The Irradiance Volume

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Gene S. Greger

August 1996

Abstract

This thesis presents a volumetric representation for the global illumination within a space based on the radiometric quantity irradiance. We call this representation the irradiance volume. Although irradiance is traditionally computed only for surfaces, its definition can be naturally extended to all points and directions in space. The irradiance volume supports the reconstruction of believable approximations to the illumination in situations that overwhelm traditional global illumination algorithms. A theoretical basis for the irradiance volume is discussed and the methods and issues involved with building the volume are described. The irradiance volume method is tested within several situations in which the use of traditional global illumination methods is impractical, and is shown to provide good performance.

Biographical Sketch

The author was born in Sheridan, Wyoming on January 29, 1968. Since that time, he has lived in Florida, Wyoming again, Arizona, and finally New York state. In 1986, Gene started his Freshman year at Rensselaer Polytechnic Institute in Troy, NY. While at RPI, he met his future wife Helen Weltin.

After graduating with a Computer Science degree in 1991, Gene spent two years in Troy doing various stuff, including getting married to Helen in 1993. He entered the Cornell Program of Computer Graphics in the fall of that year.

This work is dedicated to my wife, family, and friends whose love and support have made this possible.

Acknowledgements

First and foremost, I would like to thank my wife and family for their love and support.

I am very grateful to Professor Donald Greenberg for providing me with the opportunity to study at the Cornell Program of Computer Graphics. He has created a truly amazing environment in which to learn and contribute to the field of computer graphics.

Professor Michael Tomlan deserves special thanks for being a great advisor and instructor, who cares deeply about his field and his students.

This thesis (and my graduation) was made possible by the incredible amount of help and advice provided by Peter Shirley. I am also indebted to Philip Hubbard, whose encouragement and guidance have made this a much better work. I consider myself very fortunate to know them both.

I would like to acknowledge my officemates Dani Lischinski, David Zareski, Andrew Kunz, Mark Piretti, and Richard Coutts and my classmates Bretton Wade and Patrick Heynen for their companionship, humor, and patience.

Dan Kartch and Sebastian Fernandez not only provided invaluable technical help, but also proved worthy opponents in inter-office computer battles. Thanks

also to Jonathan Joseph, Jon Blocksom, and Ben Trumbore for being great disc golf partners.

Jim Ferwerda, Hurf Sheldon, Redstar Cleveland, Jonathan Corson-Rikert, Ellen French, Linda Stephenson, Ben Trumbore, and a host of others keep the lab going on a daily basis. Their support and expertise are much appreciated.

It is hard to do justice in a few short paragraphs to the help and friendship one has been given throughout three years; thanks to all who have done so.

This work was made possible by the generous support given to the Program of Computer Graphics by the Hewlett Packard Corporation. I would also like to thank the Evans and Sutherland Corporation for their generous donation of equipment.

This work was supported by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (grant number ASC-8920219). The rabbit models are courtesy of Stanford University and the University of Washington.

Table of Contents

List of Figures

Chapter 1

Introduction

One of the major goals in the field of computer graphics is realistic image synthesis. To this end, illumination methods have evolved from simple local shading models to physically-based *global illumination* algorithms. Local illumination methods consider only the light energy transfer between an emitter and a surface (direct lighting) while global methods account for light energy interactions between *all* surfaces in an environment, considering both direct and indirect lighting.

The visual effects of global illumination are important in most real scenes. For example, consider the office shown in Figure 1.1, where most of the visible illumination is indirectly reflected from the ceiling. Simulating this scene with only local, direct lighting calculations would yield erroneous results.

Even though the realistic effects that global illumination algorithms provide are frequently desirable, the computational expense of these methods is often too great for many applications. Consider a dynamic environment in which objects can be

Figure 1.1: *An office lit indirectly by a halogen lamp.*

added or change position. Traditionally, when some aspect of a globally-illuminated environment changes, a new illumination solution is computed from scratch because the current solution is now incorrect. There has been a considerable amount of work done to extend global illumination methods to dynamic environments; often, however, the gain in rendering speed from these methods is not large enough, or especially in the case of radiosity methods, limited in the kind or complexity of the environmental changes.

Environments with complex geometry also pose problems for many global illumination methods. When a large number of surfaces are necessary to accurately represent complex environments many global illumination algorithms are overwhelmed. Methods have been developed to reduce the computations required to render intricate environments but they are not fast enough in most situations.

As a less expensive alternative, many current systems compute only the direct lighting of a surface and use a constant *ambient lighting term* to roughly approximate the indirect component. Figure 1.2 compares this approach to direct-only illumination and full global illumination. Note that some geometric features disappear on the box illuminated with the ambient term.

A different approach to calculating the global illumination of objects is presented in this thesis. Instead of striving for accuracy at the expense of performance, the goal is rephrased to be a *reasonable approximation with high performance*. This places global illumination effects in the reach of many applications in which visual appearance is more important than absolute numerical accuracy.

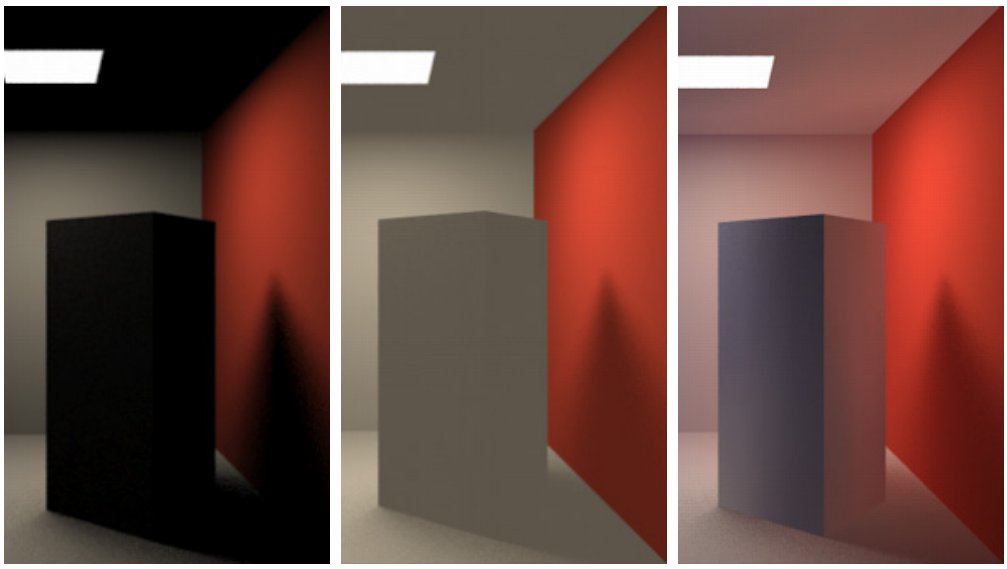


Figure 1.2: *The image on the left was rendered with direct lighting only. The center image was rendered with direct lighting and an ambient light term. The image on the right was computed using both direct and indirect lighting.*

Consider a set of surfaces enclosing a space (e.g., the walls of a room). The visual appearance of a diffuse surface within the space can be computed from the reflectance of the surface and the *irradiance* at the surface. Calculating the irradiance analytically requires a cosine-weighted integral of radiance for all incoming directions. For complex environments, this is a resource intensive computation that overwhelms most applications.

We extend the concept of irradiance from surfaces into the enclosure. This extension is called the *irradiance volume*. The irradiance volume represents a volumetric approximation of the irradiance function. The volume is built in an environment as a pre-process, and can then be used by an application to quickly approximate the irradiance at locations within the environment. This thesis represents an exploratory study on the feasibility and effectiveness of sampling and storing irradiance in a spatial volume.

By considering light transport through space instead of between surfaces, the irradiance volume method yields high performance in several important situations for which traditional global-illumination algorithms are too slow. These include "semi-dynamic" environments and environments with a very large number of small geometric features. Another relevant application includes light flow visualization systems for lighting designers and computer graphics researchers, in which interactive rates are essential. Due to the ease with which the irradiance volume can be represented in a data structure, querying the irradiance takes nearly constant time independent of the complexity of the environment the irradiance volume is built in.

Chapter 2 of the thesis discusses previous work in image synthesis and extensions

to these methods for dynamic environments. Rendering methods which use a volume storage methods are also covered. Chapter 3 examines radiance and irradiance, and provides a definition of an irradiance volume. Chapter 4 discusses how sources of radiance data such as rendered environments can yield irradiance volumes, and outlines some of the properties of irradiance volumes. The details of implementing and querying the volume are presented and an extension to the irradiance volume method for non-diffuse surfaces is described. Chapter 5 explores several applications which utilize the irradiance volume. Chapter 6 summarizes the material presented in this thesis and discusses several open issues related to this work.

Chapter 2

Previous Work

This chapter summarizes prior work that relates to rendering realistic images. Section 2.1 describes rendering methods for dynamic environments. Section 2.2 describes algorithms which utilize radiometric volume storage.

2.1 Rendering Dynamic Environments

Rendering algorithms were first developed for static environments [65, 39, 91, 38]. To render a scene whose properties did not remain constant typically required executing the entire rendering algorithm each time the scene changed. Because rendering procedures are generally computationally expensive, new methods and extensions to existing methods were developed to reduce the cost of rendering non-static environments. This section covers common rendering methods and attempts to extend these methods to *dynamic environments*. We define a dynamic environment as one in which objects are added or deleted, or change position, rotation, or surface

properties through time. Changes can either be known in advance, such as the predefined motion from an animation script, or unknown in advance, such as changes resulting from simulations or interactive applications. An environment in which a user can modify and receive feedback in a responsive manner is designated an *interactive* environment.

2.1.1 Local Illumination Methods

Local illumination methods consider only the interaction between a light source and a surface, ignoring the rest of the environment. They were the earliest lighting methods developed in the computer graphics field.

One of the simplest shading models is the *diffuse illumination model* which describes the shading across a completely diffuse surface lit by a point light source [30]. Phong [65] developed a model which handled specular surface properties. Blinn [14] applied a physically based reflection model of rough surfaces based upon work by Torrance and Sparrow [82, 83]. The He model [45] extends the Cook-Torrance [26] model to conserve energy reflected from rough surfaces.

Simple local illumination methods such as the Phong method are useful for interactive and dynamic environments because of the relatively low computational costs. Due to the locality of the lighting algorithm, additional objects can be added with only a linear increase in rendering time.

Many current hardware graphics systems quickly render polygonal environments by using a local illumination technique to compute the color of the vertices. The polygons are shaded using an interpolative method such as the Gouraud shading

model [39]. Display rates of several million triangles per second can be obtained using this procedure [3].

The use of shadows adds an extra degree of realism to a rendered scene [85]. Crow [29] and Atherton *et al.* [9] developed some of the first shadowing methods. Reeves *et al.* [68] and Brotman *et al.* [16] described software-based depth-buffer shadow methods. Segal *et al.* [73] provided a real-time method of casting shadows using texture mapping hardware.

2.1.2 Ray Tracing Methods

Global illumination algorithms consider both direct and indirect light distribution in an environment when calculating the color of a pixel or point on a surface. The most common global illumination methods are *ray tracing* and *radiosity*.

Ray tracing methods compute view-dependent global illumination solutions. Ray tracing was developed by Whitted [91] and Kay [51] as an extension to the ray casting technique formulated by Appel [4]. Most of the work on ray tracing has been attempts to reduce the rendering time and to expand the range of phenomena ray tracing can render, such as volume densities [50], gem stones [94], polarization effects [92], and motion blur [27]. Figure 2.1 shows an image of an environment rendered using ray tracing.

Efforts to increase the rendering speed of ray tracing have concentrated heavily on reducing the cost of computing ray-object intersections. Methods include spatial subdivision [35], bounding volumes [90], and hierarchical intersection tests [52]. Woo [93] discussed procedures used in a modern rendering package.

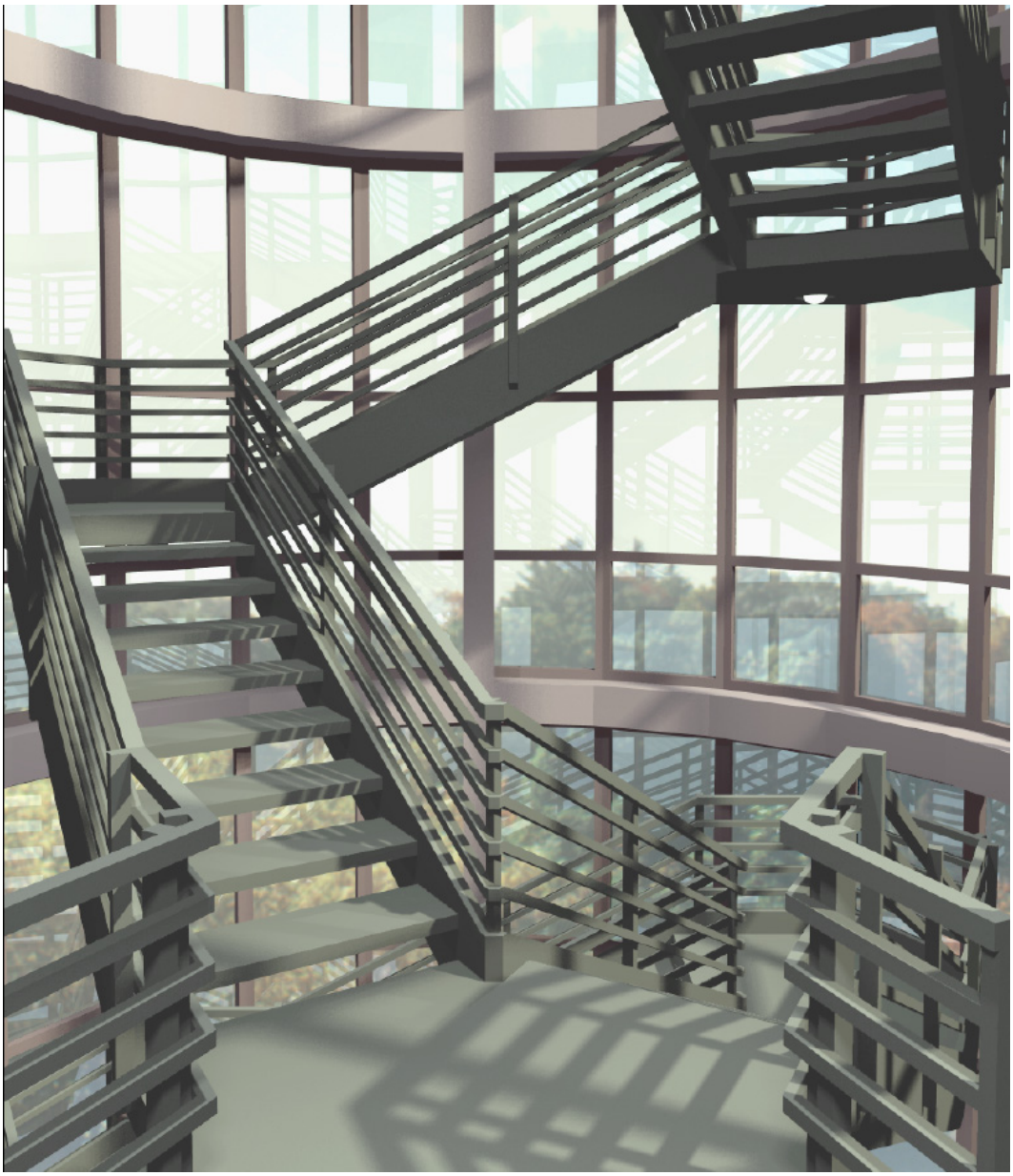


Figure 2.1: *A ray traced image modeled by Keith Howie and rendered by Ben Trumbore. Cornell Program of Computer Graphics, 1989.*

The general method of ray tracing a dynamic environment is to render a sequence of frames, each requiring a full and separate ray tracing solution. Because of the heavy computational expense, methods have been developed which take advantage of advance information about the motion of objects or information gleaned from previously rendered frames. Chapman *et al.* [19] notes that animations typically have a large amount of coherence between frames because human viewers would not be able to make sense of the sequence otherwise.

Some ray tracing algorithms which exploit temporal coherence can be classified as *screen space* methods. Screen space methods make rendering decisions based solely on pixel values in a series of frames. Rendering time is reduced for animation sequences by only computing a subset of the pixels in a frame and estimating the rest. Badt [11] presented a method which made random comparisons of corresponding pixels between subsequent frames. If the pixels differed, a "three-dimensional flood fill" across the current and surrounding frames defined a region of pixels to be re-rendered.

Chapman *et al.* [18] developed an algorithm which fully renders every nth frame of an image. Pairs of rendered frames are compared and pixels which have the same value in both frames are assumed to have that value at that location throughout the frames in between. If the value of a pixel differs, it is ray-traced on the frame lying midway between the two comparison frames. This process is recursively repeated, performing a "binary search in the temporal domain" until the animation is complete.

The screen space methods described have two major disadvantages. High fre-

quency changes in the scene will be missed, and each frame of the animation must be available during the entire rendering process.

Some methods use object coherence and exploit how geometric objects within an environment change between frames. Several methods have been developed for animations with a static viewpoint. Séquin *et al.* [74] created a method that allowed surface attributes of objects in a static scene to be changed without having to re-render the entire scene. By storing the ray tree associated with each pixel, the pixels that would be affected by changing a surface could be identified. Murakami *et al.* [60] presented an extension to this method that works for animated objects. In addition to storing the ray tree for each pixel, note is made of each spatial-subdivision voxel the rays in the tree intersect. By recalculating the voxel grid after objects move, the rays that need to be re-traced can be determined. Jevans [49] discussed a method of keeping a record of the pixel associated with each ray in the environment. Each voxel in the scene keeps track of the pixel indices of the rays that intersect the voxel. When the content of a voxel changes, the pixels associated with that voxel are re-traced.

