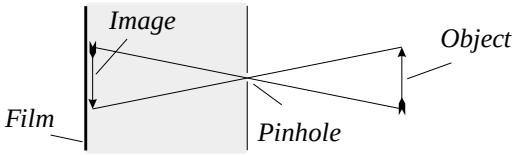

Figure 1.5 The pinhole camera.

Instead of capturing the light directly it starts with the empty image plane subdivided into a regular grid of pixels. Through each pixel a ray is traced into the scene using the modified

pinhole camera model shown in Figure 1.2. A physical pinhole camera would have the eyepoint and imageplane reversed and the eyepoint would actually be the pinhole while the imageplane would be the film (Figure 1.5). For a simulation however, the modified model is more convenient. Tracing rays this way, i.e. from the eye through the camera into the scene instead from the light sources via the objects into the eye, gave this technique its complete name *backward ray tracing*.

To find the image color at the pixel position the ray is checked against all objects or surfaces in the scene to see if an intersection point exists. If there is no intersection the pixel's color is set to the background color and the next pixel is chosen. However, if there is an intersection several components have to be summed up, corresponding to the visual effects that are to be captured. The most important contribution is the direct light due to the visible light sources. Starting from the intersection point at the object's surface a new ray is cast into the direction of each light source to determine its visibility. If the ray hits some opaque object before having traveled the distance to the light source the surface point lies in shadow relative to that light source. Therefore, these rays are called *shadow rays*. If no occlusion occurs the light source directly illuminates the questionable point, classifying the corresponding ray as an *illumination ray*.

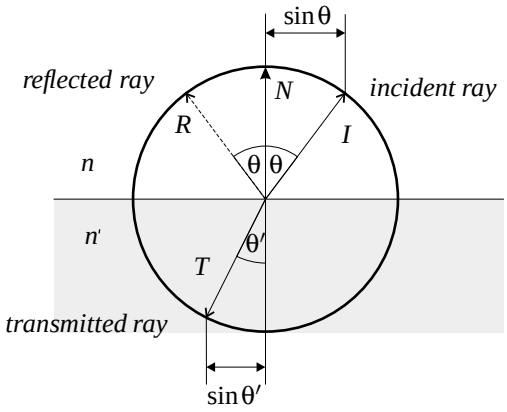

Figure 1.6 Snell's law. Computing refraction and reflection rays.

Depending on the material properties of the surface hit *reflection* or *refraction rays* have to be generated to account for specular reflection or transmission of light. To find the direction of light incident to a surface point due to reflection the incident ray from the eye has to be mirrored at the surface normal. If a ray emanating into that direction hits another surface, the color found there additionally contributes to the illumination of the first point. Transparent surfaces are treated analogically. The direction of the refraction ray however is determined by Snell's law (Figure 1.6) which takes into account the refraction indices of the two media touching at the given point (typically air which is treated as a vacuum here and the object's material).

With n and n' being the refraction indices of the two media and θ and θ' being the angles between the surface normal and the incident and transmitted ray Snell's law states:

$$n \sin\theta = n' \sin\theta' \tag{1.2}$$

Being able to compute reflection and refraction rays propagates the problem of finding the illumination at a surface point to the next intersection of the ray with an object. The same components contributing to the illumination have to be computed here to find the correct color that contributes to the illumination of the first point. This defines a recursive process which completes the ray-tracing algorithm (Figure 1.7). At each intersection point shadow rays and depending on the material properties reflection and/or refraction rays are generated. The recursion can be stopped if no more objects are hit, if the contribution of the current ray falls below a threshold or if a maximum number of recursion steps is found, whichever occurs first.

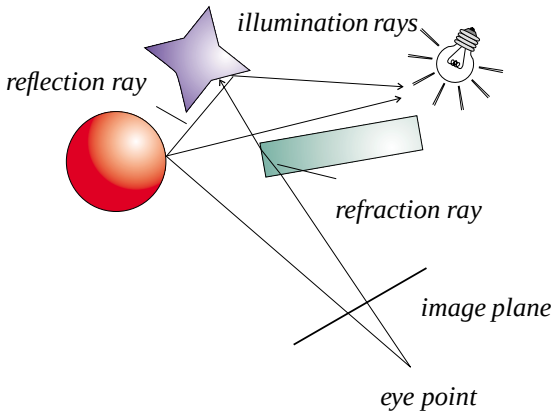

Figure 1.7 Recursive ray tracing.

The illumination at each intersection point is calculated with an extension to Formula 1.1 which takes mirror reflection and transparency into account:

$$I = I_a k_a + k_d \sum_{i=1}^{n} I_i \cos\theta_i + I_s k_s + I_t k_t \tag{1.3}$$

The coefficients k_s and k_t are material properties defining the surface's specular reflection and transmission values. The terms I_s (containing the Phong approximation) and I_t are the intensities due to the recursive ray-tracing process following reflection and refraction rays.

This simple global illumination model generates images with typical characteristics. Shadows are very sharp because neighboring points either completely 'see' the (point) light source or they are in shadow. There is no slight transition between light and dark. Another feature is the look of surfaces due to Phong's shading model, which often gives surfaces a 'plastic'-like appearance.

The main computational effort involved with ray tracing is performing the intersection tests between rays and objects. Typically simple shapes like bounding boxes or bounding spheres are tested first. If no intersection is found further expensive tests can be avoided. To minimize the number of objects that have to be tested scene structuring techniques were developed that group scene objects according to their spatial distribution [Gla89].

1.4.2 Radiosity

The radiosity method is a completely different way to compute the global illumination in a scene. The main idea is to calculate the energy flow between diffuse surfaces which results in realistically looking materials and (soft-)shadows. In 1984 results from the theory of radiative heat transfer [SH92] were applied to the global illumination problem and resulted in the first radiosity algorithms [GTGB84, NN85]. The term *radiosity* denotes the physical measure of power radiated per unit area of a surface (radiant exitance) and was established as a synonym for this category of global illumination algorithms.

The simplifying assumption that all surfaces are purely diffuse reflecting (i.e. *Lambertian* reflectors) allows for calculating a view-independent solution. In contrast to ray tracing which computes a single view-dependent solution in image space a radiosity solution is computed in object space. After the energy exchange between all surfaces is determined arbitrary views of the solution can be generated quickly. Graphics accelerators fully support this step because it exactly corresponds to the rendering pipeline (Section 1.2) without the lighting step. The lighting is precomputed by the radiosity algorithm. Walkthroughs are possible after the (time consuming) radiosity solution is obtained which give the user a chance to virtually explore a correctly lit and shaded environment in real-time.

To compute the energy flow the radiosity method uses an energy equilibrium illumination model. The reflectivity and the emissivity of the surfaces are used to simulate the bounces of light originating from the emissive surfaces (i.e. the light sources) through the environment until an equilibrium is found. Thus, indirect illumination due to reflection from other surfaces can be rendered correctly. Figure 1.8 shows a room with colored walls containig two white boxes. The only light source is directly pointing towards the ceiling thus illuminating the room indirectly.

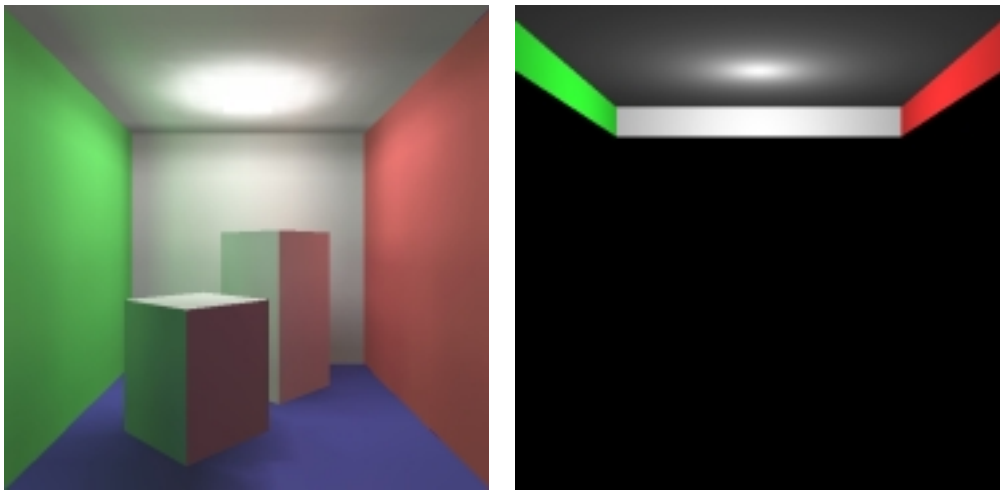

Figure 1.8 Indirect illumination: Radiosity vs. classical Ray Tracing.

The left image shows a radiosity rendering of the scene. The indirect illumination is captured very well and the interior of the room is lit correctly. Additionally the sides of the white boxes are colored according to the nearby walls of the room. The light was reflected from the ceiling to the colored walls and finally hit the boxes. This effect is often called *color bleeding*

and is due to multiple interreflections of light. As a comparison the same scene was rendered with ray tracing where only the directly illuminated parts of the scene are visible.

To find the energy equilibrium radiosity algorithms are typically formulated as finite element methods. Instead of computing the illumination for each surface point of the scene only a few representative points are considered. The scene objects are subdivided into planar surface elements defining a mesh of *patches*. For each patch an average radiosity value is computed that represents the radiosty at the center of the patch. Figure 1.9 shows a simple mesh. The radiosity of patch $p3$ in object P is computed due to the radiosities of the patches of object Q and R.

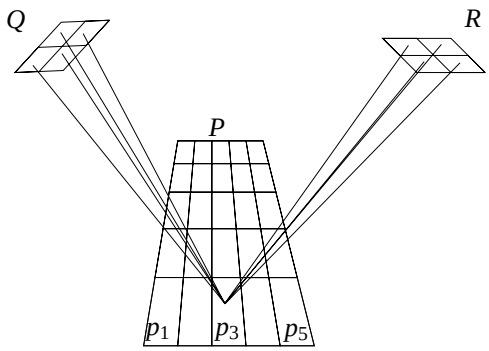

Figure 1.9 Using finite elements for radiosity.

The radiosity of each patch (B_i) can be written as a sum of the patch's own emissivity (E_i) and the radiosities of all other patches (B_j) weighted by the reflectivity of the receiver (ρ_i). The energy exchange between two patches is determined by the *form factor* (F_{ij}), a geometrical term that describes the fraction of the total energy that arrives from the sending patch P_i at the receiver P_j.

$$B_i = E_i + \rho_i \sum_{j=1}^{N} F_{ij} B_j \tag{1.4}$$

The form factor depends on the spatial position of the two patches. Most important for the computation of shadows however is the mutual visibility of the patches. Due to the visibility computation, which can be done by ray casting, the form factor calculation is the most time consuming part of a radiosity algorithm. Most research, including parts of this thesis, focused on reducing the number of form factors that have to be computed for a radiosity solution. If the radiosity equation is written down for each patch of the scene, a system of N linear equations is obtained. With the radiosities being the N unknowns the equations can be grouped to a matrix and a linear equation solver is used to compute the radiosities.

Early radiosity methods set up the matrix of form factors first and solved the full matrix containing N^2 entries if the scene was meshed into N patches [GTGB84]. The quadratic memory and time consumption extremely limited the scene complexity until the progressive refinement method was introduced [CCWG88]. Instead of storing the full matrix only a single column was used and form factors to other patches were computed on the fly. The most important contribution over the last years however, was the introduction of the hierarchical radiosity algorithm

[HSA91]. By modelling the energy exchange at different resolution levels of the mesh several entries in the form factor matrix could be combined to blocks and represented by a single value. This reduced the complexity of the radiosity method to be almost linearly in the number of mesh elements. Extending this idea to building hierarchies above the element mesh led to radiosity clustering algorithms [SAG94, Sil95]. Modelling interactions between objects and groups of objects dramatically reduced the rendering times and allowed the computation of images from scenes containing several hundred thousand patches [GH96].

More details on the derivation of the radiosity method and various ways to compute a radiosity solution are given in Chapter 2.

1.5 Thesis Contribution

This thesis contributes to the field of radiosity rendering in several ways. First, a new approach of incorporating curved surfaces in the hierarchical radiosity algorithm is presented. This approach combines ray-tracing techniques with the finite element method to obtain very accurate solutions in less time. Curved surfaces play an important role in computer graphics because most realistic models (e.g., parts of a car) are built using them. An approximation using a fixed number of planar polygons to drive the radiosity algorithm either leads to visual artifacts or to an excessive increase of running time due to the large number of very small polygons. The method introduced here uses an adaptive object-based meshing that is able to refine the mesh during the computation to an arbitrary accuracy. Because the mesh is only refined in regions critical for the visual appearance solutions are obtained very quickly.

To further improve the efficiency of radiosity implementations a new distributed hierarchical radiosity algorithm is presented. Because radiosity algorithms potentially model the interaction between all pairs of surfaces efficient parallel implementations without a huge communication overhead are hard to implement. The distributed algorithm introduced here combines object-based techniques from our curved surfaces algorithm with an efficient load balancing scheme. The algorithm runs in a network of computers and quickly adapts to stress conditions that appear in a real world network. The biggest advantage, however, is that the need for a radiosity mesh at the client computers could be eliminated. This results in very low communication costs and efficient use of remote machines with limited memory capacities.

Finally, a new scene structuring algorithm originally developed for ray tracing is applied to radiosity clustering. It makes use of an object-based cost function to drive the subdivision of scene elements into a hierarchy of bounding volumes. This hierarchy naturally adapts even to small details of the original objects. This can be efficiently used to simulate the energy transport between objects and object clusters. To quickly achieve high quality radiosity solutions of large scenes, an error driven clustering algorithm is used. Several optimizations are proposed that allow the clustering algorithm to benefit from the deep bounding volume hierarchies that are arranged in a binary tree.

1.6 Thesis Outline

The outline of the thesis is as follows: in Chapter 2 the radiosity method is described, thereby laying the theoretical foundation for the work presented here. Chapter 3 introduces our hierarchical radiosity algorithm for curved surfaces and discusses some consequences following this object based approach. In Chapter 4 the parallelization of radiosity is discussed and a distributed hierarchical radiosity algorithm for heterogeneous networks is presented. Chapter 5 gives an overview of radiosity clustering techniques and describes a clustering method that efficiently combines data structures for ray acceleration and clustering.

CHAPTER 2

The Radiosity Method

The radiosity method serves as a tool to compute the interreflections of light between diffuse surfaces. It combines the knowledge from several fields of physics (i.e. optics and radiative heat transfer) with finite element methods to compute the energy balance in a closed environment. Due to the assumption of pure diffuse reflectors photorealistic images from the solution can be rendered quickly from arbitrary view points using graphics hardware.

This chapter describes the physical background of radiosity and the algorithms that were developed in the last decades to efficiently simulate the light transport. After an introduction to radiometry and photometry the energy balance equation is derived. The methods to solve the equation are explained and recently developed algorithms to manage the complexity involved with large scenes are presented. Further information on the radiosity method and the related terms can be found by [CW93] and [SP94].

2.1 Radiometry

The frequency range of electromagnetic waves covers about 50 octaves[1] from which only one can be perceived by the human eye [Kuc88]. This range is the visible spectrum between 390 and 770nm that is typically referred to as light. Radiometry describes the transport of light in an objective way, actually reflecting the physics behind it. Photometry, on the other hand, describes how light is perceived by a human observer due to the eye's sensibility. A detailed introduction to the related terms is given by [Gla95].

To simplify the complex physical model of light transport typically some approximations are made. The radiometric terms presented here are considered to be independent of time, polarization and wavelength. These simplifications prohibit the simulation of several visual phenomena like phosphorescence or luminescence but makes the presentation clearer. The only remaining parameters to deal with are position and direction.

2.1.1 Solid Angle

The radiant energy exchange can be described as being dependent on the *solid angle* which is the three-dimensional extension of the angle between two lines. The solid angle ω is the quotient of a spherical area and its squared radius and is expressed in steradians (sr):

[1] Octave: frequency ratio 1 : 2

$$\omega = A/r^2, \qquad [\omega] = sr = \frac{m^2}{m^2} = 1 \tag{2.1}$$

As can be seen in Figure 2.1 the solid angle measures the area of the projection of an object onto a unit sphere surrounding the point of interest. This exactly follows the definition in 2D where the angle subtended by an object equals the arclength of the projection.

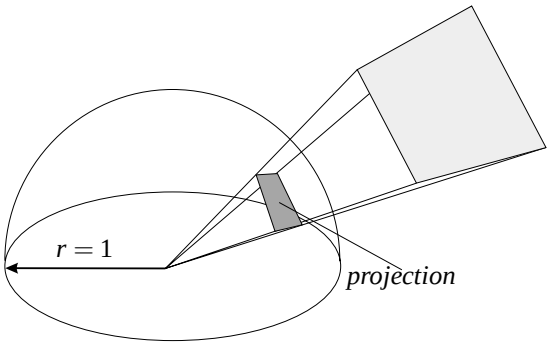

Figure 2.1 Solid angle.

For *differential* areas the solid angle can be approximated by the projection onto the tangent plane instead of the sphere. Let the differential area be ΔA and let the distance be r then the solid angle subtended is given by:

$$\Delta\omega \approx \frac{\Delta A \cos\theta}{r^2} \tag{2.2}$$

The cosine term accounts for the orientation of the surface element relative to the direction to the origin (Figure 2.2).

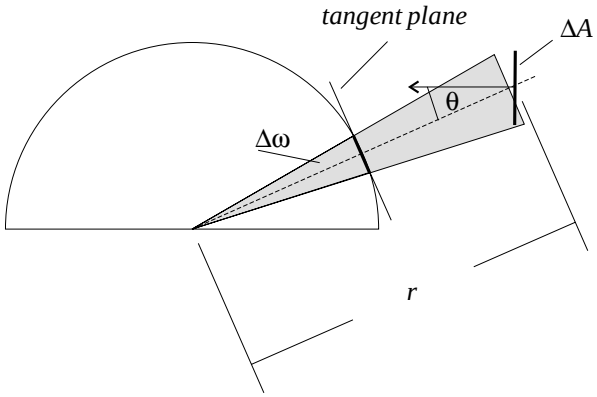

Figure 2.2 Differential solid angle.

2.1.2 Radiance

The most fundamental quantity in image synthesis is *radiance* (L). It denotes the radiant energy per unit volume and is determined by the photon density, i.e., the number of photons at a point x moving in some direction ω:

$$L(x,\omega) = \int p(x,\omega,\lambda)\frac{hc}{\lambda}d\lambda, \qquad [L] = \frac{W}{sr \cdot m^2} \tag{2.3}$$

Here, p denotes the photon density at wavelength λ and $\frac{hc}{\lambda}$ is each photon's energy, which is dependent on its wavelength and the two constants c (speed of light) and h (Planck's constant). Radiance is measured in power per unit projected area perpendicular to the direction of photon movement and per unit solid angle. Radiance has the important property of being invariant to the distance, which is expressed by its dependency from the solid angle. Due to the law of conservation of energy, the amount of radiance leaving a point along a ray remains constant if no scattering or absorbing media is in between.

2.1.3 Irradiance

To find the radiant energy arriving at some surface patch A as required by a finite element method, the incoming radiance has to be integrated over the hemisphere above the patch. This quantity is called *irradiance* (E) and is measured in power per unit area:

$$E = \int_\Omega L_i \cos\theta_i \, d\omega_i, \qquad [E] = \frac{W}{m^2} \tag{2.4}$$

The term $\cos\theta_i \, d\omega_i$ denotes the projected solid angle and is the projection of the solid angle (i.e., the differential area $\Delta\omega$ in Figure 2.2) onto the base of the hemisphere. If the projected solid angle is integrated over the complete hemisphere the projection covers the full circle, i.e., π. In contrast to radiance, irradiance drops off with the square of the distance.

2.1.4 Radiosity

The physical quantity that gave the algorithms and methods described in this thesis its name is *radiosity* (B). Analogously to irradiance it describes the radiant energy *leaving* some surface and is also called *radiant exitance*. It is measured in the same units as irradiance:

$$B = \int_\Omega L_o \cos\theta_o \, d\omega_o, \qquad [B] = \frac{W}{m^2} \tag{2.5}$$

Thus, radiosity is defined as the integral over the complete hemisphere. If we assume purely diffuse reflection (see Section 2.3.2), the radiosity that is equally reflected in all directions can directly be expressed in terms of radiance:

$$B = \pi L_o \tag{2.6}$$

2.2 Photometry

Each radiometric term has a corresponding quantity in photometry, describing the eye's responsiveness. Beside being sensitive to a limited range of wavelengths the response of the human eye is also non-uniform. This behavior is captured in the luminous efficiency function plotted in Figure 2.3.

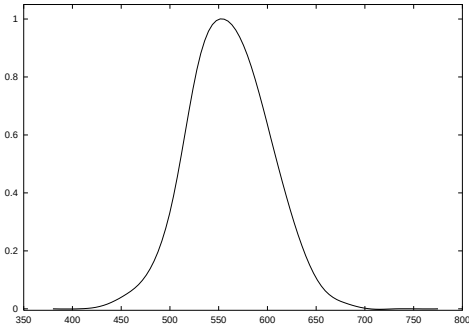

Figure 2.3 Luminous efficiency function.

Integrating the radiometric quantities against the luminous efficiency function yields the photometric quantities with their own measures (Table 2.1). Global illumination algorithms are typically formulated with radiometric terms, because the transport of light that is to be simulated is independent from the visual system. When it comes to displaying a resulting image on an output device like a monitor or a printer however, it is important to correctly map the results to the limited displayable range. This *tone-mapping* process has to be carried out with photometric quantities [TR93]. Another aspect is the optimization of global illumination algorithms with the knowledge of the limited responsiveness of the eye. Perception-based rendering techniques try to avoid computing optical effects that are hardly perceivable by the human eye. These methods are topic of ongoing research and related work can be found by [GH97] and [Mys98].

Radiometry		Photometry	
Radiance	$[\frac{W}{sr \cdot m^2}]$	Luminance	$[\frac{cd}{m^2}]$
Irradiance	$[\frac{W}{m^2}]$	Illuminance	$[\frac{lm}{m^2}]$ (Lux)
Radiosity	$[\frac{W}{m^2}]$	Luminosity	$[\frac{lm}{m^2}]$ (Lux)

Table 2.1 Radiometric and photometric quantities.

2.3 Reflection

The appearance of a material can be characterized by its reflection properties, i.e., by the fraction of light that is reflected off the surface at a specific wavelength. From the possible interactions of light with a material only the effects occurring directly at the surface are considered here.

Light is not assumed to enter the material, which would produce effects like transmission or absorption.

2.3.1 The BRDF

The ratio between reflected and incoming radiant energy at a surface point is expressed by the *bidirectional reflection distribution function* or BRDF. It relates irradiance from an incident direction ω_i to radiance in an outgoing direction ω_r by a proportionality constant (Figure 2.4):

$$f_r(\omega_i, \omega_r) = \frac{L_r(\omega_r)}{L_i(\omega_i)\cos\theta_i d\omega_i}, \qquad [f_r] = sr^{-1} \tag{2.7}$$

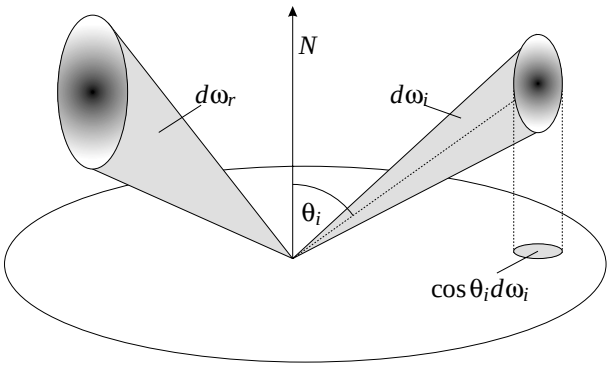

Figure 2.4 BRDF geometry.

The BRDF describes the directional distribution of reflected light and its value can range from 0 to infinity. For physically based BRDFs the Helmholtz principle states that incoming and outgoing directions can be reversed without affecting the BRDF. A BRDF is said to be anisotropic if the reflection is dependent on the rotation of the material around the normal like brushed metals. An isotropic BRDF describes a material who's surface rotation does not change the reflection properties.

Integrating the BRDF over the hemisphere of reflected directions results in the *reflectance* ρ, a much more intuitive dimensionless quantity ranging from 0 to 1:

$$\rho(\omega_i) = \int_\Omega f_r(\omega_i, \omega_r)\cos\theta_r \, d\omega_r \tag{2.8}$$

Making the BRDF independent on the outgoing direction results in purely diffuse reflection.

2.3.2 Diffuse Reflection

Radiosity methods assume ideal diffuse reflectors to achieve view independent solutions. Ideal diffuse means that the same amount of radiant energy is reflected in all directions of the hemisphere. Due to the reciprocity principle the incident direction can also be neglected resulting in a constant BRDF which in turn causes the reflectance to be constant. This constant is the

diffuse reflectance and is the only material property used by radiosity algorithms. The general reflectance equation (Equation 2.9) can now be simplified for the radiosity case:

$$\begin{aligned} L_r(\omega_r) &= \int_\Omega f_r(\omega_i,\omega_r)L_i(\omega_i)\cos\theta_i\, d\omega_i \qquad (2.9)\\ &= f_r\int_\Omega L_i(\omega_i)\cos\theta_i\, d\omega_i \\ L_r &= f_r E \qquad (2.10)\end{aligned}$$

Again, the proportionality factor between the BRDF and the reflectance in the radiosity case is π, due to integration of the projected solid angle (see Section 2.1.3).

$$\rho = \pi f_r \qquad (2.11)$$

2.4 The Rendering Equation

The reflectance equation (Equation 2.9) describes the outgoing radiance when reflected from a surface. To derive an energy balance equation the incident light distribution at a surface point must be specified. The sources of radiant energy influencing the illumination of a point are the light sources and reflectors. Summing their contribution gives the following integral equation describing an energy equilibrium:

$$L(x,\omega_r) = L_e(x,\omega_r) + \int_\Omega f_r(x,\omega_i,\omega_r)L_i(x,\omega_i)\cos\theta_i\, d\omega_i \qquad (2.12)$$

Beside the incoming and reflected directions (ω_i,ω_r) the current surface point x is specified in the radiance and BRDF terms. L_e is the emissivity of the surface which is zero for all surfaces except for the light sources. Equation 2.12 was introduced to computer graphics in a slightly different notation by Kajiya [Kaj86] and is called the *rendering equation*. To solve this integral equation where the unknown radiance L appears on both sides numerical methods are used, one of them being the radiosity method described in this chapter.

To see that the rendering equation actually describes the interreflections of light Kajiya proposes to write the equation using an integral operator acting on the radiance in the scene as the *reflection operator* $\mathcal{R}$:

$$L = L_e + \mathcal{R}L \qquad (2.13)$$

Inverting Equation 2.13 using a Neumann series delivers:

$$\begin{aligned} L &= [I-\mathcal{R}]^{-1}L_e \\ &= \sum_{n=0}^{\infty}(\mathcal{R})^n L_e \qquad (2.14)\end{aligned}$$

The physical explanation is that the contribution of all reflections is summed up, i.e., $\mathcal{R}^0 L_e$ yields the emissivity, $\mathcal{R}$ represents the first reflection, $\mathcal{R}^2$ the second and so on.

With the results from the last sections Equation 2.12 can be simplified for the radiosity case. The constant BRDF is expressed in terms of reflectance (Equation 2.11) and can be moved outside the integral:

$$L(x,\omega_r) = L_e(x,\omega_r) + \frac{\rho(x)}{\pi}\int_\Omega L_i(x,\omega_i)\cos\theta_i\, d\omega_i \tag{2.15}$$

Omitting the dependency on the outgoing direction for the reflected radiance and the emissivity results in purely diffuse reflection. Multiplying both sides of the equation by π (Equation 2.6) gives an expression for radiosity:

$$B(x) = B_e(x) + \rho(x)\int_\Omega L_i(x,\omega_i)\cos\theta_i\, d\omega_i \tag{2.16}$$

Finally, the integral over the hemisphere can be converted to a surface integral, thereby replacing the differential solid angle $d\omega$ by $\frac{dA'\cos\theta'}{r^2}$ (Equation 2.2). Because the integration still includes all surfaces in the scene, a visibility function has to take care of occlusion. $V(x,x')$ denotes a binary visibility function that equals 1 if the surface points x and x' are mutually visible, otherwise it is zero. This yields the radiosity equation:

$$B(x) = B_e(x) + \rho(x)\int_S B(x')\frac{\cos\theta\cos\theta'}{\pi r^2}V(x,x')\, dA' \tag{2.17}$$

Having derived the radiosity equation, i.e., the rendering equation for the special case of pure diffuse reflectors, the next sections will describe methods for solving this equation.

2.5 The Finite Element Approach

In general, no closed form solution is available for Equation 2.17. As a result, the radiosity for an infinite number of surface points had to be computed, which is impractical. A way to solve this kind of integral equation is a further approximation using finite element methods [Zie89]. The idea is to project the infinite dimension of the function space defined by the radiosity equation down to some finite dimension. The *dimension* of a function space is the number of discrete values needed to specify the function. A function space of finite dimension is spaned by a finite number of basis functions. A projection from a higher function space thus requires to find coefficients for each basis function. The finite element approach for radiosity commonly uses the simplest basis function i.e., the box function. The projection of the radiosity function thus results in an approximation by a linear combination of piecewise constant basis functions.

The assumption behind this approximation is that the radiosity and reflectance of a surface element or a patch is constant. This allows writing Equation 2.17 as a sum over discrete patches. With the radiosity over a single patch being the area weighted average of the pointwise radiosities:

$$B_i = \frac{1}{A_i}\int_{x\in P_i} B(x)\, dx, \tag{2.18}$$

Equation 2.17 then becomes:

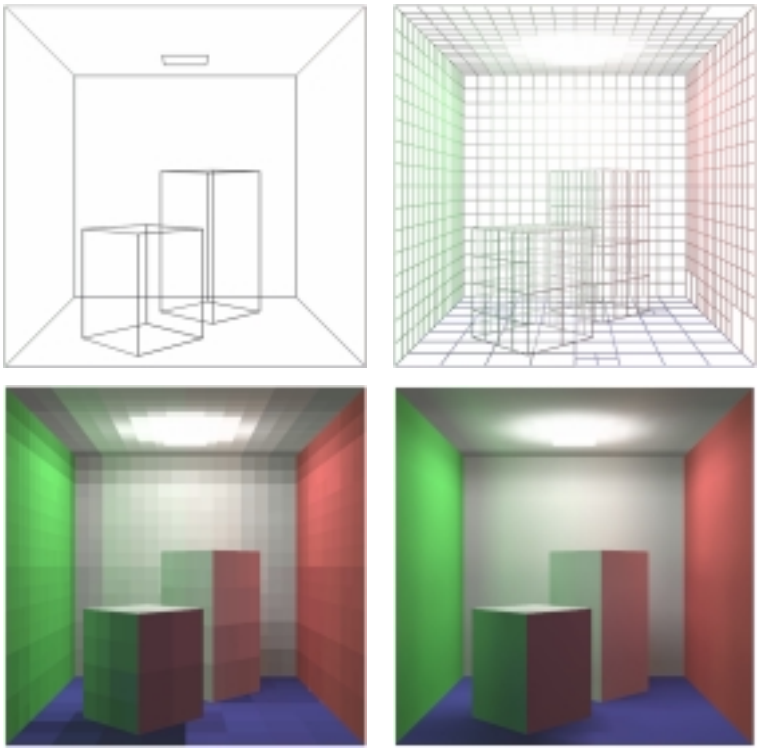

Figure 2.5 Finite element radiosity: original polygons, finite element mesh, patch radiosities and filtered solution.

$$B_i = B_{ei} + \rho_i \sum_{j=1}^{N} B_j \frac{1}{A_i} \int_{x \in P_i} \int_{x' \in P_j} \frac{\cos\theta \cos\theta'}{\pi r^2} V(x,x')\, dx' dx \tag{2.19}$$

This is the discrete formulation of the radiosity equation. The double integral on the right side describes the geometric configuration of the two patches P_i and P_j involved. It is independent on radiometric quantities and thus often written as a single expression F_{ij} called the *form factor* (see Section 2.6):

$$F_{ij} = \frac{1}{A_i} \int_{x \in P_i} \int_{x' \in P_j} \frac{\cos\theta \cos\theta'}{\pi r^2} V(x,x')\, dx' dx \tag{2.20}$$

Now, Equation 2.19 can be written more compactly:

$$B_i = B_{ei} + \rho_i \sum_{j=1}^{N} F_{ij} B_j \tag{2.21}$$

In Figure 2.5 the finite element approach to solve the radiosity equation is illustrated. The upper left image shows the original input scene shaded as wireframe. The algorithm subdivides the input polygons in a number of discrete patches forming the finite element mesh. For each patch a single radiosity value is computed as can be seen in the lower left image. Finally, as seen in the last image, a filtering step can be applied to obtain a smooth solution.

2.6 Form Factors

The form factor integral (Equation 2.20) describes the relative position and orientation of two possibly interacting patches. Actually, the form factor F_{ij} measures the fraction of the total radiant energy leaving patch P_i that arrives at patch P_j. Because the form factors are independent on the illumination, different lighting conditions can be simulated once the form factors are known. The kernel of the form factor integral is called the differential form factor and corresponds to the form factor between two differential areas as illustrated in Figure 2.6.

$$F_{dAdA'} = \frac{\cos\theta\cos\theta'}{\pi r^2}\,dA' \tag{2.22}$$

Before introducing ways to solve the form factor integral, a few properties that can be derived from Equation 2.20 are shown.

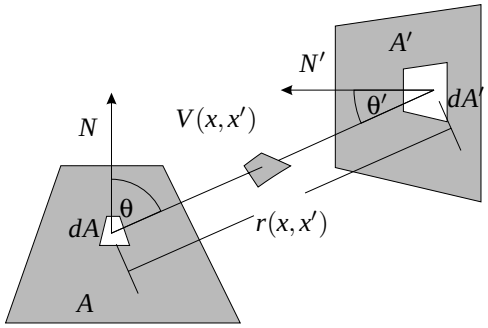

Figure 2.6 Notation for the form factor.

2.6.1 Properties

Form factors have two important properties that allow computing an unknown form factor from known ones. These properties are the *reciprocity* and the *additivity*. Both properties intuitively follow the definition of the form factor and can easily be derived. If Equation 2.20 is multiplied by A_i i.e., the area of patch P_i, a symmetric expression independent of the patch area is obtained:

$$A_i F_{ij} = \int_{x\in P_i}\int_{x'\in P_j} \frac{\cos\theta\cos\theta'}{\pi r^2} V(x,x')\,dx'dx \tag{2.23}$$

Reversing sender and receiver yields the same expression resulting in a reciprocity relationship:

$$A_i F_{ij} = A_j F_{ji} \tag{2.24}$$

An application of this principle can be seen in the radiosity equation (Equation 2.21). If this equation is written in terms of power instead of radiosity, the indices of F_{ij} can be reversed (recall from the definition that P_i is the sender). This makes the formula more clearer, because the terms being summed up can now be recognized as the sending (i.e., emitting and/or reflecting) patches:

$$A_i B_i = A_i B_{ei} + \rho_i \sum_{j=1}^{N} F_{ji}(A_j B_j) \tag{2.25}$$

The additivity rule for form factors states that the form factor from one patch to the union of two other patches equals the sum of the individual form factors. Because the form factor measures the proportion of the energy arriving at the receiver from a single source, the power received by multiple patches can be added directly. Considering three disjoint patches P_i, P_j and P_k the additivity is expressed by:

$$F_{i(j\cup k)} = F_{ij} + F_{ik} \tag{2.26}$$

To measure the energy received from multiple disjoint patches however, the individual form factors must be area weighted. The same explanation as before can be used here. Summing up multiple fractions to obtain a new fractional part makes no sense. Instead the fractions must be related to the whole environment (normalization) which is done by area averaging:

$$F_{(i\cup j)k} = \frac{A_i F_{ik} + A_j F_{jk}}{A_i + A_j} \tag{2.27}$$

2.7 Computing The Form Factor

Finding an analytic solution for the integral in Equation 2.20 was possible only for very simple geometric configurations. In the heat transfer literature [How82] examples with the known solutions can be found. For arbitrary geometries that occur in standard 3D scenes however, numerical methods have to be applied. By subdividing the interacting patches and summing the differential form factors (Equation 2.22)) an approximation to the form factor integral can be obtained.

2.7.1 The Hemicube

As noted in Section 2.1.3 the energy arriving at a surface patch is proportional to the projected solid angle of the incoming radiance. Thus, the form factor from a point x to a patch P can be interpreted as the projection of the solid angle subtended by P down to the base of the hemisphere:

$$F_{x,P} = \frac{1}{\pi} \int_{\omega_P} \cos\theta \, d\omega \tag{2.28}$$

The construction can be seen in Figure 2.7 and is known as Nusselt's analogy [Nus28].

To quickly compute the projected solid angle, the hemisphere can be approximated by a *hemicube* [CG85]. Each side of the hemicube is subdivided into discrete cells, each storing a precomputed delta form factor. These delta form factors are the differential form factors corresponding to the solid angles subtended by the cells. The actual projection then is performed with the rendering pipeline (see Section 1.2). The camera parameters have to be defined in such a way, that the camera is located at the center of the patch and a viewing angle of 90 degree is used. Rendering the whole scene onto each side of the hemicube results in the desired

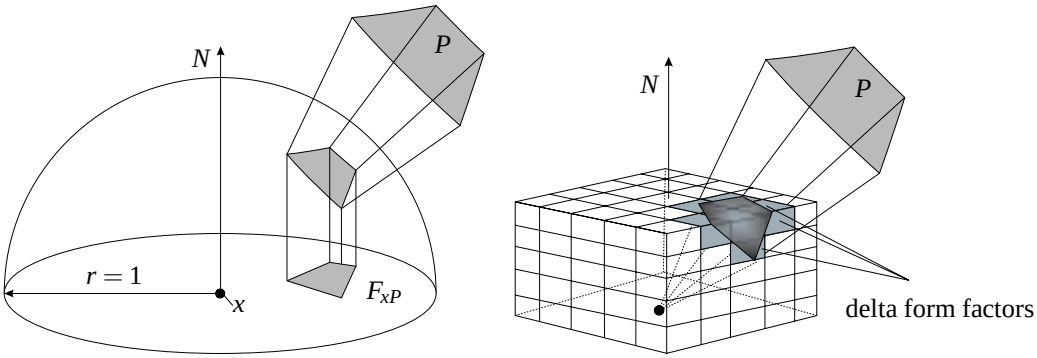

Figure 2.7 From Nusselt's Analogy to the Hemicube: The form factor corresponds to the projected solid angle.

projection. Instead of color values however, unique identifiers are attached to each polygon. These identifiers when read back from the frame buffer, indicate which polygon was visible (the so called *item buffer* technique). The form factor to a polygon in the scene is obtained by summing up the delta form factors covered by that polygon's projection. Due to hardware implementations of the rendering pipeline the algorithm performs very well, although it requires the graphics accelerator to render the scene 5 times for each patch. On the other hand, this directly computes the point-to-area form factors to all patches in the environment. The disadvantage of the hemicube however is the fixed subdivision of the hemicube sides, often resulting in aliasing artifacts. Improvements to this technique were made by using different resolutions for the top and the sides of the cube [MT93] as well as reducing the cube to a single plane [SP89].