Some methods utilize *4D ray tracing* which tests ray-object intersections in the temporal domain as well as in the three Cartesian dimensions. Glassner [36] developed a method to reduce animation rendering times by using 4D bounding boxes which enclosed an object and the volume that object occupied through time. Chapman *et al.* [19] presented a method which produced bounding volumes that represented objects and vector descriptions of their movement.

Bergman *et al.* [13] introduced the concept of *progressive refinement* for image

rendering. A progressive refinement algorithms attempts to make the most comprehensible image it can initially and then continues to refine until the final solution in computed. Painter *et al.* [63] presented a progressive refinement approach for ray-tracing.

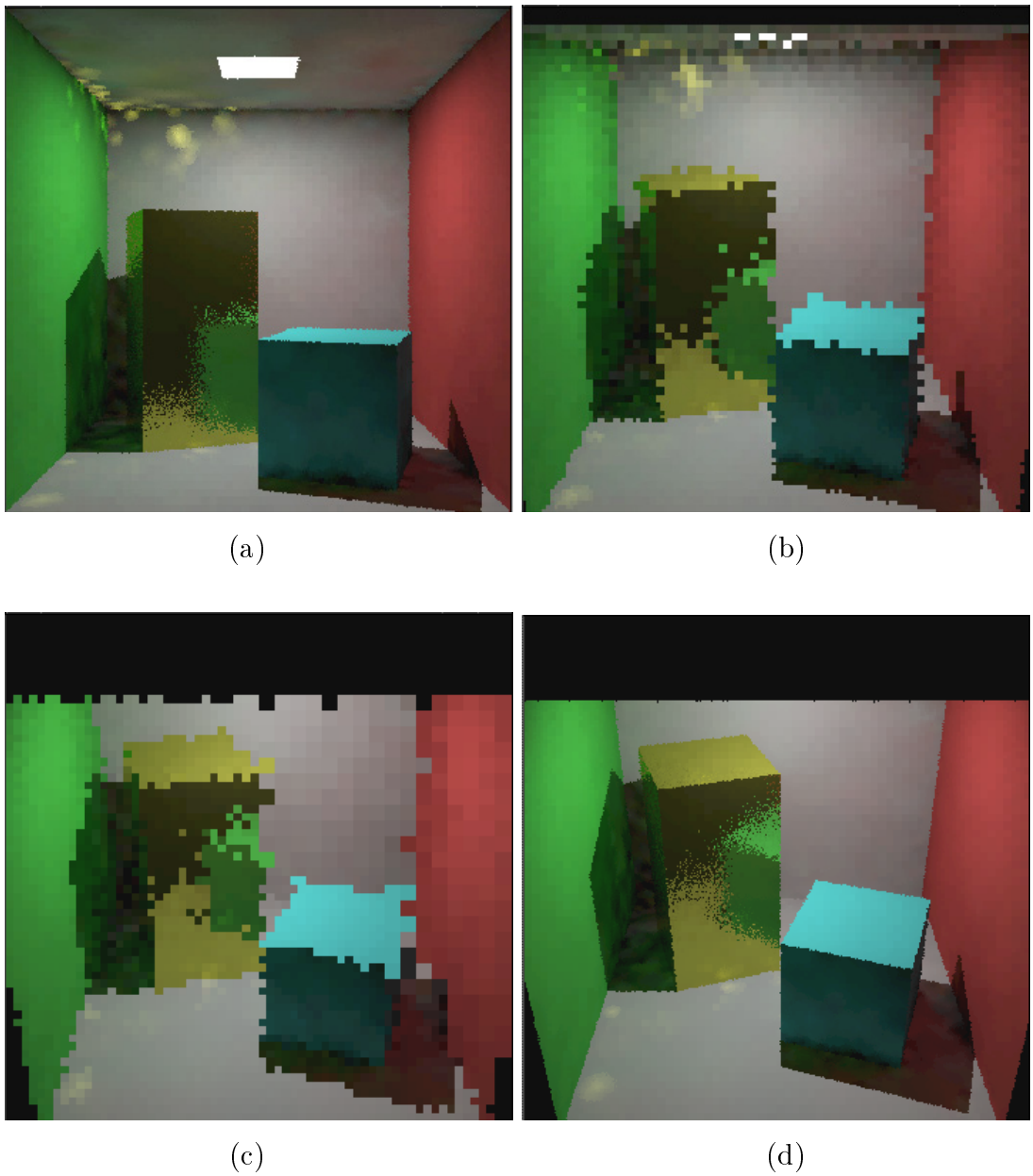
Even though ray tracing methods continue to increase in speed, at the present time relatively simple environments still cannot be rendered at interactive rates. Interactive applications which use computationally expensive rendering methods will often use progressive refinement approaches to render dynamic scenes. When motion occurs, each frame is allotted a certain amount of time to make the most thorough image it can within that time span. During a break in the motion, the algorithm continues to refine until it is done or movement begins again.

The *RADIANCE* system developed by Ward [86] uses a hybrid ray tracing algorithm to render scenes. Several stages of refinement are performed to calculate the final image. By using an interactive front end to the system [87] a user can interactively change the viewpoint of a scene while an image is being refined. Figure 2.2 shows an example. A similar front-end also exists for the popular *Rayshade* package [1].

2.1.3 Radiosity methods

The radiosity global illumination method solves for energy transfer between surfaces in environment. Radiosity methods originally developed in the heat transfer

[1]Rayshade is a public domain ray tracing package developed by Craig Kolb at Princeton University. See http://www-graphics.stanford.edu/∼cek/rayshade.



(a) (b)

(c) (d)

Figure 2.2: *User rotating between two views (panels* a *and* d*) using an interactive front end of a progressive refinement rendering package (RADIANCE).*

field [78] and were first applied to computer graphics by Goral *et al.* [38] in 1984.

Cohen [25] introduced a method for producing radiosity solutions of environments with occluding surfaces. Nishita [62] developed a shading method to reduce surface-to-surface illumination computation cost. An adaptive subdivision technique to capture better shadows was presented by Cohen *et al.* [23]. Rushmeier *et al.* [72] extended the radiosity method to include participating media such as fog and smoke. Immel *et al.* [47] produced a hybrid radiosity method to render non-diffuse surfaces.

The basic radiosity method is an $O(n^2)$ algorithm. The entire solution must be computed before it can be viewed. In 1988, Cohen *et al.* [24] developed a progressive refinement technique for radiosity. This method quickly produces an image based on an approximate solution and continues to refine towards convergence. The incremental changes to the solution can be continuously displayed as the algorithm progresses. Wallace [84] and Sillion *at al.* [79] presented methods for extending progressive radiosity to render non-diffuse surfaces.

Hierarchical radiosity was developed by Hanrahan *at al.* [43, 44] in 1991. The hierarchical method reduces the order of complexity of radiosity by reducing the number of energy transfer calculations. Extensions to hierarchical radiosity include the use of clustering to increase efficiency [80] and the ability to render non-diffuse surfaces [10].

After the energy exchange has been computed, the solution must be represented in a form suitable for display. This is usually accomplished with a polygonal mesh with radiance values at mesh vertices. Gouraud shading is used to linearly interpo-

late between vertices to produce smoothly shaded surfaces.

In many environments, discontinuities in the radiance function occur across surfaces so that smooth Gouraud shading is not appropriate. Lischinski *et al.* [55] developed a method to precompute discontinuities and produce a more accurate mesh.

Because radiosity solutions are view-independent, radiosity methods are well suited for applications such as simulated walkthroughs of static environments. Once a solution has been computed, it can be navigated as quickly as the polygonal mesh of the solution can be displayed. Puech *et al.* [67] developed an interactive radiosity-based lighting simulator. Airey *et al.* [2] described an interactive building walk-through system. Figure 2.3 shows images from a walkthrough program.

For complex scenes, display systems may be unable to quickly display the large number of polygons in the radiosity solution. Several methods that determine the subset of the environment that is potentially visible from a given viewpoint have been developed [41, 81, 2]. Display speed is increased by not displaying invisible geometry.

If the set of visible surfaces is still too large to display quickly, the geometric complexity of the visible objects can be reduced for display. Funkhouser *et al.* used varying *levels of detail* to display objects in complex scenes. Objects are displayed at differing geometric resolutions based on heuristic methods determining their importance in a particular view. Maciel *et al.* [56] presented a method of using clusters and texture maps of clusters of objects to reduce display time.

Although the radiosity method produces view-independent solutions, they be-

Figure 2.3: *Two views of a radiosity solution from an interactive walkthrough program. These images were displayed in real-time from a single solution computed using the method presented in [77]. Modeled by Gene Greger.*

come inaccurate if geometry or surface attributes in the scene change. Fully recomputing the energy exchange when the scene content changes is the obvious solution, but radiosity, like many other global illuminations methods, is computationally expensive.

Several parallel computing methods have been developed to speed computation of the energy distribution. An analysis of parallel radiosity methods can be found in [17]. The increase in speed from parallel methods is often less than expected due to system overhead and the global nature of the radiosity algorithm.

Several authors have extended the radiosity method to render dynamic environments. An early method was developed by Baum *et al.* [12] for sequences with a static viewpoint and pre-planned object motion.

George *et al.* [33], Chen [20], and later Müller *et al.* [59] extended the progressive radiosity method [24] to address dynamic environments. Extensions to hierarchical radiosity were made by Shaw [75] and Forsyth *et al.* [31]. Shaw's technique does not allow light sources to be moved. The approach taken by Forsyth *et al.* does not permit moving objects which occlude static geometry.

Although the above extensions result in a significant decrease in rendering time over the original algorithms, only relatively simple environments can be rendered at or near interactive rates. In addition, environments containing more than one object changing simultaneously often create algorithmic difficulties.

2.1.4 Picture-Based Methods

All the rendering techniques discussed so far in this chapter are geometry-based systems; illumination is computed given a set of mathematically defined geometric regions. A relatively recent body of work, *picture-based rendering systems*, also known as *image-based systems*, use pictures as the basic data type for rendering.

McMillan *et al.* [58] discussed how picture-based methods approximate the *plenoptic function* [1], also called the *radiance distribution function* [66]. The plenoptic function is defined as the view seen in all directions from all points within a space. Picture-based methods sample and attempt to reconstruct a continuous representation of this function. A real environment can be sampled using photographic techniques. Virtual environments are sampled by rendering images.

Picture-based methods can be especially useful for modeling environments which exist in reality but are too complex or time-consuming to geometrically model and render. Also, since the scene does actually exist, the "photo-realism" is inherent.

McMillan presented a method of approximating the plenoptic function by capturing cylindrical panoramas of images at various locations in a scene. Transformations and registration are performed on the images taken at each location to form a seamless cylindrical projection. By interpolating between projections, a view can be displayed for most locations and directions in the scene. With a sufficiently fast hardware system, the scene can be navigated interactively.

Chen *et al.* [22] presented a method for view interpolation. In later work, Chen [21] put forth a method similar to McMillan *et al.* for navigating picture-based environments. His work was developed with the goal of making the method

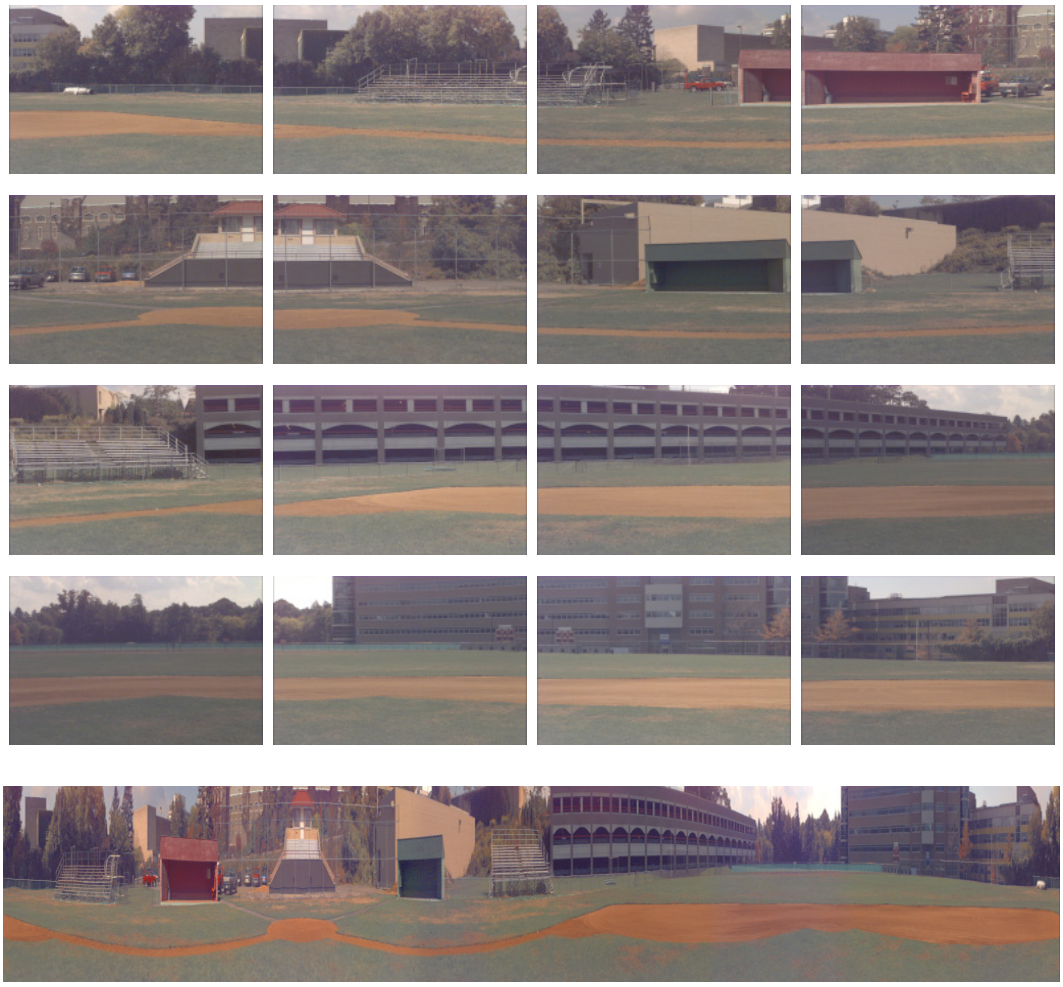


Figure 2.4: *This figure shows a cylindrical panoramic image and some of the pictures it was created from. The aspect ratio of the panoramic image was modified to fit the page. The panoramic view was computed based on the algorithm presented in [58].*

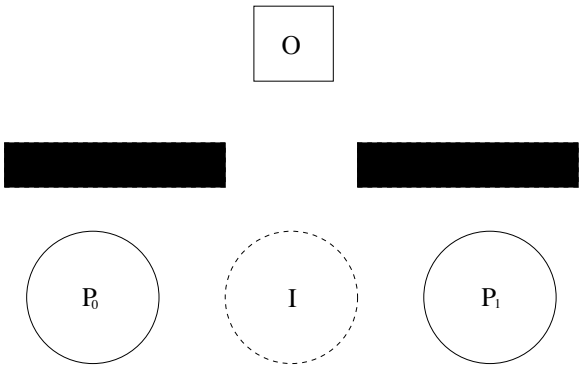


Figure 2.5: *This figure illustrates a problem with picture-based methods. In the environment shown, the viewer is at position I viewing a panorama interpolated from circular plenoptic samples at P_0 and P_1. Since object O cannot be seen from either P_0 or P_1, O will not be visible to the viewer.*

viable for interactive use on home computer systems. Nimeroff *et al.* [61] discussed the use of image interpolation in an animation system.

Because picture-based rendering systems do not utilize geometric descriptions of an environment, the potential exists for these systems to miss features within a scene. Objects which are not visible from any sample point cannot be seen from any possible viewer location. Figure 2.5 shows an example. This problem can be lessened by sampling an environment more finely, but this may be infeasible for large scenes.

Max *et al.* [57] developed a method for decreasing the rendering time of complex polygonal objects such as tree models. Instead of sampling with rendered images, parallel projection z-buffer images are used. By storing multiple z values at each pixel in the z-buffer image, forming a depth image, hidden objects can be accounted for. Images are taken from preset directions surrounding the object and are used

to reconstruct views from any angle.

Picture-based methods are currently only useful for rendering static environments. Chen suggested some methods for the composition of 3D objects into an picture-generated scene but noted that true interaction is difficult. Depth information for each image sample must be obtained, which is impractical for real scenes.

2.1.5 Summary

The ideal dynamic rendering system would provide realistic images of complex environments while not placing any constraints on motion and geometry. Local illumination methods are very fast but suffer from low realism. Ray tracing methods provide a much greater degree of realism but are view-dependent and expensive. Rendering of ray traced sequences at interactive rates for even moderately complex environments rates is still intractable. Extensions for dynamic environments decrease the rendering time but not enough for interactivity.

Radiosity methods are also computationally expensive but have the advantage of being view-independent and highly realistic. Several extensions to progressive and hierarchical radiosity procedures were discussed. Unfortunately, these methods do not perform well for complex environments or are limited in the number of moving objects the method can comfortably handle.

Picture-based methods can provide interactive exploration of very realistic spaces, but only when objects in the scene remain static. Although it is possible to approximate the plenoptic function in time by sampling the environment completely at each time step, it impractical to do so due to storage limitations and the large amount

of samples needed.

Even though many extensions have been made to rendering methods to reduce the computation time for dynamic environments, realistic rendering of non-trivial dynamic scenes at interactive rates is an unmet goal. Ray tracing and radiosity methods have been unable to become effectively interactive for dynamic environments. Picture-based methods are targeted for static scenes.

2.2 Radiometric Volume Methods

A *radiometric volume* is a data structure containing radiometric information corresponding to a three-dimensional region of space. *Radiometric volume methods* are a class of rendering techniques which utilize radiometric volumes.

Picture-based rendering methods fall into this category. Discussed previously, picture-based methods sample the radiance distribution function and attempt to generate a continuous representation of that function [58].

Several rendering methods employ radiometric volume structures to simplify and quicken computation. Patmore [64] utilizes a radiance volume to simplify ray tracing of dense foliage illuminated by an area light source. He notes that objects containing dense clusters of polygons, such as trees, do not render well when illuminated by a point light source. Using the sky or other area light source for illumination becomes prohibitively expensive due to the large amount of ray-object tests needed when sampling the area source from intersection points.

His method employs a regular grid which is placed over the polygonal object to

be rendered. The radiance is sampled in a set of directions at each point on the grid. This procedure is done as a pre-process to the rendering stage. When a ray intersects a polygon during ray-tracing, the illumination at the intersection point is interpolated from the eight locations on the grid surrounding that point. The orientation of the intersected polygon determines which directional values to query at the eight grid points.

Greene [40] implemented a similar method for calculating fractional sky visibility from complex ground scenes. Approximate visibility at a location is constructed from hemicube projections at the surrounding grid points.

Rendering algorithms which utilize volumes of radiometric data are relatively rare. Most illumination methods are concerned only with light energy leaving from and arriving at a surface. Energy interactions at surfaces can be very complex while light behavior in space is comparatively straightforward. As attempts are made to render progressively complex objects, radiometric volume methods should prove to be increasingly useful.

Chapter 3

Radiance, Irradiance, and the Irradiance Volume

This chapter discusses the radiometric quantities *radiance* and *irradiance* and defines an *irradiance volume*.

The fundamental radiometric quantity is *radiance*, which describes the density of light energy flowing through a given point in a given direction. This quantity determines the luminance of a point when viewed from a certain direction. This concept is shown in Figure 3.1, where a point $\mathbf{x}$ is shown in the center of a room with four constant colored walls. At all points on the left wall, and for all outgoing directions, the radiance is 4.0. At point $\mathbf{x}$, the radiance is thus 4.0 for all directions that come from the left wall. The radiances at $\mathbf{x}$ for many directions are shown in Figure 3.1(b). Note that the radiances are shown with arrows pointing in the direction from which the light comes; although this convention seems somewhat

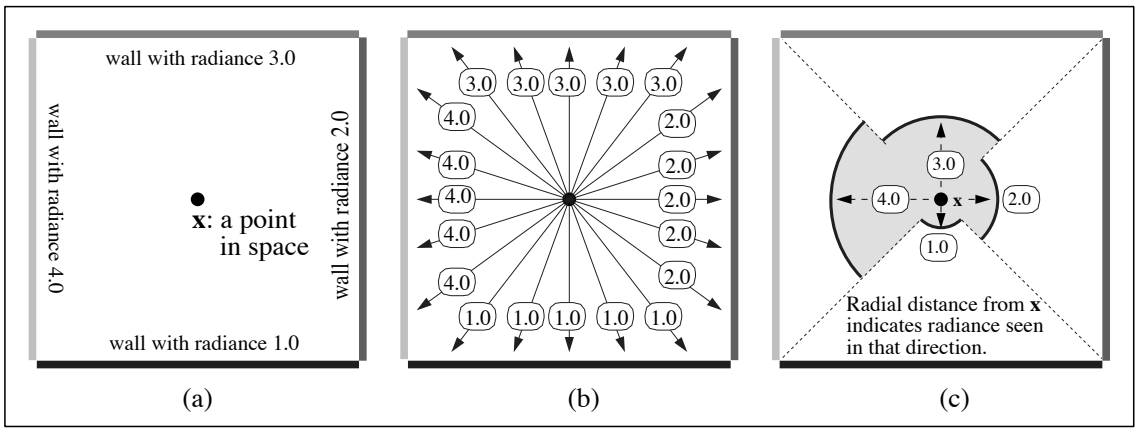


Figure 3.1: *Building a radial plot of radiance as seen from a point in space.*

backwards, it will prove convenient when dealing with irradiance.

The radiance in all directions at $\mathbf{x}$ makes a directional function called the *radiance distribution function* [6], which is shown as a radial plot in Figure 3.1(c). As can be seen, the radiance distribution function is not necessarily continuous, even in very simple environments. Since there is a radiance distribution function at every point in space, radiance is a five-dimensional quantity defined over all points and all directions[1]. Since the radiance distribution function may be discontinuous, it follows that radiance is not necessarily continuous over the space of all positions and directions.

The radiance of a surface is determined entirely by its reflective properties and the radiance incident upon it. The incident radiance defined on the hemisphere of incoming directions is called the *field-radiance* function [8]. The field radiance for a point on a very small surface is shown as a radial function in Figure 3.2(a). Because

[1]Three dimensions are spatial, and two are directional, analogous to latitude and longitude over the spherical set of directions.

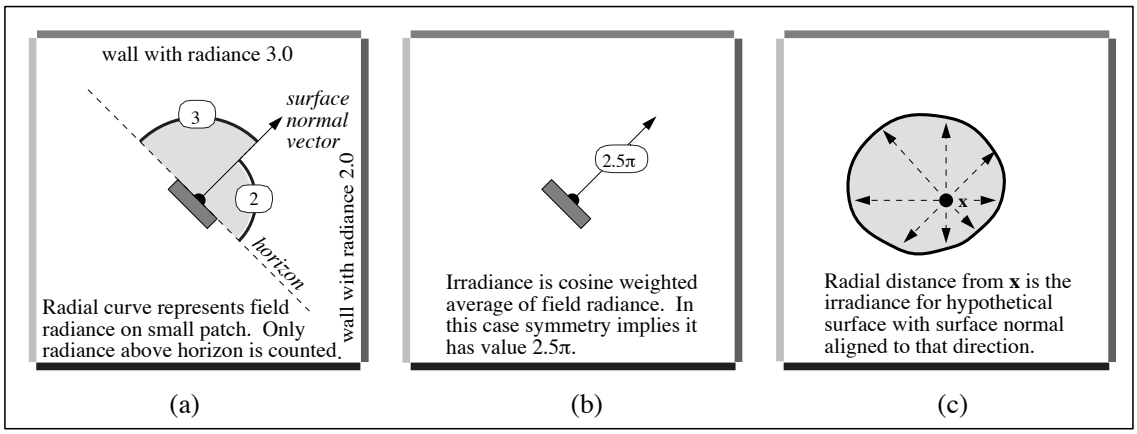


Figure 3.2: *Building a radial plot of irradiance for irradiance for all potential surfaces at a point in space.*

the surface is very small, the radiance in the room is not significantly different from the empty room of Figure 3.1.

If the surface is diffuse then its radiance is conveniently defined in terms of the surface's *irradiance*, H. Irradiance is defined to be

$$H = \int L_f(\omega) \cos\theta \; d\omega, \tag{3.1}$$

where $L_f(\omega)$ is the field radiance incident from direction ω, and θ is the angle between ω and the surface normal vector. With this definition, the radiance of the surface, $L_{surface}$, is

$$L_{surface} = \frac{\rho H}{\pi}, \tag{3.2}$$

where ρ is the reflectance of the surface. This radiance is constant for all outgoing directions.

If we rotate the small patch shown in Figure 3.2(b) by a small amount counter-clockwise, then the field-radiance function will become slightly larger on average.

Thus the irradiance will increase slightly, but will not change radically because of the smoothness of the averaging process. If we compute the irradiance for all possible orientations of the small patch, then we can plot these irradiances as a radial function, as shown in Figure 3.2(c). We call this the *irradiance distribution function* at the point. Note that this function is continuous over direction, even though the underlying field radiance is discontinuous.

An irradiance distribution function can be computed at every point in space, yielding a five-dimensional function. This function represents the irradiance of a hypothetical differential surface oriented at any location and direction within a space. We represent this five-dimensional irradiance function with the notation $H(\mathbf{x}, \omega)$. $\mathbf{x}$ corresponds to the three spatial dimensions, and ω to the two directional dimensions. Note that $H(\mathbf{x}, \omega)$ is a function which reduces five dimensions to one. Thus, it is quite different than *vector irradiance* [5], which maps three dimensions to three dimensions.

Evaluating $H(\mathbf{x}, \omega)$ efficiently is the key to the fast rendering of diffuse surfaces. Since evaluating $H(\mathbf{x}, \omega)$ would require explicit visibility and radiometric calculations for all surfaces in the environment, explicit evaluation of $H(\mathbf{x}, \omega)$ is not a practical option for complex environments. For high efficiency, we must approximate $H(\mathbf{x}, \omega)$.

Now suppose we have a volumetric approximation to $H(\mathbf{x}, \omega)$, called $H_v(\mathbf{x}, \omega)$, within the space enclosed by an environment. If we have H_v in a form we can evaluate quickly, we can avoid the costly explicit evaluation of H. This approximation is what we call the *irradiance volume*, and the key assumption is that we

can approximate H accurately enough that the visual artifacts of using H_v are not severe.

The simplest way to approximate a function is to evaluate it at a finite set of points and use interpolation techniques to evaluate the function between these known points. This approach is practical only if the function is reasonably smooth. In the case of $H(\mathbf{x}, \omega)$, $H(\cdot, \omega)$ (the function of direction while holding position constant) is continuous, while $H(\mathbf{x}, \cdot)$ is not necessarily continuous. The latter statement should make us hesitate to approximate $H(\mathbf{x}, \omega)$ using interpolation.

What gives us confidence that we can interpolate irradiance in the volume is that researchers have had great success interpolating irradiance on surfaces [25, 89, 34], even though discontinuities exist at some surface locations [55]. In practice, irradiance on surfaces is continuous in space between visibility events [88, 55, 5] (Figure 3.3). Since the smooth regions between these visibility events are large in practice, approximating the spatial variation of irradiance using interpolation is practical for most surface locations. The same facts are true in the volume, as shown by Lischinski [55] who computed discontinuity regions in space and projected them onto surfaces.

Thus, we can store the approximation to $H(\mathbf{x}, \omega)$ as point samples, and use interpolation methods to approximate the irradiance at locations between those samples. We then use this reconstructed irradiance to compute the radiance of a surface. Figure 3.4 illustrates this process in two dimensions.

The crucial issue of where to put sample points and the methods used to obtain directional samples at these points are discussed in Chapter 4.

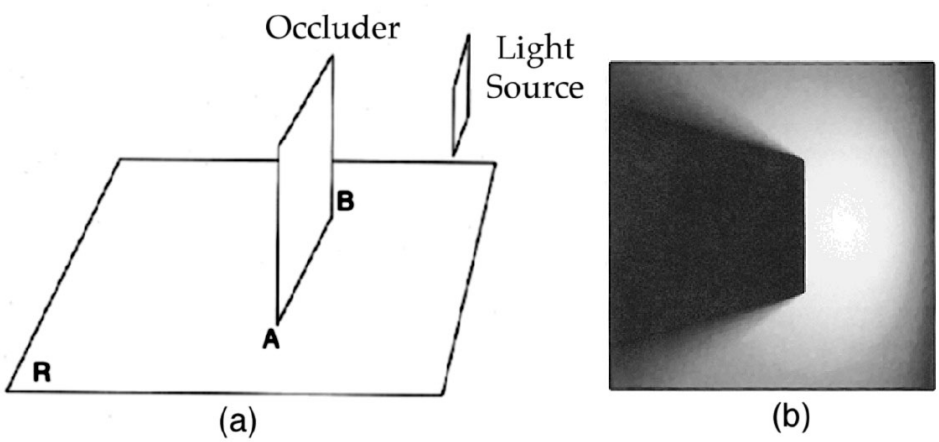


Figure 3.3: *(b) shows a discontinuity in the irradiance across a surface due to an occluding patch. (a) illustrates the geometry corresponding to the image. The discontinuity on surface R lies across the line AB. This figure was redrawn from [54].*

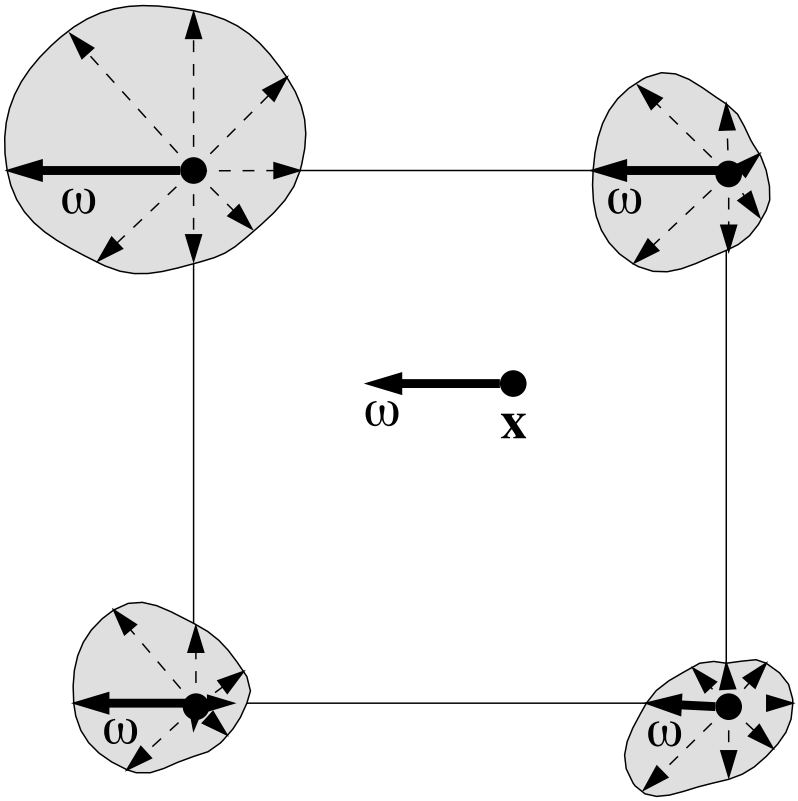


Figure 3.4: *The irradiance for normal direction ω at point* $\mathbf{x}$ *is interpolated from the values stored at the vertices of the surrounding cell.*

Chapter 4

Implementing the Irradiance Volume

This chapter describes the procedures and data structures used in implementing the irradiance volume. The first two sections discuss reconstructing the irradiance function; Section 4.1 deals with the issue of determining where to sample in the irradiance function's domain and Section 4.2 describes how sampling is performed. Section 4.3 discusses the data structures used to represent the volume and how the completed irradiance volume is queried. Finally, Section 4.4 describes an extension to the irradiance volume for illumination of glossy surfaces.

4.1 Sampling Strategy

To build the irradiance volume we need to be able to sample the radiance at any point in any direction. For environments with an explicit geometric representation,

such as one composed of polygons, acquiring the radiance can be easily done with ray-casting techniques. An irradiance volume can also be built in environments which do not contain explicit geometry, such as ones used by picture-based methods. The type of source environment does not matter, provided that radiance can be freely point-sampled. This section describes a sampling strategy designed for environments with explicit geometry. Section 5.3 in the following chapter describes how a volume might be constructed for a picture-based method.

In deciding which points and directions are sampled, and how to build these samples into a data structure, we gain some inspiration from previous work on building radiometric data structures. Patmore [64] and Greene [40] utilized radiance volumes to simplify rendering of complex scenes. Reinhard [69] used environment maps to approximate radiance [69]. LaFortune [53] and Jensen [48] used radiance volumes to guide importance sampling.

All of these methods used either regular subdivision, or k-d tree subdivision. Since they also were approximating quantities much less well-behaved than irradiance (i.e., visibility or radiance), we should expect similar data structures to work at least as well for irradiance.

An immediate question is whether to build the irradiance volume with uniformly spaced samples or adaptive samples. Uniform samples are appropriate if we expect the function to vary slowly relative to the distance between adjacent samples. We could treat the three spatial and two directional dimensions similarly, yielding a general five-dimensional data structure [7], but there are reasons to treat the spatial and directional dimensions differently. As discussed in the last chapter, any discon-

tinuities in the irradiance volume will be associated with the spatial dimensions, and thus adaptive sampling should be used only there.

We chose a bilevel grid approach for adaptive sampling in the spatial dimensions. This has the advantages of being easy to compute and having a well-defined order of complexity for search operations. Using a bilevel grid, the irradiance volume is computed as follows:

1. Subdivide the bounding box of the scene into a regular grid.
2. For any grid cells that contain geometry, subdivide into a finer grid.
3. At each vertex in the grid, approximate the irradiance distribution function.

Subdividing within cells containing geometry alleviates the most likely source of serious error in irradiance interpolation since most discontinuities are caused by interruptions of light flow by surfaces (Figure 4.1). Using a bilevel grid also allows low density sampling in open areas, while providing a higher sampling resolution in regions with geometry. Ultimately, a finite-element-style mesh based on volume discontinuities is needed to eliminate interpolation errors completely, but good results can be attained in practice with surprisingly coarse subdivisions.