2.7.2 Monte Carlo Integration

A more generic method of solving the form factor integral is using Monte Carlo integration with a ray-casting approach. This technique can easily be used to determine area-to-area form factors by sampling the areas of two patches. Consider the form factor from a differential patch to a finite element:

$$F_{dA,A'} = \int_{A'} \frac{\cos\theta \cos\theta'}{\pi r^2} V(x,x')\, dA' \tag{2.29}$$

Area A' can now be subdivided into smaller elements, approximating differential areas. Evaluating the kernel of Equation 2.29 at the center of these elements and accumulating the area weighted results gives an approximation of the form factor.

Another possibility is the summation of form factors from a differential area to a parallel disc with area A'. For this simple geometric configuration an analytic formula describing the unoccluded form factor is known as:

$$F_{dA,A'} = \frac{A'}{\pi r^2 + A'} \tag{2.30}$$

The orientation of the disc relative to the differential area can be taken into account by multiplying Equation 2.30 with $\cos\theta\cos\theta'$. The full area-to-area form factor of two patches can now be computed. A number n of randomly distributed sampling points is chosen on both patches. This effectively subdivides the patches in elements of smaller size. The sampling points are connected with rays that are checked for occluders. If an occluder is found, the corresponding differential form factor is zero. Otherwise, Equation 2.30 is evaluated, using the subdivided area $\Delta A' = \frac{A'}{n}$. Formally, the approximation of the area-to-area form factor using n discs is given by:

$$F_{ij} = \frac{A_j}{n} \sum_{k=1}^{n} \frac{\cos\theta_i^k \cos\theta_j^k}{\pi r^2 + \frac{A_j}{n}} V(x_i, x_j) \tag{2.31}$$

The ray-casting approach to simultaneously determine visibility and form factors has several advantages. Scene structuring techniques like bounding volume hierarchies or space subdivision can be used to accelerate the visibility test. The quality of the test can easily be influenced by the number of sample points, thereby trading off between accuracy and speed. Additionally, a ray tracer can typically deal with a variety of shapes. Intersection tests with spheres or cylinders can be carried out more efficiently and accurately than with the corresponding polygons.

2.8 The Radiosity Matrix

In the last sections the radiosity equation was derived and it was shown how the energy exchange between surfaces can be computed by evaluating the form factor integral. To obtain a radiosity solution that accounts for all interreflections between the surfaces of a scene, Equation 2.21 has to be computed for all patches. These n equations with n unknowns (i.e., the radiosities) form the following system of linear equations that has to be solved:

$$\begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} B_{e1} \\ B_{e2} \\ \vdots \\ B_{en} \end{pmatrix} + \begin{pmatrix} \rho_1 F_{11} & \rho_1 F_{12} & \dots & \rho_1 F_{1n} \\ \rho_2 F_{21} & \rho_2 F_{22} & \dots & \rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ \rho_n F_{n1} & \rho_n F_{n2} & \dots & \rho_n F_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} \tag{2.32}$$

which can be transformed to:

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1-\rho_n F_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} B_{e1} \\ B_{e2} \\ \vdots \\ B_{en} \end{pmatrix} \tag{2.33}$$

More compactly, this system of linear equations can be written as a matrix operating on the unknown radiosity vector:

$$\mathbf{MB} = \mathbf{B_e} \tag{2.34}$$

Solving Equation 2.33 is equivalent to inverting matrix $\mathbf{M}$, which is often referred to as the form factor matrix. The entries of the matrix can be computed with the techniques shown in the

last section. The most important property of the form factor matrix is its diagonal dominance. That means, that the sum of the entries in each row except the diagonal element is not bigger than the value of the diagonal element:

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} \left| M_{ij} \right| \leq |M_{ii}|, \forall i \tag{2.35}$$

The entries on the diagonal of matrix $\mathbf{M}$ in Equation 2.33 all evaluate to 1 because the form factor from a planar patch to itself is zero. On the other hand the form factors from one patch to all other patches in the environment must be equal to 1, otherwise energy would be created. Thus, the sum of all off-diagonal entries in a row must be smaller than 1 which makes the form factor matrix diagonally dominant. This property guarantees convergence of some iterative relaxation methods like Gauss-Seidel iteration or Jacobi iteration, that can now be applied to solve the linear system [CG85].

2.8.1 Relaxation Techniques

Linear systems like Equation 2.34 can be solved with different relaxation techniques. These methods work iteratively by providing an initial guess to the solution and then using in each iteration the approximation obtained in the previous step. For the radiosity matrix a good initial guess is provided by the vector of emissivities. The unknown radiosities are initialized with the emissive values of the patches and can be used as a first approximation. The goal of the iterative process is to minimize the difference between the actual solution and the current approximation. Because the solution is unknown, the *residual* $\mathbf{r}$ can be used as a measure:

$$\mathbf{r}^{(0)} = \mathbf{M}\mathbf{B}^{(0)} - \mathbf{B_e} \tag{2.36}$$

The superscript denotes the iteration count. If all entries in the residual vector are zero, the exact solution is found. Each iteration step thus updates the current residual and the most recent guess.

From Section 2.8 it is clear that the diagonal elements of the form factor matrix are strictly positive, i.e., $M_{ii} = 1$ for all i. This allows for computing an entry of the solution vector using all other entries:

$$B_i = \frac{B_{ei}}{M_{ii}} - \sum_{j \neq i} \frac{M_{ij}}{M_{ii}} B_j = B_{ei} - \sum_{j \neq i} M_{ij} B_j \tag{2.37}$$

Jacobi iteration

The Jacobi iteration takes each element B_i in turn and computes the solution using the current guess. When a full iteration step is finished the current guess gets updated and is used in the next iteration. Using the superscript notation, Equation 2.37 then becomes:

$$B_i^{(m)} = B_{ei} - \sum_{j \neq i} M_{ij} B_j^{(m-1)} \tag{2.38}$$

Due to the assumption that the vector containing the current guess remains constant during one iteration all entries in the solution vector can be updated simultaneously. However, this can become a costly process especially in the radiosity setting. Typically only a few patches like the light sources or the primary reflectors, contribute essentially to the visual appearance of the solution. Updating all patches of the scene in each iteration step thus is often not necessary.

Gauss-Seidel iteration

To obtain a faster convergence rate, updated entries of the solution vector can be used in the same iteration to compute new entries. The Gauss-Seidel method does not update the current guess at once, but permanently uses the same vector and updates each element in place as soon as it is available. Thus, to compute the k-th entry of the solution vector the $k-1$ entries of the current iteration and the $n-k$ entries of the last iteration are used:

$$B_i^{(m)} = B_{ei} - \sum_{j=1}^{i-1} M_{ij} B_j^{(m)} - \sum_{j=i+1}^{n} M_{ij} B_j^{(m-1)} \tag{2.39}$$

The Gauss-Seidel method can physically be interpreted as an energy gathering technique. Each step of an iteration updates one patch with the radiosities received from all other patches in the environment. As explained in Section 2.6.1 the radiosity equation actually describes the radiosity of a single patch in terms of reflected radiosities from all other patches. Thus, stepwise evaluation of Equation 2.17 in a loop over all patches directly corresponds to the Gauss-Seidel relaxation scheme. Figure 2.8 shows pseudo-code for a radiosity algorithm using this method.

```
/* initialize starting guess */
for (each i)
   B_i = B_ei
while(not converged) {
   /* loop over all patches */
   for (each i)
      B_i = B_ei + ρ_i Σ_{j=1,j≠i}^{n} F_ij B_j
   /* display current solution */
   render(B);
}
```

Figure 2.8 Pseudo-Code: Gauss-Seidel relaxation.

The vector of emissivities is used as the first approximation of the solution. Each full iteration, i.e., each loop over all patches simulates at least one reflection of light through the environment. While the Jacobi iteration always computes a single reflection, the Gauss-Seidel scheme can simulate several reflections in a single iteration which is the reason for its improved convergion speed. Thus, the first loop considers the emitters only, while each further loop incorporates subsequent reflections which corresponds to evaluating the Neumann series given in Equation 2.14. A convergence criterion for the radiosity algorithm is given by the maximum

gain of energy during one iteration. Before updating B_i the old entry must be compared to the new value and the difference is saved if it exceeds a previously stored one. Once the maximum gain drops below a user supplied threshold, the relaxation is considered to be converged.

In a single step of an iteration, the algorithm accesses one row of the form factor matrix to compute a single entry of the solution vector. In a full iteration all $\mathcal{O}(n^2)$ form factors are used and, once computed, can be re-used in the subsequent iterations. Therefore, this kind of radiosity algorithms is called *full-matrix* method, because it requires storing the complete form factor matrix in memory. Two reasons make this approach very questionable: the size and the sparsity of the matrix. A scene comprising 10,000 polygons would require about 380 megabytes just to store the matrix. Additionally, in realistic scenes many entries of the matrix are zero due to occlusion or because patches are facing away. The algorithms that will be introduced in Section 2.10 attack both problems, i.e., they do not store the full matrix and more important they will avoid computing unnecessary form factors, thereby drastically reducing the complexity of the radiosity method.

2.9 Reconstruction

Once a radiosity solution is obtained an image has to be created and rendered to the screen. Recall the rendering pipeline from Section 1.2. If the lighting step is exchanged by a radiosity algorithm the (possibly hardware accelerated) rendering pipeline can be used to display the solution. Due to the assumption of a constant radiosity per patch the result will be flat shaded. The single radiosity value per patch is only scaled to the luminance range of the monitor and then directly used as the color. This approach exhibits strong discontinuities between neighboring patches. The lower left image of Figure 2.5 shows a flat shaded image of a radiosity solution. A smooth shading as seen in the lower right image is much more appealing and gives the image a photo-realistic look.

Obtaining a smoothly shaded image from a standard radiosity solution requires the reconstruction of the radiosity function at each visible surface point. This can be done by either interpolating the computed constant radiosities or by exactly determining the radiance at each visible surface point by recomputing the point-to-polygon form factors to the environment.

2.9.1 Bilinear Interpolation

The most commonly used approach to generate a continuous shading from the constant radiosity values is interpolation. It has the great advantage of being available in hardware graphics accelerators and all standard graphics libraries and is known as *Gouraud* shading [Gou71]. A Gouraud shader requires color values at the vertices of each polygon and interpolates linearly (for triangles) or bilinearly (for quadrangles) the color values for the interior points. Thus, from the patch colors obtained by the radiosity algorithm the corresponding vertex colors have to be generated. Figure 2.9 illustrates the computation of vertex radiosities. For interior nodes the radiosity is just the average of the radiosities from the adjacent patches. Nodes along the border can be treated analogously, however, extrapolation can give better results. Using a virtual node x between nodes b and e, the radiosity at x would be $B_x = \frac{B_1+B_2}{2}$. The extrapolated radiosity at b

then evaluates to $B_b = B_x + (B_x - B_e) = B_1 + B_2 - B_e$. To obtain valid color values the resulting radiosities must be clamped to zero when using extrapolation.

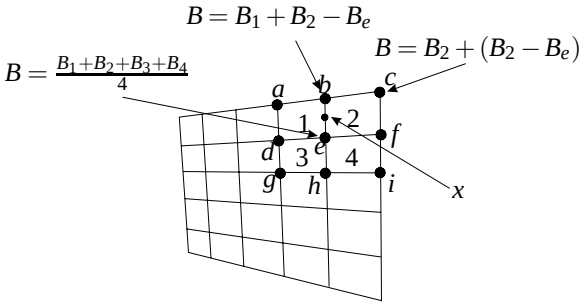

Figure 2.9 Computing vertex radiosities.

Although radiosity images look better when smoothly shaded no additional information was added. In fact the interpolation might introduce severe artifacts. The radiosity function is in general not linear, which results in different gradients over neighboring patches. Due to the eye's sensitivity for first derivative discontinuities this can easily be noticed. The interpolation of vertex radiosities should obviously be done in object space. Gouraud shading, however, is a scanline technique which works in screen space. The interpolation is carried out after the nonlinear projection of the vertices to the screen in horizontal scanlines. Thus, a perspective correction should be applied during interpolation which is only available on some graphics accelerators [CW93]. Further shading artifacts are introduced by T-vertices in the mesh. If the radiosity mesh is subdivided adaptively the degree of subdivision between adjacent patches can be different. The Gouraud interpolation over the coarse element will result in a different color value at the T-vertex then the actual color value obtained from the radiosity algorithm. To avoid T-vertices elements can be triangulated before rendering, thereby connecting T-vertices with corners of adjacent patches [BMSW91] (see Figure 2.10). If the subdivision level of neighboring elements diverges by more than one, additional elements must be created which might degrade the rendering performance.

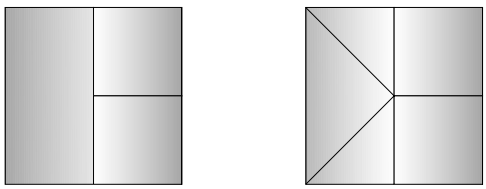

Figure 2.10 Eliminating T-vertices.

2.9.2 Radiosity Textures

Using a feature found on most modern graphics accelerators the draw-backs of bilinear interpolation using Gouraud shading can effectively be reduced. Instead of the radiosity mesh a texture

containing the same information is applied to the untesselated polygon which is then rendered using texture mapping hardware.

Texture mapping belongs to the last step of the rendering pipeline and is performed during rasterization. This allows for shading each pixel individually by looking up a color value in the corresponding texture map. Exactly like Gouraud shading the process is carried out in screen space. A perspective correction however, is standard on modern graphics adapters providing texture mapping support.

In [BGB97] the authors give an algorithm that converts an adaptively subdivided mesh in a texture map. The biggest advantage of this approach is the elimination of T-vertices. The idea is to create a texture with a resolution corresponding to the finest subdivision level of the polygon. Each entry of the texture map then receives the vertex color of the corresponding vertex of the radiosity mesh. Entries without a corresponding vertex belong to regions of a coarser subdivision. This is exactly where T-vertices occur. The empty texels can be filled by interpolating from the neighboring entries, thereby eliminating the T-vertices. Figure 2.11 illustrates this process. Numbers correspond to computed vertex radiosities and letters represent interpolated radiosities. The polygon was subdivided twice which results in a 5x5 texture.

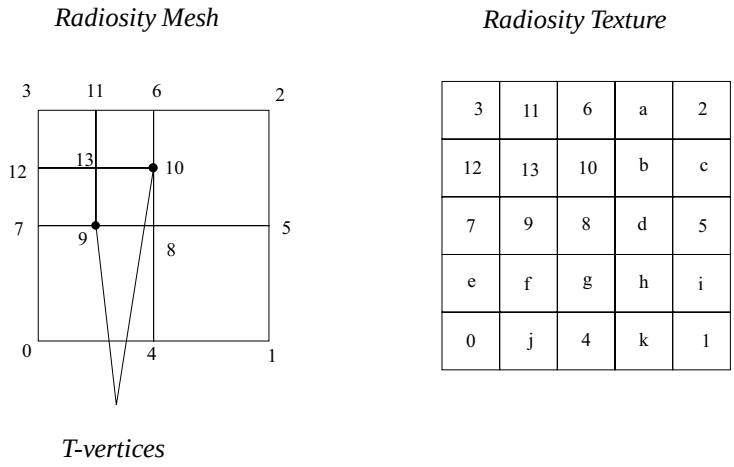

Figure 2.11 Constructing radiosity textures (after [BGB97]).

Once the textures are defined they are mapped onto the original scene polygons. This process dramatically reduces the number of vertices that have to be processed by the graphics hardware. A bilinear filter which corresponds to the Gouraud shading discussed above smoothly 'stretches' the texture over the domain of the polygon. In Figure 2.12 radiosity textures are compared to standard Gouraud interpolation for a mesh containing T-vertices.

To achieve smoothness not only in the value of the radiosity function (C^0 continuity) but also in its first derivative (C^1 continuity) Bastos et al. propose a bicubic reconstruction and filtering [BGB97]. The vertex radiosities are treated as control vertices of bicubic patches and the missing colors are obtained by interpolating the bicubic patch. The display of the texture is done using bicubic filtering available on high-end graphics hardware[2].

Another reconstruction technique to generate radiosity textures has been proposed in [KSS97].

[2]The SGI RealityEngine2 provides an OpenGL extension that allows for bicubic filtering of texture maps.

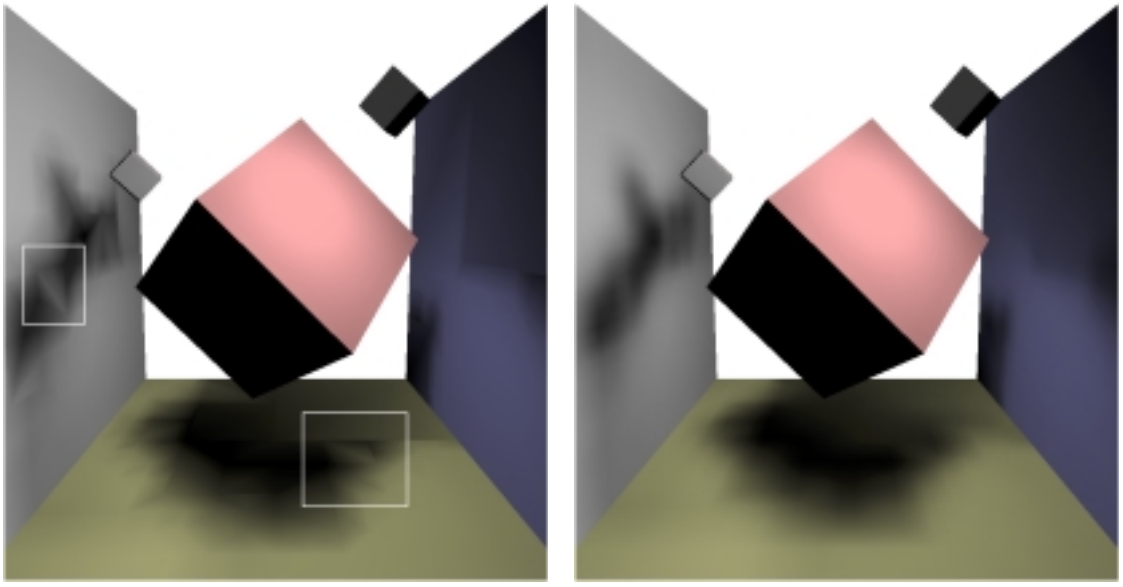

Figure 2.12 Reconstruction methods. The left image shows a Gouraud shaded image from a mesh containing T-vertices. Notice the artifacts in the marked regions. In the right image the meshes from the left wall and the bottom are replaced by radiosity textures, which results in a higher visual quality.

The authors successfully apply binary subdivision schemes used for subdivision surfaces [Loo87] to reconstruct the radiosity function over a patch with high visual quality.

2.9.3 Pixelwise Reconstruction

If the goal is to achieve a single high-quality image of a radiosity solution more effort can be spent on the reconstruction. The interpolation methods described above are capable of rendering multiple images per second, making them suitable for realtime walkthroughs. The time for pixelwise reconstruction can easily exceed the time spent for computing the radiosity solution but it can also enhance the image by view-dependent effects like mirror reflections.

Similar to the ray-tracing algorithm (Section 1.4.1) rays are traced through each image pixel into the scene. If an object is hit, the form factors from the surface point to the environment must be determined. To compute the correct radiosity for the given pixel, the radiosities from all contributing surfaces is gathered and the radiance reaching the eye is determined (Figure 2.13). During the standard radiosity algorithm data structures associated with the patches can be filled to allow quick access to the contributing patches. This process is also known as *final gathering*. The Hierarchical Radiosity algorithm that will be explained in the next sections provides a natural way to maintain the required data structures.

The pixelwise reconstruction method can also be considered as a second pass of the radiosity algorithm. The first pass computes the view-independent global illumination due to diffuse reflectors. In the second pass view-dependent effects like highlights or mirror reflections can be incorporated [Rei92]. The shading model of the second pass uses the diffuse colors obtained

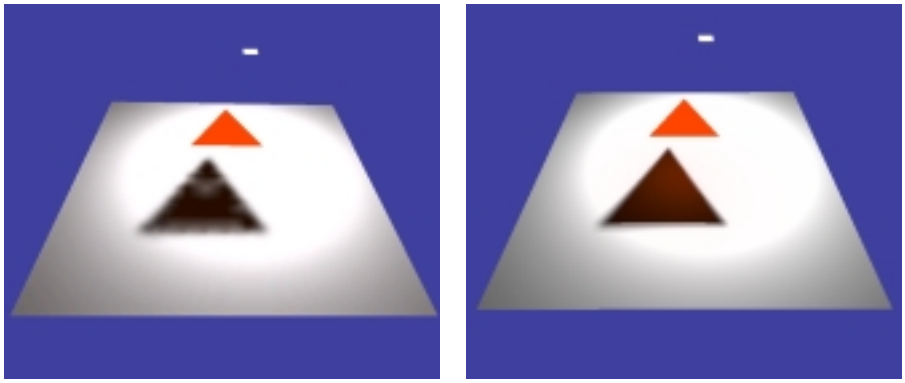

Figure 2.13 Pixelwise Reconstruction. The left image shows severe artifacts at the edges and inside of the triangle's shadow. Pixelwise reconstruction in the right image done via ray tracing completely eliminates these shading artifacts. Note the reddish color bleeding in the shadow due to reflection off the triangle's underside.

from the radiosity solution and spawns reflection rays to add the specular contribution. This can easily be achieved by a standard ray tracer.

2.9.4 Textured Surfaces

To add more realism to a scene texture maps are often applied to object surfaces [JB76]. If a radiosity solution of a textured scene is required the algorithm has to incorporate the texture detail during the energy exchange and while performing the reconstruction step.

To account for the surface texture during the solution an average texture value is computed for each patch [CCWG88]. This value is then used as the reflection parameter ρ_i to compute the average radiosity of the patch. In the rendering step however, the original texture in its full detail should be used. Using the entries of the texture map the irradiance of the textured surface can be modulated to compute a correctly lit image of the texture map. Recall from Section 2.1 that irradiance is reflected from a sending patch and converted to radiosity at the receiving patch. This conversion is performed by multiplying the irradiance by the local reflection operator ρ_i:

$$B_i(x) = B_{ei} + \rho_i(x)E_i \Rightarrow E_i = \frac{B_i - B_{ei}}{\rho_i} \tag{2.40}$$

Once the irradiance is obtained from a textured patch using Equation 2.40 the pixelwise radiosities must be reconstructed. Again, this is done by multiplying with the reflectivity that is actually stored in the texture map. Each pixel has to be multiplied with the irradiance of the patch, an operation which is supported by texture mapping hardware[3].

The method described above first computes an average radiosity for a textured patch which is then converted to irradiance to avoid accounting for the reflectivity twice. A more straightforward way to incorporate textures in a radiosity solution is to store irradiances with each patch

[3]Texture mapping hardware typically supports two modes of operation: *modulate* and *decal*. In the modulate mode each pixel of the texture is multiplied with the fragment's color before rendering. In decal mode the texture is directly applied, thereby overwriting the previous color.

Figure 2.14 Radiosity with textures.

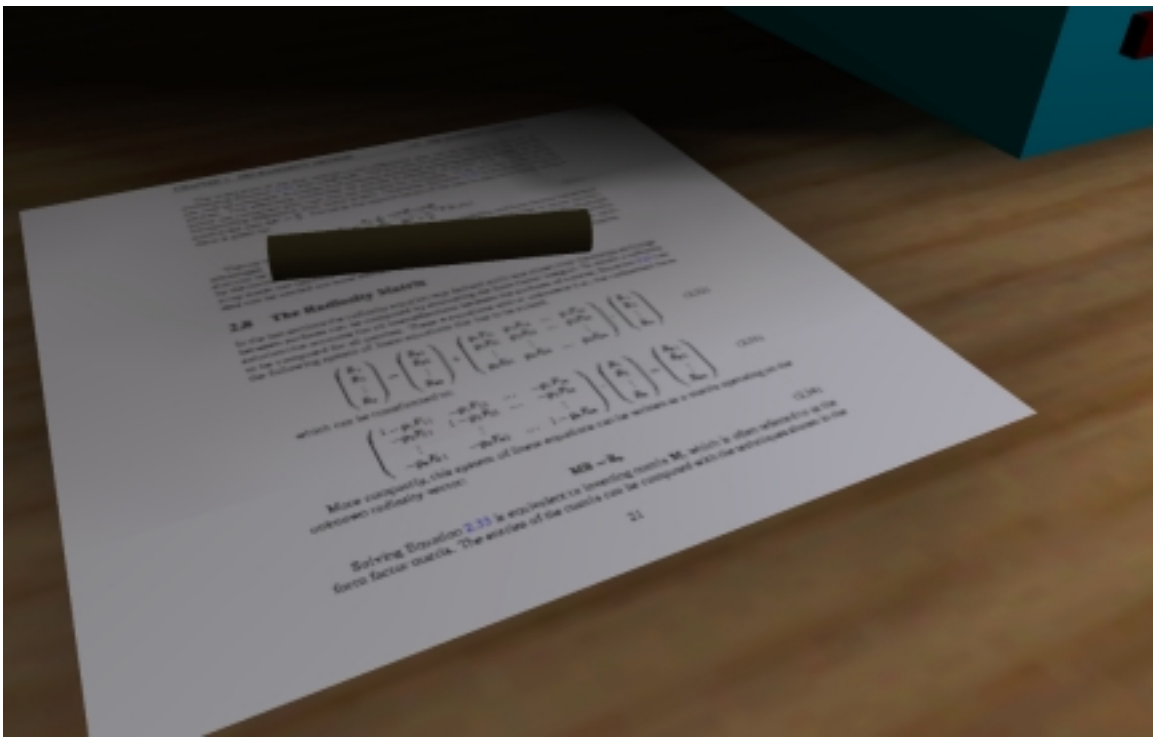

Figure 2.15 Closeup of Figure 2.14 with a high-resolution texture.

instead of radiosities [GH96]. Additionally, the average reflectivity of a patch should be computed after each subdivision which results in more accurate local reflection operators. Texture maps can also be applied to emissive surfaces. The emissivity of the patch is replaced by an average value like before in the case of textured reflectors. Because light sources are typically modeled with a zero reflectivity this approach does not interfere with the described algorithms. Figure 2.14 shows a radiosity solution of a textured environment. The monitor has an emissive texture and the material of the desk and the floor is modeled by a reflective texture. A closeup of the scene is displayed in Figure 2.15, showing the high-resolution texture that models the paper on the desk.

2.10 Advanced Radiosity Algorithms

As pointed out, the full-matrix solution technique for the radiosity matrix presented in Section 2.8.1 has several disadvantages and quickly becomes impractical when applied to realistic scenes. The following sections describe radiosity algorithms that more efficiently solve the linear system in order to obtain correct images. The progressive refinement approach (Section 2.10.1) will use another relaxation technique that is superior to Gauss-Seidel iteration in terms of memory consumption and convergence rate. The *hierarchical methods* described in Sections 2.10.3 and 2.10.4 will reduce the number of form factors that have to be calculated. Actually, they isolate blocks of entries in the form factor matrix that can be approximated by a single value.

2.10.1 Progressive Refinement

The Gauss-Seidel iteration could be used to solve the radiosity matrix due to its diagonally dominance. To find a more efficient solution technique for the special case of the radiosity setting it is important to note, that the form factor calculations influence the overall computation time the most. A relaxation technique that probably reduces the number of form factors that have to be computed is the Southwell relaxation. While the Gauss-Seidel technique could be interpreted as an energy gathering technique, the Southwell relaxation actually reverses this process. Instead of relaxing each residual in turn, the row with the largest residual will always be selected. This can be interpreted as a shooting technique, where a single patch shoots its energy to the rest of the environment (Figure 2.16).

In [CCWG88] this solution technique was used to implement the *progressive refinement* radiosity algorithm: Additionally to the current radiosity estimate B_i, each patch maintains an *unshot radiosity* ΔB_i, representing the residual entry r_i. The algorithm then always selects the patch with the greatest residual (i.e., unshot radiosity) as the current source. After computing form factors from the source to the patches of the environment the radiosity is shot to the other patches, thereby updating their radiosity estimates and unshot radiosities. The source's unshot radiosity is set to zero afterwards.

This algorithm has two advantages over the full matrix solution using Gauss-Seidel iteration. Because each iteration updates the whole environment it makes sense to display the current solution after each relaxation step. This gives the user a visual control over the simulation and allows for an interruption if the quality of the current solution is already satisfactory. The other

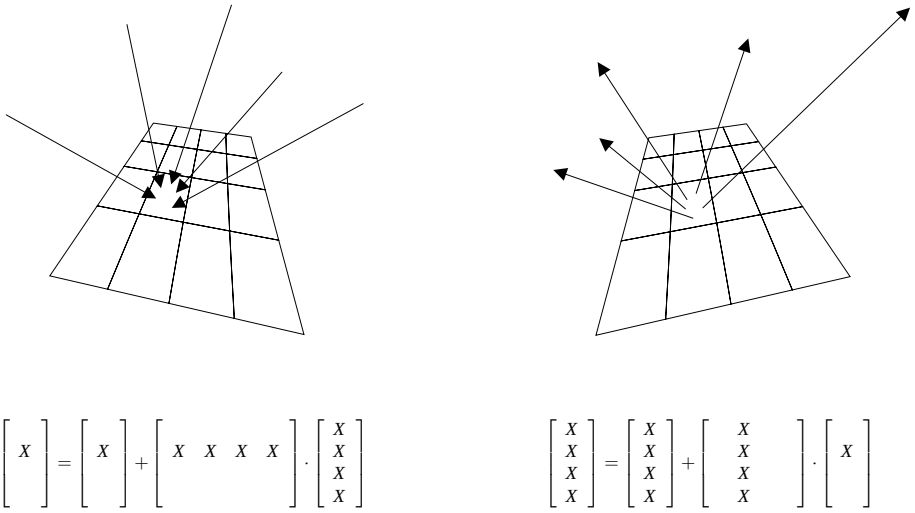

Figure 2.16 Gathering vs. Shooting. The left side shows the gathering process and the update of a single entry of the solution vector. The shooting process shown in the right side updates the whole environment at once.

advantage is the reduced memory consumption of this algorithm. The full matrix of form factors need not be present in memory to perform an iteration step. Each step only requires the n form factors from the source to all receiving patches, i.e., one column of the form factor matrix. This column is computed in each iteration, and has to be recomputed if a source is selected to shoot several times. Although this requires several form factors to be computed multiple times the savings in memory clearly justify this approach. On the other hand, form factors to patches that will never be selected as shooters, are not computed at all.

Ambient Correction

The improvement in convergence speed of the Southwell relaxation is due to always selecting the 'brightest' patch for shooting. However, in the beginning of the simulation the radiosities on the receiving patches (which make up most of the scene) tend to be rather dark. It takes several iterations until interreflections reach patches that are not directly visible from the primary light sources. Cohen et al. [CCWG88] remedy this situation by adding an *ambient correction term* to the radiosities for display. This additional term accounts for the interreflections of light in the environment that have not been computed so far and it is reduced after each iteration.

The ambient term is computed by simulating the effect of future interreflections that depends on the total unshot energy and the average reflection characteristic of the environment. The average unshot energy, which is the area weighted sum of all residual terms is given by:

$$U = \frac{\sum_{i=1}^{N} \Delta B_i A_i}{\sum_{i=1}^{N} A_i} \tag{2.41}$$

The reflection characteristic of the environment is expressed by the sum of the area weighted patch reflectivities:

$$\rho_{ave} = \frac{\sum_{i=1}^{N} A_i \rho_i}{\sum_{i=1}^{N} A_i} \tag{2.42}$$

Similar to the physical explanation in Section 2.4, the influence of the average reflectivity drops off after each reflection, resulting in the following term for the reflection factor:

$$R = 1 + \rho_{ave} + \rho_{ave}^2 + \rho_{ave}^3 + \ldots = \frac{1}{1 - \rho_{ave}} \tag{2.43}$$

Combining these equations gives an estimate on the total amount of radiosity a patch will receive during the simulation, i.e., the average of the unshot radiosities multiplied by the average reflection factor. This ambient correction term is added to the patch radiosities just for display purposes (Equation 2.44). It must not be used to update the radiosities of the simulation which are stored in the patches.

$$B_i^{disp} = B_i + \rho_i R U \tag{2.44}$$

After each iteration U gets updated and finally goes to zero while the radiosity solution process converges.

2.10.2 Substructuring and Adaptive Subdivision

The quality of a radiosity solution based on a regular mesh heavily depends on the degree of subdivision that is applied to the initial surfaces. The radiosity function has to be approximated good enough by the constant elements to allow for a reconstruction step free of artifacts as described in Section 2.9. A regular mesh has to be very fine to capture all shadows and other variations in the radiosity function. In regions without shadows however, this high accuracy is not needed but only increases time and memory consumption of the algorithm.

Reducing the size of a patch, and thereby increasing the total number of patches, should be restricted to the receiving patches. The higher resolution is needed to capture the local details of the illumination. When acting as a sender however, the higher resolution will not severely change the illumination of a distant patch. In fact, an average value over several subdivided patches will be accurate enough.

This observation led to an algorithm that uses a two-level hierarchy to distinguish between the high accuracy needed when receiving energy and a lower accuracy when emitting or reflecting energy [CGIB86]. The scene is first subdivided into a mesh of patches. These patches are then further refined to elements which are stored with each patch. When a patch is about to receive energy, the finer resolution is used to increase the accuracy of the radiosity representation. Because the initial patches and not the elements are used as senders, the number of iterations that have to be performed is reduced. This was the first radiosity algorithm, that used multiple (i.e., 2) levels of detail for the representation of the radiosity function. The hierarchical radiosity algorithm (Section 2.10.3) extends this idea to a full hierarchy of multiple levels over each input polygon.

The problem with the simple patch-element approach is that the mesh has to be created *a priori*, i.e., in advance. An adaptive mesh refinement, that subdivides the elements of a patch automatically as needed, results in a much better simulation. Changes in the radiosity function

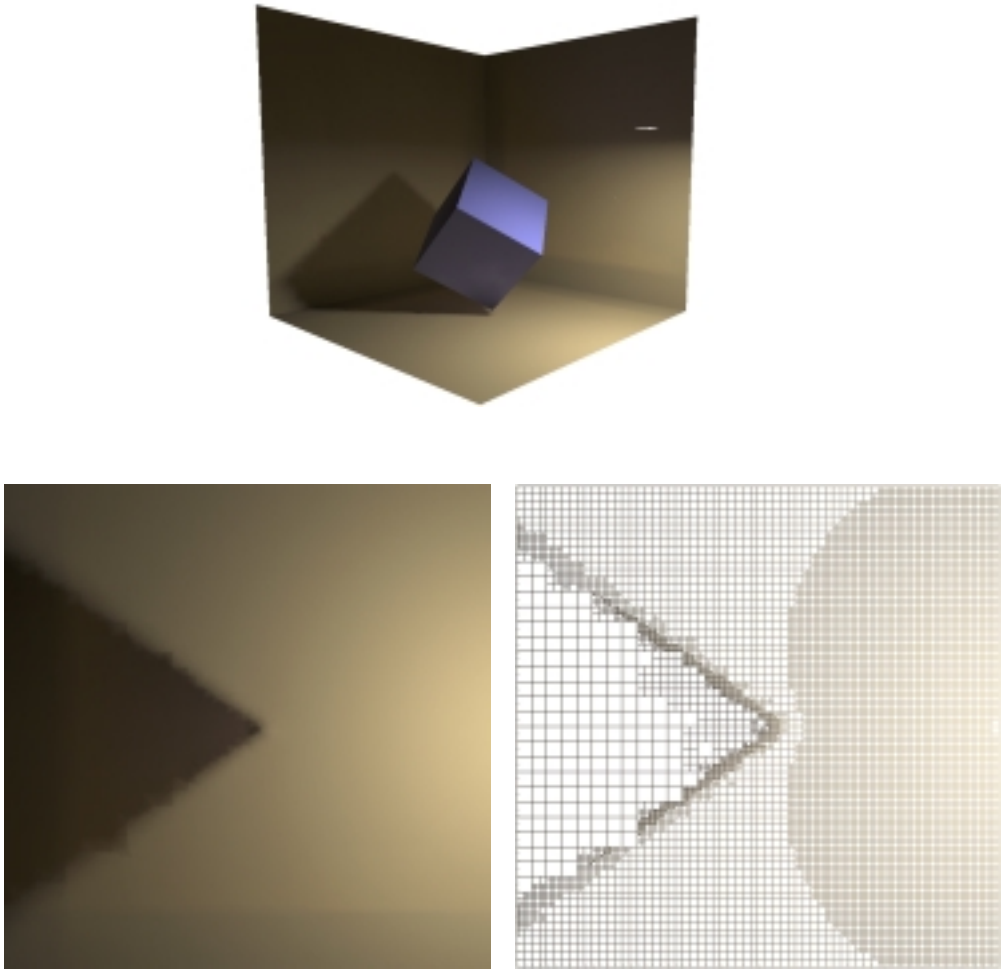

Figure 2.17 Adaptive subdivision. A cube is illuminated from above and casts a shadow on the floor and the walls of a room. The shadow boundaries and the changes in the radiosity function are captured by an adaptive subdivision scheme. The lower pictures show the floor polygon of the simple scene.

over a surface can not be predicted in advance, thus the mesh of elements has to be refined when more information of the radiosity function is available. A quadtree data structure can be used to represent the subdivision of a patch into elements. This data structure also allows for a simple error estimation of the radiosity function. Because the local neighbors are easily accessed in a quadtree, the radiosity gradient over a patch can be determined and used as a subdivision criterion [CGIB86]. As long as the radiosity gradient over any patch exceeds a given threshold the patch is subdivided and a new iteration is started. Figure 2.17 shows a simple scene, where the mesh has been created during the simulation by an adaptive subdivision scheme.