Figure 4.2 shows an example of a bilevel grid. First, a 3×3×3 first-level regular grid is built within the room, with each sphere representing a vertex of the grid. Next, every grid cell containing geometry is subdivided into a second-level 3×3×3 cell. Note that the regular grid has actually been placed slightly within the outer

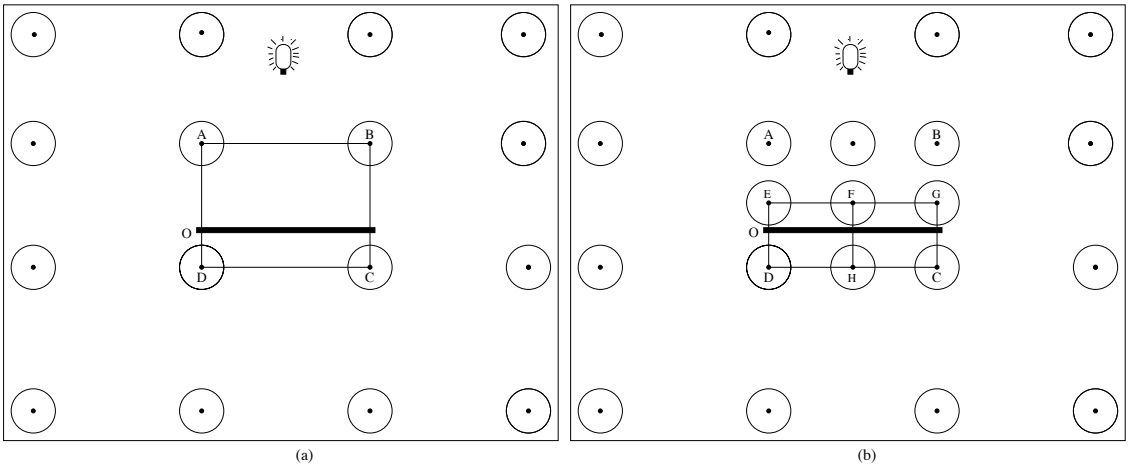


Figure 4.1: *(a) shows a room with a 4×4 first-level grid, with circles representing spatial volume sample locations. There is a discontinuity in the irradiance function in the cell $ABCD$ across the surface O; samples C and D are in shadow whereas samples A and B can see the the light source. Because of this, interpolated irradiance within the cell will be incorrect. The irradiance at locations above O will appear unnaturally dark due to the contribution of C and D, and brighter than it should at locations below. (b) shows a 3x3 second-level grid placed within the cell. Only cells $EFHD$ and and $FGCH$ will now yield bad approximations, halving the erroneous area. The region could be improved even further by using a second-level grid with a higher resolution.*

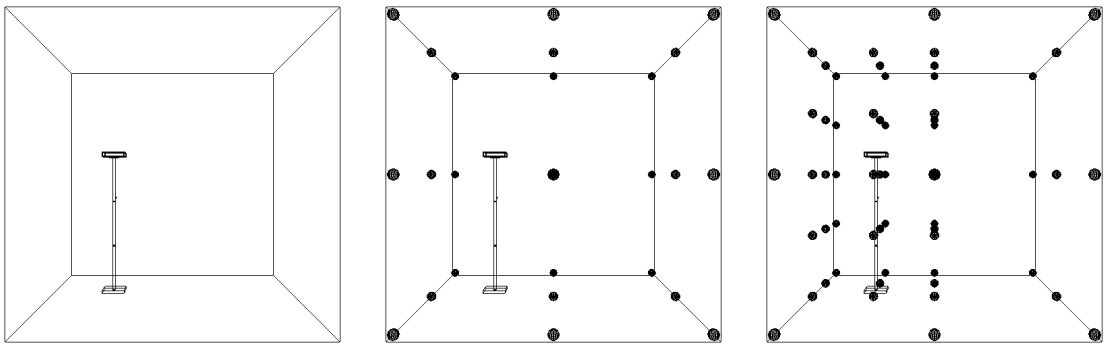


Figure 4.2: *A bilevel grid being built in a room with a lamp. The center panel shows the first-level grid. In the right panel, second-level grids have been built in the the cells containing the lamp.*

walls of the volume so that the grid does not subdivide around them. This is often done in practice so that the grid vertices at the edges of a volume do not intersect pieces of geometry.

We have not discussed any heuristic for choosing a first or second-level sampling rate; currently, the rates are chosen manually. The sampling resolution needed depends greatly on the size of an environment and the density of objects contained within. Large open areas of space often only need a sparse sampling whereas "cluttered" spaces generally need more. Other situations which may affect the needed sampling rate include placement of the luminaires in the scene and the requirements of the application using the volume.

4.2 Approximating the Irradiance Distribution Function At a Point

The irradiance distribution function at a point is approximated using the following procedure:

1. Given a directional sampling resolution, choose a set of directions over the sphere of directions in which to sample radiance.
2. In each direction chosen, determine and store the radiance.
3. Using the stored radiance, compute the irradiance distribution function.

These steps are performed at each grid vertex to form the irradiance volume. Each step is discussed in detail in the following sections.

4.2.1 Computing Sampling Directions

When gathering the radiance at a point $\mathbf{x}$, we point sample the radiance in a set of directions over the spherical set of directions $\vec{S^2}(\theta, \phi)$. This results in a piece-wise constant approximation of the radiance distribution, each sample defining a region of constant radiance over $\vec{S^2}$.

Given that we have chosen to sample uniformly over the two directional dimensions, we would like to pick a good method for selecting a set of directions for a specified sampling resolution. In addition to covering $\vec{S^2}$ as uniformly as possible,

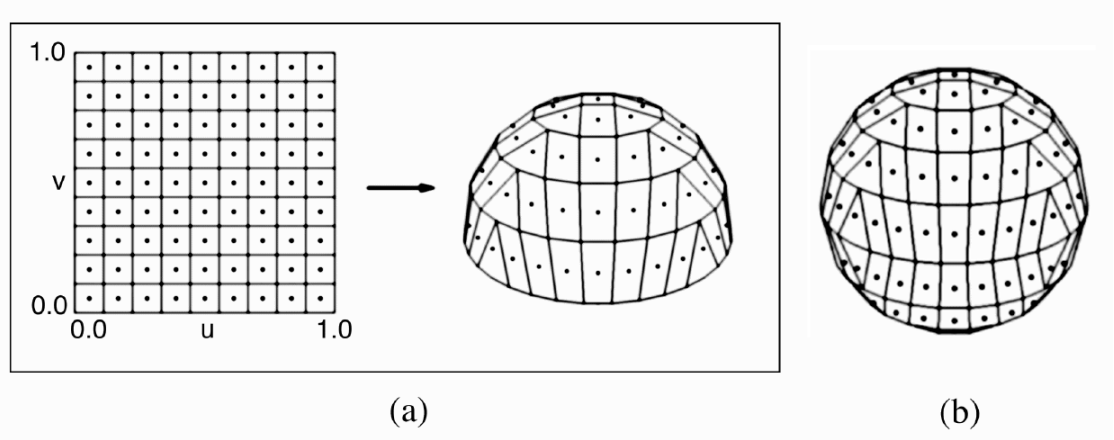


Figure 4.3: *(a) shows the hemispherical mapping of a grid. (b) shows a sphere composed from two hemispheres.*

we would like the regions surrounding the sample points to have equal area in order to simplify the computation of the irradiance distribution function later on.

Shirley *et al.* [76] provide a method of mapping a unit square to a hemisphere while preserving the relative area of regions on the square. By using this procedure, we are afforded the mathematical convenience of being able to uniformly subdivide to a given resolution on a square. This mapping procedure expresses better behavior than latitude-longitude mapping because the mapped areas have better aspect ratios, reducing distortion.

Figure 4.3(a) shows a 9×9 grid on a unit square and its hemispherical mapping. The dot in the center of each grid square represents a direction in which the radiance will be obtained from a point at the center of the sphere. Each grid cell shows the size of the region that the sample represents. The mapping is performed twice to form the upper and lower hemispheres of the sphere shown in Figure 4.3(b).

As the resolution of the sampling grid increases, the approximation of the radi-

ance distribution becomes more accurate. Super-sampling can be also be performed by taking more than one radiance sample per region and then averaging the results. This may sometimes be preferable to using a finer grid since the cost of computing the irradiance distribution is proportional to the number of radiance samples.

4.2.2 Sampling Radiance

Given a point in an environment and a direction, the radiance, $L(\mathbf{x}, \omega)$, is computed by casting a ray from $\mathbf{x}$ in the direction ω and determining the first surface hit. If that surface is diffuse, the radiance is queried from that surface. Otherwise, the ray is specularly reflected off the surface and the next intersection is found. This process is continued until the ray intersects a diffuse surface or meets some other stopping criteria, such as the attenuation of the ray being below a specified tolerance [42]. If a diffuse surface is ultimately hit, the remaining specular coefficient after attenuation from intersections with specular surfaces is multiplied with the diffuse color to give the radiance; otherwise, a constant quantity analogous to an ambient component is returned. To reduce the number of ray-object intersection tests performed when ray-casting, a uniform spatial-subdivision grid [35, 93] is computed in the environment before the volume is built.

At each grid vertex in the volume, the radiance is sampled in a pre-computed set of directions, $\{\omega_1, \ldots, \omega_n\}$, obtained by using the method discussed in the previous section. Each radiance value is stored in an array and is used to compute the irradiance distribution at that point once the sampling is finished.

Figure 4.4 illustrates this procedure in a diffuse environment. Panel (a) shows

a point $\mathbf{x}$ in the center of a room. In panel (b), the radiance is sampled in a set directions; each line represents a ray from $\mathbf{x}$ and is colored with the value of the radiance seen in that direction. A radial plot of the red component[1] of the gathered radiance is pictured in panel (c).

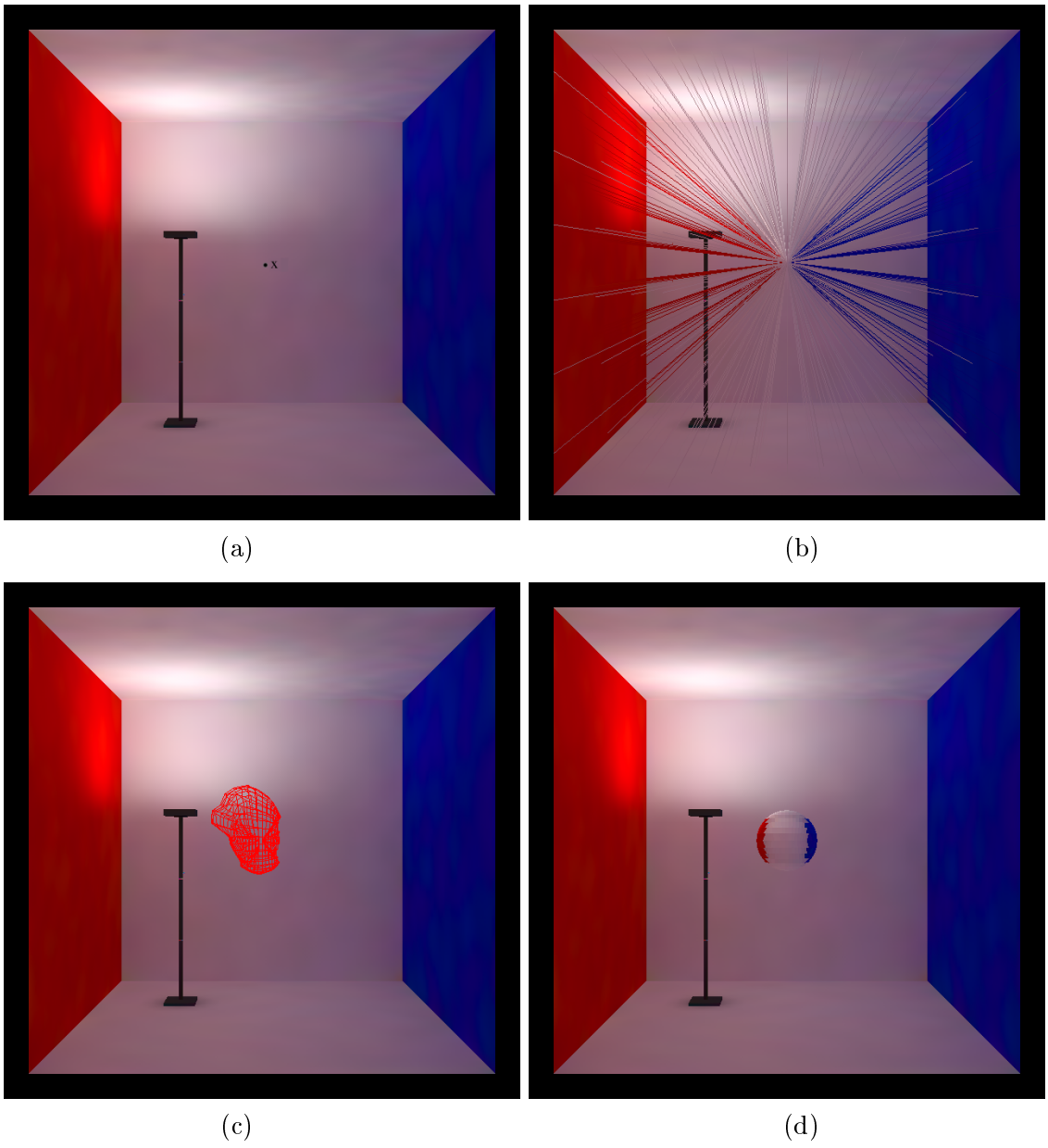
In addition to radial plots, another useful representation of directional radiance data at a point is the *radiance sphere*, shown in Figure 4.4(d). The radiance sphere can be thought of as a magnification of an infinitesimal sphere centered at $\mathbf{x}$. The color of a point on the surface corresponds to the radiance seen in that direction from the center of the sphere. Each polygon comprising the sphere corresponds to the region, or "bin", of constant radiance defined by a radiance sample taken through its center.

Figure 4.5 shows close-up views of three radiance spheres with differing resolutions. The spheres are placed at the same location as the one in the previous figure, but viewed from behind. Note that as the sampling resolution increases, the sphere increasingly takes on the appearance of a completely reflective ball.[2] This is what we would expect, since a radiance sphere reflects what can be seen of the environment from a particular location.

The number of radiance samples needed to ensure a good approximation of the irradiance distribution function at a point depends heavily on the environment. In general, unless the sampling resolution is very high, some surfaces in the environ-

[1]When sampling the radiance and computing the irradiance, each independent color component needs to be considered separately. In this thesis, the standard RGB color channels are used.

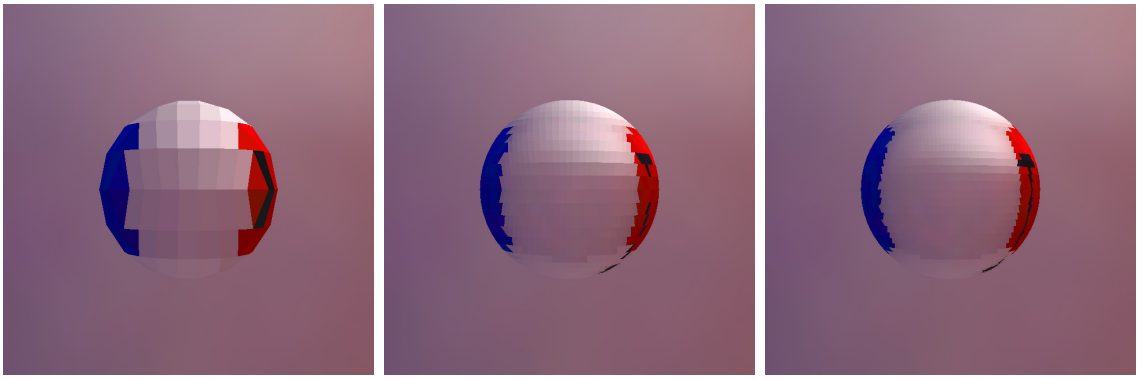
[2]It is also unlike a mirrored ball in the sense that it is view-independent; the color of a point on the sphere remains unchanged when viewed from different angles.



(a) (b)

(c) (d)

Figure 4.4: *Sampling radiance at a point. Note that the front wall of the room is not shown for display purposes.*



(a) (b) (c)

Figure 4.5: *Radiance spheres at different resolutions: (a) 162 bins, (b) 1682 bins, (c) 4802 bins.*

ment will be missed by radiance queries. Since the irradiance is a weighted average of the radiance, missed surfaces whose luminance does not differ greatly from the surrounding area will not greatly effect the irradiance calculation. Larger inaccuracies result when a surface with a relatively high luminance, such as a light source, is passed over by the sampling. Currently, the only way to ensure that no surfaces are missed is to increase the sampling rate; for large environments with many small surfaces, this may not be practical. To fully address this problem with a reasonable number of samples, an informed sampling strategy [37] should be used.

Another factor to consider is the effect of the number of radiance samples on the computation time of the irradiance distribution. As will be shown below, calculating the irradiance in a particular direction involves all the radiance samples. If the directional resolution of the irradiance computations is large, even a reasonably small increase in the number of radiance samples can have a significant effect.

4.2.3 Computing Irradiance

After the radiance is gathered at a point, it is used to compute the irradiance for each directional bin on the sphere. For convenience, we compute the irradiance at the same resolution and in the same directions as the radiance. Since irradiance is a smoother function than radiance, we can assume that a higher sampling rate is not needed. The optimal rate for sampling is an open issue; in practice, fewer irradiance samples may actually be needed.

The irradiance in a given direction is computed for a hypothetical surface which is normal-aligned in that direction. Given a directional set of radiance samples $\{\omega_1, \ldots, \omega_n\}$, taken at a point in space $\mathbf{x}$, the irradiance in direction ω is calculated using the following formula:

$$H(\omega) = \sum_{i=1}^{N} L(\omega_i) \max(0, \omega_i \cdot \omega) \Delta\omega_i. \tag{4.1}$$

L is the radiance seen in direction ω_i from point $\mathbf{x}$, $\Delta\omega_i$ is the solid angle of the bin associated with ω_i, and N is the number of directional bins on the sphere. This formula is a different form of Equation 3.1; instead of analytically integrating the radiance over a hemisphere to find the irradiance, numerical quadrature is used instead to produce an approximation.