2.10.3 Hierarchical Radiosity

The combination of substructuring and adaptive subdivision led to one of the most important radiosity algorithms called *Hierarchical Radiosity* [HS90, HSA91]. Inspired by ideas to solve the

N-body problem, receivers and emitters are substructured into several levels of detail, resulting in a multilevel hierarchy for each input polygon (Fig. 2.18).

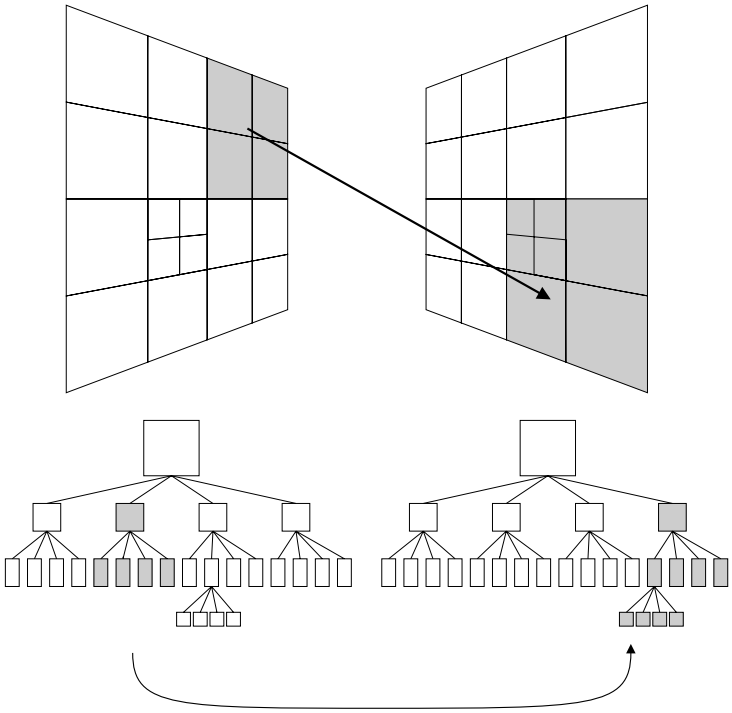

Figure 2.18 Quadtree interaction. Polygons are substructured and interact at an appropriate level. The corresponding quadtrees are shown below.

The N-body problem is the problem of computing the reaction of n objects or particles to the gravitational forces that each of them excerts on the other $n-1$ particles. In order to avoid computing the $\mathcal{O}(n^2)$ interactions, a clustering approach was made, that groups several particles to a single object. This was motivated by the observation, that the accuracy to compute distant forces was typically much higher than the accuracy of the representation of the model. Because the gravitational force drops off with the squared distance (exactly as in form factor computations), most of the $n(n-1)/2$ possible interactions hardly influence the computable solution and can be approximated using much fewer and coarser interactions.

The biggest similarity between the N-body problem and hierarchical radiosity is the use of multiple levels of detail to compute interactions with a given accuracy. The construction of the hierarchy, however, is quite different. Polygons are not clustered to bigger entities but large input polygons are subdivided during the algorithm in a top-down manner. The extension to a full clustering algorithm for radiosity was developed later and will be explained in Section 2.10.4.

The ideas of the N-body problem can easily be transferred to the radiosity setting as demonstrated in the following example. Consider a room with a table and a small object like a book on that table. To compute the shadow that is cast by the book on the top of the table, the resolution of the table polygon must be very high to capture all details. The illumination of a distant wall due to the table, however, will probably not change even when the book is moved. In this case

the table polygons need only be represented by a coarse resolution. Thus, depending on the distance and the direction of the interaction, several representations of the same polygon should be used to accurately model the flow of light.

Hierarchical Refinement

In contrast to earlier radiosity algorithms, input scenes for Hierarchical Radiosity do not need to be subdivided in advance. Starting point for the algorithm is the set of untesselated polygons that describe the scene objects. The subdivision hierarchy associated with each input polygon (Figure 2.18) is created during the solution process as needed. This is performed by comparing all input polygons with each other and, due to some refinement criterion (the *oracle*), subdividing them in a quadtree-like manner. The goal of the subdivision step is to establish *links* between pairs of polygons that can be used for the energy transport in the radiosity solution phase. The refinement criterion guarantees that the amount of energy that is transported over each link is nearly the same for all links. This ensures that the accuracy of the solution is well balanced.

The subdivision process can be formulated by a recursive procedure that is called for each pair of input polygons. If a link can be established at the current level the function returns after creating the link. If the refinement criterion requires a subdivision one of the input polygons is subdivided and the function is called recursively, this time trying to establish links between the other polygon and the new childnodes. A user supplied area threshold A_ε guarantees that polygons are only subdivided to a certain degree. Figure 2.19 shows pseudo-code for the hierarchical refinement procedure.

```
refine(p, q)
{
   if(oracle(p,q) == OK or
           (area(p) < Aε and area(q) < Aε)) {
      link(p,q)
   } else {
      if(subdiv(p,q) == p) {
         // p was subdivided
         for all children c of p
            refine(c,q)
      } else {
         // q was subdivided
         for all children c of q
            refine(p,c)
      }
}
```

Figure 2.19 Pseudo-Code: Hierarchical Refinement.

The link function computes the form factor from p to q by one of the methods presented in

Section 2.7 and appends a link node consisting of the form factor and a pointer to the sender to a list that is maintained by the receiver. Links that were established between leave nodes of two hierarchies directly correspond to entries in the form factor matrix. They represent the energy exchange between two single elements in the mesh. Links that connect arbitrary nodes in the hierarchies therefore represent groups of interactions. These groups of interactions are approximated by a single average value (i.e., the form factor stored with the link) and can be interpreted as blocks in the form factor matrix. Figure 2.20 shows the result of the refinement algorithm when applied to two perpendicular polygons.

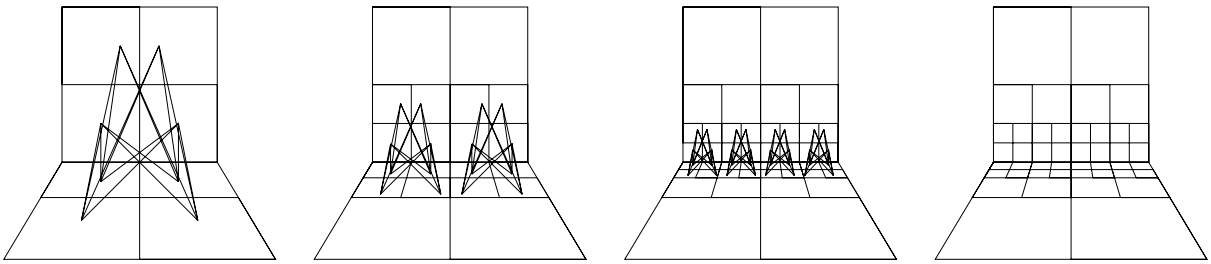

Figure 2.20 Hierarchical subdivision and links at various levels. (after [HSA91])

The Oracle

The refinement criterion or *oracle* that decides if a link between two polygons should be established at a certain level must be designed carefully. It is obviously responsible for the quality of the solution and, as can be seen in Figure 2.19, it is called very often thereby influencing the total running time of the algorithm. In order to balance the energy transport over all links, the oracle proposed in [HSA91] computes an approximation of the form factor by evaluating the geometrical expression inside the form factor integral (defined in Equation 2.20):

$$\frac{\cos\theta\cos\theta'}{\pi r^2} \tag{2.45}$$

The oracle is actually used in the algorithm as an error estimator. If the error induced by creating a link at a given level exceeds some user supplied threshold F_ε the corresponding polygons must be subdivided in order to reduce the error. The oracle given in Equation 2.45 computes a rough approximation of the unoccluded form factor which is fast to evaluate but also inaccurate. This value is then directly used as a measure of the error of the simulation. Because large form factors do not always indicate large errors in the energy transport, unnecessary subdivisions might be introduced.

Oracles that compute bounds on the error of the radiosity transfer have been successfully used by [GH96] and [SSS96]. In section 2.10.4 the use of error bounds will be discussed and its application in hierarchical clustering algorithms will be shown.

Solving the system

During the refinement procedure an element mesh and interacting links containing form factors are created. The radiosity system can now be solved using the Jacobi or Gauss-Seidel iteration

method. For each polygon the list of links is traversed and energy is gathered over the incoming links. This gathering procedure is performed repeatedly until a convergence criterion is met.

When radiosity is gathered over a link the current radiosity approximation of the sending element is multiplied with the form factor stored in the link and weighted by the receiving element's reflectivity. Because each element is part of a hierarchy representing a single input surface however, the gathered radiosities can not directly be stored. Radiosities gathered at any level of the hierarchy must be propagated through the complete hierarchy to guarantee that the correct radiosity values are stored with each node. Because radiosity has units of power per area the radiosities gathered at a certain level can directly be added to the nodes of the next level. This *pushes* down the radiosities to the leave nodes of the hierarchy where the leave's own gathered radiosity and emissivity (if available) are added. To correctly update the inner nodes of the hierarchy, the radiosities of the child nodes must be averaged on the way back to the root. From the form factor properties derived in Section 2.6.1 directly follows, that the radiosity of a parent node is the area average of its children radiosities. The upward way therefore *pulls* radiosities from the leaves to the root which gives the complete traversal the name *push/pull* process.

The hierarchical radiosity algorithm can now be formulated as follows: In an initial linking step all pairs of input polygons are linked to create a starting point for the recursive link refinement procedure. Once a network of links is established the iterative solution process starts gathering radiosity over the links and subsequently propagates the radiosities throughout the hierarchy using the push/pull procedure described above. The gathering and push/pull procedures are repeated until convergence. Separating gathering and propagation through the hierarchy corresponds to the Jacobi iteration where the solution vector is updated only after a full iteration. Using the faster Gauss-Seidel technique for hierarchical radiosity is discussed below. Figure 2.21 shows pseudo-code for the full algorithm. The gathering and push/pull procedure is given in Figure 2.22.

```
/* initial linking */
for (each polygon p)
   for (each polygon q)
      if(p != q)
         refine(p,q);

/* solving */
while (not converged) {
   for (each input polygon p) gather(p);
   for (each input polygon p) pushpull(p, 0);

}
```

Figure 2.21 Pseudo-Code: Hierarchical Radiosity.

```
gather(p)
{
   p.B_g = 0   /* initialize gathered radiosity */
   for (all incoming links l) {
      p.B_g = p.B_g + l.formfactor * l.q.B_s * ρ_p
   }
   for (each child c of p)
      gather(c);
}

pushpull(p, B_down)
{
   if(p has children) {
      B_up = 0
      for (each child c of p)
         B_up = B_up + A_c/A_p * pushpull(c, p.B_g + B_down)
   } else {
      /* p is leave node */
      B_up = p.E + p.B_g + B_down
   }
   p.B_s = B_up
   return B_up
}
```

Figure 2.22 Pseudo-Code: Gathering- and Push/Pull-Procedure.

Analysis

To compute the complexity of the algorithm, the number of links must be counted. Each link represents one interaction and requires the calculation of a form factor. For a full matrix solution this would result in a complexity of $\mathcal{O}(n^2)$. The form factor matrix that results from the hierarchical refinement, however, has fewer than n^2 blocks. As can be seen in Figure 2.20, the number of links leaving a patch is the the same for all subdivision levels. Due to the oracle, which compares the form factor estimate for a link with a constant threshold, the form factor stored with each link is bounded by F_ε. The sum of all form factors associated with a leave element must equal 1 as pointed out in Section 2.8, thus the form factors for a leave node and all its ancestors is approximately equal to the constant $1/F_\varepsilon$. Because each patch does not interact with all other patches but a constant number dependent on F_ε the number of blocks in the form factor matrix is proportional to n. Thus, the complexity of the hierarchical radiosity algorithm is $\mathcal{O}(n)$, which is asymptotically better than previous radiosity approaches.

It is important to note, that the above analysis relates the complexity of the algorithm to n, the number of links which is proportional to the number of elements in the finite element mesh. The initial linking step however, compares all pairs of input polygons before the solution process

starts. This results in a total complexity of $\mathcal{O}(k^2+n)$, with k being the number of input polygons. From this observation it is clear, that scenes containing mainly large polygons are best suited for the hierarchical radiosity algorithm. Polygons that are too small to be further subdivided quickly turn the algorithm in a quadratic algorithm similar to the full matrix solutions discussed earlier. This situation arises for examples when curved surfaces are part of the scene. To approximate the geometry of a curved object, many small polygons are needed. These polygons typically will never be subdivided by the refinement procedure, resulting in a 'worst-case' input. A solution to this problem will be presented in Chapter 3.

Improvements

BF-refinement The adaptive refinement induced by the oracle function results in high subdivision levels in regions of large form factors. Due to the $1/r^2$ term in the form factor, this is typically the case where polygons are close to each other or share a common edge. The idea of the oracle function is to balance the energy transport over all links, i.e., the error should be the same for all transport paths. A more accurate error estimate can be achieved when the radiosity that is transported over a link is also considered. Thus, large errors in the form factor can be tolerated if the amount of transported energy is small. The algorithm outlined above must be changed slightly to incorporate this BF-refinement (radiosity times form factor) strategy. Because the radiosity that will be transported over a link is not known in advance, a progressive algorithm must be used. The initial linking now just links the input polygons but does no hierarchical refinement. This can be achieved by using a temporary large area threshold in the first pass to prevent polygons from subdivision. In the following loop energy transport and refinement are performed alternately. The refinement procedure checks existing links for the amount of transported energy and possibly refines them by deleting the current link and creating finer links in the next level. After the next energy exchange all links are tested again and so on until all links transport approximately the same amount of energy. Pseudo-code for the modified main loop of the hierarchical radiosity algorithm is given in Figure 2.23.

Multigridding During the progressive BF-refinement, it is also possible to gradually change the threshold BF_ε which is used by the oracle to determine if a link should be refined. The algorithm starts with a large error threshold and once the system is solved, the threshold is reduced and the system is solved again, using the links from the last pass and refining them where necessary. This *multigridding* technique first computes a coarse solution and progressively refines the radiosity mesh to the desired accuracy. The result is a more balanced mesh and the solution often converges more quickly because the coarse solution which can be obtained in short time is a better starting point for a more accurate link refinement than the original mesh.

Gauss-Seidel iteration The original Hierarchical Radiosity algorithm [HSA91] used the Jacobi iteration to solve the linear system. Gathering and push/pull was formulated in two separate passes which requires more iterations to converge. In [Gib95] a single pass solution is formulated that updates radiosities in place which corresponds to a Gauss-Seidel iteration. Radiosities are gathered and directly pushed down the hierarchy from the current level. At the

```
tmp = A_ε
A_ε = ∞
/* initial linking */
for (each polygon p)
   for (each polygon q)
      if(p != q)
         refine(p,q);
A_ε = tmp

done = FALSE;
while(done == FALSE) {
   /* solving */
   while (not converged) {
      for (each input polygon p) gather(p);
      for (each input polygon p) pushpull(p, 0);
   }
   done = TRUE;
   /* refinement */
   for all links l
      if(refineLink(l) == FALSE)
         done = FALSE;
}
```

Figure 2.23 Pseudo-Code: BF-refinement.

leave nodes the emittance is added and the radiosities are pulled up using the area averaging as described for the push/pull procedure. The pseudo-code for the Gauss-Seidel gathering procedure is given in Figure 2.24.

2.10.4 Clustering

The analysis of the hierarchical radiosity algorithm resulted in a time complexity of $\mathcal{O}(k^2+n)$. Thus, if k^2 is much bigger than n, the costs of the initial linking step are predominant which makes the algorithm unusable for scenes containing many small polygons. By extending the hierarchical radiosity algorithm to object hierarchies and allowing energy exchanges between objects or *clusters* of objects to take place, the linking costs could be lowered dramatically [Sil95]. Links that had previously to be established between all pairs of surfaces and that often transported only a small amount of energy can be avoided by using cluster links. This is especially useful when objects are well separated. As soon as objects are close enough, the original hierarchical radiosity algorithm is used and links are established between single polygons or mesh elements to provide a sufficient accuracy. Figure 2.25 shows an example of cluster links and polygon links between two objects.

```
gather(p, B_down)
{
   for (all incoming links l) {
      B_down = B_down + l.formfactor * l.q.B * ρ_p
   }

   if(p has children) {
      p.B = 0
      for (each child c of p) {
         gather(c, B_down)
         p.B = p.B + c.B * A_c/A_p
      }
   } else {
      /* p is leave node */
      p.B = p.E + B_down
   }
}
```

Figure 2.24 Pseudo-Code: Gathering procedure using Gauss-Seidel iteration. (after [Gib95])

Volume Form Factors

The idea of Sillion's clustering algorithm is to treat object clusters as volumetric objects that scatter light isotropically. An object that contains a number of polygons with arbitrary orientation can be seen as a volume with an *extinction coefficient* κ, that describes the reduction of light when it travels through the volume due to absorption and scattering. The reduction of light due to scattering alone is called the *albedo*. The *transmittance* of a medium along a path describes the fraction of light that is neither absorbed nor scattered and is given by:

$$\tau(s) = e^{-\int_0^s \kappa(u)du} \tag{2.46}$$

The power that is emitted by an isotropic volume element k can be computed using the *equivalent area* of a volume [RT87]:

$$P_k = 4\kappa_k V_k B_k \tag{2.47}$$

The term $4\kappa_k V_k$ is used analogously to the area of a surface element when computing the power emitted by a surface, thus it can be seen as the equivalent area of a volume element. On the other hand, if a volume V is given that contains a number of polygons with the total area A, the extinction coefficient κ can be computed by:

$$\kappa = \frac{A}{4V} \tag{2.48}$$

Using volume scattering methods, the energy exchange between volumes and surfaces can be computed very similarly by the standard radiosity formula:

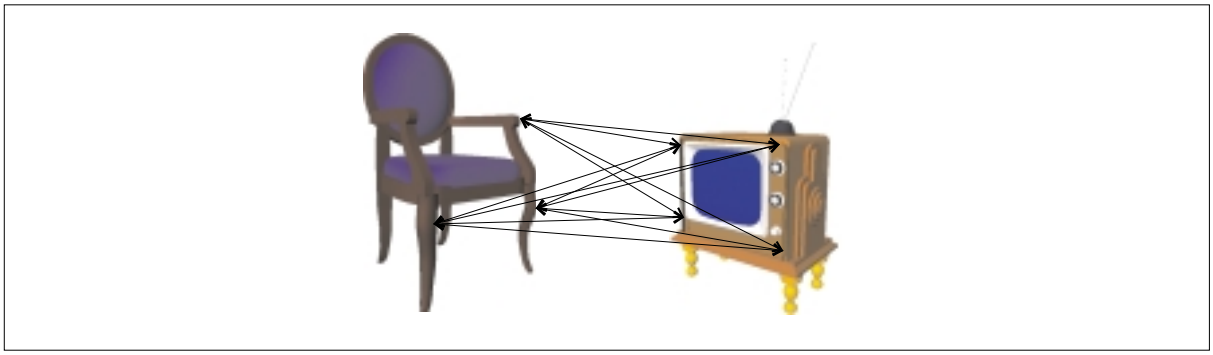

Figure 2.25 Cluster links and polygon links. Links for well separated objects can be represented by a single cluster link that must be refined for closer objects.

$$B_i = B_{e_i} + \rho_i \sum_{j=1}^{N} F_{ij} B_j \tag{2.49}$$

where ρ_i denotes the albedo for volumes and the reflectance for surfaces respectively. Form factors between the possible combinations of volumes and surfaces can be calculated using the formulas given in Table 2.2.

$F_{row-column}$	Surface j	Volume k
Surface i	$\frac{1}{A_i}\int_{A_i}\int_{A_j}\frac{\tau\cos\theta_i\cos\theta_j}{\pi r^2}\,dA_j dA_i$	$\frac{1}{A_i}\int_{A_i}\int_{V_k}\frac{\tau\kappa_k cos\theta_i}{\pi r^2}\,dV_k dA_i$
Volume m	$\frac{1}{V_m}\int_{V_m}\int_{A_j}\frac{\tau\cos\theta_j}{4\pi r^2}\,dA_j dV_m$	$\frac{1}{V_m}\int_{V_m}\int_{V_k}\frac{\tau\kappa_k}{4\pi r^2}\,dV_k dV_m$

Table 2.2 Form factors between volumes and surfaces. (after [Sil95])

Monte Carlo integration as described in Section 2.7.2 can be used to evalute the form factor integrals, where the sample points are distributed over the surface or inside the volume. Using these formulas the hierarchical radiosity algorithm can be extended to volumes. Links are now

established between volumes and/or surfaces to be able to transport energy between hierarchy levels above the input polygons. Link refinement is also driven by an oracle function that evaluates the current error of energy transfer.

Link refinement

In contrast to the surface algorithm where a polygon is never linked to itself, the clustering algorithm uses self-links for clusters to represent the energy exchange that may occur inside the cluster. In fact, the whole algorithm starts with a self-link of the top-level cluster enclosing the whole scene geometry. This results in the most important advantage compared to the surface based hierarchical radiosity algorithm: the quadratic initial linking phase is eliminated and replaced by a single self-link that is refined when the algorithm proceeds. The refinement of a self-link requires linking of all pairs of children inside the cluster, where child clusters receive a self-link for future refinement (see Figure 2.26). The refinement of a self-link can be performed in constant time, thus the complexity of the hierarchical radiosity algorithm using clustering drops from $\mathcal{O}(k^2+n)$ to $\mathcal{O}(k+n)=\mathcal{O}(n)$.

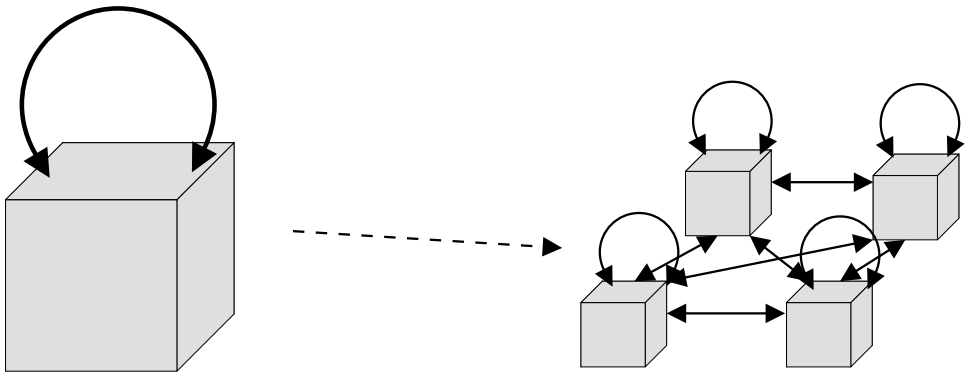

Figure 2.26 Refinement of self-links.

Hierarchy Creation

The clustering algorithm relies on a hierarchy above the input polygons. To automatically group polygons into clusters that can be hierarchically refined, several techniques can be used. In [Sil95] using an axis-aligned *k-d* tree or an octree is proposed. Hierarchies of bounding volumes [GS87] are also possible but care must be taken to find the optimal place to insert new surfaces into the hierarchy to minimize the overlap of bounding volumes [Gib95]. In Chapter 5 this topic will be discussed in more detail and the application of a new scene structuring algorithm for radiosity clustering will be presented.

Related work

Several authors have worked on the subject of radiosity clustering improving its usefulness for very large radiosity computations even more. Different linking techniques, the use of error bounds and the extension to non-diffuse surfaces have been proposed.

Linking Smits et al. [SAG94] developed their clustering algorithm simultaneously to Sillion but without the ideas from the volume scattering method. They present a different linking scheme that comes at a higher cost but may have a better accuracy than the isotropic volume assumption. When linking two clusters which contain m and n surfaces respectively, the goal is to avoid creating $m * n$ polygon links as required by hierarchical radiosity. Instead the transfer from all source polygons is averaged by a single value that is distributed to the receiving polygons. The averaging and distributing takes the orientation and visibility of each polygon into account. This reduces the costs of linking to $m + n$ operations and is called α-linking. A more inaccurate and faster approach are β-links that are similar to Sillion's method. By ignoring the orientation of the polygons inside each cluster and assuming a constant distance between the source and receiver polygons, linking between two clusters can be achieved in constant time (Figure 2.27).

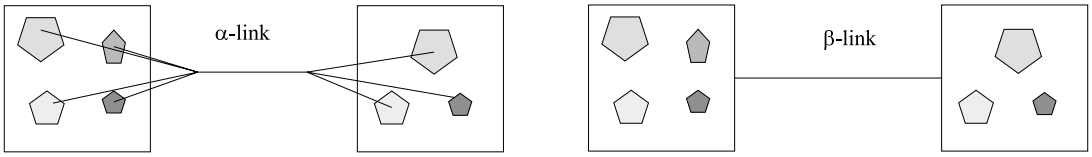

Figure 2.27 α-links and β-links between clusters. (after [SAG94])

Error bounds The BF-refinement strategy relies on the assumption that the size of the error that occurs in an energy transfer is proportional to the product of radiosity and the form factor. A larger value indicates a larger error and results in a subdivision of the sender or the receiver. Computing bounds on the error can result in a better error estimate that leads to fewer or more accurate subdivision. Lischinski et al. [LSG94] used an error bounding technique for hierarchical radiosity that computed upper and lower bounds on the form factor and the radiosity for each transfer. His gathering- and push/pull-procedures treat the upper and lower bound of the radiosity exactly like the 'real' radiosity value to ensure a correct propagation of error bounds through the hierarchy. These ideas were applied by Gibson to radiosity clustering [GH96]. In contrast to [LSG94] where only the variation of the kernel function is considered he extends the computation of error bounds to the visibility term and to the varying reflectance of surfaces due to texture maps. Stamminger used error bounds to efficiently compute the radiosity for general reflectors [SSS97b] by incorporating bounds on the curvature of a surface.

Radiance The clustering technique has also been applied to non-diffuse surfaces. Computing radiance solutions for large environments quickly becomes impractical, due to the complexity of these algorithms. The extension of hierarchical radiosity to non-diffuse surfaces resulted in an $\mathcal{O}(k^3 + n)$ algorithm for the initial linking that only allows a few hundred input surfaces to be used [AH93]. Clustering seems to be a promising approach for this problem. In [SDS95] Sillion extends his radiosity clustering algorithm by storing with each surface or cluster a number of directional distributions representing the radiant properties of the element. To

efficiently store these distributions spherical harmonics are used[4]. Using these data structures and a modified gathering and push/pull procedure leads to up to 3 times more computation time for purely diffuse scenes when compared to the traditional approach. Thus, the integration of non-diffuse reflectors in a clustering algorithm is possible at a moderate increase of costs.

In [CSS96] Christensen et al. combine their wavelet radiosity algorithm with the clustering technique presented by Smits et al. ([SAG94]) to derive a radiance clustering algorithm. Wavelets were successfully applied to the global illumination problem before ([SGCH93, GSCH93, SH94]). The hierarchical nature of wavelet bases suggests the approximation of the radiance function by linear combinations of a wavelet basis and using this representation in the hierarchical radiosity environment. Christensen used a mapping of the directions of the hemisphere to points on the unit square to be able to construct a wavelet basis for radiance using the domain $[0,1]^4$. This avoids the large storage costs induced by the spherical harmonics used by Sillion. To achieve high quality solutions, images are rendered using the final gathering technique as described in Section 2.9.3. The illumination for each surface point is recomputed using all the basis functions that contribute to the radiance solution at that point. The costs involved with this technique however, easily exceed the costs of the radiance solution.

2.11 Summary

In this chapter, the radiosity method was introduced along with several improvements to achieve faster solutions and to obtain a better visual quality. The most important step towards making radiosity practical was the introduction of the Hierarchical Radiosity algorithm and its extension to clustering which allow for the computation of a radiosity solution in linear time.

The algorithms presented here do not make any assumption on the input data. A list of unconnected, arbitrarily oriented, and planar polygons (typically triangles) is used as a description of the environment that is to be rendered. Although clustering algorithms group input polygons to object clusters, no additional information is created.

If the radiosity algorithm had some knowledge of the original *objects* that were tesselated to produce the input polygons, many operations could be implemented more efficiently. A single object, or its description, can be used more easily than the corresponding polygons. If the polygons are only needed at a certain stage of the algorithm it is useless to deal with a large amount of polygons throughout the whole algorithm. If the access to the parent object is always possible, information implicitly stored with the object can be used whenever it may be useful.

The next chapters will introduce radiosity algorithms that use object-based techniques to improve the speed and accuracy of radiosity solutions. In Chapter 3 it will be shown how curved surfaces can be efficiently rendered with the Hierarchical Radiosity algorithm using object information. Due to the quadratic costs of the initial linking step curved surfaces that are represented by many small polygons are impractical to render by the classical approach.

When scenes are too large to be computed in reasonable time on a single computer, parallelization may be a possibility to reduce rendering times. Due to the network of links however, the Hierarchical Radiosity algorithm induces a large communication overhead when distributed

[4] Spherical harmonics basis functions form an orthogonal basis of the set of distributions on the unit sphere and allow the representation of any square-integrable function by a set of scalar coefficients.

over a network of computers. Using object information during the computation can help to drastically reduce the communication costs of the algorithm. An efficient distributed hierarchical radiosity algorithm will be presented in Chapter 4 that scales very well in a local area network of computers.

Finally, a solution to the previously mentioned hierarchy creation problem for clustering algorithms will be proposed in Chapter 5. An optimized bounding volume hierarchy based on a cost function will be used to compute both, visibility and energy transport. Combined with an error driven refinement procedure accurate radiosity solutions can be generated quickly. The curved surfaces algorithm of Chapter 3 will fit easily into this error driven clustering algorithm.

CHAPTER 3

Hierarchical Radiosity on Curved Surfaces

3.1 Introduction

The development of the radiosity method as summarized in chapter 2 led to efficient global illumination algorithms with a common property. Because the physical quantity radiosity can also be computed by other means in order to generate an image, the application of finite element methods clearly characterizes the radiosity method. The ability to render images from any desired viewpoint without essential further computational effort, once a solution has been computed, makes this approach so attractive. Walk-through applications can use the finite element mesh produced by a radiosity computation to directly render images from the solution. The polygons that represent the mesh elements are simply delivered to a hardware renderer where the reconstruction of the radiosity function (Section 2.9.1) and the projection of the three dimensional polygons onto a two dimensional viewing device is performed (Section 1.2). Therefore, radiosity algorithms typically require a polygonal representation of the scene geometry that will be the starting point for the finite element mesh, which in turn is finally rendered by a polygon renderer. In this polygon-based approach, each polygon is dealt with independently, i.e., the origin of each polygon and the connectivity to adjacent polygons is not used. Actually this information is unknown to the algorithm, although it was available when the scene was modeled.

The polygon-based approach works well if the initial tesselation of the scene objects is identical or at least very close to the objects' shape. The advanced radiosity algorithms that employ adaptive refinement further do subdivide the input polygons, but only to improve the representation of the radiosity function (Section 2.10). For lack of object information, errors due to geometrical inaccuracies can not be reduced or eliminated. A polygon can only be subdivided within its planar domain, thereby keeping its initial shape. As a result, the application of these algorithms to curved objects is limited in several ways. Considering the hierarchical radiosity algorithm, the classical way of using curved objects contradicts the idea of hierarchical refinement, thereby destroying the main benefits of the algorithm and making its application useless. This will be clear by revisiting the analysis of the algorithm's time complexity.

The hierarchical radiosity algorithm can be separated in two parts: the initial linking stage and the solution stage. The initial linking is basically a classical radiosity algorithm, where form factors are computed between all pairs of input polygons. Thus, the time complexity is quadratic in their number. The solution stage with its linear time complexity due to the hierarchical refinement is the reason for the algorithm's strength. However, the input data must be chosen carefully to not loose the benefits induced by the hierarchical approach. The algorithm is best suited for input data consisting of relatively few and large polygons compared to the num-

ber and size of the elements of the solution. The polygonal approximation of a curved surface however, is the opposite of this kind of input data. Many small polygons are required to tesselate a curved surface accurately. If too few polygons are taken, the real curvature of the object will not become apparent. Additionally, shadow computations involving curved occluders will be inaccurate due to the low resolution. To avoid these artifacts and because the algorithm can not improve their geometrical approximation, the final quality of curved objects must be chosen beforehand. This also includes the determination of the accuracy of the radiosity representation over their surfaces, which contradicts the error-driven refinement. If the oracle function demands the subdivision of a polygon that represents a part of a curved surface, the visual quality of the solution will not be improved noticeably.

Thus, the application of hierarchical radiosity to curved surfaces results in the following drawbacks:

- quadratic time complexity (many small input polygons)
- no hierarchical refinement (the polygons are too small or the subdivision is useless)
- no adaptive quality control (the mesh resolution is chosen in advance)

These problems can be eliminated if the algorithm deals with curved objects in the same way as with planar objects. To be able to create a hierarchy of several levels of detail regarding the representation of the objects, the *shape* of a curved object's approximation must not be fixed. The initial linking stage needs access to the coarsest representation to efficiently create the top-level links of the patch hierarchies. Subsequent refinement steps then access finer object representations to model the interaction of light with the surface more accurately. Provided that the different levels of detail are given or that they could be generated on demand, there is the problem of finding the proper place in the hierarchy to switch between them. The transition from a coarse sphere to a more detailed one probably introduces too many polygons from one hierarchy level to the next. Thus, in the same way as the hierarchical radiosity algorithm subdivides polygons at nearly arbitrary positions (thereby ignoring the creation of T-vertices), curved surfaces must be refinable *locally*.

A local refinement procedure for curved surfaces is only possible if the algorithm has a knowledge of the underlying object's geometry. If the object information created during the modeling stage is conserved and provided to subsequent rendering algorithms, more efficient image synthesis techniques can be developed. These improvements apply not only to the rendering speed but also to the visual quality of the resulting image.

In this chapter, an object-based refinement scheme for Hierarchical Radiosity will be proposed. Together with a topological data-structure and an object-oriented rendering environment, curved surfaces can be rendered efficiently, bypassing the problems of the classical approach outlined above. Modifications to the energy transport and visibility computations with an inherently high accuracy lead to an improved quality of the resulting radiosity solutions. An object-based final gathering procedure allows for the generation of high resolution images and can easily be combined with other shading techniques, like bump-mapping, that work in object-space.

After giving a motivation to the application of object-based methods, the previous work on rendering of curved surfaces in the global illumination context will be revisited. The remaining

sections describe the chosen data-structure and the details of the new algorithm. Finally, the results will be shown and the application to other global illumination algorithms as well as the integration into a commercial rendering package will be discussed.

3.2 Motivation

As soon as radiosity algorithms are used to visualize *realistic* scenes, the simplifying assumptions inherent to the radiosity approach will become apparent. Scenes that were modeled independently of a special rendering algorithm and that contain objects and surfaces resembling real things are not well suited for polygon-based rendering. The fact that most surfaces resulting from industrial design, like parts of airplanes and automobiles or cases of electronic devices are curved, emphasizes the need for rendering algorithms that take these object properties into account.

The foundation for an object-based rendering algorithm is a rendering framework that uses abstract objects to encapsulate the geometrical properties and materials of the real objects. Most rendering platforms that were developed for research purposes (MRT [Fel96], Vision [SS95], MoCaRT++ [GMP96], ART[1]) follow the object-oriented paradigm and profit from inheritance mechanisms, data encapsulation, and extensibility to integrate existing algorithms or to develop new approaches. The decision to base a rendering system on an object-oriented design becomes clear by looking at the ray-tracing method, which is one of the most powerful and popular techniques for image synthesis. Computing the intersection of a ray with a sphere requires the encapsulation of the sphere's exact geometry in the intersection test routine. The same is true for all other ray-surface intersection algorithms [Gla89]. Using an object-oriented programming language like C++ [Str91], which is probably the most popular one, spheres would be implemented as a single *class* derived from a *base class*. The base class forces all derived classes to provide the implementation of an intersection *method*. Thus, an object of type *sphere* can be asked if and where a given ray intersects this particular sphere.

Object-oriented design is also commonly used in commercial graphics applications. The modeling, animation, and rendering software *3D Studio MAX* from Kinetix uses the concept of objects not only for the encapsulation of scene objects but also to provide easy third-party extensibility [EM96]. Nearly every function is designed as a *plug-in* component that can be replaced and modified. Additionally, changes that are made to objects with the provided modeling tools are kept in a stack of operations, making the original object always available.

From this situation, a promising concept to enhance Hierarchical Radiosity for the efficient use of curved objects can be developed. If modeling and rendering are used in combination and not separated in two distinct processes, an object-based meshing scheme can be realized. With the knowledge of their parent object, curved surfaces can be refined locally to adaptively improve a radiosity solution. The acceptance and broad availability of object-oriented design in digital image synthesis suggests and supports this approach.

[1] http://www.cg.tuwien.ac.at/research/rendering/ART/

3.3 Previous Work

Several papers in the radiosity literature have addressed the problem of rendering curved objects by global illumination algorithms. Enhancements of form factor calculations between curved patches have been proposed by Bao and Peng [BP93]. All surface patches are subdivided into triangles and a variation of the radiosity function over the triangle is allowed, thus breaking with the constant radiosity approach. After computing delta form factors to each vertex of a source triangle a bilinear interpolation scheme is applied. The radiosities are computed at arbitrary sample points, thereby taking the true patch geometry into account. Another approach describes the direct evaluation of form factors from points to B-spline surface patches [BP94]. Here, spatial polygons are used instead of planar ones. The algorithm tries to find a subdivision of the original curved patches into curved triangles that subtend the same solid angle as the source patch. Form factors are then calculated to these pseudo-triangles by an analytic formula. The subdivision of the curved patches is done by subdividing the boundary curve instead of subdividing planar polygons that approximate the curved patch.

A similar approach has been presented by Nishita and Nakamae [NN93]. They assume that all curved objects are represented by bicubic Bézier surfaces. Subdivision always results in Bézier patches and not in planar polygons. The form factor itself is calculated by contour integration whereas visibility calculation is done by a scanline algorithm using Bézier clipping. The same scanline algorithm is used for the display step, resulting in high precision renderings. Jones et al. [JCC+93] mainly focus on the image rendering step and can effectively reduce the number of required subdivisions. They introduce an elaborate ray-casting algorithm that computes the radiance for each pixel by sampling and adaptive anti-aliasing. This algorithm only uses a coarse mesh and is able to render curved surfaces and shadows by accessing the original scene geometry during form factor calculation and final rendering.

To even further reduce the meshing step or making it obsolete at all, results from the finite element literature have been applied to solve the global illumination problem [Zat93, TM93]. The Galerkin radiosity method uses higher order polynomials instead of constant basis functions to express the radiosity of a surface. A technique is presented which uses radiosity coefficients instead of radiosities during energy transfer. Curved surfaces can directly be incorporated if an object can compute its surface normal at any given intersection point during kernel sampling. Shadows have to be represented by texture like shadow-masks, because high radiosity gradients near shadow edges are difficult to represent by piecewise smooth basis functions. The final rendering is done in a ray-casting step.

A similar approach to the one presented in this thesis was developed independently by Stamminger et al. [SSS97a]. By computing the cone of normals for a given object, which is an indicator of the variance of the surface normals, curved objects could be integrated in hierarchical radiosity computations. This technique also focuses on a strong object-oriented design and thus can be considered most similar in spirit to this work. If an object is capable of efficiently computing its cone of normals, it can be integrated in the radiosity algorithm. As will be shown later, our approach requires no additional methods to be implemented for each object. In general, the intersection test and the computation of the surface normal, methods which are the foundation for any ray tracer, will suffice [Sch97].

All of the presented algorithms make use of the original underlying scene geometry to incorporate curved objects, thereby achieving high accuracy solutions. However, most of the

algorithms require a view dependent rendering step based on ray tracing [Zat93, JCC+93] or scanline processing [NN93] or they require the overall use of computationally expensive free form surfaces [BP93, BP94]. The only hierarchical algorithm was introduced in [SSS97a].

This chapter presents a new approach for the incorporation of curved objects into hierarchical radiosity algorithms, thereby using advantages of the methods summarized above. The key element will be a three-dimensional extension of adaptive meshing. By accessing the real object geometry local improvements of the polygonal approximation of curved objects can be made at any stage of the algorithm. The accuracy of the final solution is scalable and hardware accelerated rendering can be used to speed up image display. This makes the method well suited for walk-through applications. High quality still images can be generated by an object-based reconstruction step using final gathering.

3.4 Topological Data Structures

To store the polygonal model of a geometric object, several data structures can be used. The simplest form is to define a polygon by the coordinates of its vertices that are arranged in a list of k entries. An object is then defined by a list of n polygons. Because the boundary of an object is closed however, many vertices are shared by adjacent polygons. Thus, storing each vertex separately wastes memory. If this data structure is used to directly render the object by a polygon renderer, it is not only memory but also bandwidth that is used inefficiently. Each vertex is sent multiple times through the rendering pipeline where computationally intensive transformations and lighting calculations are performed. A simple way to reduce the unnecessary computations is the application of triangle strips [Sil93]. A triangle strip is a sequence of adjacent triangles that belong to an object's polygonal approximation. Because each triangle in a strip is the neighbor of its predecessor it shares exactly two vertices with it. Thus, to define the next triangle at any point in the strip a single vertex is sufficient. Inside of a strip, each vertex is stored and sent only once. Hardware support for this data structure is available on most modern graphics processors. Although this technique reduces memory and bandwidth limitations, there are still vertices that are processed multiple times. In general, an object can not be represented by a single triangle strip. As soon as more strips are used, the same vertices appear in neighboring strips. Because no information about their adjacency is available, the vertices are treated separately.

In a *topological* data structure full adjacency information is maintained for all vertices, edges and faces (i.e. polygons) of an object's polygonal approximation. This guarantees that all edges leaving a vertex, or all faces that share a common edge or vertex can be found efficiently. To define operations that work on the topology of a polyhedron a planar graph can be used. This graph is obtained by embedding the polyhedron into a plane, which is accomplished by extending a single face until all edges can be projected on that face without any intersection (see Figure 3.1). This technique restricts the use of the data structure to polyhedra without self intersections which are objects that are homomorph to a sphere.

Because an edge always connects two vertices but it also always separates exactly two faces, vertices and faces are topologically equivalent. Thus, if the vertices and faces of the graph are exchanged, the dual graph is obtained (Figure 3.1). As a consequence, operations that modify the representation of the polyhedron always occur pairwise. In the same way as a face can be

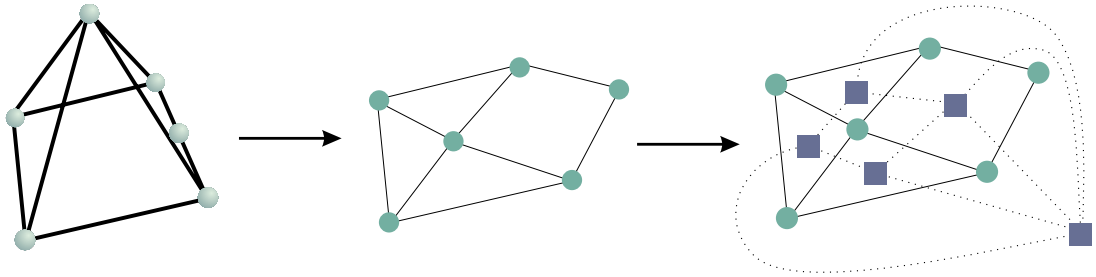

Figure 3.1 Embedding of a polyhedron results in a planar graph. The dual graph is obtained by exchanging the roles of vertices and faces. (after [Ben97])

split by inserting a new edge, a vertex is split into two vertices that are connected by an edge. The reverse operation to a *split* is the *join*-operation. Together, these operations can perform any allowed modification of the graph that results in an object that is still homomorph to a sphere. Thus, an implementation that creates and modifies polyhedra represented by a topological data structure need only support these four basic operations, that are known as Euler operators [Män88]. Because these operators always result in a valid planar graph, their exclusive use maintains the object's topology and gives access to the full adjacency information after any modification of the graph. This can be used to robustly implement various computer graphics algorithms like point reduction to compress polygon meshes or the application of boolean operations on two polyhedra (constructive solid geometry) which is a non-trivial task [Ben97].

The implementation of a topological data structure can be based on the *winged-edge* data structure as proposed by Baumgart [Bau72]. The core of this data structure is an edge that connects two vertices and that provides pointers to the two adjacent faces and to the four adjacent edges leaving the edge's endpoints. This allows for the enumeration of all vertices of a face in clockwise and in counter clockwise direction. In Figure 3.2 the components of the data structure connected to an edge are shown.

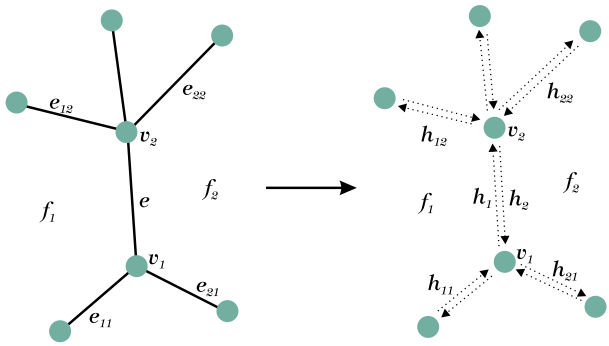

Figure 3.2 Winged-edge data structure. The introduction of half edges guarantees a unique representation of edges. (after [Ben97])

The representation of an edge in the data structure is ambiguous regarding its direction. The same edge can be denoted by the two symmetrical expressions $e = (v_1, v_2, f_1, f_2, e_{11}, e_{12}, e_{21}, e_{22})$ and $e' = (v_2, v_1, f_2, f_1, e_{22}, e_{21}, e_{12}, e_{11})$. If the full edge e is split into two half edges h_1 and h_2

(Figure 3.2), the representation becomes unique and an implementation will be more straight forward. Basing the winged-edge data structure on half edges has an additional advantage when the data structure is used for solid modeling. Normal vectors can be associated with half edges to accurately model sharp object edges. If normals would be associated with vertices it would be impossible to find the correct normal for the corner of the box, which is modeled by a single vertex. The half edge approach enables the normal to be stored unambiguously for the only adjacent face. As will be describe later the radiosity reconstruction step additionally stores color values with each half edge to obtain better shading results.

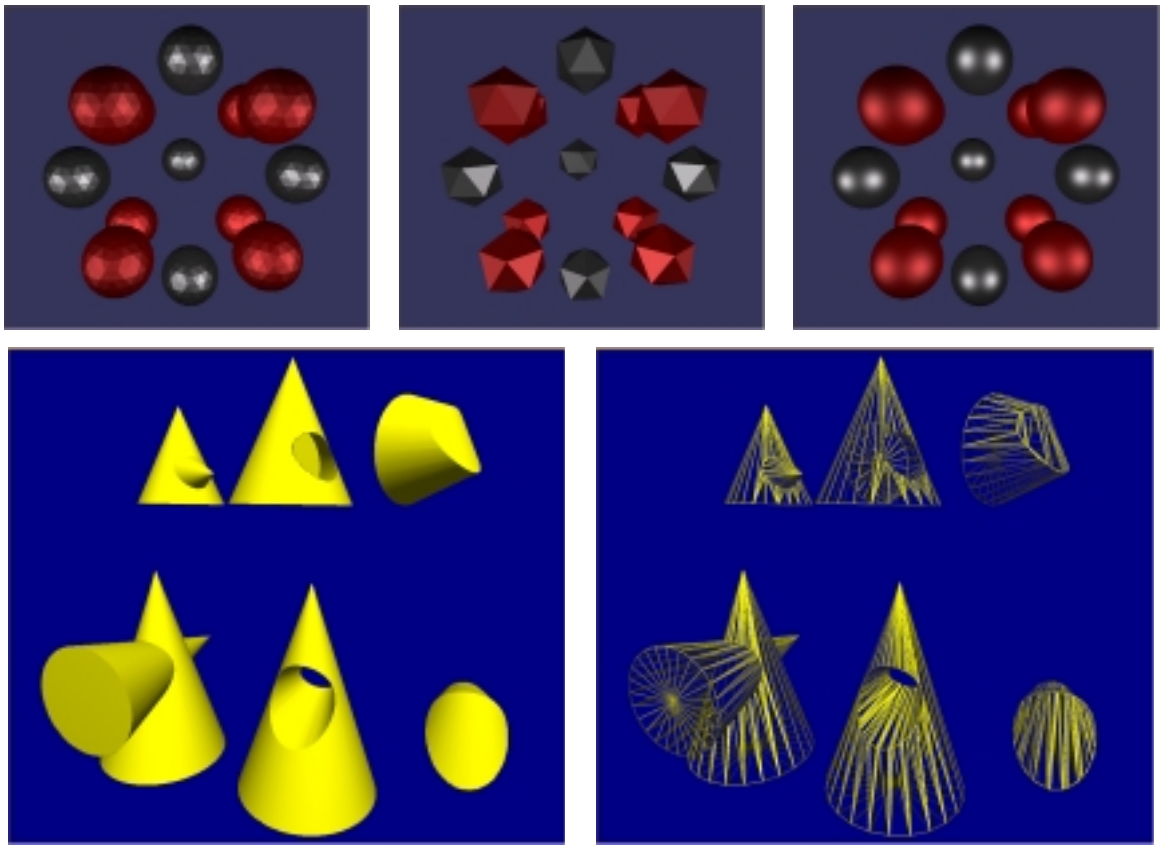

Figure 3.3 Solid modeling using boundary representations. The spheres in the upper row show different levels of detail generated by Euler operators from the initial model in the upper left. A more complex application is the combination of different polyhedra according to boolean expressions.

The winged-edge data structure combined with Euler operators is a powerful tool for solid modeling. Once a valid polyhedron is defined from an object description, various modifications can be applied easily. One application is the computation of different levels of detail. To maintain a constant frame rate in dynamic applications, the resolution of the displayed geometry must be variable. Figure 3.3 shows in the upper row spheres in different levels of detail. For performance reasons, the initial boundary representation (*BRep*) of the spheres is hard coded. Once all vertices and edges are known, the topology is built. Any further modification is performed by applying Euler operators that maintain the topology. In the bottom row an object

modeled by different boolean expressions are shown. The boundary representation of two objects can be intersected, which inserts new edges. In a second step, the boundary representations are cut along the intersection edges and combined to a single object. Each face is then visited in turn to separate the representation in an intersection and a union. The final object can then be composed according to the evaluation of the boolean expression.

3.4.1 Application to Hierarchical Radiosity

The availability of full adjacency information is very useful for radiosity applications [BFS96]. Many steps of the algorithm rely on the accessibility of neighboring vertices or patches. In the subdivision step T-vertices can be detected and avoided and radiosity gradients can easily be computed. Similar to the improvements achieved by using triangle strips for polygon rendering, expensive vertex based radiosity operations can be optimized. Because vertex colors are needed for the final display of a radiosity solution it is often useful to compute radiosities per vertex and not per face. In this case, the form factor computation is restricted to point-to-patch form factors. For vertices that are shared between adjacent planar patches the topological data structure helps to compute the corresponding form factors only once.

The hierarchical radiosity algorithm as described in Section 2.10.3 is typically implemented using a quad-tree data structure. Each initial polygon is associated with a quad-tree of which the root node is the polygon itself. The hierarchical refinement generates additional quad-tree levels by regularly subdividing the root polygon into four children. Further iterations will possibly subdivide the children as illustrated in Figure 2.18. This subdivision scheme is easy to implement, but it favors the use of polygons with quadrilateral shapes. A quad-tree stores adjacency information implicitly. The four children of a node are always arranged in a fixed order that directly defines the access to neighboring patches. These patches however, must reside in the same tree. As soon as the boundary of the initial polygon is reached, the neighbors to a patch can not be enumerated completely. This has an impact on the quality of the reconstructed solution. When linear interpolation is used for the reconstruction (i.e., gouraud shading) only the interior of a subdivided polygon provides all required radiosity values. In border regions the values that are unknown have to be guessed. Although some heuristics like mirroring the values from the interior of the polygon can provide reasonable results, the eye is very sensitive to gradient changes. As a result, the boundaries of the initial polygons are still visible.

The operations performed by the hierarchical radiosity algorithm during energy exchange and subdivision are independent of adjacency relationships. Because the oracle measures the error of an interaction and not the error of the radiosity representation, radiosity gradients are not used. Adjacency information is only used in the reconstruction step.

The application of a topological data structure thus can be limited to the lowest level of the hierarchy. An hierarchical topological data structure, that would be difficult to maintain, is not needed. Instead, a simple data structure that mainly stores the radiosity values is used at each node. A pointer is provided that points to the next level if it is an inner node. The leaf nodes of the hierarchy, i.e., the finest representation of the input polygons, are connected to the corresponding faces of the winged-edge data structure (Figure 3.4).

Combining a light-weight hierarchy with a powerful data structure has several advantages. The hierarchy is not limited in its branching factor. Any number of children can easily be maintained in any node, which supports the use of arbitrarily shaped input polygons. Because the

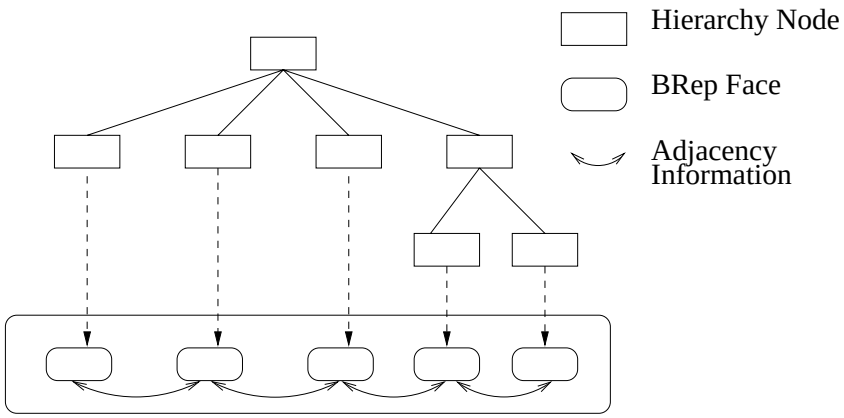

Figure 3.4 Hierarchy on top of a winged-edge data structure. Adjacency information is only needed in the leaf nodes.

subdivision of a patch can only occur in a leaf node, Euler operators that work on the winged-edge data structure can be used. This helps to separate the geometrical parts of the algorithm from the radiometric parts inside the hierarchy, which makes an implementation more robust and easier to maintain. Additionally, the only step where adjacency information is actually needed is now fully supported because all polygons of the original object's boundary representation are directly accessible.

3.4.2 Meshing

The hierarchical refinement procedure subdivides patches to improve the representation of the radiosity function. Although the refinement of links occurs in any level of the hierarchy, patch subdivision is only performed in leaf nodes. Depending on the shape of the root polygon, different subdivision schemes must be provided. Two levels of a regular subdivision for quadrangles and triangles are illustrated in Figure 3.5.

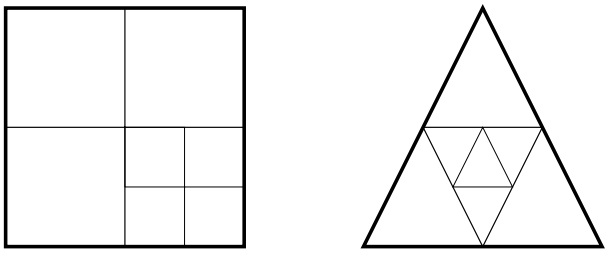

Figure 3.5 Regular subdivision scheme for quadrangles and triangles.

Refinement of a leaf node is performed by subdividing the corresponding face from the boundary representation. For each created face a new leaf node is attached to the hierarchy. The child pointers of these nodes are initialized with pointers to the corresponding faces. The child pointer of the original node is then redirected to point to the new child nodes.

The face subdivision requires a split operation for each enclosing edge and the insertion of edges to define the new faces. In a topological data structure splitting of an edge influences both neighboring faces. Vertices introduced by earlier split operations should be used as anchor points for new splits. The problem is how to decide quickly if the proper anchor point already exists or if a new vertex must be inserted by splitting the edge. Just testing if the midpoint of the current edge coincides with an existing vertex can lead to numerical problems and requires some calculations. The quickest way is to tag each face and vertex with the current subdivision level. Before the meshing starts, all tags are reset. If a face with subdivision level l has to be split all vertices with level $l+1$ serve as anchor-points. New vertices of level $l+1$ are inserted where two adjacent vertices of level $\leq l$ occur. The resulting faces have level $l+1$. Figure 3.6 illustrates this meshing scheme for a quadrilateral shape.

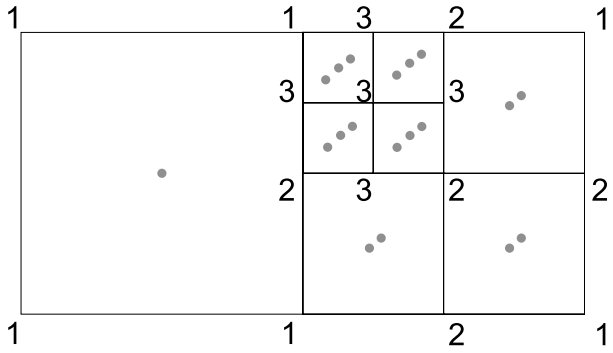

Figure 3.6 Meshing scheme using level tags.

By using this technique, a meshing scheme appropriate for hierarchical radiosity can be realized. The problem of T-vertices however, which is common to any unrestricted meshing scheme (Section 2.9.1), can occur at shadow boundaries. To achieve a smooth transition between a finer subdivided illuminated region and a coarser subdivided shadow region, a T-vertex elimination must be applied. Using adjacency information this can easily be achieved. By examining the level tags, introduced during the meshing, T-vertices are detected and then connected to the nearest corner of the adjacent face.

3.4.3 Weighted Reconstruction

The situation for a reconstruction step using Gouraud shading is slightly different when using a topological data structure. Instead of well separated polygons that are independent on each other, the winged-edge data structure describes the boundary of the complete object. During the radiosity calculation, constant radiosity values per patch are assumed, i.e., each face of the final boundary representation receives a single color. For the purpose of Gouraud shading however, a color value in each vertex is needed that can be interpolated over the corresponding polygons. A simple approach, which works well for planar environments, computes the vertex radiosity as the average of the adjacent faces' radiosities. This achieves good results inside a subdivided polygon, but at the boundary no information about the neighboring polygon is available. Vertices lying on the boundary thus can receive different color values depending on the current polygon. If the approximated object is not planar but curved, the problem gets even

worse. The different normal of a direct neighbor can result in a high gradient of the radiosity function, which is probably missed.

An obvious method for computing vertex radiosities while making use of the underlying topological data structure, is the adaption of the above algorithm. Because the boundary representation is closed all vertices can be enumerated and the radiosities of all adjacent faces can be averaged. Solids containing many input polygons thus receive better shading values. But another problem arises. In the polygon based approach, a box for example would be modeled with six independent polygons. Each interpolation would only consider the patches making up one side. If the box is modeled as one consistent object however, the simple interpolation scheme fails. Radiosity values would be smoothed over the corners and edges of the box, thus introducing severe artifacts if one side lies in shadow and an adjacent side is lit.

The introduction of topological data structures above motivated the use of half edges to store surface normals. To improve the quality of the reconstruction step for complex objects, the same approach is taken here. Originally vertex based color values are stored with each half edge. For all adjacent faces of a shared vertex, one half edge starts and one half edge ends in that vertex. If the colors are stored in these half edges, different vertex colors for the same vertex are possible. The calculation of vertex colors and the final rendering are thus edge based. A face is always asked for all edges which in turn point to the shared vertex but contain the vertex color for the current face. This technique is also used for storing the normals, which are located in the edges for the same reason.

Using edge colors an algorithm can take into account the spatial position of adjacent patches and store the vertex color in the corresponding edge. The average radiosity of all patches surrounding a vertex must be weighted. By introducing the cosine of the angle between the current edge's normal and the normal of an adjacent edge, the corresponding face color can be weighted. The weighted average radiosity from all neighbor faces is then stored in the current edge (Figure 3.7).

```
for all faces f {
   for all edges e of f {
      B_e = 0
      weight = 0
      for all edges l leaving e {
         weight = weight + max(0, N_e · N_l)
         B_e = B_e + B_face(l) * max(0, N_e · N_l)
      }
      B_e = B_e / weight
   }
}
```

Figure 3.7 Pseudo-Code: Weighted Reconstruction.

This mechanism is easy to implement and results in a smooth shading of curved objects and preserves D^0 discontinuities along boundary edges. Figure 3.8 shows the effect of the weighted reconstruction in comparison to the normal reconstruction of vertex radiosities.

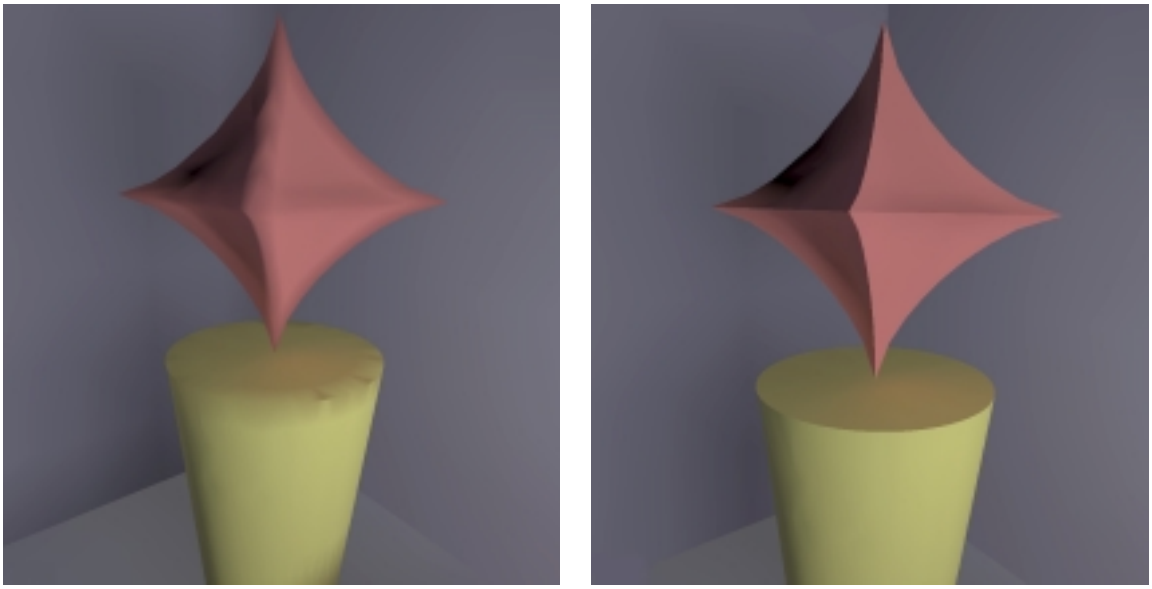

Figure 3.8 Weighted reconstruction for solid objects. The left image shows the standard approach while in the right image the patch orientation was taken into account.

Although the weighted reconstruction step leads to a better visual quality, rendering times for curved objects are still high. The next section introduces a novel meshing scheme that, combined with this reconstruction technique, improves speed and quality of the rendering of curved objects.

3.5 Object-based Meshing

As pointed out earlier, the key to an efficient hierarchical radiosity algorithm for curved surfaces is a local mesh refinement. Similar to the planar approach, a mesh describing an object's complete boundary must be able to refine its representation at any location. As a result, the algorithm can start with a coarse approximation of the scene geometry and refine the representation in order to reduce the simulation error.

Local refinement A refinement procedure that adapts the mesh to the object's shape can be implemented in a two pass approach. The first step is a standard planar refinement that basically refines the domain of the particular patch. For an underlying winged-edge data structure this involves inserting several new vertices and updating the adjacency information to reflect the new topological structure. In a second step, the coordinates of the new vertices are adjusted which moves the vertices to a location on the original object's surface. Figure 3.9 shows the result of this approach being applied to a sphere.

To properly adjust a vertex to its correct position, the knowledge of the underlying object's shape is required. This knowledge is implicitly stored in several methods that determine an object's behaviour in a rendering system. As a requirement for the implementation of an object-based meshing scheme, each object must provide two functions:

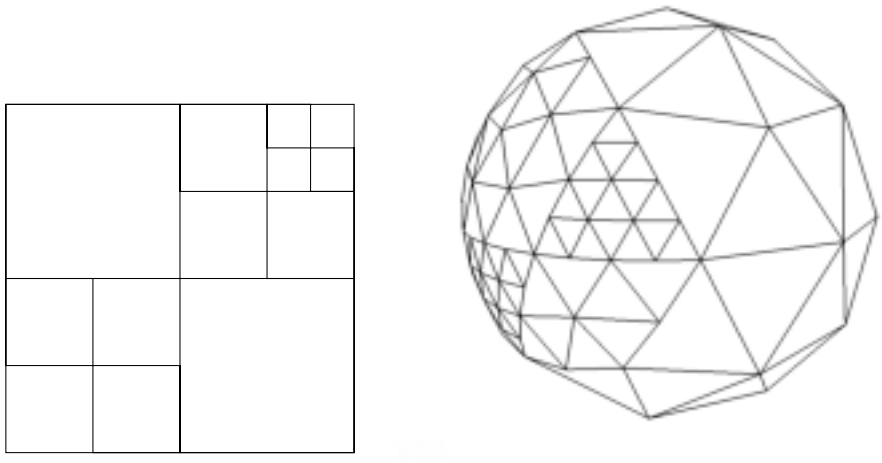

Figure 3.9 Polygon-based and object-based meshing. Curved objects can be refined adaptively like a polygon that is represented by a quad-tree. The representation is improved by moving vertices to the underlying object's surface.

- surface normal computation
- ray intersection test

These methods belong to the core functionality of any rendering package that supports ray tracing. Because ray tracing is also commonly used for visibility test in the computation of form factors (Section 2.7.2), its availability in a radiosity system is very likely. Both functions can be combined to a new *generic* function that, independently on the particular object, moves a vertex to the real boundary. The computation of the surface normal is used to find the direction of the object's hull, whereas the intersection test gives the exact surface position. Thus, to adjust a vertex that is the result of a planar subdivision, a ray is cast from the vertex position into the direction of the surface normal. The result of the intersection test gives the nearest surface point of the original object. Adjusting the coordinates of the vertex to that point moves the vertex to the boundary. As a result, the object's shape is improved and its representation is locally refined.

If the intersection test did not succeed the ray has left the object without hitting its surface. This situation indicates a non convex object and requires the direction of the ray to be reversed. The reason for this behaviour is the way in which the winged-edge data structure is initially filled. All vertices of the data structure are initialized with exact surface points of the geometrical object. These points are then connected with edges to build the faces of the boundary representation. For convex objects, these faces always reside completely inside the object boundary. During vertex adjustment, the faces move outwards. Due to this construction a non convex object has faces outside of the boundary. To approach the refined faces towards the real boundary, the opposite direction of the normal must be used. In Figure 3.10 C++ code for a generic `adjustVertex()`-Function is given. Implementing this method in a C++ base class will provide the functionality of local mesh refinement for all derived objects that support surface normal computation and an intersection test.

```
Object:: adjustVertex(Vertex& v)
{
   Ray ray;

   ray.origin = v.point;
   ray.dir    =  normal(v.point);

   // intersect(ray,pos) stores the first
   //   intersection found in pos
   // if no intersection is found it returns NULL
   if( intersect(ray, v.point) == NULL) {
      ray.dir = -ray.dir;
      intersect(ray, v.point);
   }
}
```

Figure 3.10 The adjustVertex method in C++ syntax. This generic implementation moves vertices onto the real object's boundary.

Optimization The above method can be optimized in several ways and should be regarded as the default implementation for an object class. If the objects provide more functionality, the test for reversing the ray's direction can be improved. The default implementation requires two intersection tests for non convex objects which might be inefficient. A method that directly tests if a given point lies inside of the object is often found in ray-tracing environments to speed up the rendering of boolean or CSG-expressions. According to this method, the direction of the ray can be chosen in advance to eliminate the need for a second intersection test. Once a vertex has been moved to its final position it should be tagged. All vertices that make up the initial boundary representation provide the same tag. This avoids multiple adjustments of the same vertex or the adjustment of vertices that are already in the correct place. Because all vertices are shared in a winged-edge data structure this is very likely to occur.

The most obvious optimization is the definition of an object specific version of `adjustVertex()`. For objects that are not curved the implementation is empty and does not even check the tags introduced earlier. In order to generate a surface point, the u, v-parametrization of an object can also be used. The intersection test for parametric surfaces is typically much more expensive than just evaluating the corresponding equation. Thus, the subdivision is performed in parameter space and the resulting vertices are directly computed at the correct position.

For objects with discontinous surface normals the direction in which to shoot the ray might be ambiguous. The result of the surface normal computation might not be defined if the given point does not already coincide with the object's surface. Objects like cylinders or cones consist of curved and flat regions that, when using a winged-edge data structure, are direct neighbors and thus are sharing common edges and vertices. This helps to detect the necessity of adjusting a vertex. If the vertex lies on the flat part of the boundary no adjustment is needed. Traversing the half-edges that are leaving the vertex in the data structure and checking if they store the

same normal, identifies flat parts of the boundary. A winged-edge data structure should use references to normal vectors to save memory and to avoid numerical errors when comparing them. The test just described thus will only compare memory pointers which is much faster and more reliable than comparing coordinate values. If the test fails, the vertex belongs to a curved part of the boundary and the ray-casting algorithm described above can be used.

Flatness test The hierarchical radiosity algorithm drives its planar subdivision scheme by an oracle that enforces subdivision in regions of large error. To prevent patches from endless subdivision due to conservative error thresholds or due to numerical problems, an area threshold has to be supplied. Patches are only subdivided if their area still exceeds an area threshold A_ε. If curved objects are subdivided with the object-based meshing scheme, using the same area threshold often leads to unsatisfactory results. The area of a patch is not suited as an error measure that describes the quality of a curved object's approximation. Instead the local curvature must be used. An easy way to compute the local curvature of the approximation is to look at the normals stored in the winged-edge data structure for each (half-)edge. The more an object is refined the less the normal vectors around a face diverge. Thus, regarding the normals, the face's flatness increases. To compute the local flatness of the surface covered by the approximating face, the minimal dot product between all adjacent normals can be used. This corresponds to the largest deviation of all considered normal vectors and measures the quality of the approximation. This test is always possible because the original surface normal of an object is stored after each vertex creation or adjustment. An optimization similar to the one proposed above is a separate implementation for non-curved objects. By returning the constant 1.0 they always indicate perfect approximation regarding their local curvature without the need of computing dot products. To speed up frequent access to the flatness value it can be stored during the polygon creation after any subdivision step. Figure 3.11 illustrates the use of the flatness test to subdivide curved objects.

By introducing a new user supplied threshold $C_\varepsilon \in [0;1[$ to the hierarchical radiosity algorithm, the subdivision of curved objects can be influenced by an error measure. A flatness test that evaluates the local quality of the approximation is used in analogy to the area threshold for planar objects. This completes the object-based meshing scheme to fit seamlessly in the refinement procedure of Hierarchical Radiosity.

3.6 Energy transport

To guarantee the correct energy balance throughout the whole subdivision hierarchy, the hierarchical radiosity algorithm uses a *push/pull* procedure (Section 2.10.3). Energy that is gathered at different hierarchy nodes is propagated accordingly in the tree. Because radiosity is measured in power per area child nodes receive the same value as their parent node, whereas parent nodes receive the area averaged values of their children. The object-based meshing scheme however, violates the implicit assumption that a node in the hierarchy is the direct union of its children. Due to the adjustment of vertices, the geometry actually changes during the solution of the radiosity system. In general the orientation of the polygons representing the child nodes does not coincide anymore with the orientation of the input polygon. In this case, the radiosity of a node can not be computed using the area average of the child nodes. Instead, the *projected*

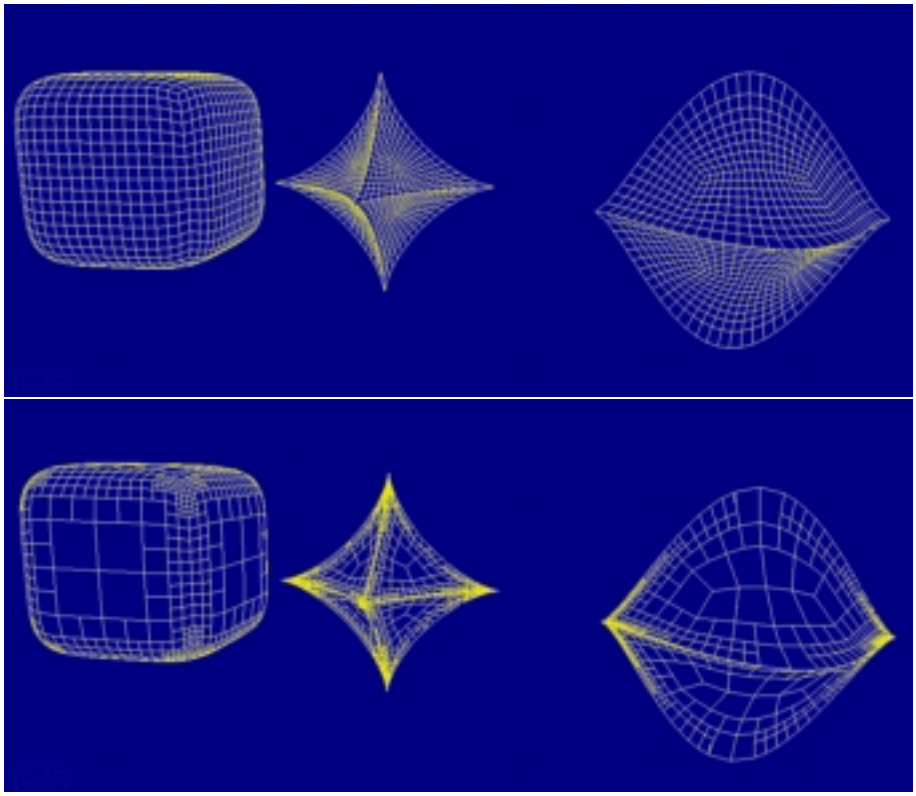

Figure 3.11 Curvature driven subdivision. The upper row shows three superquadrics with different parameters that were subdivided regularly. In the bottom row, the flatness test was applied during subdivision.

area of the child nodes in the direction of the parent must be used. Figure 3.12 illustrates the relationship between parent and child nodes after the application of object-based meshing. The original object is a sphere that is approximated by few large polygons. During the radiosity solution the parent node is subdivided due to a link refinement from patch q. New links are established from q to the children that now have a different orientation than the parent node.