If the sphere of directions is subdivided into regions of equal area, then

$$\Delta\omega_i = \frac{4\pi}{N}. \tag{4.2}$$

and Equation 4.1 becomes

$$H(\omega) = \frac{4\pi}{N} \sum_{i=1}^{N} L(\omega_i) \max(0, \omega_i \cdot \omega). \tag{4.3}$$

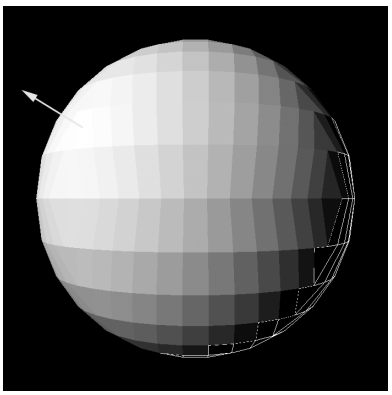
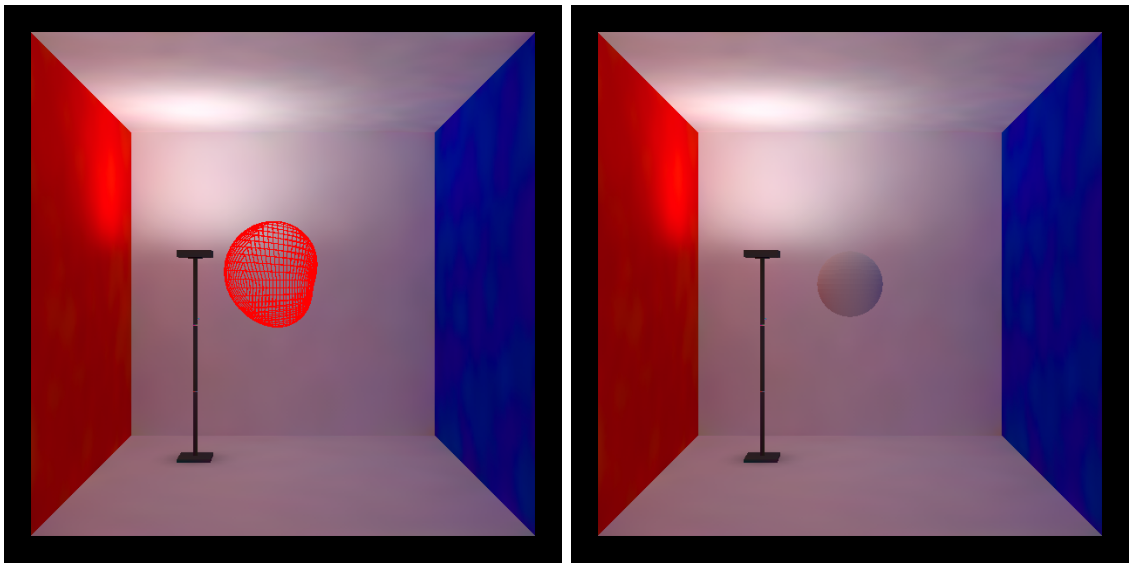


Figure 4.6: *Calculating the irradiance for a single direction. Each shaded bin corresponds to a contributing radiance value. The intensity of each bin indicates the amount of contribution, brighter areas affecting the irradiance more.*

This equation is evaluated for a number of directions at $\mathbf{x}$ to obtain a set of directional irradiance samples. For each irradiance direction ω, the irradiance is computed by summing the cosine-weighted contribution of each radiance sample on the hemisphere oriented in that direction (Figure 4.6). The approximate irradiances form an approximation of the irradiance distribution function at x.

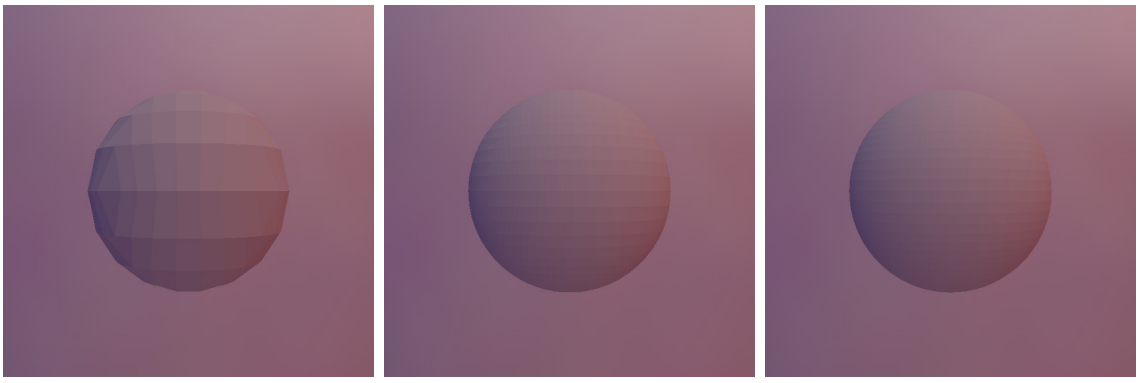
The irradiance distribution function can be displayed in a manner analogous to the radiance sphere as an *irradiance sphere*. Figure 4.7 shows both an irradiance sphere and a radial plot of irradiance in the environment shown in the previous section. Note how much smoother the irradiance is compared to the radiance (Figures 4.5). Figure 4.8 shows three irradiance spheres at the same position as the radiance spheres in Figure 4.5.

Figure 4.9 presents a completed irradiance volume. An irradiance sphere is displayed at each grid vertex, representing the irradiance data stored at that point



(a) (b)

Figure 4.7: *The irradiance distribution at a point in the center of the room. Panel (a) shows a radial plot of the red component, and (b) shows an irradiance sphere.*



(a) (b) (c)

Figure 4.8: *Irradiance spheres at several resolutions: (a) 162 bins, (b) 1682 bins, (c) 3362 bins.*

in the volume.

4.3 Querying the Irradiance Volume

The irradiance volume is represented as three distinct data structures: *samples*, *cells*, and *grids*. Samples contain directional irradiance values corresponding to a particular point in the environment (Section 4.2.3). A cell represents a box[3] in space bounded by eight samples, one at each of its corners. A grid is a three-dimensional array of cells. Each cell may also contain another grid, forming a bilevel structure. Figure 4.10 shows examples of each of the three structures. Figure 4.11 presents a map of the how the data structures are used to represent the volume shown in Figure 4.9.

Given a point $\mathbf{x}$ and a direction ω, querying the irradiance from the volume requires the following steps:

[3]The term box is commonly used as shorthand for the three-dimensional analog of the rectangle, which is more precisely known as the "rectangular parallelepiped".

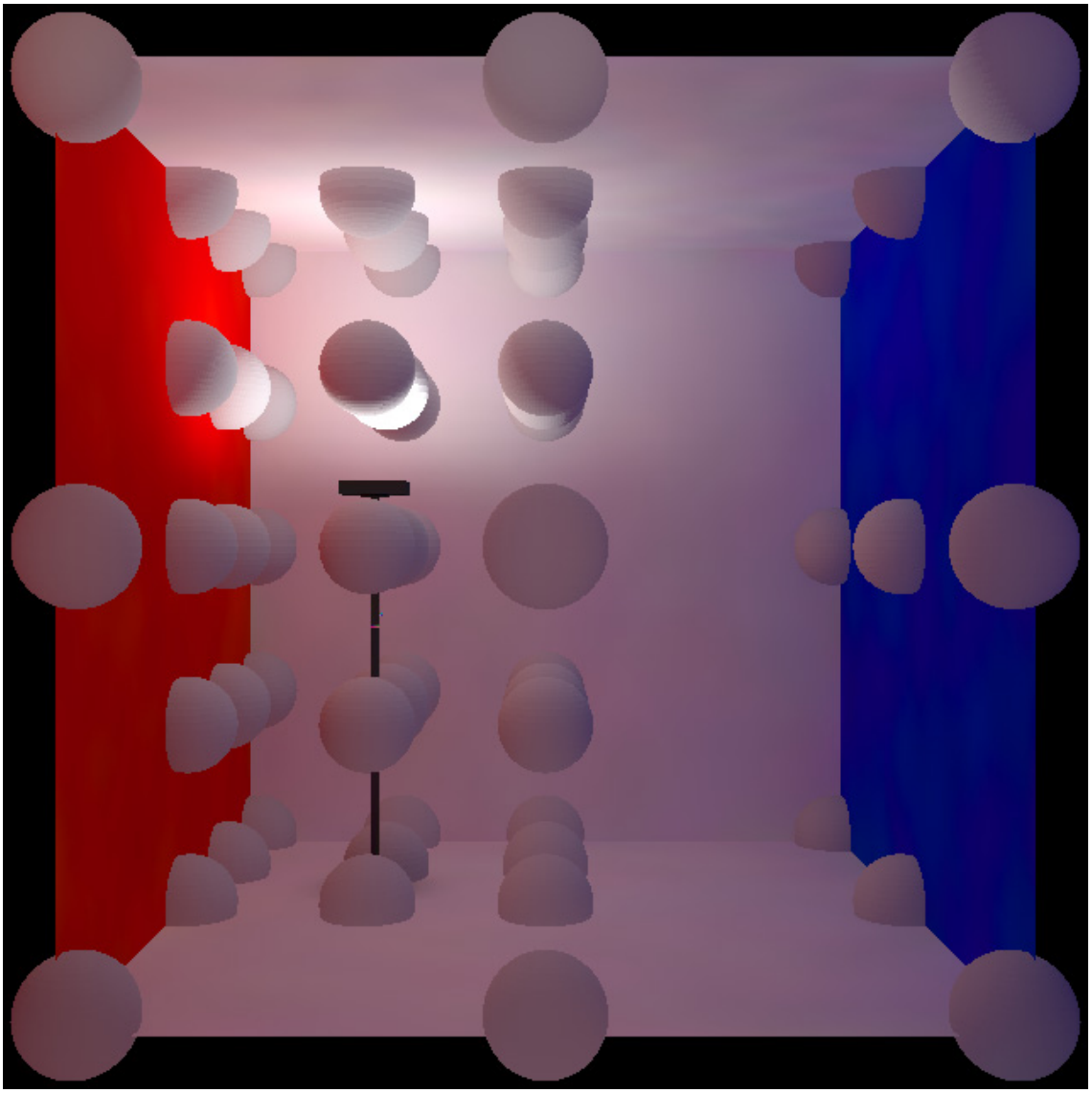
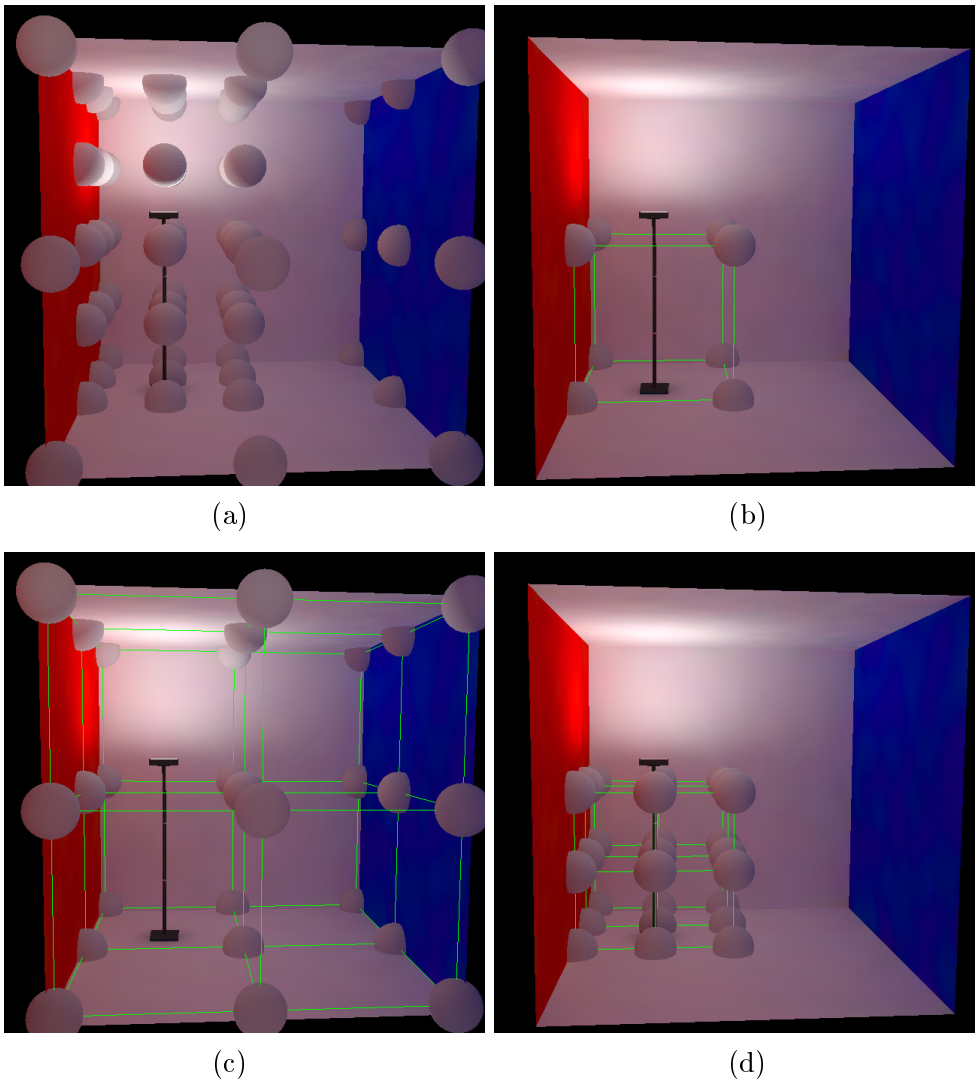


Figure 4.9: *A completed irradiance volume.*



(a) (b)

(c) (d)

Figure 4.10: *Volume data structures: samples (a), a cell (b), the first-level grid (c), and a second-level grid (d).*

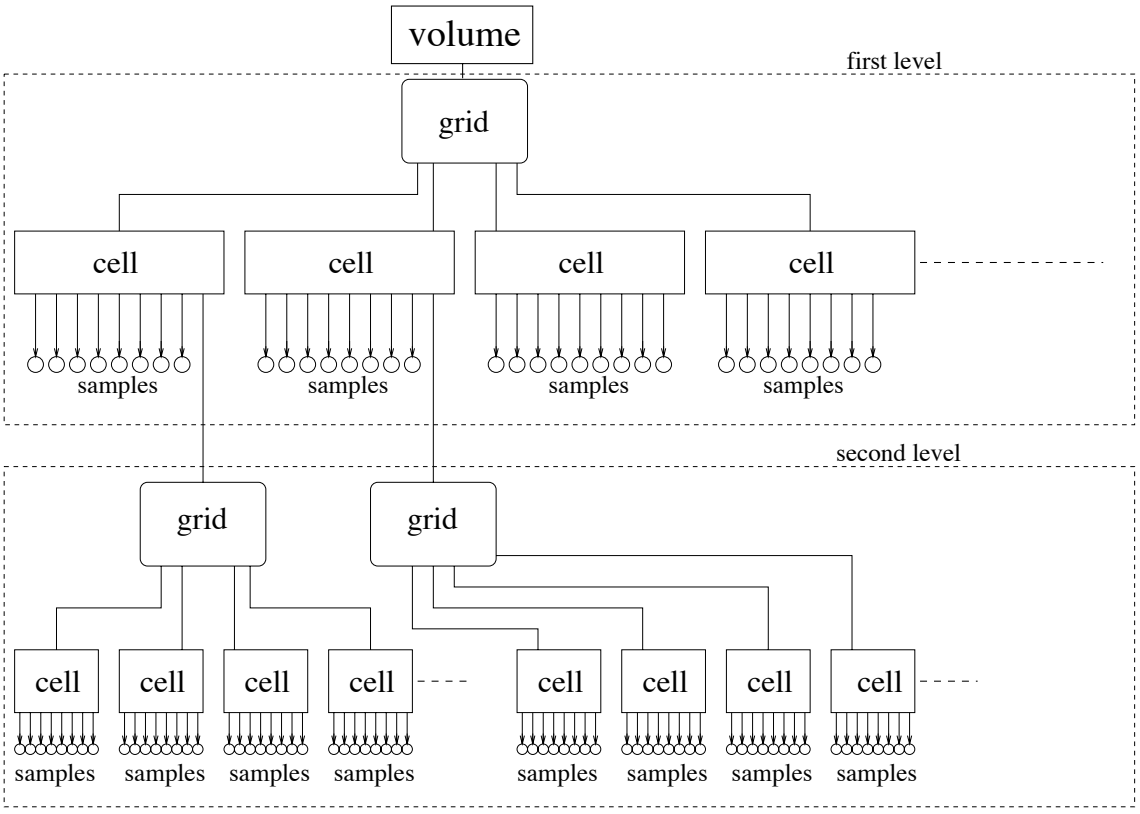


Figure 4.11: *A graph of the data structures used to store the volume. The first-level grid contains 2×2×2 cells. Two of those cells contain geometry (the lamp) and include second-level grids, each containing another eight cells. Each cell has pointers to eight of the sample structures; adjoining cells may share samples.*

1. Calculate which cell in the first-level grid contains $\mathbf{x}$.
2. If the cell contains a grid, calculate which cell in the second-level grid $\mathbf{x}$ lies in.
3. Find out which data value in the cell's samples corresponds to ω.
4. Given the position of $\mathbf{x}$ within the cell and the eight values from the surrounding samples, interpolate to get the irradiance.

To find which cell in a grid structure contains a point $\mathbf{x}$ and the fractional offset of that point within the cell, the following quantity, q_d, is computed for each dimension $d \in (x, y, z)$:

$$q_d = \frac{\mathbf{x}_d - grid\_min_d}{grid\_max_d - grid\_min_d} \times num\_cells_d, \tag{4.4}$$

where $grid\_min$ and $grid\_max$ are the minima and maxima of the grid's bounding box and num_cells is the size of the grid in a given dimension. The cell containing the point $\mathbf{x}$ is element $[cell_x, cell_y, cell_z]$ of the grid structure, where $cell_d$ is the integer component of the above quantity:

$$cell_d = \lfloor q_d \rfloor . \tag{4.5}$$

The fractional offset of the point within the cell, used for interpolation, is the fractional component of the quantity in Equation 4.4:

$$offset_d = q_d - cell_d, \tag{4.6}$$

$offset_d \in [0.0, 1.0)$. Figure 4.12 shows an example of these quantities.

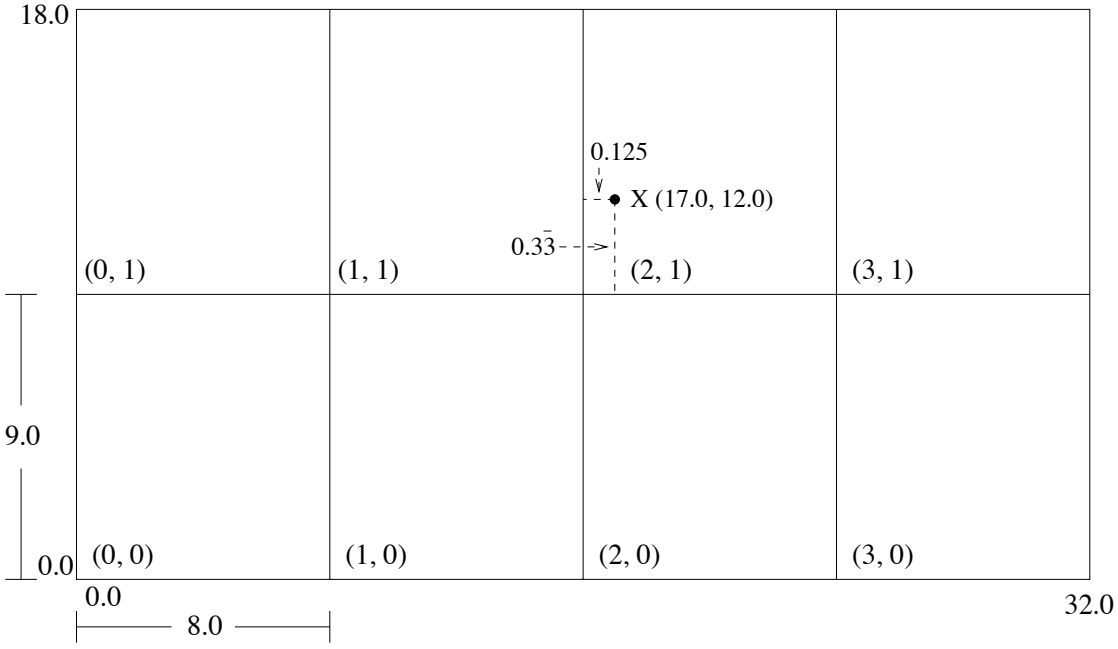


Figure 4.12: *Finding out which grid cell contains a point. In the above two-dimensional* 4×2 *grid, point* $\mathbf{x}$ *lies at coordinates* $(17.0, 12.0)$. *The numbers at the lower left of each cell indicate the index of that cell in the grid array. Using Equation 4.5,* $cell_x = \left\lfloor \frac{17.0-0.0}{32.0-0.0} \times 4 \right\rfloor = \lfloor 2.125 \rfloor = 2$ *and* $cell_y = \left\lfloor \frac{12.0-0.0}{18.0-0.0} \times 2 \right\rfloor = \lfloor 1.3\bar{3} \rfloor = 1$, *indicating that point* $\mathbf{x}$ *lies in the cell contained in grid array element [2,1]. Using Equation 4.6, the fractional offsets are* $offset_x = 0.125$ *and* $offset_y = 0.3\bar{3}$.

Every sample is stored as two two-dimensional arrays. Each array represents the irradiance computed over a hemisphere, with each array element corresponding to a direction in (u, v) space (Figure 4.3). A simple check is first performed on ω to determine which hemisphere it belongs to. Then, an inverse mapping of the method used in Section 4.2.1 is used to find the which bin on the unit square the direction lies in. After the eight values are obtained, trilinear interpolation is used to get the irradiance at point $\mathbf{x}$.

Given a surface with reflectance ρ, and an irradiance value from the volume H, the formula for the radiance of the surface is:

$$\text{radiance} = \frac{\rho H}{\pi}. \tag{4.7}$$

Figure 4.13 shows an icosahedron at several locations within the volume. The volume is queried to obtain the irradiance at the vertices of each face in the direction of the face's normal. The patch is then displayed with Gouraud shading.

4.4 Illuminating Non-Diffuse Objects

The irradiance volume can be extended to simulate non-diffuse effects by storing higher order moments than irradiance [6]. This allows the simulation of such phenomena as Phong-like glossy reflections, as long as the specularity is not too high[4].

[4]One of the main assumptions in the creation of the irradiance volume is that we can interpolate between samples to get a good approximation of the irradiance. As we move away from irradiance towards increasingly specular functions, this assumption becomes less valid.

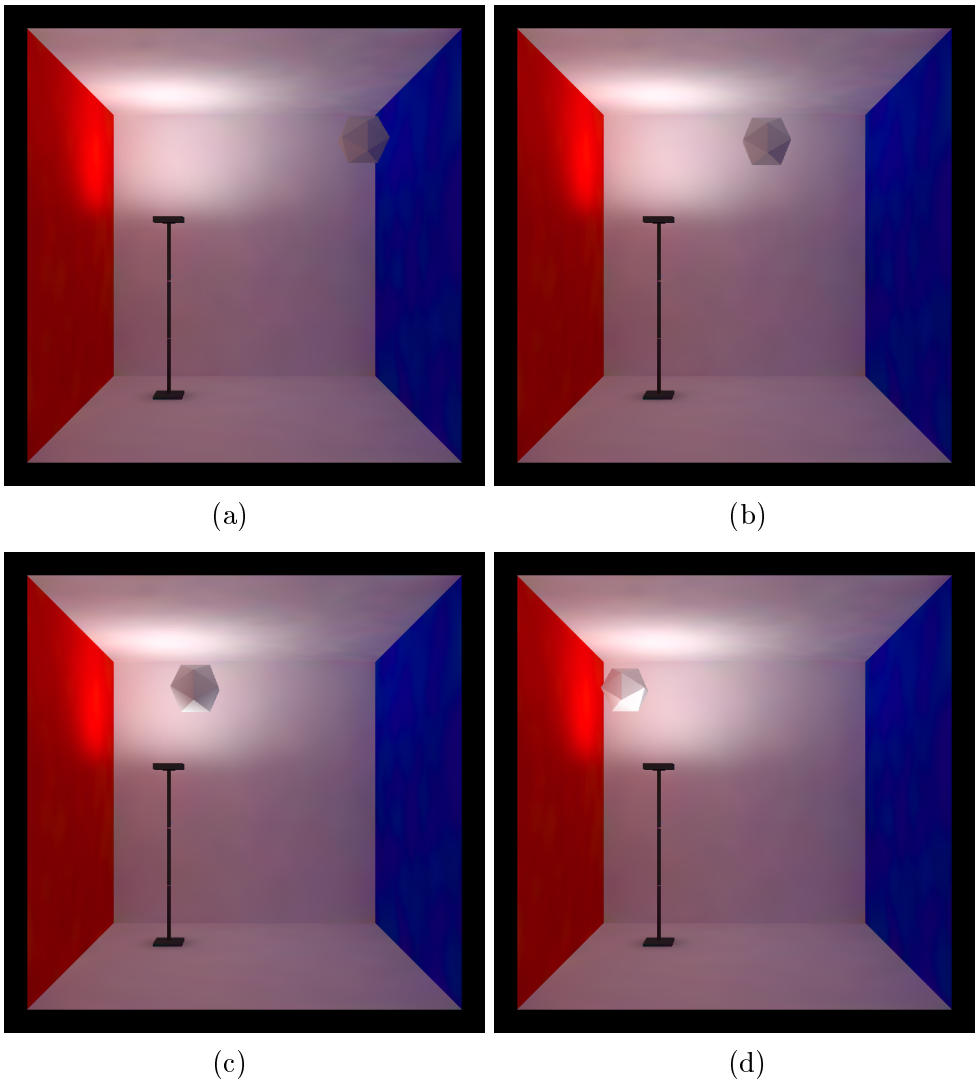


Figure 4.13: *An object illuminated from the volume, shown at several locations.*

To build a volume with higher-order moments, we use the following formula[5] instead of Equation 4.3:

$$H^n(\omega) = \left(\frac{n+1}{2}\right)\frac{4\pi}{N}\sum_{i=1}^{N} L(\omega_i)\max(0, \omega_i \cdot \omega)^n, \tag{4.8}$$

where n is the order of the moment. Note that if $n = 1$, the above equation becomes

$$H(\omega) = \frac{4\pi}{N}\sum_{i=1}^{N} L(\omega_i)\max(0, \omega_i \cdot \omega), \tag{4.9}$$

which is exactly same as Equation 4.3, the method we use to compute irradiance.

As discussed in Section 4.2.3, the irradiance in a particular direction ω is a cosine-weighted average of radiance samples on the hemisphere oriented around ω. As n increases, it has the effect of weighting the integration of the radiance further towards ω. This is shown in a series of graphs in Figure 4.14. The vertical axis designates the weighting a radiance sample on the hemisphere would receive at a given angle (indicated on the horizontal axis) from ω. As n increases, the curve approaches the ideal specular function. Figure 4.15 shows several orders of irradiance moment spheres[6] in the environment used in Figures 4.5 and 4.8. Figure 4.16 shows a complete irradiance moment volume; contrast this with the irradiance volume shown in Figure 4.9.

When a higher-moment volume is used in an application, changes must be made to the volume query function. Since we are no longer storing and using irradiance to shade objects, the illumination on a surface is no longer view-independent. Queries to the volume must take into account the position of the viewer. Instead of

[5] The notation H^n is used here to indicate the nth moment of H.

[6] Analogous to irradiance spheres.