The projected area of a child node can be computed by multiplying the area with the cosine of the angle between the normal and the direction to the parent. This direction is given by the surface normal of the parent. Thus, with the radiosity B, the area A and the normal vectors $\vec{n}$, the correct propagation of radiosity in the tree can be computed by:

$$B_{parent} = B_{parent} + \frac{(\vec{n}_{child_1}\vec{n}_{parent})A_{child_1}B_{child_1} + (\vec{n}_{child_2}\vec{n}_{parent})A_{child_2}B_{child_2}}{A_{parent}} \tag{3.1}$$

3.7 Form Factors

The accurate computation of form factors has a great impact on the visual quality of a radiosity solution. Missed shadows let objects seem to float in the environment instead of being fixed to the ground. Correct shadows help the observer to easily recognize the spatial relationship be-

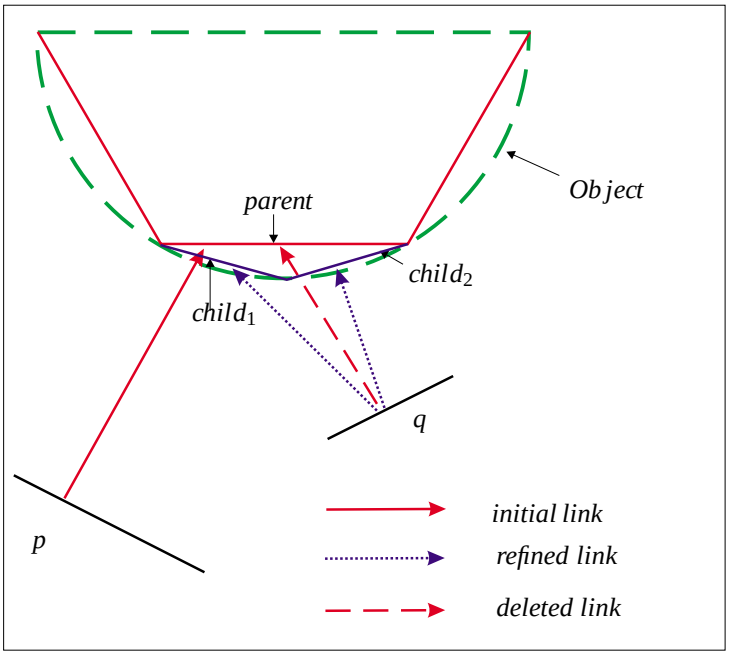

Figure 3.12 Geometry of object-based meshing. Subdivision of a parent patch results in two children of different orientation. The energy received by the parent due the children is computed by projecting the child nodes onto the parent node.

tween objects. In Section 2.7.2 the use of ray casting to determine the visibility and to compute the form factor kernel between two patches was discussed. By extending this technique to an object-based method, accuracy and speed of form factor computations for curved objects can be improved.

The object-based meshing scheme introduced above allows the hierarchical radiosity algorithm to start with very coarse approximations of curved objects. This reduces the time needed for the initial linking step and, because objects can be refined locally, should improve the quality of the solution. However, this approach renders the application of a polygon-based ray-casting technique useless. The coarse approximation of a curved object will lead to several errors in the form factor computation. Because the polygons are mainly located inside or outside of the original object's surface their use as sources or targets of rays will lead to wrong form factor results. The following problems will occur:

- wrong or missing shadows because objects are not hit
- wrong form factors because the patch normal is used
- wrong form factors because the patch-to-patch distance is used

Figure 3.13 illustrates these cases using a simple 2D geometry. Each ray exhibits one of the possible errors.

Basing the ray-casting process on the underlying objects avoids these problems. Because the algorithm still uses finite elements to represent the radiosity values, again a combination of

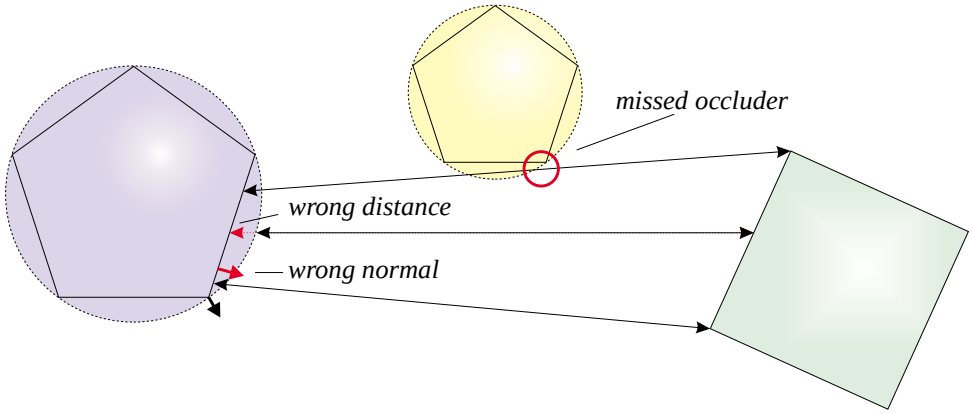

Figure 3.13 Form factor computation with curved objects. A coarse approximation of curved objects results in form factor and visibility errors. Instead of using the patches all calculations must be based on the real objects.

patches and objects is used. It is assumed, that the ray tracer is not aware of the radiosity mesh, i.e., patches do not belong to the scene as active geometry. Because an object is responsible for its own boundary representation, this should be an obvious design decision. Evaluating the form factor integral using Monte-Carlo integration normally works by randomly selecting sample points on the source and on the receiver patch. The same approach is taken here. Rays are cast from the sample points on the source into the direction of a sample point on the receiving patch. Because the ray tracer does not 'see' the patches it only checks if an intervening object exists between the source and the receiver. If both patches belong to planar objects the results can be used directly. If curved objects are involved, start and end point will never be exactly on the object's boundary. Thus, in case of an intersection, it must always be checked if the source or receiver patch belongs to the hit object. Typically, the first encountered intersection when launching a ray from a convex object's patch is the object itself. In this case a new ray is launched, emanating from the last intersection point. For non-convex objects the situation is different. The object might cast a shadow on itself, in which case an intersection must not be ignored. For a correct decision the surface normal at the intersection point must be checked. If it points towards the origin of the ray, the receiver patch is invisible and the object partially occludes itself. Checking the surface normal at each intersection point imposes an additional effort to the form factor computation. However, it solves the second problem listed above. In the unoccluded case, the real surface normal at the intersection points of the interacting objects is automatically obtained. Thus, the real angle between the surface normal and the direction of transfer can be computed. Together with the correct distance between the two intersection points formula 2.31 can now be solved exactly, thereby avoiding all inaccuracies of the polygon-based approach.

3.8 Reconstruction

The reconstruction of a radiosity solution depends on the application's needs. For a walk-through application an accurate polygon mesh must be provided whereas high-resolution still

images have to be rendered with a final gathering step. Both techniques must be tuned for the use of a radiosity solution obtained by the algorithm described so far.

3.8.1 Remeshing

The use of an oracle in the polygon-based meshing scheme guarantees that the representation of the radiosity function is refined if the energy transport can not be computed accurately. The mesh is only responsible to store the constant radiosities. For curved objects however, the mesh has to store the object's shape as well.

Consider the BF-refinement procedure that tries to balance the energy transport over all links (Page 40). Regions that are directly lit by primary light sources tend to get heavily subdivided. Dark regions that are in shadow or that do not face any light source however, keep their coarse representation. This can lead to severe artifacts for curved objects. When the radiosity solution has converged, the geometry can still be in an unsatisfactory state. Regions of curved objects that are in shadow were probably not refined at all, which leads to a very irregular mesh representing that object. Similar to T-vertices in the planar case, non-planar polygons result in regions where the degree of subdivision changes abruptly, e.g., at shadow boundaries over curved objects.

As a solution, another application of the object-based meshing scheme together with the flatness test is used. After a radiosity solution has been obtained, all curved objects are remeshed locally until the curvature is within the earlier introduced error threshold C_ε. This guarantees homogeneous polygon meshes, because the radiosity algorithm used the same error threshold during subdivision. The new patches that are generated by the remeshing step need a radiosity value to be rendered properly. Because the radiosity representation in these regions already satisfied the oracle (otherwise further refinement had taken place), the radiosity values from their parent nodes can be copied directly. This technique improves the shape of the corresponding object without increasing the simulation error and, most importantly, without the need for additional iterations.

Once the remeshing step is finished, the weighted reconstruction for color interpolation introduced in Section 3.4.3 is applied. The resulting mesh containing vertex colors and planar polygons can then be rendered by any polygon renderer or stored in a 3D file format like *Open Inventor* [Wer94].

3.8.2 Final Gathering

In Section 2.9.3 final gathering was introduced as a pixelwise reconstruction technique that is capable of generating high-quality renderings. A ray tracer is used to find the first intersection point of a ray from the eye with a scene object. At each visible point in the scene the radiance outgoing towards the eye is computed. Using this technique arbitrary radiosity gradients can be visualized. To compute the radiance value all surfaces that contribute to the illumination of a particular surface point must be visited. For each surface the point-to-patch form factor and visibility is then evaluated to obtain the exact solution. The hierarchy of links that were built by the hierarchical radiosity algorithm during its iterations can be used efficiently to enumerate all contributing patches. Starting from the lowest level, the gathering process is recursively performed for all ancestors in the hierarchy.

In this section an object-based final gathering technique is proposed, that was successfully applied to hierarchical radiosity, wavelet radiosity and wavelet radiance [Bül99]. It fits into the object-based framework developed in this chapter and further enhances the rendering of curved surfaces.

The problem that must be solved is the correct identification of the corresponding patch for a given surface point. As mentioned above, the patches of the radiosity mesh are invisible to the ray tracer. Only the original scene objects are checked for intersection. The links however, are connected to the patch hierarchies. In Section 3.5 the ray tracer was used to move the vertices of a patch to the closest point on the object's surface, now, the problem is reversed. The corresponding patch to a surface point is a leaf node of the hierarchy associated with one of the input polygons. For planar objects the corresponding patch can be found easily, because the root node of a subdivision tree exactly coincides with the object surface. The tree is then traversed from the root node, until the leaf node is determined that contains the intersection point.

For curved objects the root node of the hierarchy is not a part of the object's surface. Thus, the patch closest to the intersection point must be found. Additionally, this patch's normal should be similar to the surface normal. If the faces of an object's boundary representation are regarded as a scene of polygons, ray casting can be used to find the proper patch. This does not contradict the object-based approach, where only complete objects are regarded by the ray tracer. The object intersection is used for the exact surface normal and distance calculations. Additionally, the coordinates of that point are later used for further visual enhancements like texture mapping or bump mapping. These rendering techniques are based on an inverse mapping of a 3D point into a 2D map that stores additional parameters. Thus, both intersection points (object *and* patch) are needed to achieve an optimal result.

The following test were performed to find a reliable assignment of surface points to hierarchy nodes [Bül98]:

1. Find the closest patch by casting a ray into the direction of the surface normal.
2. Like 1. but additionally the patch hierarchy is used as a search tree.
3. Select from several patches with the minimal distance the one with a normal that is closest to the real surface normal.
4. Like 3. but additionally the patch hierarchy is used as a search tree.

A combination of methods 1. and 3. was found to be the most promising approach. Only if method 1. is not successful, method 3. is applied. Method 1. fails if the approximation does not cover all parts of the object, e.g., at the boundary of the disc shaped bottom of cylinders or cones. Although these methods are the most expensive ones, the total running time of the final gathering process is dominated by the visibility tests when traversing the links.

Using this technique results in high-quality images with accurate interreflections and perfectly curved objects. Because the direct illumination has the greatest impact on the visual quality of shadows it can be useful to treat the direct illumination separately. Very coarse approximations of curved objects tend to provide too few links to primary light sources. Gathering

light directly from all primary light sources thereby ignoring the link hierarchy remedies the situation.

Still, final gathering is an expensive rendering technique. Due to the overall use of ray tracing it can be optimized by applying ray-acceleration techniques like bounding volume hierarchies or space subdivision. Stochastic methods have also been applied to improve the method's efficiency [UT97]. If high-quality renderings of curved objects are required, the object-based methods introduced so far can be combined very efficiently. The pixelwise reconstruction requires only a coarse radiosity solution that is normally obtained after very few iterations. Due to the object-based meshing scheme, the first iterations that include the initial linking step, could be improved essentially. Thus, the final gathering process has a minimal 'preprocessing' time. Additionally, its quality is improved by accessing the original scene objects.

3.9 Results

The object-based methods developed in this chapter have been implemented in a rendering system to illustrate their general applicability. The modified hierarchical radiosity algorithm was used to compute radiosity solutions for various scenes containing curved surfaces. The images presented here will show the progress of the algorithm, the quality and speed of its solution, its application to complex objects and its combination with other rendering techniques.

Object-based meshing. To visualize the object-based meshing scheme, snapshots of several stages of the algorithm are used to illustrate the progress of the algorithm. The image sequence of Figure 3.14 shows an urn that was modeled as a solid of revolution. A coarse approximation of only 176 polygons was used to initialize the algorithm. Proceeding from the upper left image to the right and from top to bottom, two things can be noticed. The shape of the urn improves from a crude approximation to an object of accurate curvature. Simultaneously, the global illumination in the scene gets updated and lighting effects and shadows appear until a satisfactory result is obtained. In the initial linking phase 30800 potential links had to be checked. Due to many backfacing pairs of polygons 6108 links were actually established. On a MIPS R5000 processor running at 150MHz initial linking took 61 sec. If the object is to be rendered with the standard hierarchical radiosity algorithm to the same accuracy, the input model must consist of 1160 polygons. This results in 232112 potential initial links, which is 40 times more than for the object-based approach.

Form factors and remeshing. Figure 3.15 shows another simple scene, where a sphere is directly lit by two light sources. The sphere model started with 20 triangles that were adaptively refined during the radiosity solution. Because the back side of the scene is hardly illuminated, only few polygons had to be refined. In directly illuminated regions however, the sphere's mesh was refined to a high degree which results in very sharp highlights on the sphere. Trying to achieve the same quality with a standard radiosity algorithm would require an enormous effort due to the extremely fine tesselation. The severe artifacts at the shadow boundary on the sphere's surface could be eliminated by applying the remeshing step discussed earlier. The curved shadows on the floor are an indicator of the correct form factor evaluation. Because the sphere and not the coarse mesh were used, the visibility computation is accurate.

The corresponding mesh to the solution and its correction can be seen in Figure 3.16. The reason for the artifacts is obvious in the lower region of the sphere (left image), where the initial tesselation is still apparent.

Reconstruction. In Figure 3.17 two reconstruction schemes are compared. The medium sized scene contains a superquadric, a cylinder and several spheres and is lit from the small quadrangle to the right. Another light source illuminates the picture on the wall. The left image shows the solution reconstructed by the weighted reconstruction technique described in Section 3.4.3 and was rendered with gouraud shading. The right image was obtained after applying the object-based final gathering process, that could be started after only 3 iterations of the radiosity solution. A closeup of both images can be seen in Figure 3.18. Due to the object-based approach, all curved surfaces are rendered exactly. The solution took 4 min to complete with an additional 83 min for the final gathering step (R4400/200MHz) at a resolution of 600x600 pixels [Bül99].

Complex objects. Figure 3.19 documents the applicability of the algorithm to a wide variety of curved objects. The museum scene contains a sphere flake of 91 spheres, 6 cylinders, 4 superquadrics, several Bézier patches, another sphere and the urn model from Figure 3.14. All objects started with their coarsest approximation available (e.g., 8 polygons for the superquadric in the front). The number of polygons that had to be compared in the initial linking stage could thereby be reduced from about 15000 to only 2500. The scene is illuminated homogeneously from the large white quadrangles below the ceiling. Note the shadows of the light sources on the ceiling as a result of light being reflected back from the floor. Due to this kind of illumination, no remeshing step was required. Gouraud shading was used in the reconstruction step.

The advantages of combining the new algorithm with a topological data structure can be seen in Figure 3.20. The cup is modeled by a boolean expression combining a torus with the difference of two cylinders. The whole cup is represented by a single (CSG-)object. The mesh was created with the technique described in Section 3.4. Curvature and shadows are rendered accurately. The adaptive object-refinement due to the single light source is illustrated by the wireframe view. Parts of the cup that were exposed to the light source, even inside the cup, were subdivided accordingly.

Further applications. The object-based hierarchical radiosity algorithm could successfully be integrated in a commercial rendering package. Figure 3.21 shows a snapshot of the modeling and rendering software *3D Studio MAX* from Kinetix, where the algorithm was implemented as a plug-in [Thi99]. The top-most window contains the radiosity solution of the scene that was previously modeled in the viewports behind. The plug-in accesses the internal scene graph and converts all objects into data structures used by the radiosity algorithm. Currently, only unmodified objects can be used. As soon as an object's boundary is transformed (bended, twisted, etc.) this information must be taken into account by the meshing scheme. Future work will use the stack of transformations that is kept internally, to improve the applicability of the algorithm to this extended class of objects.

The combination of the final gathering approach for curved objects with other rendering techniques is illustrated in Figure 3.22. Bump mapping is typically used by ray tracers to

visually increase the complexity of a model. At each surface point hit by a ray, the original surface normal is slightly modified according to an image map. Although the geometry is not changed, surface structures appear like a carving in the object. Because the final gathering approach used here is based on ray tracing of the original objects, this technique can easily be used. The result is a global illumination solution with perfectly curved objects and a high visual complexity.

The object-based meshing scheme is not restricted to the radiosity case. To illustrate its applicability to other classes of finite element global illumination algorithms, it was incorporated in a radiance algorithm using wavelet bases [Chr95]. The higher complexity of the radiance case, in contrast to radiosity, is due to the variation of the reflected radiant energy at a surface point depending on the outgoing direction. The incorporation of curved surfaces can be seen analogously to the radiosity case. Starting with highly tesselated approximations of curved objects results in an excessive computational effort and prohibits an adaptive refinement that actually improves the quality of the simulation according to the user's needs. Reducing the complexity of this algorithm while avoiding to predetermine the final mesh quality obviously could enhance radiance computations. Figures 3.23 and 3.24 show a glossy reflecting cylinder that is lit by a single light source. The images show the same radiance solution from different viewing angles. Note that in contrast to the radiosity case, the highlight moves over the surface when the viewing position changes. The solution was obtained by a wavelet radiance computation using the object-based meshing scheme [Bül99]. The algorithm started with the roughest approximation of the cylinder containing only five polygons (two triangles for top and bottom and three quadrangles for the cylinder coat).

3.10 Summary

In this chapter it was shown, that using modeling information during the whole rendering process can efficiently improve hierarchical radiosity computations. Accessing the original objects during refinement, for visibility and form factor computations and in the reconstruction step leads to better results in terms of speed and quality. Thus, curved objects could be efficiently incorporated in the hierarchical radiosity algorithm. Combined with a powerful data structure that provides full adjacency information, the presented algorithm is applicable to scenes containing a wide variety of complex objects. The final display of the solution benefits from both, the underlying data structure and the object-based approach, depending on the chosen reconstruction method. Due to the combination of ray tracing and radiosity, rendering techniques that require the pixelwise evaluation of surface points are directly available to further enhance the quality of the solution.

The requirements for the new algorithm are small and should be available in any rendering software providing ray tracing. As a proof of concept the algorithm was incorporated in a commercial rendering package.

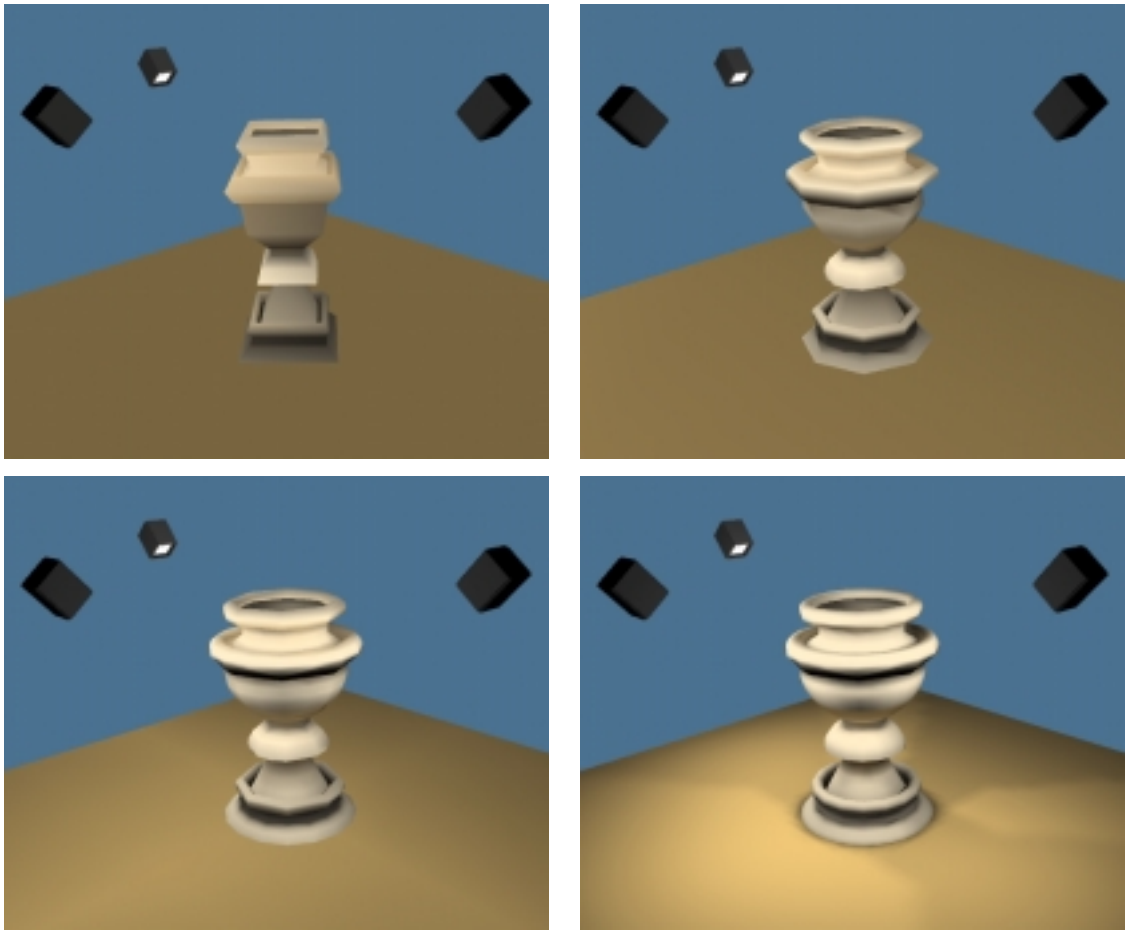

Figure 3.14 Hierarchical radiosity on curved objects. During the radiosity simulation the object's shape improves. The object-based meshing scheme allows for a very coarse input model without degrading the quality of the solution. These snapshots were taken after 1, 2, 3 and 9 iterations of the modified hierarchical radiosity algorithm.

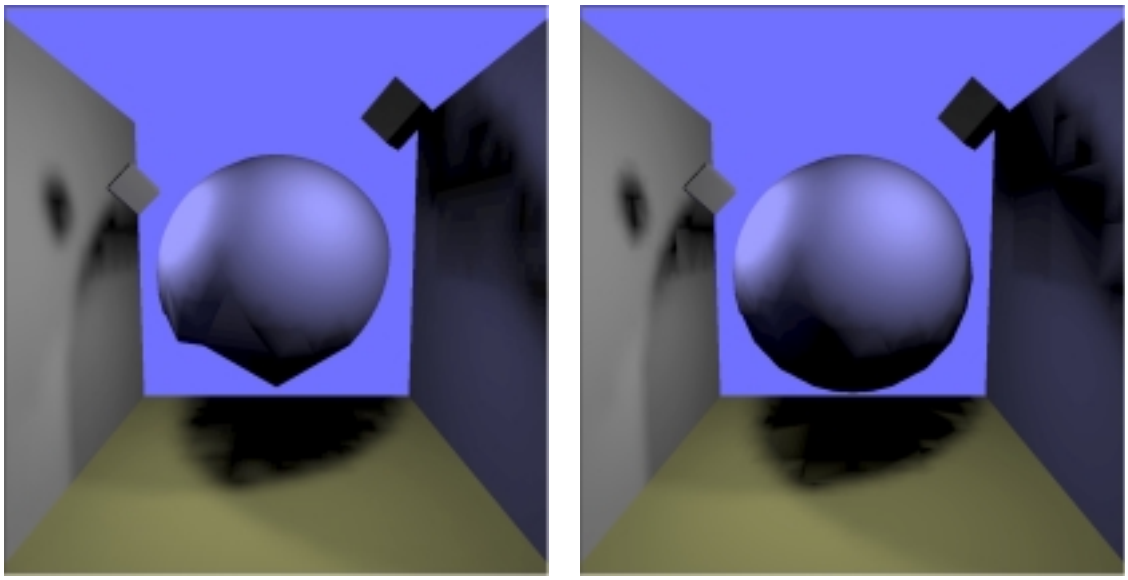

Figure 3.15 The remeshing step. The left image shows an adaptively refined radiosity solution. The inhomogeneous refinement causes severe artifacts in shadow regions of the sphere. After a remeshing step the sphere's shape is correct. Note the sharp highlight on the sphere and the curved shadows.

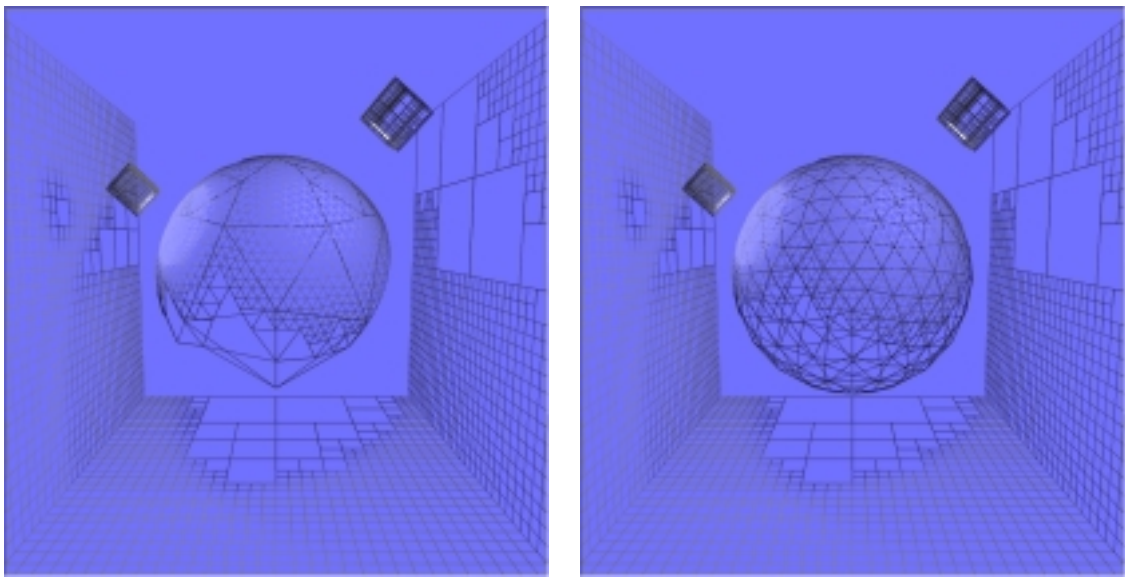

Figure 3.16 These two images show the corresponding meshes to Figure 3.15. On the left side, the sphere's coarse start approximation is still visible in the lower region of the sphere.

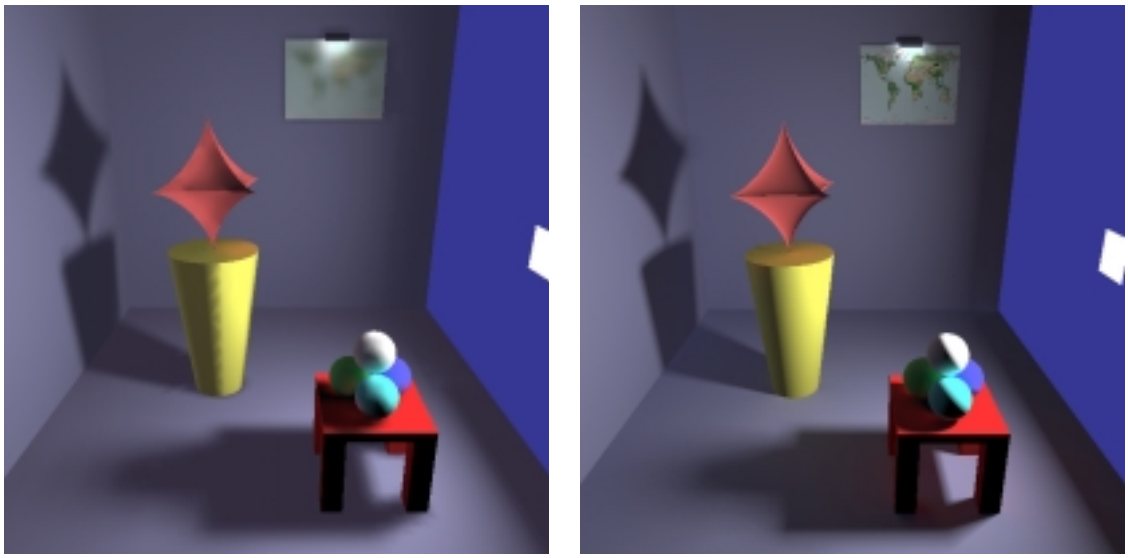

Figure 3.17 Final gathering. On the left image reconstruction was done by gouraud shading, whereas for the right image the object-based final gathering step was used. The radiosity solution was computed by a wavelet radiosity algorithm. (from [Bül99])

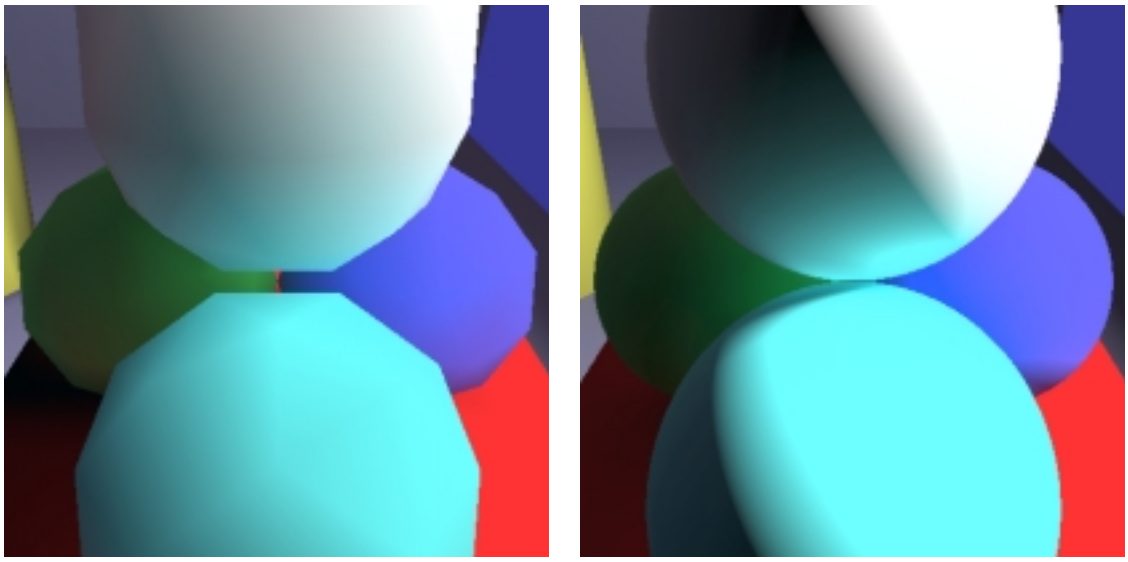

Figure 3.18 These images show a closeup of the solution in Figure 3.17. Note the high quality of the final gathering reconstruction scheme when compared to polygon-based bilinear interpolation. (from [Bül99])

Figure 3.19 A museum of curved objects.

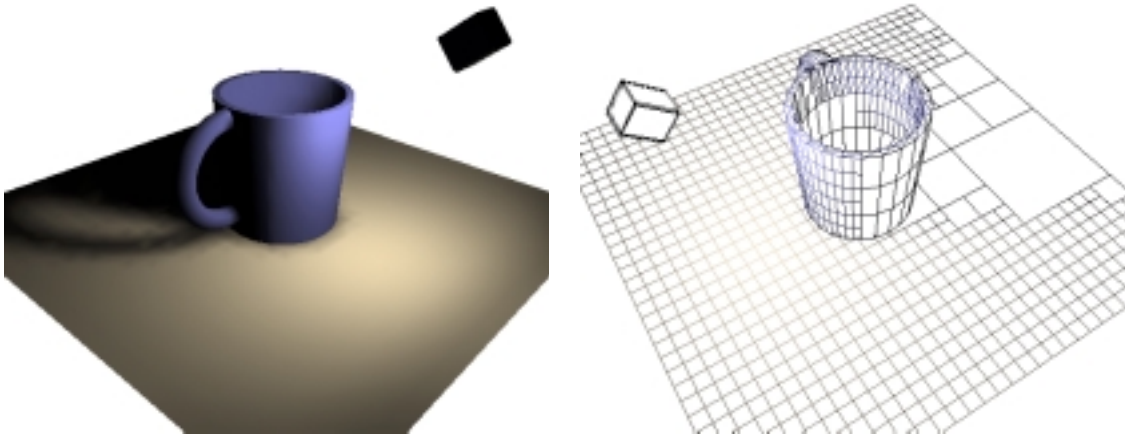

Figure 3.20 Hierarchical radiosity on CSG-objects. The cup is represented as a single object, being the union of a torus and the difference of two cylinders.

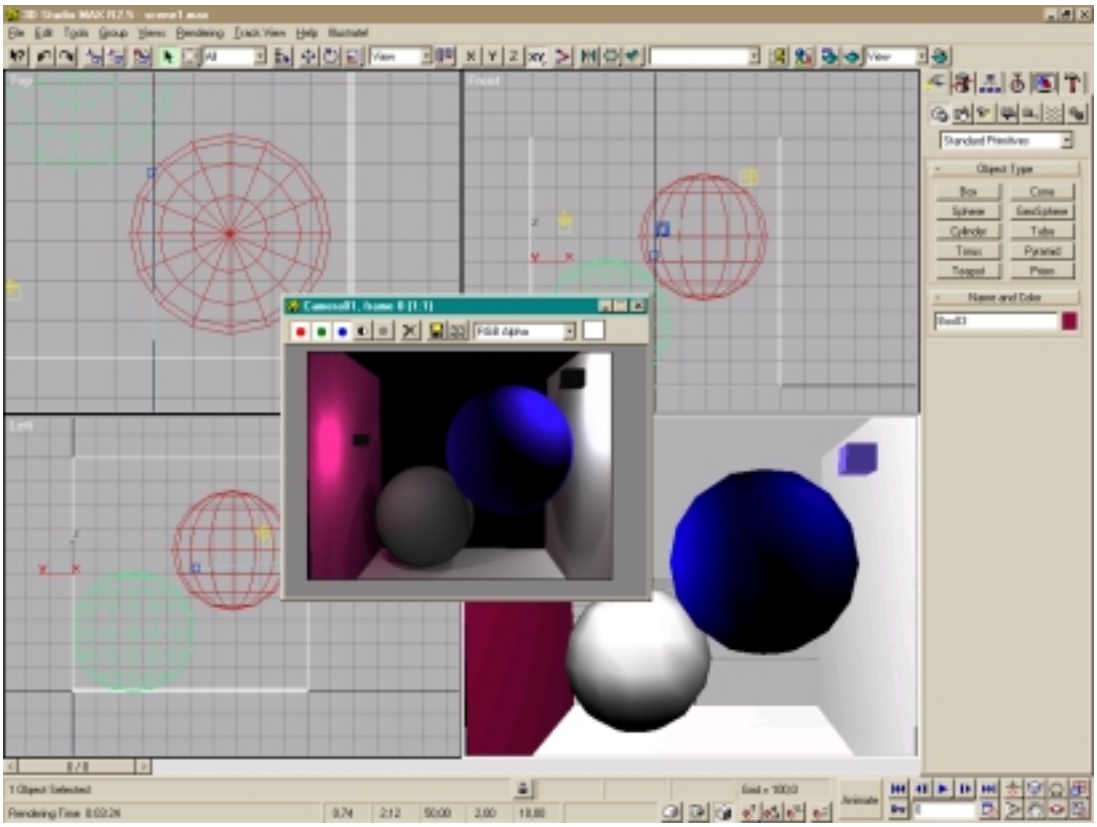

Figure 3.21 Integration in 3D Studio MAX [Thi99]. The object-based hierarchical radiosity algorithm is well suited for integration in object-oriented rendering packages.

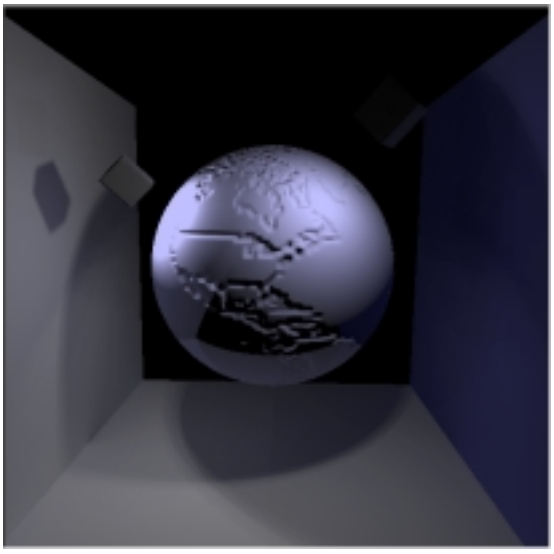

Figure 3.22 Radiosity with bump-mapping. During the final gathering process, the surface normals of the sphere were perturbated according to a 2D map.

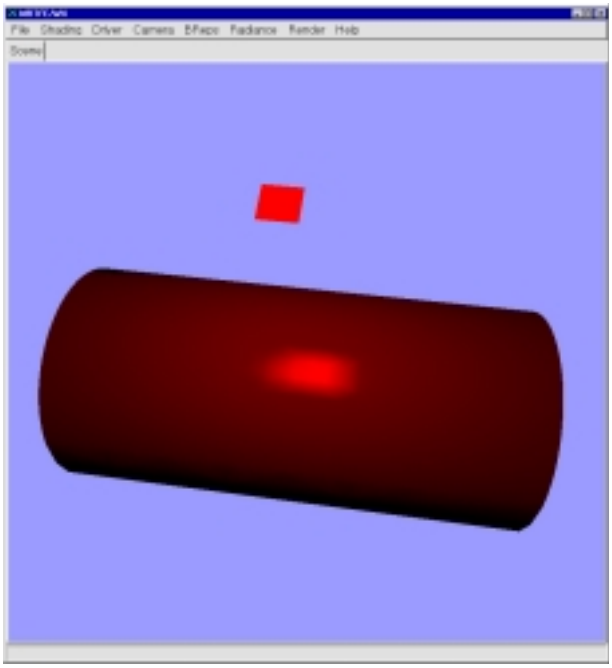

Figure 3.23 Radiance solution for a glossy cylinder lit by a red lightsource. The cylinder started with a mesh of 5 polygons.