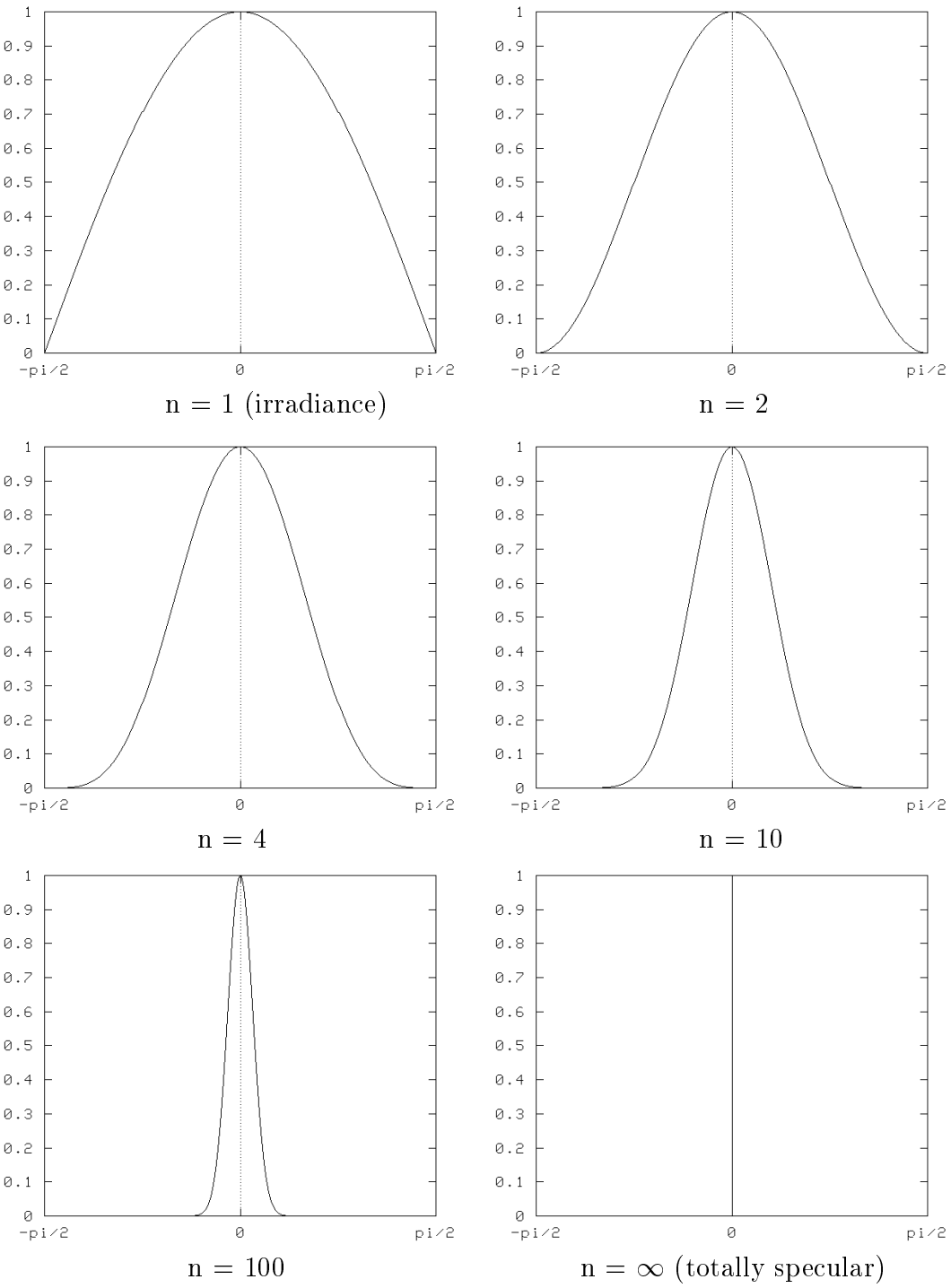


Figure 4.14: *Weighting graphs.*

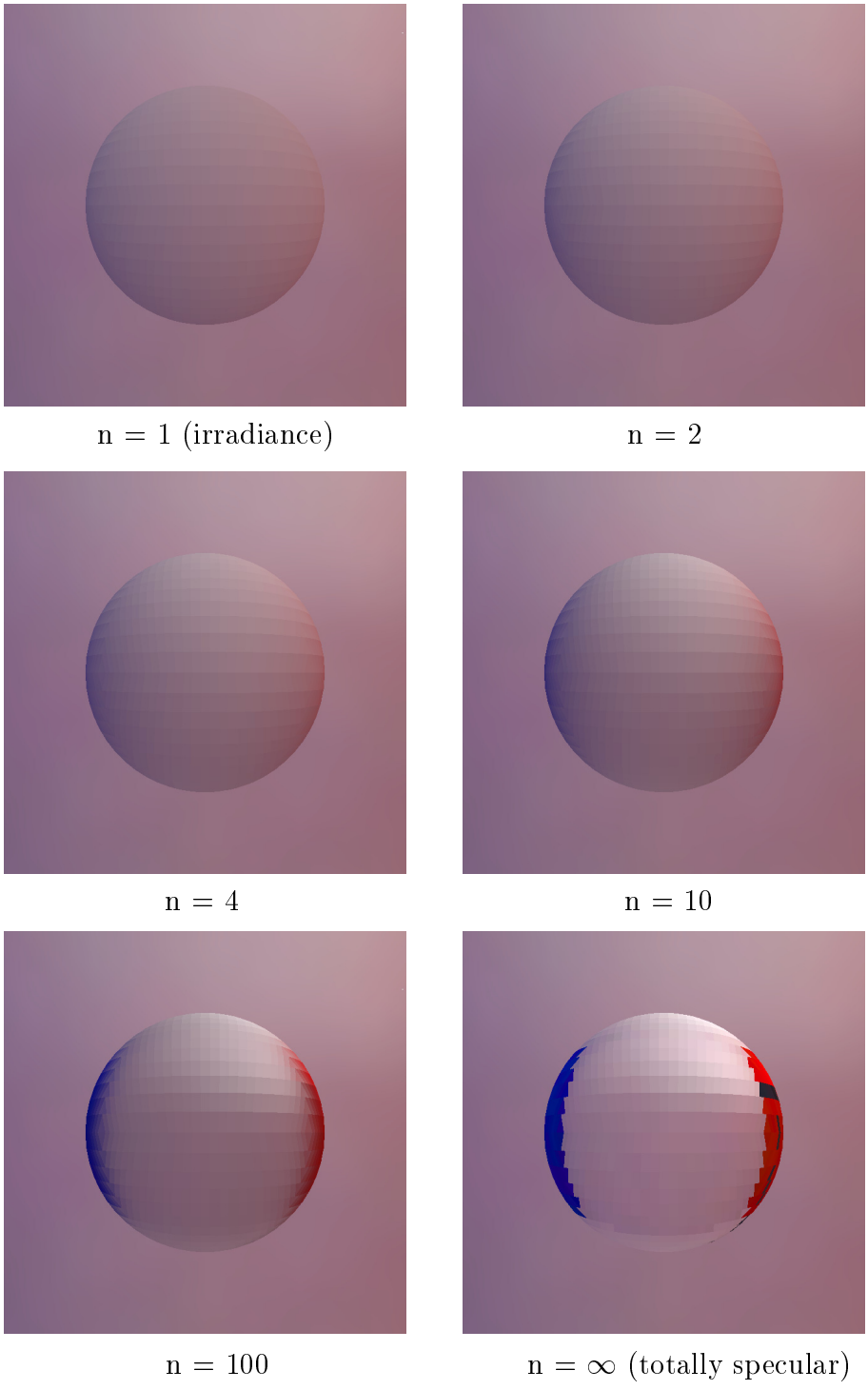


Figure 4.15: *Irradiance moment spheres.*

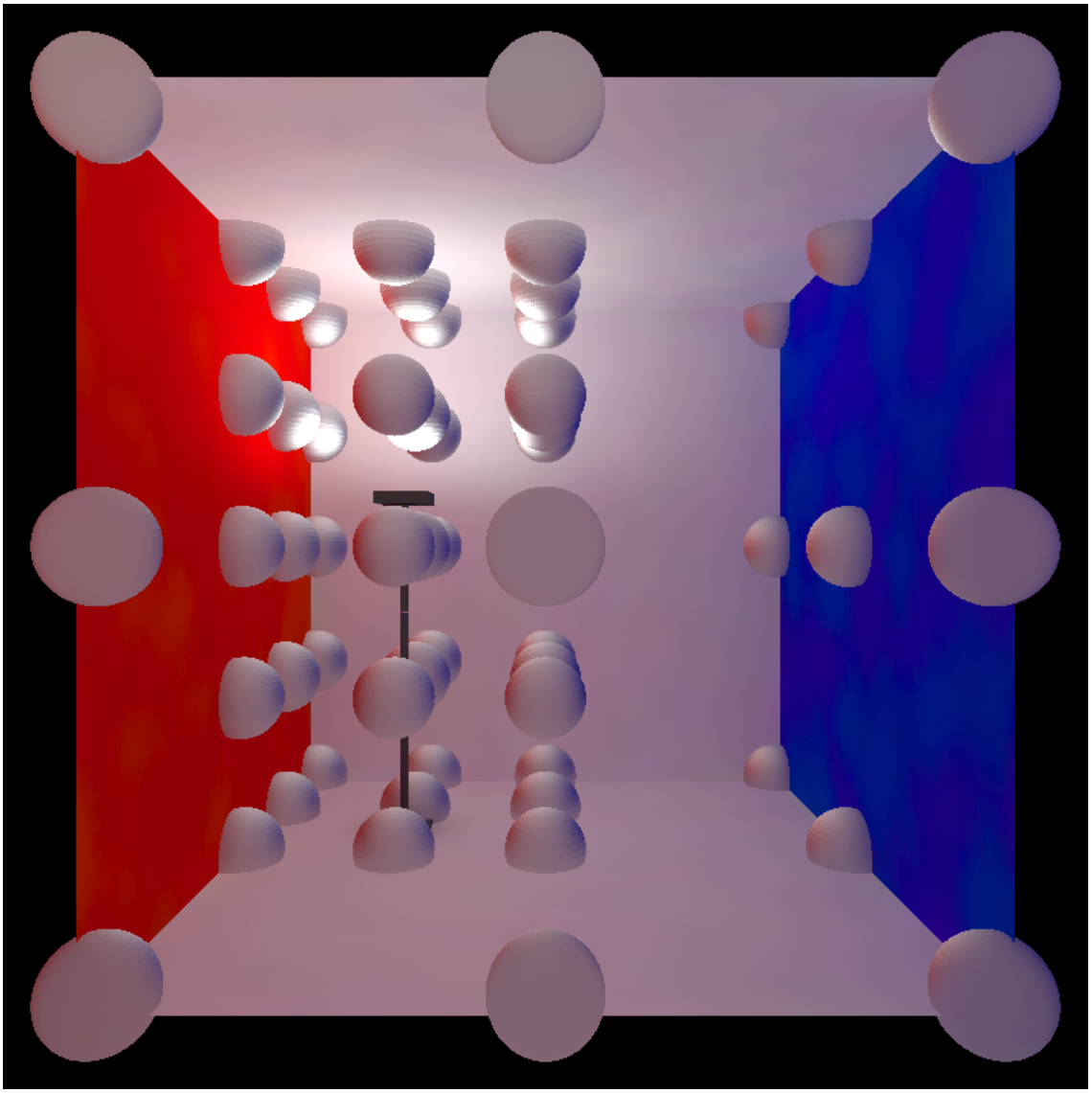


Figure 4.16: *A completed irradiance moment volume,* $n = 10$.

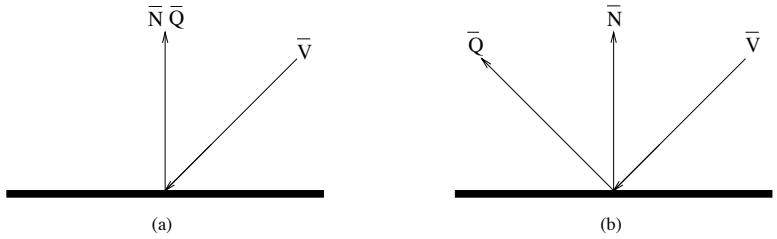


Figure 4.17: *Volume query directions for irradiance (a), and higher-order moments of irradiance (b). $\bar{N}$ is the surface normal, $\bar{V}$ is the viewing vector, and $\bar{Q}$ is volume query direction.*

querying in the direction of the normal of the surface, the view vector is mirrored about the normal to obtain the query vector in the direction of specular reflection. (Figure 4.17).

As n increases, the frequency of the illumination function also rises. As it becomes more specular, higher directional and spatial sampling rates are needed to maintain a good approximation. In addition, interpolated shading across a surface (such as Gouraud) becomes less accurate. Objects which query the volume to get the illumination at their vertices may need to have finer meshes for good results.

Figure 4.18 shows several views of an object illuminated by three volumes of differing moment orders.

Figure 4.18: *An object (a rabbit) illuminated from volumes with different orders of irradiance moments.* $n = 1$ *in the left panel,* 2 *in the center panel, and* 10 *in the right panel.*

Chapter 5

Applications

The previous chapter described the technical details of building an irradiance volume; this chapter discusses how irradiance volumes might be used for several different types of generic simulations.

5.1 Rendering Dynamic Objects in Semi-Dynamic Environments

Our definition of a *semi-dynamic* environment is one in which most surfaces are stationary, and the few dynamic objects are small relative to the scale of the environment. For example, positioning an object in an architectural application would fall into this category. The use of an irradiance volume is well suited for semi-dynamic environments and can be used to approximate the global-illumination of dynamic objects. Such a system starts with a preprocessing stage that builds an

irradiance volume from the static objects, as described in Chapter 4. Once the volume is built, it is queried to acquire the illumination of non-static surfaces.

This section presents an example implementation of an interactive semi-dynamic system and discusses several issues related to using the irradiance volume method in this context. Our implementation builds an irradiance volume in a radiosity-rendered environment and allows a user to move an object, illuminated from the volume, around the environment. The model used for this example, an office scene, is pictured in Figure 5.1. The dynamic object positioned under user control, in this case a grey polygonal model of a rabbit, is shown illuminated by the volume in Figure 5.2.

Figures 5.3 and 5.4 each show a sequence of frames of the rabbit in motion. In Figure 5.3 the rabbit moves from an open area of the office to a position under the desk, becoming darker as it enters the desk's shadow. Figure 5.4 shows color-bleeding effects as part of the rabbit takes on a yellow tinge as it approaches a divider.

5.1.1 Performance

Building the Volume

The irradiance volume used consists of a 7^3 (343) sample first-level grid and 72 $3 \times 3 \times 3$ second-level grids, equaling a total 2287 samples. Each sample has a directional resolution of 2×17^2 (578) angular bins. The volume took approximately

Figure 5.1: *Two views of the office model.*

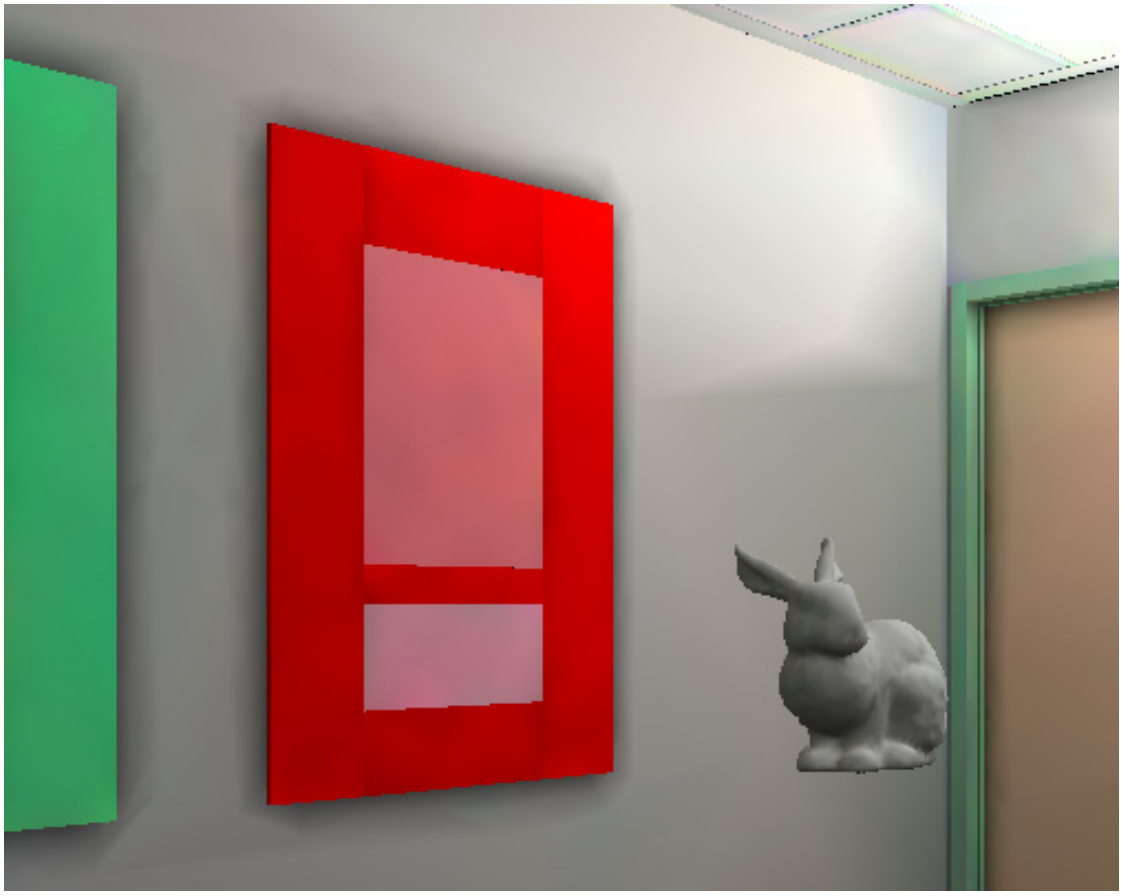


Figure 5.2: *Rabbit shaded by irradiance volume.*

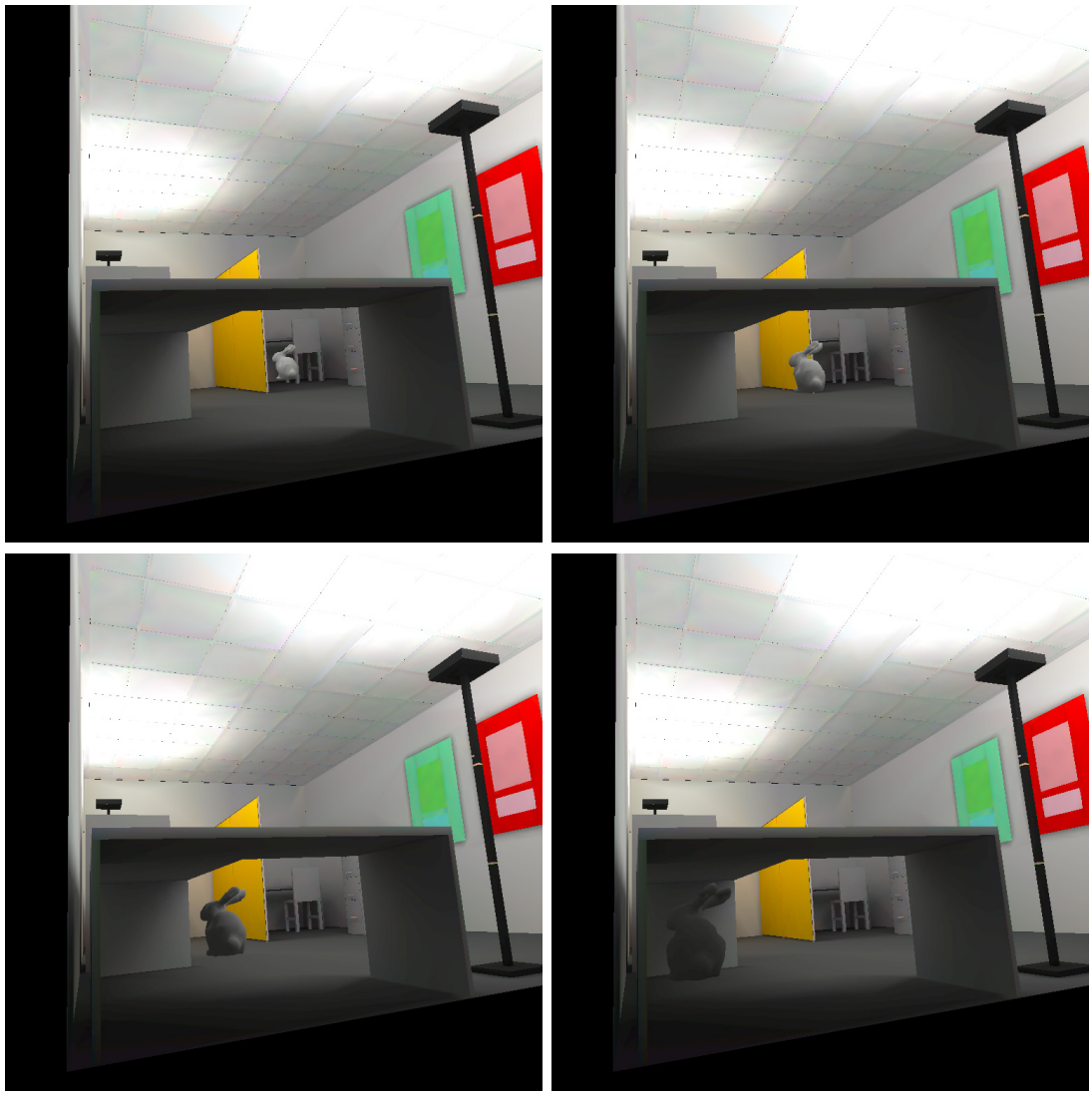


Figure 5.3: *Rabbit moving under desk.*

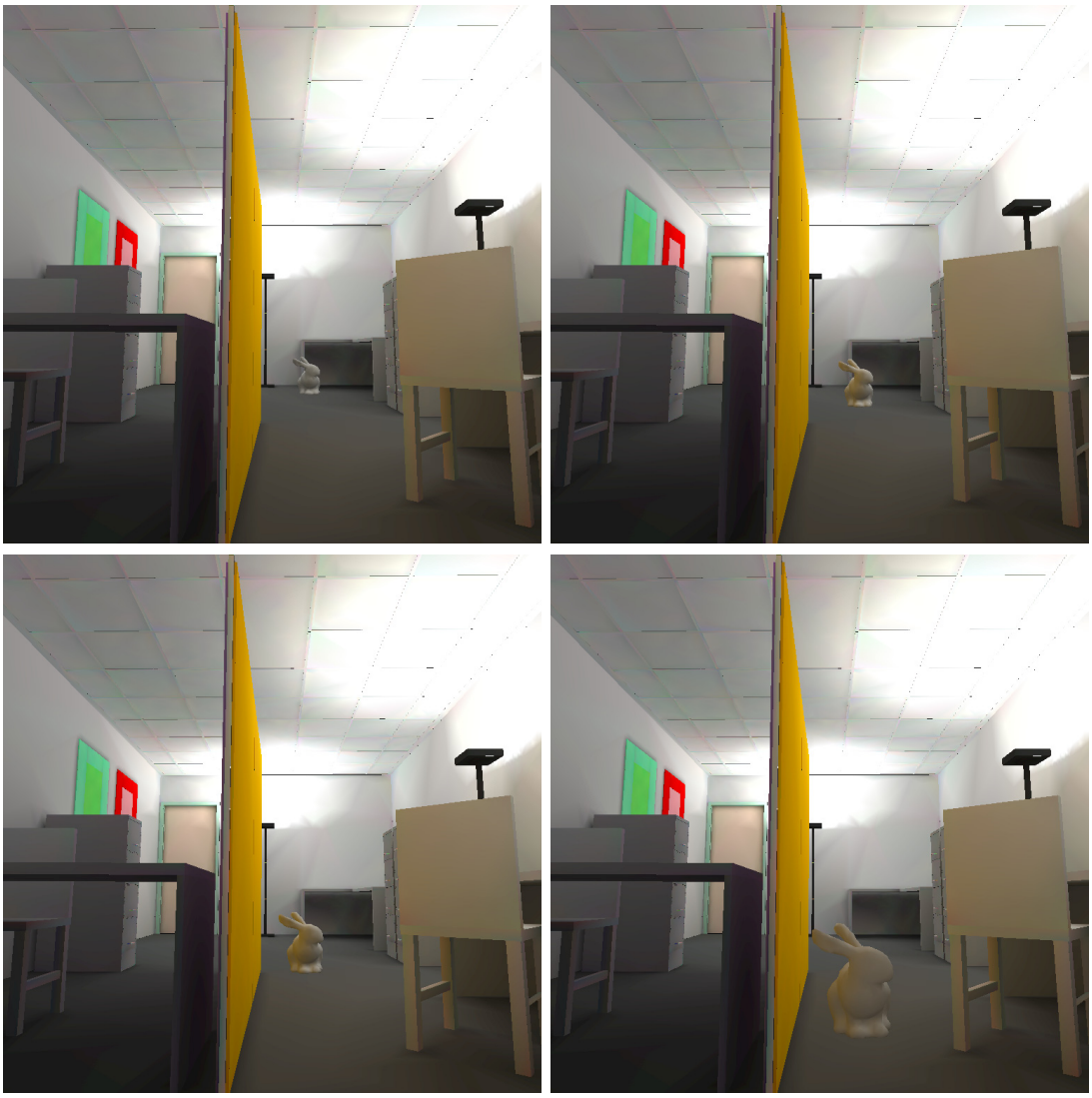


Figure 5.4: *Rabbit moving by divider.*

24 minutes to compute[1] and used about 18 megabytes of main memory for storage. Figure 5.5 shows the structure of the completed volume.