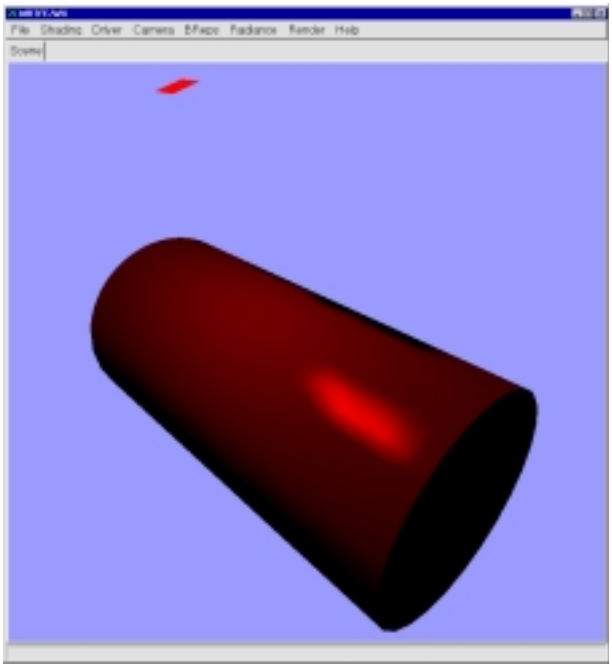

Figure 3.24 The radiance solution of Figure 3.23 seen from another viewing position. Note the different position of the highlight. Both images were extracted from the same solution by an interactive viewer [Bül99].

CHAPTER 4

Distributed Hierarchical Radiosity

4.1 Introduction

With the increasing size of input scenes, radiosity computations get more and more impractical. Rendering times and memory consumption clearly limit the usability of finite element algorithms. A possible way to reduce rendering times is the parallelization of these algorithms. Due to the large amount of interactions between mesh elements however, communication costs quickly become a new bottleneck. In this chapter, a new approach to the parallelization of Hierarchical Radiosity will be introduced. By further exploiting the object-based techniques developed in the last chapter an algorithm is formulated that minimizes communication costs and scales very well when the number of processors is increased.

The design of a parallel algorithm heavily depends on the target platform addressed. Special purpose hardware typically provides the best support for parallel solutions, however their accessibility is often limited and their acquisition and maintenance is expensive. A much cheaper and widely available resource for parallel computing are workstations and PCs that are interconnected by a local area network (LAN). These machines are easily accessed and often have enormous computational power that is unused most of the time. Algorithms targeting this environment must deal with two problems that make their design rather specific. The bandwidth provided by a LAN is very small when compared to explicit parallel architectures, which increases the demand for efficient communication. Additionally, processors are often idle but during a lengthy computation they can probably not be used exclusively. A load balancing comparable to scheduling techniques found in operating systems must be applied.

The distributed hierarchical radiosity algorithm presented here is designed to run in a network of workstations. Together with an efficient communication protocol a load balancing scheme is introduced that adapts to the changing availability of CPU time in a typical LAN. Due to minimal changes to the original algorithm, its integration in existing hierarchical radiosity systems is easy to accomplish [FSZ98].

This chapter is structured as follows. After discussing previous work on distributing hierarchical radiosity computations a new object-based parallel algorithm will be presented. The scheduling scheme that led to an efficient load balancing is explained in Section 4.4. Finally results regarding the efficiency of the algorithm will be presented. A summary concludes this chapter.

4.2 Previous Work

Several successful approaches have been made to improve the hierarchical radiosity algorithm by parallelization.

In [SHT+92] and [SGL94] Singh et al. describe their implementation of hierarchical radiosity on the Stanford DASH multiprocessor system. The DASH machine has a shared address space and is equipped with 48 MIPS R3000 processors. As the basic computational task a patch and all its interactions (including the interactions of its children) or a single patch-patch interaction is chosen. Task queues are provided for each processor that are initially filled with the top level polygon interactions. New interactions are added to the processor's queue that subdivided the patch. Load balancing is performed using task-stealing: if a queue is empty, tasks from another queue are transferred to the empty queue. New tasks are always created at and dequeued from the head of a queue, whereas task-stealing only occurs from the tail. This algorithm resulted in good speedups for a simple scene of about 150 polygons. The good speedup is the result of exploiting the coherence of subsequent visibility tests using a BSP tree. Because the same part of the tree can be reused, the processor caches work efficiently. The communication costs of the task stealing process however, prevent its application in a network of workstations.

In order to achieve a better load-balancing the overall costs involved with a patch were investigated by the same authors. Although they found that the costs decrease during a radiosity computation, using it as a hint for the scheduling algorithm on how to distribute new tasks did not succeed.

Zareski describes in [Zar95] an implementation similar in spirit to the one that will be presented in this chapter. The author also uses a client-server model in a network of workstations to distribute ray-patch intersection tests. The server uses the results to compute form factors and performs the refinement steps. The scheduled tasks however, are very fine grained, thereby resulting in high communication costs. As a result, the algorithm suffers from a bad scalability.

Bohn et al. describe an implementation that uses the parallel supercomputer *ConnectionMachine 5* [BG95]. They use a very fine grained parallelism by allowing each stage of the radiosity algorithm to be computed in parallel. This is achieved by distributing all patches and links over the processors. If some required data is missing at a processor, a request is sent. These requests are kept in a local stack to be able to proceed with other tasks while waiting for the requested data. Due to the massive communication costs, the achievable speedup is sublinear.

Funkhouser was the first author who presented a parallel hierarchical radiosity algorithm based on a group iterative approach [Fun96]. Using a BSP technique, the master processor partitions the scene into cells and constructs a visibility graph. Using this graph, clusters that are mutually visible are distributed to the clients, where local radiosity computations are performed. The results are recombined by the server, using a step of a group iterative approach of which the convergence is mathematically proved. Load balancing is performed by the master processor that distributes clusters to the clients. Using an estimate of the task size depending on the number of form factor evaluations and gathering steps, tasks are distributed with decreasing size to the available clients. The algorithm is implemented in a client-server environment and runs in a network of workstations.

An implementation that can be used in a network of workstations or by parallel computers is presented in [MB98]. It targets large architectural environments and applies graph partitioning

to the visibility graph to break the scene into clusters. These clusters are distributed to the clients where the gathering of energy and patch refinement is performed. All data is kept on a shared disk, using one file per cluster. When all the processors have performed gathering for all their clusters, the current iteration of a global iterative resolution method is completed. Although this concept is similar to the approach presented by Funkhouser, it does not depend on expensive client-server communication.

An improved implementation of the one developed by Singh et al. (described above) has been published in [PRR98]. Using a special parallel hardware, the SB-PRAM which is a shared memory machine that provides uniform memory access times, an impressive scalability could be achieved. In contrast to other shared memory machines, locality is not an issue here. By exploiting the facilities of the execution platform a speedup for up to 2048 processors was obtained. The main differences to the original implementation is the parallel construction of the BSP tree and a parallel loop over the input polygons to perform the initial linking. Additionally the granularity of visibility and form factor computations could be refined, due to the advantages of a uniform memory access.

Most of the algorithms summarized above try to distribute all stages of the hierarchical radiosity algorithm. Due to the high dynamic of a radiosity computation, communication costs play an important role in an efficient implementation. Therefore, the algorithms greatly differ from their sequential counterpart which makes a robust implementation difficult and which requires a complete rewrite of existing (sequential) code. The algorithm presented in this chapter is a novel approach in two aspects. First, the object model developed in the last chapter is used as the base of the algorithm. Besides the efficient support for curved objects it provides the key to keep communication costs very low. The second aspect is the reuse of existing and stable code. The modifications to the sequential algorithm are minimal. Of course, code for the scheduler must be added.

4.3 The Parallel Algorithm

4.3.1 Execution Platform

The algorithm presented here is designed to run in a computer network. The workstations may be single or multi-processor architectures of different computing power. A shared file system (e.g., NFS) is used to store the scene database although it is possible to transport the geometric description directly over the network. Bidirectional communication is done via TCP-stream-sockets that provide fast and synchronized communication channels. The design of the algorithm will follow the client-server principle, therefore a server machine that is equipped with enough RAM to store the complete radiosity solution is required. As will be described later, the client processors have much lower memory requirements. Their processor speed however, will significantly influence the execution time of the distributed radiosity computation.

4.3.2 Choosing Computing Tasks

To parallelize a sequential algorithm it must be decomposed into independent tasks that can be computed simultaneously without interfering each other. An essential decision is the choose of

a proper granularity that ensures a partitioning of the problem into well sized subproblems. In the best case, all subproblems are of the same size. This would allow for a simple decomposition of the algorithm into evenly distributed tasks. Typically, an algorithm consists of very different subproblems that have arbitrary and probably changing computational demands. The synchronization of these tasks requires high communication efforts to schedule new tasks to the otherwise idle processors. Additionally, small tasks, i.e., tasks that can be computed very quickly, result in a bad ratio of communication time to processing time. Especially in a client-server environment where each task must be scheduled separately by a central server, small computing tasks have a severe impact on the total execution time. Thus, similar sized and computing intensive tasks are best candidates for an efficient distributed computation.

The hierarchical radiosity algorithm consists of a variety of differently sized computing steps. In the starting phase the input polygons are compared and the initial links are established. During the solution, an unknown number of new patches is generated by subdividing the input polygons. These patches unpredictably interact with each other, thereby requiring access to the other patches for visibility computations. Especially due to the increasing number of geometry data, task sizes vary and permanently increase the communication load of the system. To find one or more tasks that can be distributed efficiently the various steps must be compared in terms of their computational size and locality. A task has a high locality if its execution does not require the access to other processors.

Subdividing a polygon is a very local and simple process, where no other patches are involved. Gathering energy over one or more links requires visiting a number of other patches. Due to the hierarchical approach, gathering is a local operation to a certain degree because only a subset of the environment has to be accessed. Compared to the few arithmetic operations (addition and multiplication) that are performed in a gathering step however, the communication costs are rather high. Distributing energy in the hierarchy once the gathering is completed, requires access to all patches of the same input polygon. Depending on the storage scheme (polygon- or patch-based) the degree of locality can be very different. The most time consuming task that remains is the computation of form factors and visibility, which is typically performed simultaneously (Section 2.7.2). Actually, the visibility computations in complex scenes need up to 90% of the total execution time of a radiosity solution.

Detecting an independent part in the algorithm that clearly dominates the overall execution time makes it a preferred candidate for distribution. The locality of visibility computations is typically very low, because any patch in the scene is a potential occluder. Using ray-acceleration structures effectively improves the situation. However, visibility computations from or to subdivided patches still requires their expensive distribution to all processors.

The next sections describes a parallel algorithm that is based on the distribution of form factor and visibility computations but that avoids the increase in communication costs described above. The key to minimizing the communication costs will be the object-based form factor computation method developed in Section 3.7.

4.3.3 The Data Flow

The standard hierarchical radiosity algorithm must be slightly reformulated to separate the form factor computation from the other tasks. Consider the original algorithm described in Section 2.10.3 using BF-refinement. Form factors are calculated in the initial linking stage and during

further mesh refinement. This is directly coupled with the creation of new links that are established if the error in transport exceeds a given threshold. Because the form factor is stored in the link data structure it is computed whenever a new link is created. The pseudo-code given in Figure 2.23 shows the context of this procedure. An iteration is performed over all links in the system and checks them for refinement.

If the form factor computations would be distributed directly, the whole communication with the clients had to be integrated deeply into the algorithm. The responds of the clients that are sending back their results would permanently interrupt the refinement process thereby requiring additional synchronization effort.

The pseudo-code in Figure 2.19 illustrates the standard hierarchical refinement procedure. It can be seen that all further refinement, once a link was established, is independent of that link. Especially the form factor stored with the link has no influence on the further subdivision of other mesh elements. Therefore, to isolate the form factor computation it can be moved outside the loop. All links are established as before, but with a place holder instead of the real form factor.

Keeping in mind that the functions `refine()` and `refineLink()` now create empty links, i.e., links with an invalid form factor of -1, the pseudo-code of the algorithm's inner loop is illustrated in Figure 4.1. Changes to the original code of Figure 2.19 are set in bold.

The changes to the original algorithm are minimal and the most time consuming steps are now isolated in a central function call. The efficient parallel computation of the form factors and their integration in the server's main loop is the next step.

4.3.4 Distributed Form Factor Computation

Using a patch based form factor computation in a parallel implementation typically requires the replication of large parts of the radiosity mesh to the clients. As mentioned earlier, the resulting communication costs will reduce the gain from distributing the most expensive part of the algorithm over multiple processors.

The approach presented here is based on the same object-based form factor computation that was successfully used to render curved surfaces with the hierarchical radiosity algorithm (Section 3.7). Although it is based on the mesh elements to select the sample points, the evaluation of distances, angles and, most important, visibility only needs access to the original scene objects. Once the sample points are determined, form factors are computed by casting rays through the *untesselated* scene. This suggests the following setup of the proposed client-server environment:

The server machine is responsible for the communication and solves the radiosity system. The main loop illustrated in Figure 4.1 runs exclusively on this central machine. The client computers are 'only' ray tracers and they do not load a radiosity program. Instead, they load the same scene description as the server but never tesselate the scene into polygons. During the simulation ray-tracing tasks are sent to the clients that can be evaluated without a mesh. This setup results in several advantages:

- The mesh is never sent over the network.
- Accurate object-based form factor computation.

```
tmp = A_ε
A_ε = ∞
/* initial linking */
for (each polygon p)
   for (each polygon q)
      if(p != q)
         refine(p,q); /* creates empty links! */
A_ε = tmp

/* fill empty links with form factors */
distributeFormFactors();

done = FALSE;
while(done == FALSE) {
   /* solving */
   while (not converged) {
      for (each input polygon p) gather(p);
      for (each input polygon p) pushpull(p, 0);
   }
   done = TRUE;
   /* refinement */
   for all links l
      if( refineLink(l) == FALSE ) /* creates empty links! */
         done = FALSE;

   /* fill empty links with form factors */
   if(done == FALSE)
      distributeFormFactors();
}
```

Figure 4.1 Pseudo-Code: Delayed form factor computation.

- Support for curved surfaces as described in Chapter 3.
- Ray-acceleration structures to speed-up ray tracing need to be initialized only once. (The scene is static for the clients.)

The amount of data that has to be transported over the network depends on the formulation of the ray-tracing tasks. To avoid any dependencies of the mesh elements, each sample ray could be regarded as a task. With the knowledge of the mesh geometry, the server would determine the sample points on both interacting patches to let the clients launch the connecting rays. Assuming 16 samples per patch, which is a typical count, 32 3D-coordinates had to be transferred for each form factor. The floating-point representation would require $32 * 3 * 4\ bytes = 384\ bytes$ if single precision numbers (4 bytes) are assumed. This large amount can be reduced by letting the clients choose the sample points.

Recall the point-to-disc form factor approximation that is used in the Monte Carlo integration approach described in Section 2.7.2:

$$F_{dA,A'} = \frac{A' \cos\theta_i \cos\theta_j}{\pi r^2 + A'} \tag{4.1}$$

The parameters to specify a disc to be used in this approximation are the disc's area A', the center and the normal. This sums up to a scalar value and two 3D-coordinates for each patch or $2 * (1 + 2 * 3) * 4\ bytes = 56\ bytes$ per form factor task. Once these parameters are sent to the client, sample points over the corresponding patch can be generated there. The reduction in communication costs is about a factor of seven.

Thus, the `distributeFormFactors()` call in Figure 4.1 can be formulated as follows. All links in the hierarchy are visited and checked if their form factor field is empty. For all empty links the corresponding patch data for sender and receiver (area, center, normal) is collected and put in a list. Finally the list is traversed and the form factor tasks are sent to the clients.

The clients execute the form factor calculation as described in Section 3.7. One important detail of this method is the identification of a patch's parent object when launching a ray. To avoid regarding the object itself as an occluder, each hitpoint must be checked carefully as discussed earlier. A sequential implementation can base this test on a pointer comparison. In a parallel implementation this is not possible due to different address spaces on the clients. Instead an additional ID must be transmitted that uniquely identifies the parent object of each patch. This ID increases the data volume by 4 bytes per form factor (two 16 bit integers) to a total of 60 bytes.

Once all clients have finished their tasks and have sent back the form factors, the server fills the empty links and starts the radiosity simulation.

4.3.5 Compacting Form Factor Tasks

The iteration over all links to collect the form factor tasks can be optimized. Because all gathering links must be stored or be accessible from within the patch, the loop should iterate over all patches. At each patch, all gathering links define form factor tasks with the same receiver, i.e., the patch itself. This can easily be used to compact a list of form factor tasks by sending the receiving patch's parameters only once, followed by the parameters of all sending patches.

Together with the number of links (a 32 bit integer), this list defines a collection of tasks that can be solved by a single client.

For medium and large scenes this technique essentially reduces the amount of data that has to be sent. Without compacting, each form factor task requires 60 bytes or $2 * n * 30$ bytes for a list of n form factors. Sending the receiver only once plus the length of the list results in $30 + 4 + n * 30$ bytes. For large n the size of the sending patch's data can be neglected and the required amount of data is approximately $n * 30$. This effectively reduces the communication costs to 50%.

4.4 Scheduling

In the client-server architecture proposed above, the server is responsible for the efficient distribution of rendering tasks, i.e., form factor computations, to the clients. A scheduling algorithm has to decide which jobs (or how many) should be scheduled to which client.

This scheduling algorithm should minimize the total running time of the application. The arising load balancing problem has to comply with some conditions: all clients start simultaneously, they work without breaks and nothing is computed twice. If all clients finished their last job at the same time (i.e., without *idle-time*), optimal execution time would be achieved. However, job execution times are unpredictable. Even under less strict assumptions, the load balancing problem is **NP**-complete [GJ79]. Every client workstation has different (and probably changing) performance (due to varying workload) and jobs are not preemptable without some cost. Therefore a simple heuristic for dynamic load balancing is applied here, which was subsequently improved to achieve good results in "real world" environments. "Real world" means, that the scheduling algorithm is applicable to networks that are not dedicated to the rendering task. The client computers connected by the network may be concurrently used by other processes.

A situation that has to be avoided is to have some clients still working on their last job while others are already finished and are idling. This *idle-time* is bounded by the maximal job execution time. The only assumption about job execution times that can be made is that smaller jobs will take less time. Therefore the first approximation to an optimal schedule is to use many (small) jobs and to distribute them using a first-come first-served algorithm. In the proposed distributed radiosity algorithm, the problem of idle-time reappears periodically. After each refinement step, a parallel computation takes place, thereby accumulating performance losses due to potentially idle processors.

Unfortunately a higher number of jobs causes more communication, which degrades the overall performance. This is even more important in LAN environments with high communication costs compared to explicit parallel architectures. Empirical tests can be used to find a good jobsize to meet both requirements. Measuring rendering times of various example scenes (real world as well as artificially constructed, simple and complex) revealed that five to eight times the number of clients is a reasonable number of jobs (see Figure 4.2 for an example).

A typical LAN connects workstations of varying performance characteristics. By taking the estimated performance of each client workstations into account, improvements regarding the reduction of idle-time can be made. Generally, a computer with twice the computational power will take half the time on the same job than its counterpart with "normal" power. By using

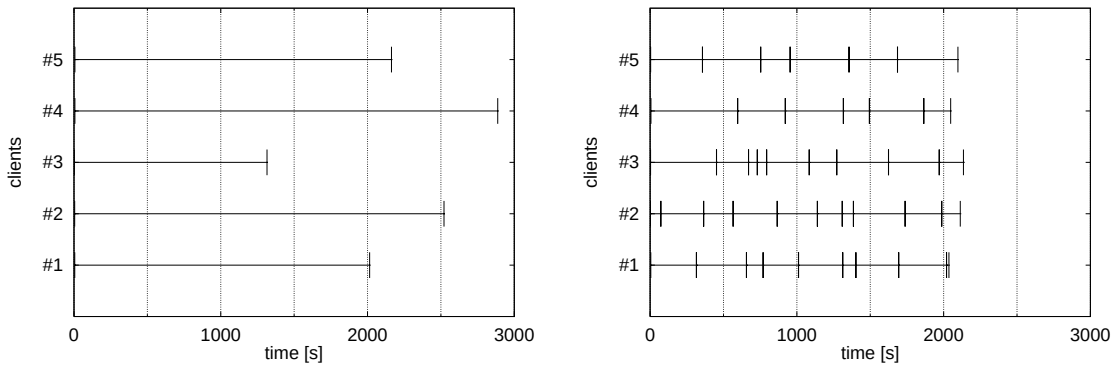

Figure 4.2 These Gantt charts show the positive effect of increasing the number of jobs on a problem. The left one uses one job per client while the right one uses eight. The distinct horizontal bar segments represent the jobs that each of the five clients has computed.

a single *speed index* to describe the relative speed of each computer, jobsizes can be adjusted accordingly. The result is a decrease in the number of jobs (less communication overhead) without an increase of maximal job execution time and thus idle-time.

To find the appropriate jobsize for a given client, a normalized jobsize can be computed that is weighted by the client's speed index [Zen97]:

$$JS_{norm} = \frac{PS}{NS \cdot \sum_{i=1}^{n} SI_i} \tag{4.2}$$

where PS is the total problem size (i.e., the number of form factors), NS the number of scheduled jobs per client, and SI the speed indices of the clients. The individual jobsize for client i is then given by:

$$JS_i = JS_{norm} \cdot SI_i \tag{4.3}$$

The scheduling algorithm described so far schedules more than one job to each client (constant NS in Equation 4.2). Obviously, the last job of any client will be scheduled with a higher probability in a later part of the total radiosity computation. As only these last jobs are responsible for idle-times it seems reasonable to make them small and increase the size of earlier jobs instead. The tests mentioned above showed that throughout the scheduling process jobsizes could be decreased linearly by a factor of five, i.e., the first job was five times bigger than the last one.This effectively reduces idle-times without increasing the number of jobs (and thus communication overhead) compared to static jobsizes.

Due to the assumption that the network may be concurrently used by other processes, it is never known how much computing performance will actually be available from any client. Some user could probably start a computational intensive process on a computer that is currently working on its last job of the radiosity algorithm. This job will take much more time than expected and make the server wait while the other clients are idling. Thus, the speed index introduced earlier must be made dynamic to control a client's jobsize. The ratio between real-time and actual cpu-time for the last job has proven to be a good measure for the available

amount of a workstation's computational resources. Therefore it is used to compute the *actual speed index*:

$$SI_i^n = D \cdot SI_i^{n-1} + (1.0 - D) \cdot SI_i^0 \cdot \frac{t_{cpu}}{t_{real}} \tag{4.4}$$

The constant D is used to damp the current speed index SI_i^n, using the previous speed index and the initially specified speed index SI_i^0. A value of $D = 0.25$ has found to be reasonable [Zen97].

Despite these efforts, in a "real-life" environment the situation can still arise where some clients are working and others waiting, thereby wasting resources and increasing total computation time. To reduce this effect (as it can not completely be avoided) a *rescheduling* mechanism is used. Jobs from the list of currently active jobs are subsequently rescheduled to clients that have finished their last job and would idle otherwise. This is done again using a simple first-come first-served scheme. The last scheduled job is rescheduled first and only once, at least as long as there are other jobs which are not yet rescheduled. The first result of multiple instances of such a job is saved and all other instances are aborted. With this technique, the last job can possibly be computed in less time by a faster client which would result in a reduction of total computation time. For a successful application of rescheduling see Figure 4.3.

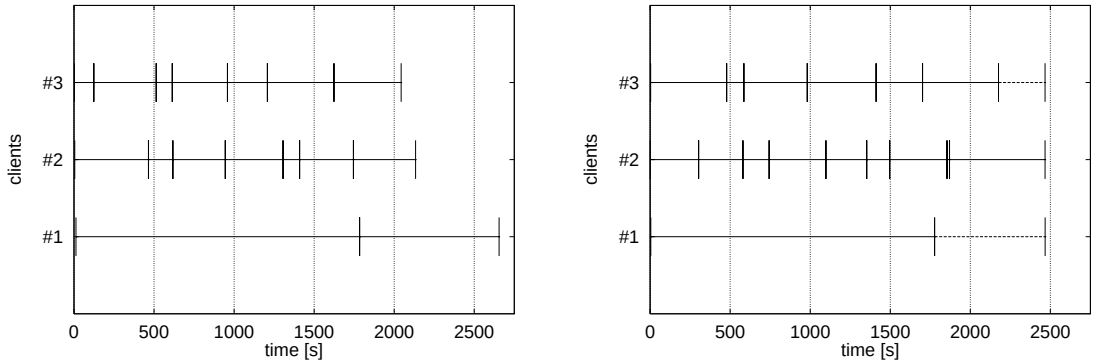

Figure 4.3 These figures show the effect of rescheduling. The left computation has a high idle-time because client #1 worked very long on its last job. On the right, this job was rescheduled and finished quicker by client #2, reducing idle- and total-time. The dotted line-segments denote aborted job-instances.

Additionally, this algorithm is very useful if some clients fail completely (e.g., they crash) before finishing their last computation. Because the failed job is rescheduled and another client can finish it, the algorithm will terminate correctly – a point of robustness.

4.5 Results

Using the empirically obtained scheduling constants from above, the distributed hierarchical radiosity algorithm was applied to several test scenes. All tests were performed using the implementation described in [Zen97]. The tests were run in a network of up to eight Silicon

Graphics Indy Workstations with MIPS R4600 processors running at 100 MHz. The machines were equipped with 32MB of RAM and they were connected by a 10 MBit network. The server machine was a SUN UltraSPARC equipped with 240 MB of RAM. Following the premise that a parallel computation is only efficient when the computing time of the distributed tasks clearly outweighs the communication time, scenes of different complexity were tested. Here, complexity does not only mean the size of the scene. Form factor computations of complex objects take longer than those for simpler ones, because the objects' ray-intersection routines dominate the object-based form factor computation.

The museum scene (Figure 3.19), that was already used for the curved-surfaces-algorithm in Chapter 3, can be regarded as a complex scene. Thus it should be well suited for the distribution of form factors. As is illustrated by Table 4.1, applying the parallel algorithm resulted in a nearly linear speedup.

# of processors	1	2	3	4	5	6	7	8
total time [s]	32695	16548	11402	8373	7550	5676	5215	4650
speedup	1.0	1.98	2.9	3.9	4.3	5.8	6.3	7.0
max. comm. time [%]	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
max. idle time [%]	0.4	2.9	3.2	3.3	4.9	4.2	6.8	9.4

Table 4.1 Statistics for the museum scene. The complex form factor computations result in a nearly linear speedup and neglectable communication costs.

During the computation, 57.3 MB of data were sent to the clients and 7.1 MB of data were sent back to the server. Thus, the communication time of 0.3% can be neglected. The moderate idle time of about 10% contains the time of distributing the energy at the server and the complete mesh refinement. With an increasing number of processors, the percentage grows because more processors are waiting for the server at the same time. The scheduling for the case of 3 processors is visualized in Figure 4.5. It can be seen that the total jobsizes are decreasing towards the end of the computation. This is a result of the hierarchical refinement of the radiosity algorithm. Most refinement steps occur at the beginning of the algorithm where the overall error in transport is still very large. After further refinement steps, fewer links have to be refined, thereby requiring a decreased number of form factors to be computed for the next iteration.

A moderately complex scene is the office scene displayed in Figure 4.4. It shows two office rooms with some furniture, that are modeled using differently sized boxes. The scene complexity is thus lower than the museum scene and contains about 1000 polygons. In Table 4.2 the corresponding statistics are given. For 8 processors a good speedup of 6.9 was obtained. The server sent 26.8 MB to the clients and received only 3.3 MB of form factor data. Due to the simpler geometry, the idle time is about 12% which is slightly worse than for the museum scene.

Finally, a very simple scene was constructed. It contains some boxes and spheres, that can be ray traced very efficiently which improves the object-based form factor computations. The communication overhead should be obvious. Figure 4.4 contains a rendering of the scene.

The results are given in Figure 4.3. Good speedups are only achievable for up to 4 processors. Using more than 7 processors for a scene of this complexity results in a performance loss

# of processors	1	2	3	4	5	6	7	8
total time [s]	4240	2190	1483	1134	926	782	681	609
speedup	1.0	1.94	2.9	3.7	4.6	5.4	6.2	6.96
max. comm. time [%]	0.99	1.3	1.3	1.2	1.3	1.5	1.4	1.6
max. idle time [%]	1.4	4.2	5.8	7.3	9.1	9.7	10.1	11.6

Table 4.2 Statistics for the medium sized office scene. Even simpler form factor tasks can lead to a nearly linear speedup.

# of processors	1	2	3	4	5	6	7	8
total time [s]	2697	1456	1052	852	714	627	546	553
speedup	1.0	1.9	2.6	3.2	3.8	4.3	4.9	4.9
max. comm. time [%]	3.8	4.2	5.2	4.9	5.3	5.2	5.4	6.5
max. idle time [%]	4.5	13.0	19.2	23.6	26.4	25.8	31.0	33.9

Table 4.3 Statistics for the simple spheres scene. A good speedup is only possible for up to 4 processors. Note the high fraction of idle time when compared to the more complex scenes.

due to the high idle time of more than 30%. The amount of data that was transferred from the server was 69.3 MB while only 8.2 MB were sent back. This is also indicated by the highest communication costs of the test scenes, ranging at more than 5%. The Gantt-chart in Figure 4.5 visualizes the communication and idle time of the processors. The empty spaces denote the computing intervals and reflect the lower bound of a useful problem size for this algorithm.

A summary of the different speedups and their scalability together with renderings of the test scenes is displayed in Figure 4.4.

4.6 Summary

In this chapter a novel approach to the distributed implementation of Hierarchical Radiosity was presented. It is based on distributing the most time consuming part of the algorithm over a network of workstations. By exploiting an underlying object-model, form factor computations could efficiently be parallelized without the enormous communication costs typically found in client-server approaches. Additionally, scenes containing curved surfaces benefit from the quality and speed improvements presented in the previous chapter.

A scheduling algorithm was developed that has found to be well suited for the target environment, i.e., a workstation cluster of unpredictable and changing CPU load. By dynamically changing jobsizes, incorporating an adaptive speed index and rescheduling of jobs at the end of each computation cycle, idle times could be minimized. As a result linear speedups for a moderate number of processors could be achieved.

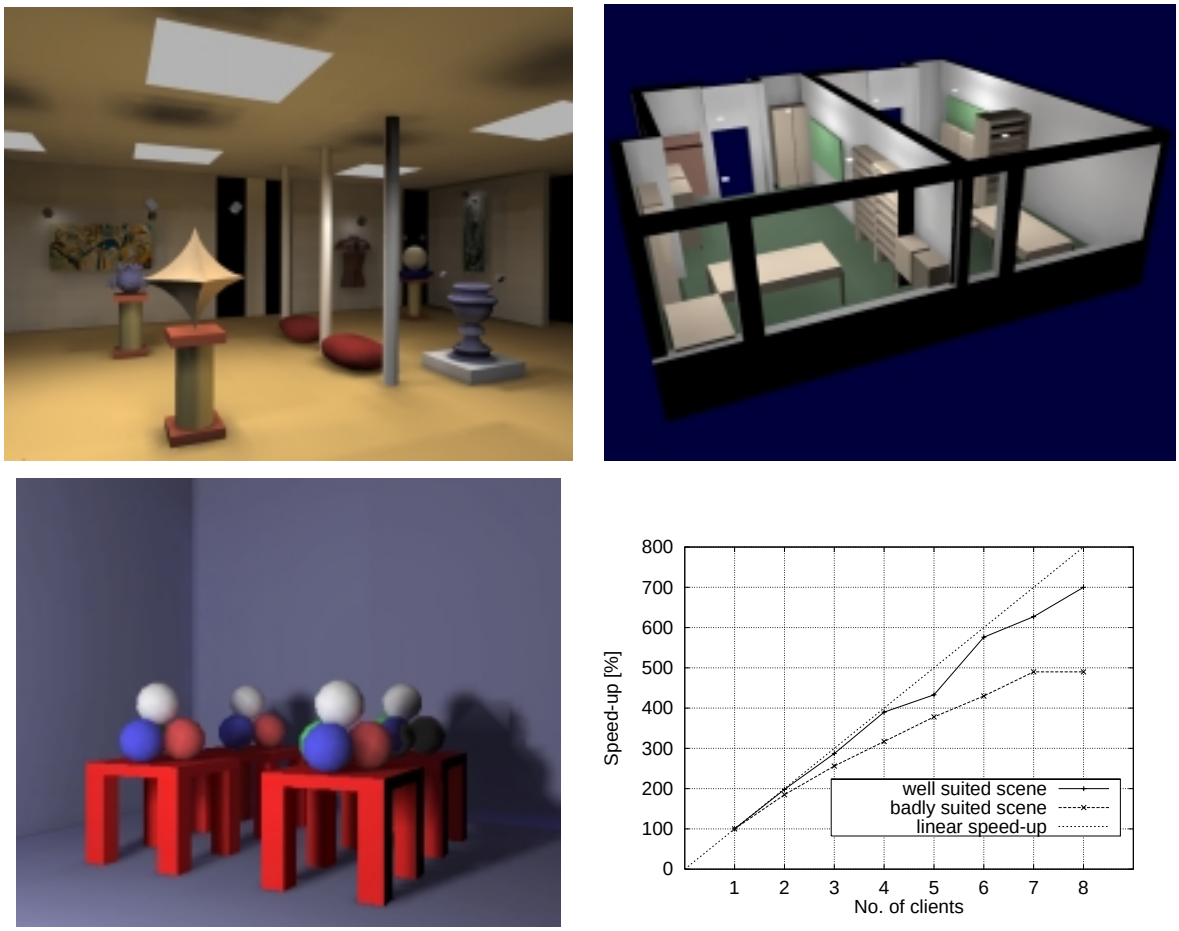

Figure 4.4 Renderings and speedups of the test scenes. From left to right and from top to bottom: museum, office and spheres. The speedups for museum and spheres are plotted in the diagram.

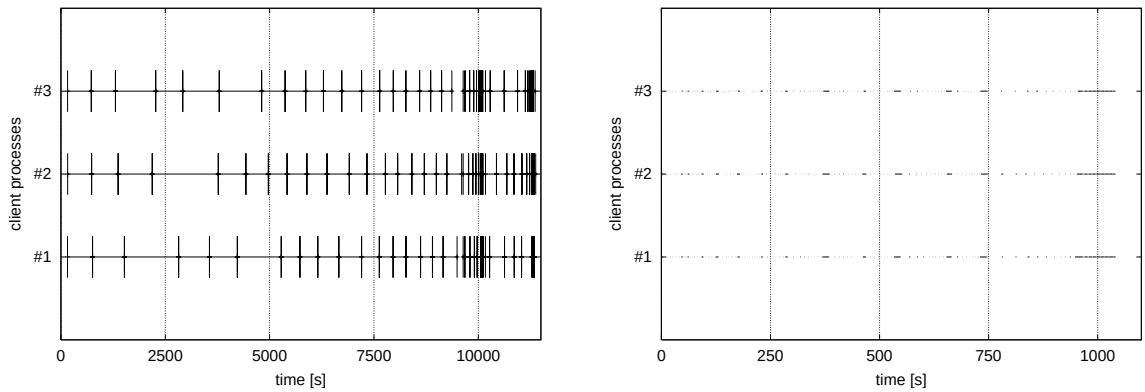

Figure 4.5 The left Gantt-chart shows the scheduling for 3 processors computing the museum scene. The right chart shows communication and idle times while computing the spheres scene.

CHAPTER 5

Efficient Clustering

5.1 Introduction

The extension of Hierarchical Radiosity to the use of object clusters greatly improved the efficiency and thus usability of radiosity computations. Due to the quadratic starting phase and excessive memory consumption of the original polygon based approach, large scale radiosity simulations for scenes containing hundreds of thousands of polygons were not feasible. In Section 2.10.4 a clustering algorithm based on isotropically scattering volumes was presented. Although this method can efficiently be applied to very complex scenes, the quality of the solution is not always satisfactory. The result is directly influenced by the structure of the employed hierarchy and the oracle that drives the refinement process. The cluster hierarchy must adapt very well to the underlying geometry to justify an approximation of many fine interactions by a single, coarser one. Additionally, the quality of the hierarchy has an impact on the execution time of the radiosity simulation. Because visibility computations between patches are too inaccurate when using the volume scattering method, ray casting must be used. The object hierarchy therefore should efficiently improve ray-surface intersection tests. Often, an additional data structure is used for this purpose, thereby increasing the algorithm's memory consumption. The second prerequisite for an accurate solution is an error-driven refinement strategy. The oracle function should incorporate error bounds on all components involved in the energy transfer.

In this chapter a new clustering strategy will be applied to Hierarchical Radiosity and its implication on visibility computations and energy transfer will be discussed. The hierarchy is built using a hybrid structuring algorithm that combines an optimized bounding volume hierarchy together with uniform spatial subdivisions for regions with regular object densities. The hierarchical subdivision is driven by a cost function, that assures the efficient use of the resulting data structure for visibility computations. The hierarchy of bounding volumes will be stored in a binary tree.

When aiming at high quality radiosity solutions an error-driven refinement strategy must be combined with a more elaborate energy transfer. Object clusters typically do not reflect light isotropically, therefore more hierarchy traversals are needed to account for the individual orientation of the clustered polygons. These quality improvements require an additional computational effort. Due to using a binary tree however, the accumulated hierarchy traversal costs may quickly become overwhelming. In order to reduce the required number of full traversals an efficient single pass gathering will be used.

The chapter is structured as follows: after presenting the clustering strategies used in the

literature so far and discussing the properties of a 'well-suited' hierarchy, the use of error bounds in radiosity clustering algorithms will be revisited. The subsequent sections describe the new clustering strategy and its advantages for radiosity clustering. Finally, it will be shown how the hierarchy can efficiently be used for radiosity computations and results obtained from rendering several large industrial scenes will be presented.

5.2 Clustering Strategies

Several different clustering strategies have been used in the context of hierarchical radiosity. They differ in the underlying data structure and in the way, how objects are inserted in the hierarchy. Often, these strategies are based on ray-tracing acceleration methods that early introduced hierarchical data structures to reduce the complexity of the ray-tracing approach [Gla89].

After describing the most often used data structures and hierarchy creation algorithms, a guideline to evaluate clustering strategies will be given.

5.2.1 Data Structures

The use of hierarchical bounding volumes reduced the time complexity of the ray-tracing algorithm from linear to logarithmic in the number of objects [RW80]. Once a larger volume is found to not being intersected by a given ray, the enclosed bounding volumes do not need to be tested anymore. For ease of use the bounding boxes are aligned to the main coordinate axis.

Another hierarchical acceleration technique is the use of octree data structures [Gla84]. An octree is built using a recursive subdivision process. Rectangular volumes are split into eight subvolumes or octants until some criterion is met. The criterion can be that a given object most tightly fits into an octree cell or that no more than a specified number of objects is contained in each cell. If a list of intersecting objects is attached to an octree cell, it can be used to quickly determine the object which is intersected by a given ray. Only a short list of candidates has to be traversed for each octree cell intersected by the ray.

To circumvent the strict cubical subdivision of an octree, *k*-d trees use planes with no fixed position to subdivide space. The position is chosen depending on the distribution of objects. This results in more tightly fitting cells, both for the objects and for the empty space between them.

The described data structures can be filled in several ways. Hasenfratz et al. classify the hierarchy construction algorithms in top-down approaches and bottom-up approaches [HDSD99]. Top-down approaches are typically very fast, because objects are inserted subsequently based only on a local decision. Bottom-up approaches however, optimize the process of insertion by using all existing objects.

5.2.2 Algorithms

In [Sil95] Sillion uses a *k*-d tree that is filled in a top-down approach. Starting with a single cell, objects are inserted in the lowest level of the hierarchy that contains them entirely. New cells are created by subdivision as needed to find the appropriate hierarchy level. Because objects are always completely enclosed by their cell no intersections between objects and cell boundaries

occur. This is important for the later energy transfer, because objects that are spread over more than a single cell contribute multiple times to the solution. However, if scenes contain many large objects they all tend to be inserted at very high levels in the hierarchy.

A top-down approach that uses an octree to store the hierarchy is presented by Christensen et al. [CLSS97]. They start with the bounding volume of the entire scene as the root cluster which is split into eight octants. Each patch that has a smaller extent than the size of an octant is inserted into the octant containing its centroid. Otherwise it is attached as a child to the current cluster. The same procedure is recursively performed for each octant. Once the number of patches in a cluster is sufficiently small, the process terminates. As a result, each patch is assigned to a single cluster and a hierarchy of bounding volumes is created.

The following clustering strategies (as well as the one used in this chapter) are based on the bounding volume algorithm of Goldsmith and Salmon [GS87]. In order to predict the quality of a bounding volume hierarchy for ray tracing, they use a cost function to find the proper insertion level for an object. This cost function is based on the observation, that the conditional probability of a ray hitting a given node if it hits the root node can be approximated as the area of the given node's bounding volume divided by the area of the root node's bounding volume:

$$Pr(r \text{ hits } B | r \text{ hits } A) = \frac{S(B)}{S(A)} \tag{5.1}$$

where r denotes a random ray, bounding volume B is enclosed by bounding volume A and $S(V)$ is the surface area of volume V. Using this cost function the construction algorithm works as follows: All objects are subsequently inserted into a tree. To find a node's suitable subtree for insertion, the subtree chosen is determined by the smallest increase in surface area of the current node's bounding volume if the object would be inserted. If, due to equal costs, multiple subtrees are possible they can be searched individually for the best location. At each node of a subtree, the same cost function is evaluated to find the optimal insertion level.

The order in which the objects are inserted into the tree can have a great impact on the quality of the resulting hierarchy. The authors tested several data orders and found, that simply shuffling the data before insertion resulted on average in the best quality. Due to the fast hierarchy construction time, when compared to the actual rendering process, several seeds can be tried for shuffling. The best hierarchy according to the cost functions is finally chosen.

Smits et al. applied the Goldsmith-Salmon algorithm in their radiosity clustering approach [SAG94]. The bounding volumes are directly used as clusters and for the ray-casting based visibility tests. Without giving further details, however, they made few modifications to the bounding volume data structure.

Gibson et al. note, that the proposed shuffling method not necessarily results in optimal hierarchies suited for radiosity computations. Instead, surfaces should be inserted in order of spatial location and decreasing size [GH96]. Their bottom-up approach uses a hierarchy of grids with decreasing cell sizes. Each level is subdivided into $2^i, i \in [0; n[$ voxels and each surface is inserted into the level whose voxel size fits its bounding box. Finally, the grid is traversed top down and each level's objects are inserted into the final bounding volume hierarchy as described above.

Hasenfratz et al. introduced two new clustering strategies by extending the k-d tree approach to allow for overlapping cells [HDSD99]. Thus, depending on a user parameter, large objects are not always inserted at high levels of the tree. In order to get rid of the empty clusters, they

finally arrange the bounding volumes of the non-empty cells in a hierarchy of bounding volumes. Because the distribution of polygons among the resulting clusters can often be rather unbalanced, a second algorithm is proposed. It uses the output of the first algorithm and applies a bounding volume optimization based on the Goldsmith-Salmon algorithm to clusters containing more than a dozen polygons.

5.2.3 Evaluation of Clustering Strategies

In order to evaluate and compare clustering strategies for Hierarchical Radiosity, Hasenfratz et al. recently presented a thorough analysis of the algorithms described above [HDSD99]. Their evaluation of clustering strategies is based on investigating the following requirements: those affecting the quality of the simulation and those affecting the overall efficiency.

Simulation Quality The quality of the light transfer is mainly influenced by the approximation quality of the clusters. Because clusters represent the enclosed geometry during the energy exchange, they should be as close to the individual original objects as possible. To avoid objects to contribute to a cluster twice, overlapping clusters must be avoided. Small object detail should be detected and enclosed by separate clusters, otherwise large empty space is contained. This contradicts the central idea of clustering, that clusters should represent the 'average' of the enclosed geometry. Thus, an optimal cluster is filled with a random distribution of similarly-sized objects.

Efficiency In order not to loose the performance gains obtained by radiosity clustering, the hierarchy creation phase must be relatively short. The preprocessing step of sophisticated optimization processes like bottom-up techniques that compare all objects with each other might take too long to justify the achievable acceleration. Depending on the application (e.g., dynamic environments) a cluster hierarchy must probably be rebuilt (at least partially) for every new frame, thereby making the time complexity of the creation step much more critical.

During the radiosity simulation, the applicability of the hierarchy to visibility computations is a very important requirement. Most visibility tests are performed using ray casting as described in the Monte-Carlo approach in Section 2.7.2. The efficiency of the hierarchy with respect to ray tracing thus greatly influences the overall performance of the radiosity simulation. The authors find that good ray-tracing hierarchies often are bad suited for radiosity clustering. The Goldsmith-Salmon algorithm [GS87], that is driven by a ray-tracing cost function tends to produces elongated bounding boxes that enclose too much empty space. This is contradictory to the requirements regarding the simulation quality as discussed above. The use of an extra ray-acceleration structure is considered to be too expensive. Large model radiosity computations already have an enormous demand on memory, therefore a single hierarchy should be sufficient.

Finally, the branching factor of the hierarchy greatly influences both, the efficiency of the simulation and the efficiency of the ray acceleration. As soon as a link to a cluster is refined, new links have to be established to all of its children. A large branching factor therefore may result in long computational times. The same is true for the accelerated ray-casting process, where branches of the hierarchy are pruned as soon as an intersection with the corresponding

nodes becomes impossible. To generate as many tree-pruning operations as possible, the tree should have a small branching ratio [GS87].

Evaluation The presented clustering strategies were applied to several large test scenes comprising hundreds of thousands of polygons. Analyzing the results with respect to the above requirements, the authors make the following observations: Clustering schemes based on hierarchical bounding volumes typically behave the most predictable but do not always result in best performance. Due to the tight fitting clusters, they generally model the approximative energy transport very well. The combination of a *k*-d tree with a local bounding volume optimization in densely populated inner nodes gives good ray-acceleration results, thereby improving overall performance (but not necessarily image quality). Finally, bottom-up construction methods (i.e., those with an optimization pass) generally produce better object hierarchies although the costs for the creation step can easily become quadratic.

In Section 5.4 a new clustering algorithm will be presented that combines an efficient ray acceleration with a tight bounding volume hierarchy in the same data structure. Thus, most of the requirements for both, the simulation quality and the overall efficiency can be met. Because the clustering strategy is applied to an error driven radiosity algorithm, the use of error bounds will be revisited in the next section.

5.3 Error Bounds

In Section 2.10.4 the use of error bounds was mentioned to improve the quality of hierarchical radiosity solutions. The adaptive mesh refinement examines the error in energy transfer to decide if an interaction at a given level is accurate enough. Due to the finite element approach however, the radiosity method only computes an approximation of the radiosity function, thus the exact error of an interaction is unknown. By exploiting the geometric and radiometric properties of the interacting objects reliable upper and lower limits on the energy transfer can be computed. These limits can be used to bound the error of an interaction, thus arriving at a more accurate mesh refinement and representation of the radiosity function.

In this Section several methods to compute upper and lower bounds on the energy transfer will be presented.

Error bounds can be classified into conservative and non-conservative error bounds [LSG94]. Conservative error bounds guarantee to always contain the exact error everywhere, which might be too pessimistic. Non-conservative error bounds on the other hand are cheaper to compute and provide tighter error bounds – with the drawback of not being always totally reliable.

5.3.1 Kernel Bounding Techniques

Lischinski et al. present a radiosity bounding algorithm for Hierarchical Radiosity that uses conservative error bounds [LSG94]. The exact radiosity $B(x)$ is bound from below and from above by two piecewise constant functions $\underline{B}(x)$ and $\overline{B}(x)$, such that for all surface elements S_i:

$$\underline{B}_i \leq B(x) \leq \overline{B}_i \tag{5.2}$$

To actually compute these bounds, variations on the kernel function expressed by the form factor are considered. Upper and lower bounds on the form factor are computed by storing the maximum and the minimum of the unoccluded point-to-polygon form factor while sampling the receiving patch. During energy exchange, these values are used to maintain upper and lower bounds of a patch's radiosity. The bounds are treated similar to the standard patch radiosities, i.e., they are gathered and propagated through the hierarchy. While gathering upper bounds however, care must be taken to guarantee convergence of the solution process. The sum of the upper form factor bounds must never exceed 1, otherwise the corresponding form factor matrix would not necessarily be diagonally dominant (Equation 2.35). Thus, only the brightest sources should be considered which might require an additional sorting step while updating the bounds at a leaf node.

Assuming that the values of B(x) over a surface element are uniformly distributed between $\underline{B}_i$ and $\overline{B}_i$ the L_1 norm can be used to express an upper bound on the local error of an element:

$$||B(x) - B_i||_1 \approx A_i(\overline{B}_i - \underline{B}_i)/4 \tag{5.3}$$

The L_1 norm corresponds to a bound on the total power and can be computed for any inner node as the sum of its children's error in L_1 norm.

The original hierarchical radiosity algorithm uses a brightness-weighted refinement, i.e., a link is refined if the product of radiosity B and form factor F exceeds a given threshold (BF-refinement, see Page 40). As a result, all links transport roughly the same amount of energy [HSA91]. Using error bounds however, links can be detected and refined that have the greatest effect on the total solution error. Lischinski et al. compute the error in transferred energy as $(\overline{K}_{ij} - \underline{K}_{ij})\overline{B}_j$, where K_{ij} denotes the form factor between the two corresponding surface elements including visibility [LSG94]. If this error exceeds a threshold, the patch that contributes to the error the most must be refined. To decide whether to refine the receiving or the sending patch, the following heuristic is suggested: the receiver i is refined if the form factor error is too large and the sender j is refined if the error in radiosity is too large. More formally, receiver node i is subdivided if:

$$(\overline{K}_{ij} - \underline{K}_{ij})\overline{B}_j \geq \overline{K}_{ij}(\overline{B}_j - \underline{B}_j) \tag{5.4}$$

In [SAG94] Smits et al. describe two strategies for computing non-conservative bounds on the energy transfer in a clustering algorithm. Again, only variations of the kernel function are considered. The algorithm uses the two linking strategies (i.e., α-links and β-links) that were already introduced on page 45 in Section 2.10.4.

To bound the energy transfer between clusters using α-links, the maximum value of the kernel function (i.e., the unoccluded form factor) between all pairs of enclosed patches can be used to find an upper bound. This would require $\mathcal{O}(mn)$ operations if m and n are the numbers of the surfaces contained in the corresponding clusters. By splitting the kernel function $k(x,y)$ into a source related term k^s and a receiver related term k^r, these bounds can be computed more efficiently:

$$k(x,y) = \rho(x)\frac{\cos\theta_1 \cos\theta_2}{||x-y||^2} \Rightarrow$$

$$k^r(x,y) \equiv \rho(x)\cos\theta_1, \quad k^s(x,y) \equiv \frac{\cos\theta_2}{||x-y||^2}$$

Thus, when computing the maximum value of k^s over the sending cluster and then computing the maximum value of k^r over the receiving cluster only $\mathcal{O}(m+n)$ operations must be performed. Assuming zero as a pessimistic lower bound, the energy transfer can be bound from below and from above.

A much simpler and less expensive bounding technique is achieved using β-links. If the kernel function is approximated without taking the orientation of the surfaces into account, it can be written as a function k^d:

$$k(x,y) \leq k^d(x,y) \equiv \frac{1}{||x-y||^2} \tag{5.5}$$

Thus, k^d can be used to compute an upper bound on the transfer by bounding the cosines from above by 1.

Gathering energy using both link types works by maintaining minimum, maximum and average values of the functions k^s, k^r and k^d over each cluster. The function values are obtained by sampling each cluster's bounding volume. Link refinement between clusters works by first examining the β-bound. If it is too inaccurate, the more expensive α-bound is checked. If it is still not accurate enough, the larger cluster is refined. If both cluster hierarchies reach the surface level the standard hierarchical radiosity refinement is performed.

The following subsection describes more complete error bounding techniques that take bounds on all components into account that influence the error in radiosity transfer. This includes bounds on the visibility, on the reflectance and on the irradiance of a surface. Additionally non-zero lower bounds can often be computed which results in tighter error bounds.

5.3.2 Bounds on the Radiosity Transfer

Gibson et al. [GH96] propose storing irradiance with each element instead of radiosity to have a better control on the error incurred by varying reflectance values on both sides of a link. The use of texture maps as reflectors and as emitters suggests this approach. Because the local reflectance operator converts irradiance at the receiver patch into radiosity (Section 2.9.4) the approximated radiosity $\tilde{B}$ over a patch can be written as:

$$\tilde{B}_i = \rho_i(F_{ij}V_{ij})(\rho_j E_j) \tag{5.6}$$

where ρ is the reflectivity, F the unoccluded form factor, V the visibility term and E the irradiance of the source. In the following, non-conservative upper and lower bounds on all components of Equation 5.6 will be computed.

Bounds on the unoccluded form factor can be obtained by sampling source and receiver and storing minimum and maximum values as described above. Form factors between clusters and between clusters and surfaces are computed using the methods proposed by Sillion [Sil95] as described in Section 2.10.4. This approach is based on the extinction coefficient that depends on the density of surface patches in a cluster. To arrive at a better form factor approximation and thus better error bounds, the orientation of the surfaces in a cluster must be taken into

account. This can be achieved by using the projected area (i.e., the area weighted by the cosine between the normal and the direction of transfer) instead of simply the area as in [Sil95]. Thus, the directional extinction coefficient must be recomputed for each transfer by enumerating all surfaces in the source cluster and computing the cosine values.

If source and receiver are partially occluded, the conservative visibility bounds zero and one can always be used. Due to the visibility sampling approach however, a large error would also be reported if only a single ray was occluded. Because this will later drive the refinement process, unnecessary subdivision might occur. To force subdivision of interactions that are occluded at about 50%, which typically occurs at shadow boundaries, Gibson et al. propose the following lower visibility bound:

$$|\overline{v} - (1 - \overline{v})| = |2\overline{v} - 1| \tag{5.7}$$

where $\overline{v}$ denotes the partial visibility of a patch. Actually, Equation 5.7 corresponds to the difference of the visible and invisible fractions. For performance reasons this technique is only applied to visibility computations between surface patches because it requires more rays to accurately compute $\overline{v}$. For visibility between clusters a lower bound of zero can be used.

Reflectance bounds can be computed by precomputing a reflectance map for each texture applied to a surface. The texture is sampled at appropriate levels of decreasing resolution to find minimum, maximum and average reflectance values that can be used in the energy transfer between surface elements.

Bounds on the irradiance are computed during the gathering step as explained above ([LSG94]) by using the upper, lower and average values of the form factor.

Having computed upper and lower bounds on the transfer error according to Equation 5.6 it must be decided if the source or the receiver has to be refined in order to decrease the error. Similar to the approach of Smits et al. [SAG94] an expression of the receiver-related error ε_r and one for the source-related error ε_s can be used:

$$\varepsilon_r = (\lceil \rho \rceil_i \lceil F \rceil_{ij} - \lfloor \rho \rfloor_i \lfloor F \rfloor_{ij}) \rho_j E_j$$
$$\varepsilon_s = \rho_i F_{ij} (\lceil \rho \rceil_j \lceil E \rceil_j - \lfloor \rho \rfloor_j \lfloor E \rfloor_j)$$

where the notation $\lceil x \rceil, \lfloor x \rfloor$ denotes the upper and lower bounds respectively.

Thus, the variation of the receiver reflectance and the unoccluded form factor are compared to the source's reflectance and irradiance bounds. Subdivision of the source occurs if $\varepsilon_s \geq \varepsilon_r$, otherwise the receiver is subdivided.

A general framework to compute tight conservative bounds on the light transport is given by Stamminger et al.[SSS98]. All nodes in the cluster hierarchy are considered as objects that must provide a common set of methods. These methods are used to query the various geometric and radiometric properties needed to actually compute the error bounds. Therefore, no further distinction between clusters and surfaces is needed and optimized versions of these methods are possible to support arbitrary objects.

A basic requirement for each object is the computation of its bounding box, that can be used to bound several geometric values. The distances and directions of rays between two objects can be bound by using the union of both bounding boxes. The projected area, that influences the

amount of energy received from a sender, can be bound by using the projected bounding box. Using these bounds, the solid angle of the sender can be bound by the quotient of the bounds for the projected area and the distances.

In contrast to other error bounding techniques Stamminger et al. take the possible self-interaction of non-convex objects into account. By assuming a closed scene, that can always be guaranteed by enclosing the whole scene by a sphere, form factors from a non-convex object that do not sum up to 1 indicate self-interaction. Thus, form factors from an object to itself can be approximated by $F_{ii} = 1 - \sum_j F_{ij}$.

To decide if a link has to be refined, the authors propose to use a measure similar to the L_1-norm (Equation 5.3). To better support arbitrary objects however, the radius of the object's bounding sphere is used instead of the area.

Earlier error bounding techniques always computed non-conservative error bounds on visibility due to the inaccurate sampling approach. Sampling a cluster in several directions and using the resulting transparency value ([Sil95]) can be arbitrarily wrong. Two small patches that have a cluster in between might be considered occluded although the patches in the cluster are not uniformly distributed and thus no occlusion takes place. To obtain better visibility bounds, the approximated visibility due to the sampling approach may only be used, if the size of the cluster is smaller than the cross section of the set of all rays between the objects.

5.4 A New Clustering Strategy

In this Section a new clustering strategy for Hierarchical Radiosity will be presented that conforms to most of the requirements discussed in Section 5.2.3 [MSF99a, MSF99b]. The technique is based on an optimization of the Goldsmith-Salmon algorithm [GS87] but evaluates the cost function in a different way [MF99]. It takes all locally possible orders in which to insert objects in the hierarchy into consideration while still having a time complexity of $\mathcal{O}(n \log n)$.

5.4.1 Overview

The clustering algorithm creates a hierarchy of bounding volumes which define the individual clusters for the radiosity process. It examines the existing objects and their mutual spatial relationship. This yields very high quality hierarchies with structures which are often called *intuitive* or *natural*. The cluster hierarchies adapt well to the spatial distribution of objects:

- Spatially separated geometry is identified and placed into separate clusters.
- Overlaps between clusters are reduced and tight bounding volumes are built by finding bounding volumes with minimal surface areas.

The basic idea behind the construction scheme is close to the median cut scheme described in [Grö95, KK86] which recursively computes partitions of the objects in two equally sized subsets based on their spatial location relative to a coordinate axis. This scheme is extended by introducing a cost function similar to [GS87] to gain significantly better scene partitions. It will be shown that the data structure can be built very efficiently.

5.4.2 Construction

The hierarchical bounding volume optimization method recursively subdivides the set of scene objects into two disjoint subdivision entities. Each node of the corresponding binary tree represents a specific subscene of the whole scene. At each subdivision level, the individual objects are assigned to exactly one subdivision entity of that level. No objects are split, hence no new objects (e.g. polygons, triangles, etc.) are created by this method.

Starting from the root node, the objects are sorted along all major coordinate axes, where the center of an object's bounding box serves as sorting key. Based on these sorted lists, the potential partitioning positions along each axis for each entry in the respective list are evaluated by splitting the sorted list of objects into a *left* and *right* part. In contrast to the median cut scheme applied in [Grö95], no predefined partitioning position is used. Instead, a cost function is used that describes the approximated costs computing the ray/scene-intersection for a specific subdivision position. By minimizing the cost function over *all* partitioning positions, an optimal subdivision position is obtained which generates two new subdivision entities; one containing the *left* objects, one containing the *right* objects (Figure 5.1). The subdivision process terminates for subscenes which contain only a single object.

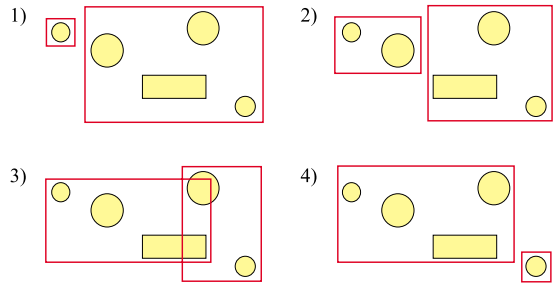

Figure 5.1 Possible object partitions along a single coordinate axis for a simple example scene containing 5 objects. The partition with minimal costs according to function C will be used in the subsequent recursion step.

The cost function used has already been successfully applied in a ray-tracing environment and has shown its superiority compared to other bounding volume schemes [MF99]. The costs of a subdivision entity H, with children H_{left} and H_{right}, is given by:

$$C_H(axis) = \frac{S(H_{left})}{S(H)} \cdot |H_{left}| + \frac{S(H_{right})}{S(H)} \cdot |H_{right}| \tag{5.8}$$

where $|H|$ is the number of objects within hierarchy H, $S(H)$ is the surface area of the bounding box associated to scene H and *axis* is one of $\{X, Y, Z\}$.

Applying hierarchies based on this cost function to radiosity clustering is a direct consequence of the requirements discussed in Section 5.2.3:

- An efficient ray-acceleration scheme is needed for fast visibility computations based on ray casting.
- Results in [MF99] show that this algorithm generates hierarchies that adapt well to the distribution of objects in the scene. E.g., polygons of individual objects are detected and

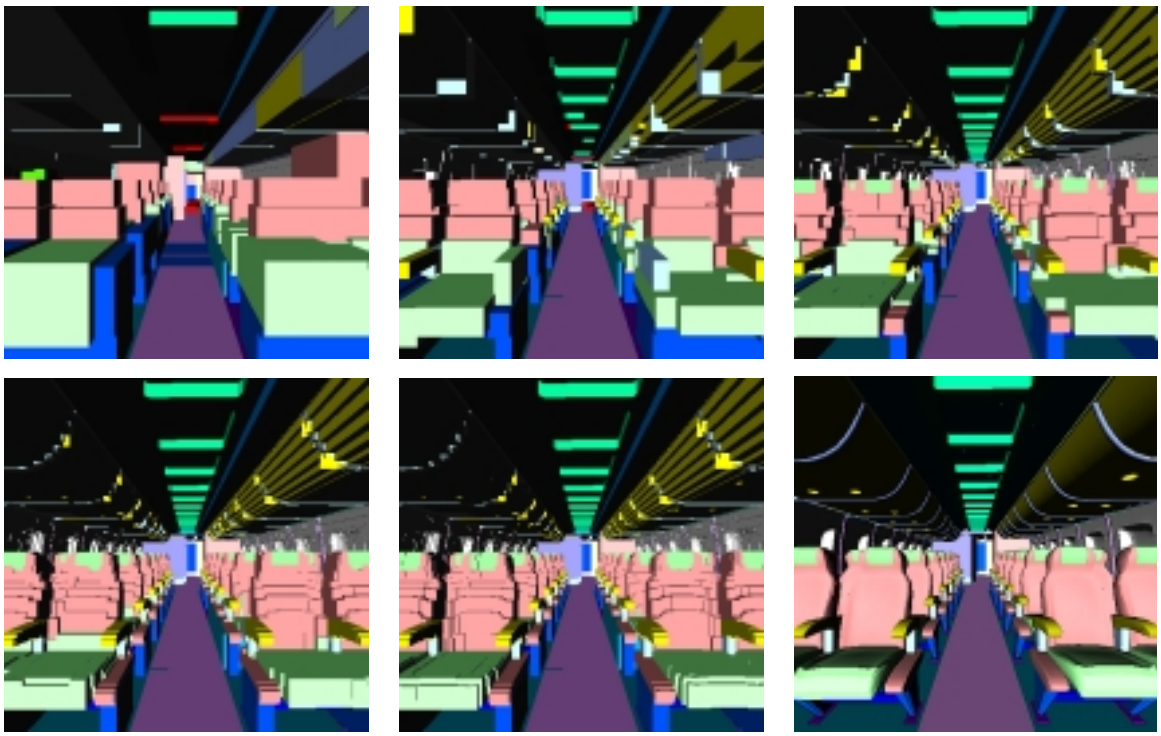

Figure 5.2 Visualization of bounding volumes of the aircraft test scene at tree levels 10, 12, 14, 16, and 18. The image at bottom right shows the Gouraud shaded original scene model.

clustered together, and overlaps of bounding volumes are minimized (although overlaps are still possible).

Construction Costs

Many previously proposed bounding volume schemes fail for large scenes because of the high computational complexity involved in the respective construction method. Their complexities easily become quadratic by checking mutual spatial relationships, thereby making them unusable for large geometries composed of many objects. The method proposed here, will need only time $\mathcal{O}(n \log n)$ in the average case while guaranteeing hierarchies of very high quality comparable to bottom-up methods.

When performing a subdivision step at an inner node of the hierarchy, the cost function has to be evaluated for all potential subdivision positions and coordinate axes. This can be done in time linear in the number of objects by incrementally unifying bounding volumes assigned to the elementary objects. The sorting should be done in a pre-process for each coordinate axis, since the mutual relative object positions will not change throughout the construction algorithm. For the analysis of run-time complexity, randomly chosen split positions are assumed for object partition, which leads to an overall run-time complexity of $\mathcal{O}(n \log n)$ in the average case. The actual subdivision position is determined by the cost function, though.

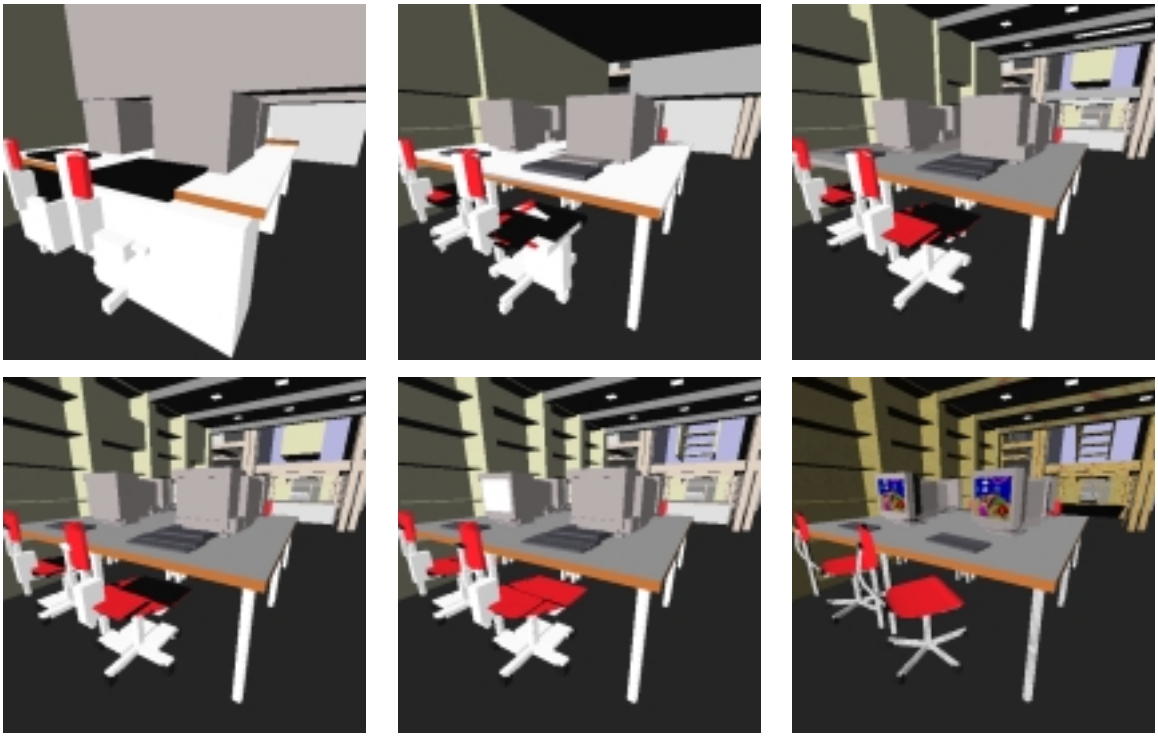

Figure 5.3 Visualization of bounding volumes of the vrlab test scene at tree levels 10, 12, 14, 16, and 18. The image at bottom right shows the Gouraud shaded original scene model.

Quality of Hierarchies

Figure 5.2 illustrates the resulting bounding box hierarchy when the bounding volume optimization is applied to a test scene (*aircraft*) containing more than 180,000 triangles. Even at a very low level of hierarchy (level 10) individual basic structural objects like seats and overhead compartments emerge. At level 12 details of the individual objects appear like armrests, backrests, or headrests. The total number of boxes at level 12 is less that 2^{12}, which is only about 2% of the total number of scene objects.

Figure 5.3 shows the results for an architectural scene containing more that 36,000 polygons. Again, close fitting bounding volumes emerge at a relatively low level of the hierarchy. At level 12 details of the scene like chairs, tables, cubicles, monitors, and keyboards are detected at a very abstract but informing degree. At level 14 smaller details like a ladder to the second floor or book shelves emerge.

The clustering scheme even detects inhomogeneous detail at a very early level of hierarchy and is thus suitable for a vast range of scene arrangements.

5.4.3 Optimizing Ray Acceleration

Besides the quality of the cluster hierarchy the performance of the radiosity algorithm is mostly influenced by the efficiency of the underlying ray-acceleration scheme. Although bounding volume hierarchies deliver acceptable performance in many cases, these schemes involve high

tree traversal costs which can often be reduced by using hierarchical grid approaches. To answer visibility queries, a novel hybrid ray-acceleration scheme has been presented in [MF99] that combines the advantages of bounding volume hierarchies with uniform grid approaches. It could be shown that the performance is at least equal to similar schemes like [CDP95] and [KS97]. In the following, the major subdivision algorithm will be summarized.

Given a bounding volume hierarchy, the goal is to detect inner scene nodes that are suitable base nodes for a local space subdivision. To achieve this goal, scene nodes are recursively classified based on the

- **surface area** of neighbor hierarchy nodes
- **volume** of neighbor hierarchy nodes
- **average size** of elementary objects below a hierarchy node

An inner scene node is classified as a suitable base node for a uniform space subdivision if the surface area and the volume of the node are not significantly larger than respectively summed values of its child nodes. Also, the elementary objects below each child node must have similar size to justify a locally uniform space subdivision for this sub-scene. The involved threshold constants were determined empirically.

Additionally, all scene nodes hold a counter representing the number of *uniform* classified sub-nodes which will be used in the actual space subdivision phase. In the next phase, uniform space subdivisions are built for sub-scenes marked in the previous step. This is efficiently achieved by recursively subdividing the bounding box of a scene node along the dominant coordinate axis. The node counter is used to determine the number of voxels or subdivisions. The available bounding volume hierarchy can be used to speed-up the voxel initialization significantly by applying hierarchical voxel membership tests.

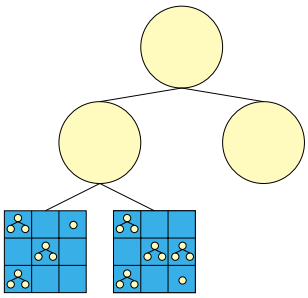

Figure 5.4 Combining bounding volume hierarchies and space subdivisions.

Figure 5.4 shows an example of a resulting data structure. A scene is partitioned into two (possibly distant) subscenes using a bounding volume hierarchy. One subscene contains many subscenes uniformly distributed in space, thus we are building a uniform spatial subdivision to represent the subscene. Note that highly inhomogeneous detail contained in one of the resulting voxels, could again be modeled using a hierarchy of bounding volumes.

Construction Costs

Because run-time complexity and space complexity of the space-subdivision approach is linear in the number of scene objects, the overall complexity of the cluster construction is dominated by the time to build the bounding volume hierarchy. This is a remarkable fact, because it implies, that the whole scene voxelization can be done in a very short time, provided the scene hierarchy from section 5.4.2 is available. The scene hierarchy could be computed *once*, and the result could be saved to an external storage medium. A fast voxelization based on the precomputed scene hierarchy could then be computed whenever using the scene.

5.5 Radiosity with Optimized Clusters

In order to achieve accurate radiosity solutions of very large scenes in short time, an efficient clustering strategy must be combined with an error driven solution process. The use of error bounds as discussed in Section 5.3 has led to severe improvements in simulation quality of the hierarchical radiosity algorithm and gives more predictable results than the simpler BF-refinement. Thus, its integration in a radiosity clustering algorithm is mandatory. The possible increase in execution time due to additional computing steps however, especially when computing very large solutions, must be addressed. The clustering algorithm of the previous section produces very deep binary trees. In order not to waste the effort of finding a good cluster hierarchy, the number of expensive tree traversals during the energy transport and while computing error bounds must be minimized. This will be the topic of this section.

5.5.1 Overview of the Implementation

The clustering algorithm has been combined with the error bounding technique of Gibson et al. that was presented in Section 5.3.2. It computes non-conservative bounds on the form factor, visibility, reflectance and irradiance, thereby taking the orientation of clustered polygons into account. When compared to standard BF-refinement, the error driven approach leads to fewer links in favor of more subdivisions at the element level. This improves the detection of shadow boundaries and thus enhances the overall visual quality of the radiosity solution [GH96].

An important detail that was not discussed in Section 5.3.2 is the way how energy is transferred from and to clusters. Energy received by a cluster is directly deposited on the contained surfaces obeying their orientation with respect to the source. An iteration process that pushes the energy down the tree is started once the top level surfaces are reached. When energy is gathered from a cluster, surface orientation inside that cluster also plays an important role. To account for the non-diffuse reflection property of a surface cluster, the reflection of each contained surface is weighted by its orientation to the receiver.

Visibility calculations between patches or between patches and clusters are done via ray casting. Only if visibility between two clusters is needed, the approximative visibility approach based on a voxel grid with extinction coefficients [Sil95] is used.

5.5.2 Using the Cluster Hierarchy

The algorithm described in section 5.4.2 builds a binary tree containing a bounding volume hierarchy. If the algorithm detected regions of regular object densities inner nodes encapsulating these regions are specially marked. Inside these regular nodes, however, the bounding volume hierarchy still persists. This is important, because the algorithm might find regular regions quite early, thus probably resulting in no hierarchy at all. Because the same hierarchy should be used for ray tracing (i.e. visibility checks) and for clustering both data structures have to be combined. Ray tracing can benefit from regular regions whereas the radiosity algorithm needs the full hierarchy. Using object oriented programming, nodes with a regular structure can be derived from standard inner nodes. The method performing a ray-intersection test is then overloaded to exploit the local data structure. Instead of just propagating the test to their child nodes, regular nodes use a local voxel array created during the construction of the hierarchy.

When applying the radiosity clustering algorithm using this data structure to very large scenes, the time spent during gathering energy over the links and propagating it through the hierarchy dominates the execution time. The refinement steps consisting of form factor calculations, visibility checks, and subdivision of links however, only uses a fraction of the gathering time. The reason for both observations lies in the nature of the employed hierarchy. The subdivision of links is very fast due to the branching factor of two in a binary tree. The visibility checks are performed either approximatively or based on ray casting which greatly benefits from the tight fitting bounding volumes and the local voxel arrays. The problem encountered in the gathering step however, is the depth of the binary tree.

Optimizing the Directional Transfer

As mentioned above, propagation of energy in a cluster is performed by depositing energy directly to the inner surfaces. This means traversing the hierarchy below a cluster to enumerate all leaves and computing a dot product of each surface normal with the direction of transfer. Thus, performing this step for each link and at all levels of the hierarchy results in an excessive number of partial sweeps through the hierarchy. Because the same traversal has also to be done for the sending cluster, deep hierarchies degrade the performance of this process. To optimize the number of iterations on both sides (sender and receiver) the gathering step for clusters first loops over all links. While visiting each link, the sender's weighted radiosity is computed once and pushed on a stack together with the direction of transfer. After all links are visited, the receiving cluster's resembling surfaces are enumerated and the saved radiosities are accumulated on these children, again weighted by their orientation. This results in only two traversals for each link. This approach is similar to the α-links proposed by Smits et al. [SAG94] that were discussed in Section 5.3.1. In the implementation proposed here however, the direction of transfer for each individual contribution to the receiver's radiance is taken into account. Smits computes a single average value for the sender and uses it to update all patches of the receiving cluster which can be wrong if most patches are oriented sideways regarding the direction of transfer. Using the technique described above a higher accuracy is obtained without increasing the complexity. Figure 5.5 illustrates the need for a more sophisticated energy transfer between clusters. The local transfer direction typically diverge essentially from the averaged transfer direction.

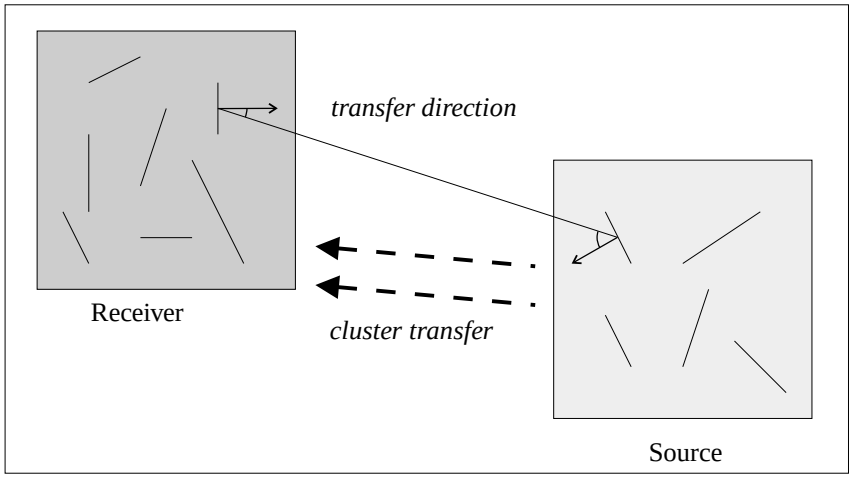

Figure 5.5 Directional transfer. The mutual orientations of the interacting surfaces must be accounted for during the energy transport between clusters.

Efficiently Solving the System

The original hierarchical radiosity algorithm [HSA91] and subsequent clustering algorithms [Sil95, SAG94] split the solution process in two passes. In a gathering step the energy is transferred across the links, which is done for all links of all nodes. In a second pass the radiosities must be swept to each object's parents and children to obtain the proper amount of energy at all levels of the hierarchy. The radiosity received by a parent node is pushed down the hierarchy to the children and on the way up the parents receive the area-weighted average radiosities from their children. The separation of a *gathering* and a *push-pull* pass corresponds to the Jacobi iteration, where the solution vector is only updated after a full iteration step [SP94].

In Section 2.10.3 it was shown how the faster converging Gauss-Seidel iteration can be used to speed up the surface based hierarchical radiosity algorithm. See Figure 2.24 for the corresponding pseudo-code. As noted in [SSSS98], for radiosity clustering algorithms this technique can not be used directly. The surface based approach assumes convex initial surfaces that can not gather energy from itself. Thus, the push-pull step can be performed whenever needed, i.e., after visiting all links at the current hierarchy level. In the clustering approach, this would result in an inconsistency of the energy balance. To always keep the hierarchy consistent, the gathering of energy at all levels of the hierarchy has to be finished before the radiosity can be propagated through the hierarchy.

To benefit from the faster convergion rate of the Gauss-Seidel iteration, which is crucial to minimize the number of full tree traversals, the push-pull step must be modified slightly [GH96]. Instead of propagating the gathered radiosity through the hierarchy of clusters, only the surfaces are considered. This can be achieved by immediately storing radiosity gathered by a cluster together with the direction of transfer onto the surface children, thereby skipping all cluster nodes in between. As soon as the gathering process reaches the surface level (i.e., the leaves of the cluster hierarchy), the stored radiosities are pushed down the surface hierarchy. Area averaging is used on the upward pass to compute an average radiosity value for each cluster. Thus, energy can be propagated in a single sweep without an additional push-pull step

as required for Jacobi iterations.

5.6 Results

To test the overall behavior and efficiency of the new clustering algorithm radiosity solutions for several large scenes containing up to 180,000 polygons were computed (see Figure 5.6).

The scenes chosen cover a broad range of features that are typically not found in 'synthetic' test scenes. The *aircraft* is a highly tesselated model of the interior of an aircraft with mainly curved surfaces[1]. *wichmann* is an architectural model of an office building containing moderately subdivided furniture and many large light sources. Most surfaces of the scene are textured. The *vrlab* scene has lower overall complexity than the former one but here the chairs are highly tesselated. The *atrium* scene has the lowest complexity of the test scenes, but it is rather irregular due to several trees on the ground floor.

Solution times and additional statistics are given in Table 5.7 (all tests were run on a 250MHz R10k with 2GB RAM). The hierarchy creation time is the time that was used to construct the optimized bounding volume hierarchy. This step has to be regarded as a preprocess. To measure the quality of the clustering algorithm, the error threshold was chosen to not subdivide more than about 20% of the initial polygons. This ensures that the timings reflect the quality of the bounding volume hierarchy as well as the ray-tracing speed. Finer error thresholds shift the main computational effort towards ray tracing.

A problem that can be noticed in the rendering of the airplane model is the overlap of bounding volumes in the cluster hierarchy. The result is an inhomogeneous illumination of nearby small patches.

Solutions of higher quality were computed for scene *wichmann* (see Figure 5.8) and for another architectural scene (see Figure 5.9). The architectural scene comprises about 20,000 polygons that are completely textured. Rendering time was about 15 min. All images were rendered using bilinear interpolation provided by graphics hardware.

The successful application of the new algorithm to these large scenes confirms the results regarding the quality of the bounding volume hierarchy documented in Figures 5.2 and 5.3. Beyond these observations the new clustering strategy closely matches several requirements found to be useful in clustering algorithms as discussed in Section 5.2.3: The creation phase of the hierarchy is very time efficient and tightly fitting bounding volumes are generated. These bounding volumes are stored in a binary tree, which allows for fast link refinement steps. Additionally, the pure ray-tracing speed of the bounding volume hierarchy combined with voxel grids has been shown to be superior to most other approaches [MF99], that have typically been used previously for radiosity clustering.

5.7 Summary

In this chapter the application of the radiosity method to scenes containing hundreds of thousands of polygons has been discussed. The use of clustering methods has been proven to be

[1] The aircraft model was kindly provided by LightWork Design Ltd.

a possible way of dealing with these large scenes that were impossible to handle by the surface based hierarchical radiosity algorithm. The grouping of surfaces or objects to a hierarchy of clusters that efficiently supports the needs of the radiosity algorithm is a non trivial task. The structure of the hierarchy has a great impact on both, the quality and the efficiency of the solution. Additionally, these properties are influenced by the refinement strategy that should incorporate error bounds to most accurately approximate the error in energy transport.

The requirements of a hierarchy well suited for radiosity clustering were presented. This includes an accurate adaptation to the expected energy flow and the efficient support for visibility computations based on ray casting. Following these guidelines, an algorithm was developed that closely meets these requirements. In order to achieve solutions of high visual quality, error bounding techniques were presented and combined with the new clustering strategy. The additional computational effort could be reduced by optimizing the energy transport with regards to the presented data structure, that uses in very deep hierarchies. The number of full tree traversals could be minimized which resulted in short solution times even for scenes containing up to 180,000 polygons.

Figure 5.6 Renderings of the four test scenes. From left to right and from top to bottom: aircraft (Light-Work Design Ltd.), wichmann, atrium, vrlab

	aircraft	*wichmann*	*atrium*	*vrlab*
number of polygons	183,114	59,844	13,558	36,337
max. hierarchy depth	26	27	25	27
bvol hierarchy creation [min:sec]	0:53	0:22	0:03	0:12
solution time [min:sec]	14:37	7:01	3:29	6:55

Figure 5.7 Solution times and statistics for the rendered test scenes.

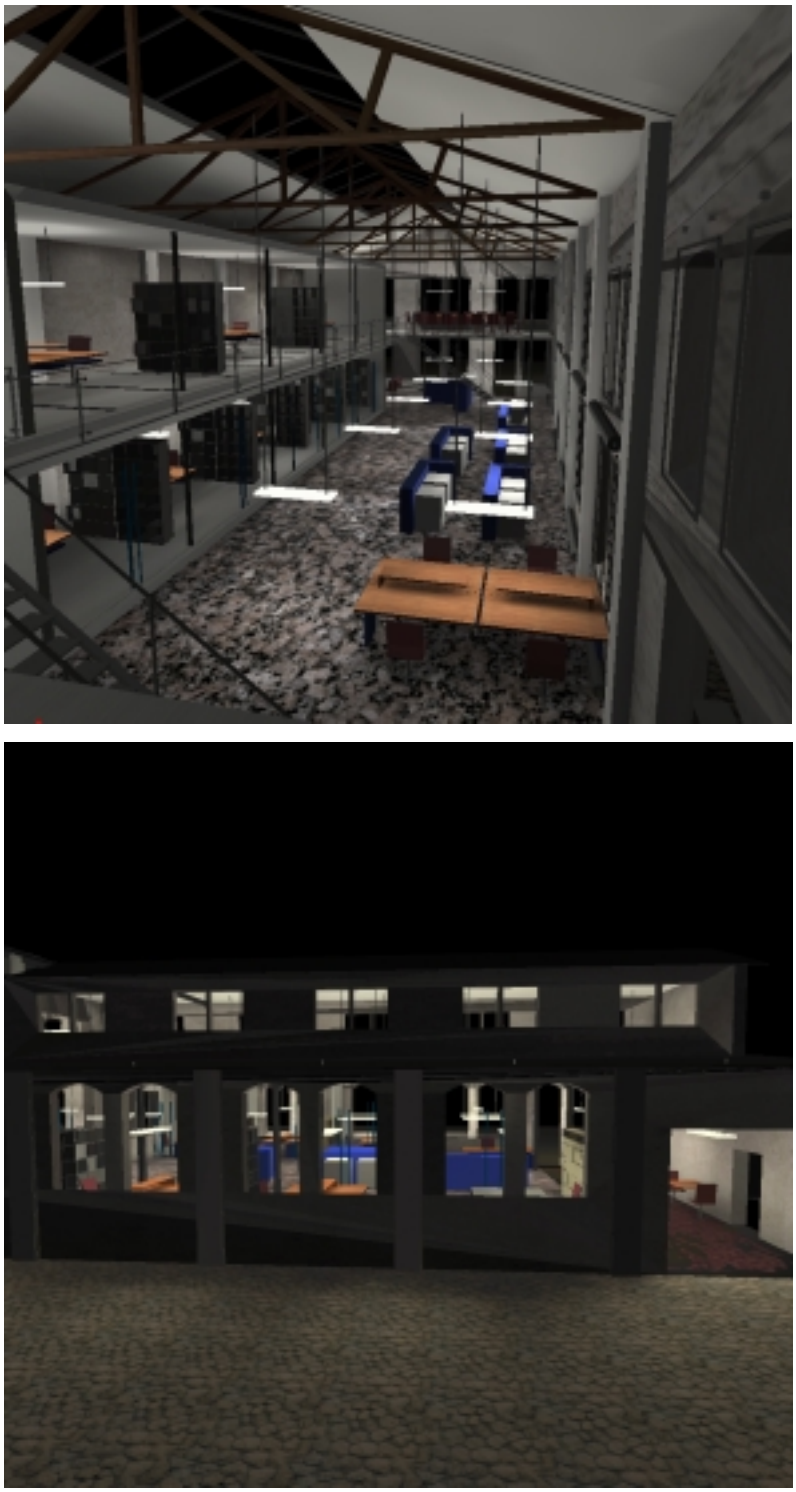

Figure 5.8 High quality solutions of the scene wichmann (47 min)

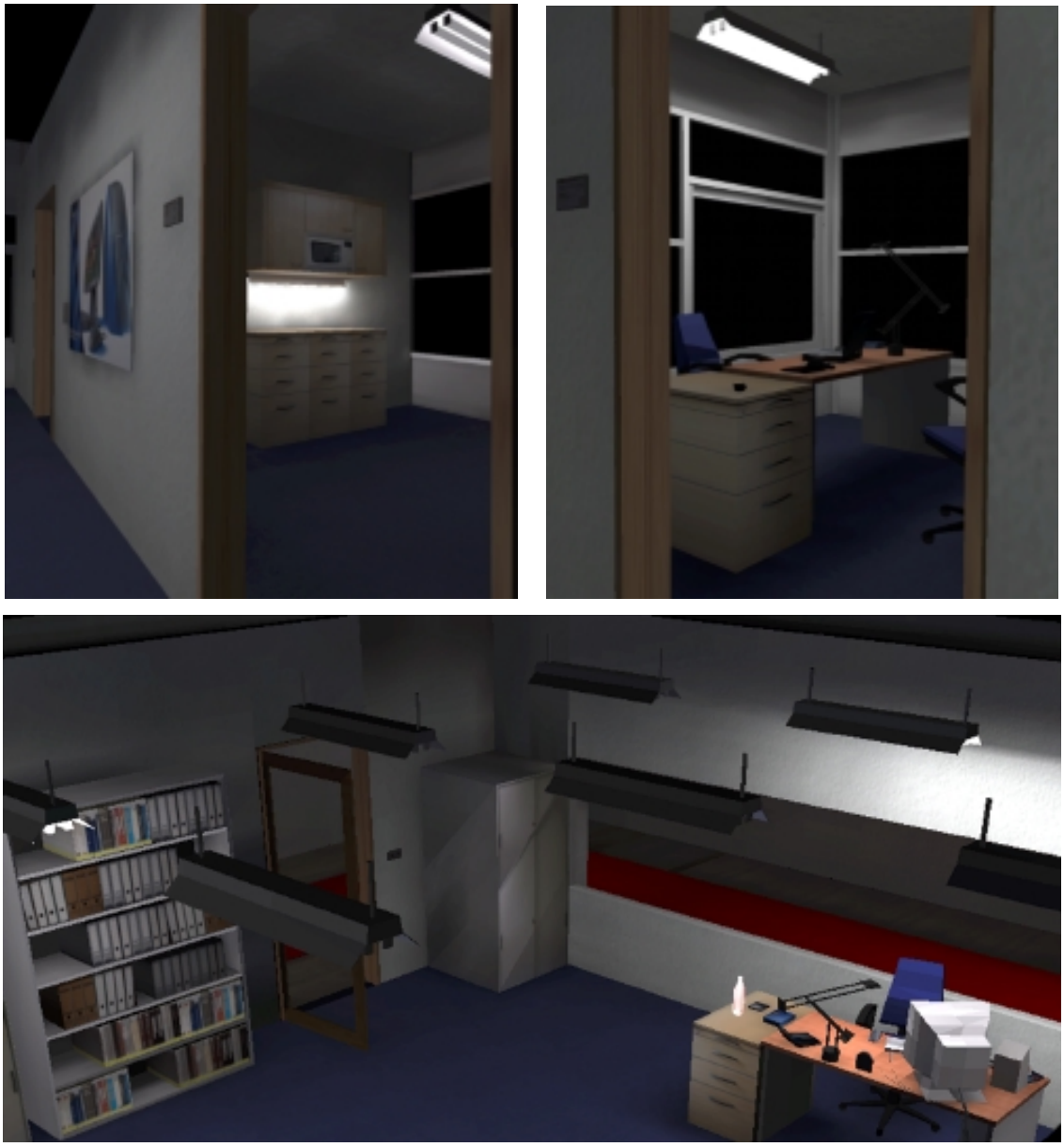

Figure 5.9 High quality solutions of an office scene (15 min)

CHAPTER 6

Conclusion

In this thesis we have presented several methods to improve hierarchical radiosity computations. In contrast to other approaches, the work of this thesis was led by the paradigm of using object information wherever possible. Most classical approaches, including the hierarchical radiosity algorithm ignore the origin of the polygons that are used to approximate a scene and that are the input to finite element algorithms. If the origin of the polygons is considered, additional information that was created during the modeling stage can be exploitet at no extra cost. As a result, the speed and the quality of rendering algorithms can be improved effectively.

6.1 Thesis Summary

By using object information several contributions to the field of hierarchical radiosity rendering could be made. The development of an object-based meshing scheme drastically reduced the time complexity of Hierarchical Radiosity when rendering curved surfaces. By providing an additional parameter that controls the curvature of the radiosity mesh, quality improvements of the solution can be obtained with linear costs. In contrast to the original algorithm, mesh refinement of curved surfaces can be performed at any stage of the algorithm. This avoids the need to define the final mesh quality in advance which suffers from the quadratic starting phase of Hierarchical Radiosity. The basic idea to efficiently support curved surfaces was the use of ray tracing to adjust mesh vertices after a planar subdivision. With the knowledge of the underlying object, the mesh can always follow the object's shape, thereby improving the approximation while simultaneously computing the radiosity solution.

The use of parallel processing has always been proposed as a valuable tool to speed up scientific computations. The hierarchical radiosity algorithm however, mainly consists of computing tasks that are dependent on each other. Computing the energy balance in a scene requires for each surface to consider most other surfaces' incfluence. This results in high communication costs, that often prohibit a performance increase linear in the number of emplyoed processors. Therefore, most researchers used special purpose hardware to implement their parallel algorithms. A parallel computer typically provides fast data paths, that effectively support high bandwidth communication. A more economic approach is to use workstations that share a common network like Ethernet that can be found in most computing environments.

We have developed a distributed hierarchical radiosity algorithm that runs in a network of workstations. By exploiting object information communication costs could be minimized which is even more important in a comparable slow network. The algorithm distributes form factor calculations that can be computed by client computers. In contrast to earlier approaches, the

clients have no knowledge of the radiosity mesh but use the analytic description of the scene objects to compute accurate form factors. This reduces communication costs and improves the quality of the solution. Using a scheduling algorithm that takes care of the special requirements of a network of workstations that is not dedicated to the radiosity computation, a linear speedup for a moderate number of processors could be achieved.

The computation of large radiosity solutions for scenes of hundreds of thousands of polygons requires a clustering approach ontop of the hierarchical radiosity algorithm. Nodes in the hierarchy of clusters represent geometry collections and can be used to model the exchange of energy at a very coarse level. The structure of the cluster hierarchy has a great impact on the quality and performance of the radiosity solution. The clusters must approximate the enclosed geometry very well to accurately simulate the energy flow. To reduce the memory consumption of the algorithm, the cluster hierarchy should efficiently support visibility computations based on ray casting. This avoids the need for an additional data structure whose size depends on the size of the input scene.

We have presented a new clustering strategy that is based on a construction algorithm that was developed to speed up ray tracing. It uses an optimized hierarchy of bounding volumes. By minimizing a cost function that takes the ray intersection costs of the scene objects into account, objects are inserted at an optimal place in the hierarchy. Local grids in nodes of homogeneous object density are used to improve the ray-tracing performance when using this structure. The algorithm has been proven to create a hierarchy well suited for radiosity clustering. The resulting bounding volumes are very tight and the structure explicitly accelerates ray casting. Due to the deep binary tree resulting from the construction algorithm, the traversal costs can easily dominate the performance of the radiosity algorithm. Several techniques to minimize the number of full tree traversal have been proposed. Especially the use of error bounding techniques that require a more sophisticated gathering procedure can lead to additional traversal steps. We have shown that radiosity simulations for large industrial and architectural scenes can be computed at low costs while still using error bounds to achieve solutions of high quality.

6.2 Future Work

The distributed hierarchical radiosity algorithm presented in Chapter 4 is currently limited to the surface based approach. An extension to clustering would be an interesting research topic. If the same setup is chosen, i.e., a client-server architecture, the clients could be activated as soon as the refinement procedure reaches the surface level. Visisbility computations between clusters however, would probably not benefit from distributed computing. Especially when using approximative visibility based on extinction coefficients, communication costs would be too high.

The clustering strategy developed in Chapter 5 opens several perspectives for future research. The reliable detection of almost arbitrary scene details, even at moderate hierarchy depths could be further exploited. Rapid prototyping applications using the tight bounding boxes as an approximation of a scene could improve the efficiency of various algorithms. A radiosity computation could be performed on only a fraction of the number of the original scene elements. Projecting the results obtained by this method onto the real scene polygons (i.e. using texture mapping hardware) would deliver high-speed approximative solutions, applicable to

dynamic environments.

The problem of overlapping bounding volumes should also be addressed in future research. Although the data structure is well suited to speed up radiosity and ray-tracing tasks, the quality for scenes containing many small polygons suffers from the same geometry contributing to different clusters. A simple approach could be to detect an overlap during the creation phase and to mark the corresponding clusters. The radiosity algorithm then would not subdivide a cluster if the resulting clusters will have an overlap. Instead, the enclosed surfaces would be used as the next hierarchy level. The ray acceleration, however, would still use the complete bounding volume hierarchy.

Bibliography

[AH93] L. Aupperle and Pat Hanrahan. A Hierarchical Illumination Algorithm for Surfaces with Glossy Reflection. In *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)*, pages 155–162, 1993. 45

[Bau72] Bruce G. Baumgart. Winged Edge Polyhedron Representation. Artificial Intelligence Project Memo AIM-179 (CS-TR-74-320), Computer Science Department, Stanford University, Palo Alto, CA, October 1972. 53

[Ben97] Heinzgerd Bendels. Eine topologische Datenstruktur und ihre Anwendungen im 3D-Graphiksystem MRT. M.Sc. thesis, Institut für Informatik III, Friedrich-Wilhelms Universität Bonn, Bonn, Germany, 1997. In German. 53

[BFS96] Heinzgerd Bendels, Dieter W. Fellner, and Stephan Schaefer. Hierarchical radiosity on topological data structures. In B. Girod, H. Niemann, and H.-P. Seidel, editors, *3D Image Analysis and Synthesis '96*, pages 111–118, 1996. 55

[BG95] Christian-A. Bohn and Robert Garmann. A Parallel Approach to Hierarchical Radiosity. In V. Skala, editor, *Proceedings of the Winter School of Computer Graphics and CAD Systems '95*, pages 26–35, Plzen, Czech Republic, February 1995. University of West Bohemia. 78

[BGB97] Rui Bastos, Michael Goslin, and Norman I. Badler. Efficient rendering of radiosity using texture and bicubic interpolation. In *1997 Symposium on Interactive 3D Graphics*, pages 71–74. ACM SIGGRAPH, April 1997. 27

[BMSW91] Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James M. Winget. Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions. In *Computer Graphics (ACM SIGGRAPH '91 Proceedings)*, volume 25, pages 51–60, July 1991. 26

[BP93] H. Bao and Q. Peng. A Progressive Radiosity Algorithm for Environments with Curved Surfaces. In Z. Tang, editor, *New Advances in Computer Aided Design and Computer Graphics (Third International Conference on CAD and Computer Graphics)*, pages 114–120, Beijing, China, August 1993. International Academic Publishers. 51, 52

[BP94] H. Bao and Q. Peng. An Efficient Form-Factor Evaluation Algorithm for Environments with Curved Surfaces. *Computers & Graphics*, 18(4):481–486, April 1994. 51, 52

[Bül98] Frank Büllesfeld. Final Gathering. internal report, 8 1998. 67

[Bül99] Frank Büllesfeld. Wavelet-basierte globale Beleuchtungsalgorithmen. M.Sc. thesis, Institut für Informatik III, Friedrich-Wilhelms Universität Bonn, Bonn, Germany, 1999. In German. 67, 69, 70, 73, 76

[CCWG88] Michael Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. In *Computer Graphics (ACM SIGGRAPH '88 Proceedings)*, volume 22, pages 75–84, August 1988. 8, 29, 31, 32

[CDP95] F. Cazals, G. Drettakis, and C. Puech. Filtering, clustering and hierarchy construction: a new solution for ray-tracing complex scenes. In *Proceedings of Eurographics '95*, pages 371–382, 1995. 102

[CG85] Michael Cohen and Donald P. Greenberg. The Hemi-Cube: A Radiosity Solution for Complex Environments. In *Computer Graphics (ACM SIGGRAPH '85 Proceedings)*, volume 19, pages 31–40, August 1985. 20, 23

[CGIB86] Michael Cohen, Donald P. Greenberg, Dave S. Immel, and Philip J. Brock. An Efficient Radiosity Approach for Realistic Image Synthesis. *IEEE Computer Graphics and Applications*, 6(3):26–35, March 1986. 33, 34

[Chr95] Per Henrik Christensen. *Hierarchical Techniques for Glossy Global Illumination*. Ph.D. thesis, Technical Report, Seattle, Washington, 1995. 70

[CLSS97] Per H. Christensen, Dani Lischinski, Eric J. Stollnitz, and David H. Salesin. Clustering for glossy global illumination. *ACM Transactions on Graphics*, 16(1):3–33, January 1997. 92

[CSS96] Per Henrik Christensen, Eric J. Stollnitz, and David H. Salesin. Global Illumination of Glossy Environments Using Wavelets and Importance. *ACM Transactions on Graphics*, 15(1):37–71, January 1996. 46

[CW93] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston, MA, 1993. 11, 26

[EM96] Steven Elliott and Phillip Miller. *Inside 3D Studio MAX, Volume I*. New Riders Publishing, Indianapolis, Indiana, 1996. 50

[Fel96] Dieter W. Fellner. MRT – a teaching and research platform for 3D image synthesis. *IEEE CG&A*, 16(3), May 1996. 50

[FSZ98] Dieter Fellner, Stephan Schaefer, and Marco Zens. *Parallel Computing: Fundamentals, Applications and New Directions*, volume 12 of *Advances in Parallel Computing*, chapter Photorealistic Rendering in Heterogeneous Networks. Elsevier Science, 1998. Proceedings of Parallel Computing '97 (September 1997, Bonn, Germany). 77

[Fun96] Thomas A. Funkhouser. Coarse-Grained Parallelism for Hierarchical Radiosity Using Group Iterative Methods. In *Computer Graphics Proceedings, Annual Conference Series, 1996 (ACM SIGGRAPH '96 Proceedings)*, pages 343–352, 1996. 78

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1990. 1

[GH96] Simon Gibson and R. J. Hubbold. Efficient hierarchical refinement and clustering for radiosity in complex environments. *Computer Graphics Forum*, 15(5):297–310, December 1996. 9, 31, 37, 45, 92, 96, 103, 105

[GH97] Simon Gibson and R. J. Hubbold. Perceptually driven radiosity. *Computer Graphics Forum*, 16(2):119–128, June 1997. 14

[Gib95] Simon Gibson. Efficient Radiosity for Complex Environments. M.Sc. thesis, Manchester, UK, 1995. 40, 42, 44

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979. 84

[Gla84] Andrew S. Glassner. Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, Oct. 1984. 91

[Gla89] Andrew S. Glassner. *Introduction to Ray Tracing*. Academic Press, New York, NY, 1989. 6, 50, 91

[Gla95] Andrew S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann, San Francisco, CA, 1995. 11

[GMP96] Robert Garmann, Robert Mencl, and Georg Pietrek. Mocart++: An object oriented system for monte carlo image synthesis. Technical Report RR 618, Universitat Dortmund, Dortmund, Germany, June 1996. 50

[Gou71] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623–629, June 1971. 25

[Grö95] Alwin Gröne. *Entwurf eines objektorientierten Visualisierungssystems auf der Basis von Raytracing*. Ph. D. thesis, Fakultät für Informatik der Eberhard-Karls-Universität Tübingen, 1995. In German. 98, 99

[GS87] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987. 44, 92, 93, 94, 98

[GSCH93] Steven J. Gortler, Peter Schroder, Michael F. Cohen, and Pat Hanrahan. Wavelet Radiosity. In *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)*, pages 221–230, 1993. 46

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the Interaction of Light Between Diffuse Surfaces. In *Computer Graphics (ACM SIGGRAPH '84 Proceedings)*, volume 18, pages 212–222, July 1984. 7, 8

[HDSD99] J. M. Hasenfratz, C. Damez, F. Sillion, and G. Drettakis. A practical analysis of clustering strategies for hierarchical radiosity. In *Computer Graphics Forum (Proc. Eurographics '99)*, volume 18, pages C–221–C–232, September 1999. 91, 92, 93

[How82] John R. Howell. *A Catalog of Radiation Configuration Factors*. McGraw Hill, New York, NY, 1982. 20

[HS90] Pat Hanrahan and David Salzman. A Rapid Hierarchical Radiosity Algorithm for Unoccluded Environments. Technical Report CS-TR-281-90, Department of Computer Science, Princeton University, Princeton, NJ, August 1990. 34

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. A Rapid Hierarchical Radiosity Algorithm. In *Computer Graphics (ACM SIGGRAPH '91 Proceedings)*, volume 25, pages 197–206, July 1991. 9, 34, 37, 40, 95, 105

[JB76] M.E. Newell J.F. Blinn. Texture and reflection in computer generated images. *Communications of the ACM*, 1976. 29

[JCC+93] G.R. Jones, C. G. Christou, B. G. Cumming, A. J. Parker, and A. Zisserman. Accurate Rendering of Curved Shadows and Interreflections. In *Fourth Eurographics Workshop on Rendering*, number Series EG 93 RW, pages 337–345, Paris, France, June 1993. 51, 52

[Kaj86] James T. Kajiya. The Rendering Equation. In *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150, August 1986. 16

[KK86] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, 20(4):269–278, 1986. 98

[KS97] K. Klimansezewski and T. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17(1):42–51, 1997. 102

[KSS97] L. Kobbelt, M. Stamminger, and H.-P. Seidel. Using subdivision on hierarchical data to reconstruct radiosity distribution. *Computer Graphics Forum (Eurographics '97 Proceedings)*, 16(3):C347–C355, 1997. Available from http://www9.informatik.uni-erlangen.de/eng/research/pub1997. 27

[Kuc88] Horst Kuchling. *Taschenbuch der Physik*. Verlag Harri Deutsch, Thun und Frankfurt/Main, 1988. 11

[Loo87] C. Loop. Smooth subdivision surfaces based on triangles. Technical report, University of Utah, 1987. 28

[LSG94] Dani Lischinski, Brian Smits, and Donald P. Greenberg. Bounds and Error Estimates for Radiosity. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)*, pages 67–74, 1994. 45, 94, 95, 97

[Män88] Martti Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, 1988. 53

[MB98] Daniel Meneveaux and Kadi Bouatouch. Parallel hierarchical radiosity for complex building interiors. Technical Report 1193, Institut National de Recherche en Informatique (IRISA), Rennes, France, May 1998. 78

[MF99] Gordon Müller and Dieter W. Fellner. Hybrid scene structuring with application to ray tracing. In *Proceedings of the International Conference on Visual Computing (ICVC'99)*, pages 19–26, Goa, India, February 1999. available online via http://www.cg.cs.tu-bs.de/people/mueller/publications. 98, 99, 102, 106

[MSF99a] Gordon Mueller, Stephan Schaefer, 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)*, Los Alamitos, CA, October 1999. IEEE Computer Society Press. 98

[MSF99b] Gordon Mueller, Stephan Schaefer, and Dieter Fellner. A rapid clustering algorithm for efficient rendering. In *Short Papers and Demos (Eurographics '99)*, September 1999. 98

[MT93] Nelson Max and Roy Troutman. Optimal Hemicube Sampling. In *Fourth Eurographics Workshop on Rendering*, number Series EG 93 RW, pages 185–200, 349–352, Paris, France, June 1993. 21

[Mys98] Karol Myszkowski. The visible differences predictor: Applications to global illumination problems. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop '98)*, pages 233–236, New York, NY, 1998. Springer Wien. 14

[NN85] Tomoyuki Nishita and Eihachiro Nakamae. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. In *Computer Graphics (ACM SIGGRAPH '85 Proceedings)*, volume 19, pages 23–30, July 1985. 7

[NN93] Tomoyuki Nishita and Eihachiro Nakamae. A New Radiosity Approach Using Area Sampling for Parametric Patches. In *Computer Graphics Forum (Eurographics '93)*, volume 12, pages C385–C398, Barcelona, Spain, September 1993. 51, 52

[Nus28] W. Nusselt. Grapische Bestimmung des Winkelverhaltnisses bei der Warmestrahlung. *Zeitschrift des Vereines Deutscher Ingenieure*, 72(20):673, 1928. 20

[Pho75] B.-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975. 3

[PRR98] A. Pohdehl, T. Rauber, and G. Runger. A shared-memory implementation of the hierarchical radiosity method. *Theoretical Computer Science*, 196(1-2):215–240, 1998. 79

[Rei92] Mark C. Reichert. A Two-Pass Radiosity Method Driven by Lights and Viewer Position. M.Sc. thesis, Ithaca, NY, January 1992. 28

[RT87] Holly E. Rushmeier and Kenneth E. Torrance. The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium. In *Computer Graphics (ACM SIGGRAPH '87 Proceedings)*, volume 21, pages 293–302, July 1987. 42

[RW80] Steven M. Rubin and Turner Whitted. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. *Computer Graphics*, 14(3):110–116, July 1980. 91

[SAG94] Brian Smits, James Arvo, and Donald Greenberg. A Clustering Algorithm for Radiosity in Complex Environments. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)*, pages 435–442, 1994. 9, 45, 46, 92, 95, 97, 104, 105

[Sch97] Stephan Schaefer. Hierarchical radiosity on curved surfaces. In Julie Dorsey and Phillip Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 187–192, New York, NY, 1997. Springer Wien. ISBN 3-211-83001-4. 51

[SDS95] Francois Sillion, George Drettakis, and Cyril Soler. A Clustering Algorithm for Radiance Calculation in General Environments. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 196–205, New York, NY, 1995. Springer-Verlag. 45

[SGCH93] Peter Schroder, Steven J. Gortler, Michael F. Cohen, and Pat Hanrahan. Wavelet Projections for Radiosity. In *Fourth Eurographics Workshop on Rendering*, number Series EG 93 RW, pages 105–114, Paris, France, June 1993. 46

[SGL94] Jaswinder P. Singh, Annop Gupta, and Marc Levoy. Parallel Visualization Algorithms: Performance and Architectural Implications. *IEEE Computer*, 27(7):45–55, July 1994. 78

[SH92] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfer, 3rd Edition*. Hemisphere Publishing Corporation, New York, NY, 1992. 7

[SH94] Peter Schroder and Pat Hanrahan. Wavelet Methods for Radiance Computations. In *Fifth Eurographics Workshop on Rendering*, pages 303–311, Darmstadt, Germany, June 1994. 46

[SHT+92] Jaswinder Pal Singh, Chris Holt, Takashi Totsuka, Anoop Gupta, and John L. Hennessy. Load Balancing and Data Locality in Adaptive Hierarchical N-body Methods: Barnes-Hut, Fast Multiple, and Radiosity. Technical Report CSL-TR-92-505, Computer Systems Laboratory, Stanford University, Stanford, CA, 1992. 78

[Sil93] Silicon Graphics Inc. *The OpenGL Programming Guide – The Official Guide to Learning OpenGL*. Addison-Wesley, Reading, Mass., 1 edition, 1993. 52

[Sil95] Francois Sillion. A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3), September 1995. 9, 41, 43, 44, 91, 96, 97, 98, 103, 105

[SP89] Francois Sillion and Claude Puech. A General Two-Pass Method Integrating Specular and Diffuse Reflection. In *Computer Graphics (ACM SIGGRAPH '89 Proceedings)*, volume 23, pages 335–344, July 1989. 21

[SP94] Francois Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann, San Francisco, CA, 1994. 11, 105

[SS95] Philipp Slusallek and Hans-Peter Seidel. Vision - An Architecture for Global Illumination Calculations. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):77–96, March 1995. Available from http://www9.informatik.uni-erlangen.de/eng/research/pub95. 50

[SSS96] M. Stamminger, P. Slusallek, and H.-P. Seidel. Bounded radiosity - illumination on general surfaces and clusters. Technical Report Technical Report 15/1996, Computer Science Department, University of Erlangen-Nuremberg, 1996. Available from http://www9.informatik.uni-erlangen.de/eng/research/tr96. 37

[SSS97a] M. Stamminger, P. Slusallek, and H.-P. Seidel. Bounded radiosity - illumination on general surfaces and clusters. *Computer Graphics Forum (Eurographics '97 Proceedings)*, 16(3):C309–C317, 1997. Available from http://www9.informatik.uni-erlangen.de/eng/research/pub1997. 51, 52

[SSS97b] Marc Stamminger, Philipp Slusallek, and Hans-Peter Seidel. Bounded clustering - finding good bounds on clustered light transport. Technical Report Technical Report 2/1997, Computer Science Department, University of Erlangen-Nuremberg, 1997. Available from http://www9.informatik.uni-erlangen.de/eng/research/tr1997. 45

[SSS98] Marc Stamminger, Phillip Slusallek, and Hans-Peter Seidel. Bounded clustering: Finding good bounds on clustered light transport. In *Pacific Graphics '98*, Singapore, October 1998. 97

[SSSS98] M. Stamminger, H. Schirmacher, P. Slusallek, and H.-P. Seidel. Getting rid of links in hierarchical radiosity. *Computer Graphics Journal (Proc. Eurographics '98)*, 17(3):C165–C174, September 1998. 105

[Str91] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, 2 edition, 1991. 50

[Thi99] Joern Thiemann. MRT4MAX: PlugIn für Discreet 3D Studio MAX. Technical report, Computer Graphics, Braunschweig University of Technology, 1999. 69, 75

[TM93] Roy Troutman and Nelson L. Max. Radiosity Algorithms Using Higher Order Finite Element Methods. In *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)*, pages 209–212, 1993. 51

[TR93] Jack Tumblin and Holly E. Rushmeier. Tone Reproduction for Realistic Images. *IEEE Computer Graphics and Applications*, 13(6):42–48, November 1993. 14

[UT97] Carlos Urena and Juan C. Torres. Improved irradiance computation by importance sampling. In Julie Dorsey and Phillip Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 275–284, New York, NY, 1997. Springer Wien. ISBN 3-211-83001-4. 68

[Wat89] Alan Watt. *Fundamentals of Three-Dimensional Computer Graphics*. Addison-Wesley, Wokingham, UK, 1989. 2

[Wer94] Josie Wernecke. *The Inventor Mentor*. Addison Wesley, 1994. 66

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980. 4

[Zar95] David Zareski. Parallel Decomposition of View-Independent Global Illumination Algorithms. M.Sc. thesis, Ithaca, NY, 1995. 78

[Zat93] Harold R. Zatz. Galerkin Radiosity: A Higher Order Solution Method for Global Illumination. In *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)*, pages 213–220, 1993. 51, 52

[Zen97] Marco Zens. Paralleles Rendering. M.Sc. thesis, Institut für Informatik III, Friedrich-Wilhelms Universität Bonn, Bonn, Germany, 1997. In German. 85, 86

[Zie89] O. C. Zienkiewicz. *The Finite Element Method*. McGraw-Hill, London, 4th edition, 1989. 17