Querying the Volume

Ignoring system factors such as memory paging, querying the volume is a near-constant time operation. Algorithmically, queries will differ at most by a "find which cell contains a point" operation, if the level of hierarchy of the queried cells are not the same (Section 4.3). Finding the correct cell is a relatively inexpensive procedure, resulting in only a small time difference between first- and second-level queries.

Because queries to the volume are well-bounded[2], the use of the volume is well suited to time-critical applications. In addition, the volume supports[3] levels of detail (LOD's) for dynamic objects. While the user is moving an object, the system can draw the lower resolution LOD, and then draw the higher resolution LOD when stationary (Figure 5.6). Finding an effective balance between speed and accuracy, coupled with a scheduling algorithm [32], could enable time-critical rendering. Figure 5.7 shows the rabbit at two levels of detail.

[1] All timings in this chapter were made on a Hewlett Packard 9000-715/100 system with a PA-RISC 7100LC processor (100 Mhz, 121 MIPS, SPECfp92 138.3).

[2] The lower bound on the time it takes to illuminate an object from the volume can be expressed as the *number of queries* $\times$ *time it takes to perform first-level query*. Similarly, the upper bound can be described as *number of queries* $\times$ *time it takes to perform a second-level query*.

[3] Some rendering algorithms attempt to take advantage of object coherence between frames, and run into problems if the object changes. The irradiance volume is meant to be used to completely re-shade an object every frame. This means that dynamic objects can be deleted, added, or modified freely.

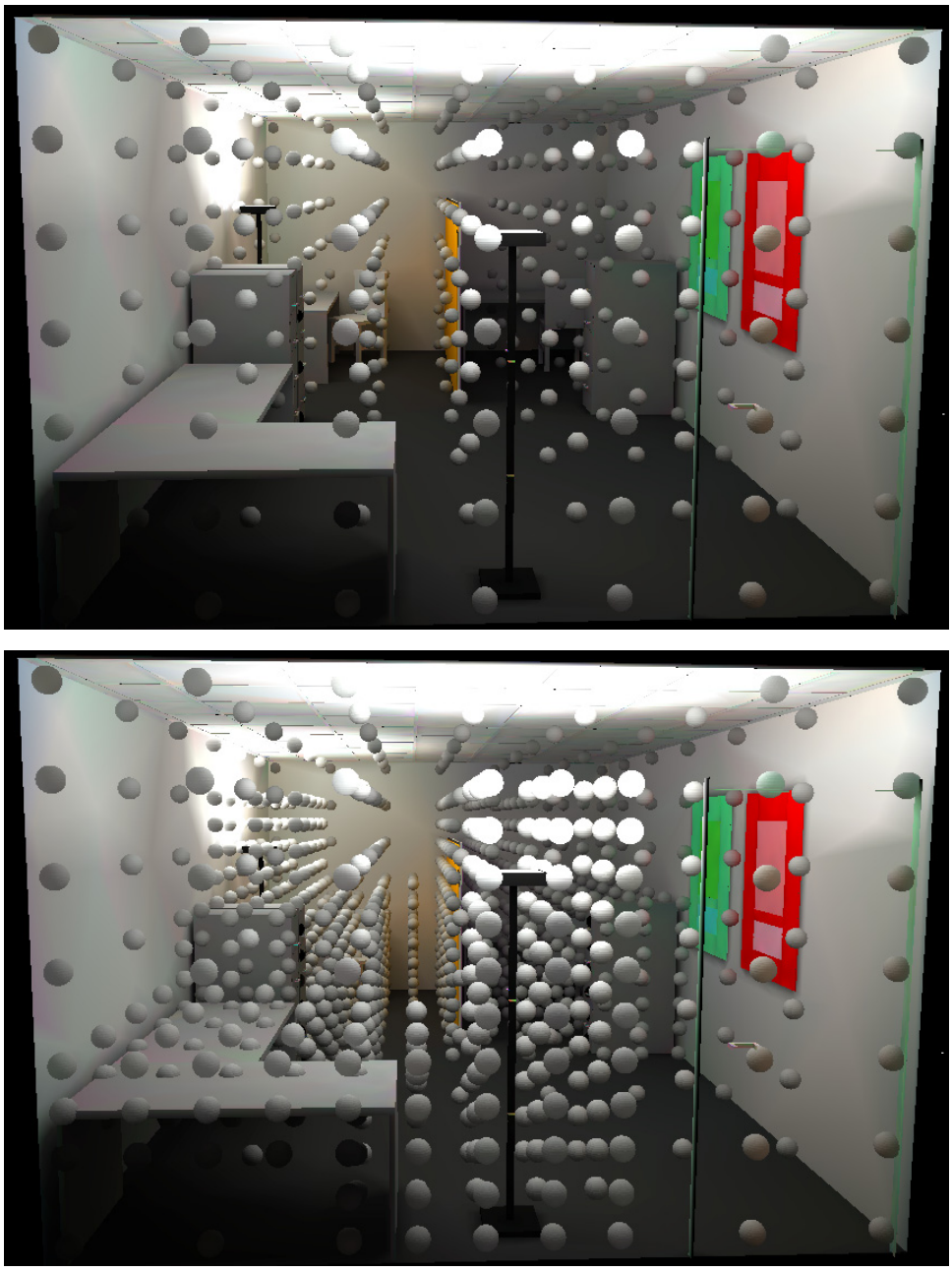


Figure 5.5: *The top image shows the first-level volume grid, and the bottom image the entire bilevel grid.*

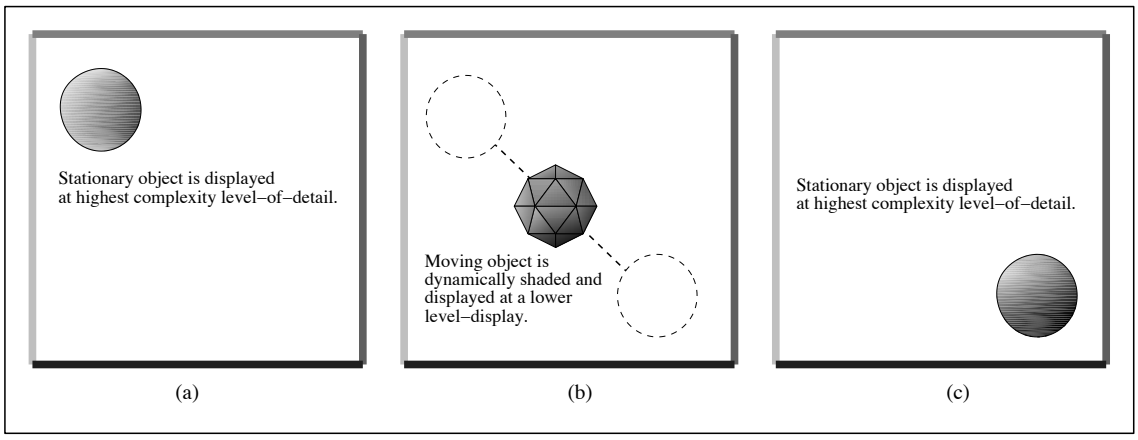


Figure 5.6: *As a dynamic object is manipulated, a less complex level-of-detail model can be used.*

To test how fast our irradiance volume implementation supports queries, a number of queries were made to the volume at random positions and in random directions in the office environment. The total time to perform 500,000 queries was 18.1 seconds, averaging approximately 27,600 queries per second.

Interacting with the Application

In an semi-dynamic application implementation of the kind presented here, three main computational procedures account for the large majority of time needed to compute a display frame. The geometry of the static objects are displayed (in this case the office), the volume is queried to obtain the illumination of the dynamic object or objects (the rabbit), and the dynamic geometry is displayed[4].

[4]Note that we consider the display of the static and dynamic geometry separately. Our implementation uses the OpenGL [15] graphics system, which allows unchanging geometry to be put into a *display list* that greatly reduces overhead. Since the rabbit is dynamic, it must be sent one polygon at a time to the display

Figure 5.7: *Two polygonal resolutions of rabbits.*

The office model contains 43946 polygons and rabbit is comprised of 1162 polygons with 601 shared vertices, resulting in 45108 displayed polygons and 601 volume queries per frame. Using high-end graphics hardware[5], we were able to achieve near real-time interactive rates, averaging approximately 5.3 frames per second.

Since a goal of the implementation is to run at interactive rates, it is useful to do an analysis of the time costs involved to perform the major procedures to determine the bottlenecks in the implementation. We measured the time for the display and query procedures as the rabbit was moved across the model in 120 frames. The results for several polygonal resolutions of rabbits are shown in Figure 5.8. For all

processor, with the associated overhead. Also, the irradiances obtained from the volume must be transformed into display colors, which is a relatively expensive procedure.

[5]An Evans and Sutherland Freedom 3000 series with 14 processors.

resolutions, querying the volume is faster than the shading and displaying of the rabbit.

5.1.2 Comparison with a Full Global-Illumination Solution

Several comparisons were made between a rabbit illuminated from the volume and a rabbit rendered using a standard radiosity method. To obtain an global-illumination solution, the polygonal rabbit model was added to the office environment and rendered using the method described in [55]. A visual comparison[6] was then made on images acquired from the two methods (Figure 5.9). As shown in the figure[7], the volume method provides a believable approximation to the analytical solution.

5.1.3 Display Issues

Self-Occlusion of Dynamic Objects

Significant inaccuracies can result in the shading of object from the volume if it has large areas of self-occlusion. As discussed in the previous chapter, for a given direction ω, the volume approximates the irradiance based upon the gathered radi-

[6] There is not a standard metric for quantifying visual differences between images because there is no accurate model of human vision [70]. Direct visual comparison is currently the most practical method to evaluate images.

[7] Note that the polygonal facets comprising the rabbits shown in the figure are much more apparent then in previous images. In the previous figures, smoother shading was obtained by averaging the normals at vertices shared by several polygons. The radiosity method used to make a comparison did not have the capability to interpolate the illumination between polygons. Because of this, the rabbit from the volume was displayed faceted for comparison purposes. Note that the facets tend to maximize visual error; if both rabbits were displayed with interpolated shading, any differences would likely be less noticeable.

Rabbit Polygons	Total Polygons per Frame	Queries per Frame	Total Time	Model Display %	Rabbit Display %	Query %	Other %	FPS
1162	45108	601	22.4	57.7	29.1	11.8	1.4	5.3
1921	45417	985	29.0	44.7	37.4	16.5	1.4	4.1
3145	46641	1606	42.7	33.4	45.9	19.3	1.4	2.8
4639	48135	2361	50.4	25.5	52.3	20.8	1.4	2.4
6598	50094	3351	66.3	19.6	56.3	22.6	1.5	1.8
(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)

Figure 5.8: *Breakdown of computational costs for our test implementation. Timings were taken over 120 frames with the rabbit in motion. Column (a) shows the polygonal resolution of the rabbit, and Column (b) shows combined number of polygons displayed each frame for the office and rabbit models. Column (c) indicates the number of irradiance volume queries each frame, equal to the number of vertices on each rabbit. The numbers in Column (d) represent the total time in seconds used by the implementation to render 120 frames.*

Columns (e), (f), and (g) show the percentage of the total time used to display the geometry and the rabbit, and to query the volume. The remainder of the time, shown in Column (h), is used by other functions in the program, such as event handling. Column (i) shows the update rates of our implementation in frames per second for each resolution of rabbit.

Figure 5.9: *The images on the left were produced using the volume, and the images on the right were obtained from a radiosity solution. The top row shows the rabbits partly under a desk, the middle row shows them by the wall paintings, and the bottom row shows them in the center of the office.*

Figure 5.10: *Shading inaccuracies caused by self-occlusion. A model of a table has been placed into the office environment. The lower and upper shelves of the table are nearly the same brightness, which is incorrect. The lower shelf of the table should be noticeably darker than the top shelf due to shadowing by the top of the table. Note also the lack of shadows on the floor from the object (Section 5.1.3).*

ance over the entire hemisphere centered around ω. When the irradiance is queried at a point on an object whose hemispherical "view" of the environment is partially blocked by the object itself, the approximation obtained from the volume becomes less accurate (Figure 5.10). The amount of potential inaccuracy caused by self-occlusion depends on how much of the hemisphere above a point on the object is subtended by the object itself.

Shadows

In our implementation, dynamic objects do not cast any shadows on the environment. Calculating shadows from an object due to indirect illumination is a difficult problem and except for very simple cases, cannot be performed quickly enough for use in interactive applications. Several high-end graphic systems[8] support *direct* illumination in hardware and can compute shadows at real-time rates, but only from point light sources. To simulate effects that would be possible with hardware shadowing, we produced a static image, shown in Figure 5.11. In this image the direct illumination is performed by ray-tracing, using classic Whitted-style direct lighting. The indirect lighting on the rabbit comes entirely from the volume; there is no ambient component.

5.2 High Complexity

As rendering algorithms continue to improve and computing hardware becomes faster, increasingly complex environments can be rendered interactively. Currently, for direct lighting algorithms, high-end systems can render scenes with millions of surfaces [3]. Unfortunately, this type of geometric complexity overwhelms traditional view-independent global illumination algorithms such as radiosity. Even though hierarchical and clustering methods have greatly reduced rendering times compared to the original radiosity algorithm, the computational expense is still too

[8] Such as a Silicon Graphics RealityEngine system, which uses *projective texturing* to create shadows [73].

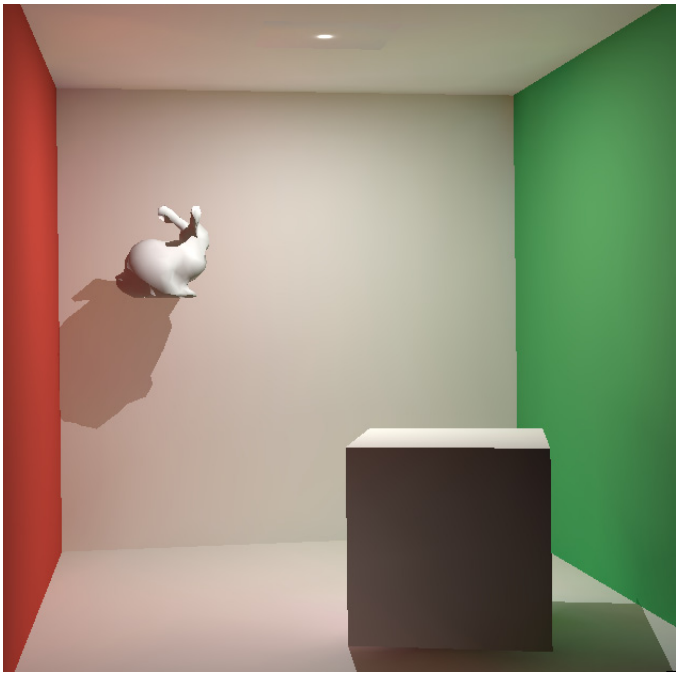


Figure 5.11: *The rabbit is illuminated using ray tracing for the direct lighting and the irradiance volume for the indirect lighting.*

large for environments with millions or even hundreds of thousands of surfaces. In addition, in order to avoid page faulting, these methods require the entire environment to reside in main memory while they are executing, placing an additional practical constraint on the size of the model.

Rushmeier *et al.* [71] discusses the use of *geometric simplification* to accelerate global-illumination renderings of complex environments. In their method, clusters of surfaces are replaced with optically similar boxes before performing the global-illumination calculation. After the global-illumination solution is obtained, it is used in rendering the original geometry.

The irradiance volume can be used in an analogous manner. After a global-illumination solution is obtained with simplified objects, an irradiance volume is built[9]. The simplified geometry is then replaced with the original complex surfaces, which are shaded using the volume. This has the advantage over Rushmeier's method by using fast queries from the volume rather than an expensive gather operation requiring tens or hundreds of traced rays.

To test the effectiveness of the above procedure, we replaced one of the flat walls in the model with a polygonal model of a cinder block wall. An irradiance volume was created from a global solution of the room with the flat wall (Figure 5.1). After the volume was built, each polygon of the cinder block wall was read into memory one at a time and rendered from the volume. The wall consists of approximately 612,000 polygons; not counting disk-access time, it took about 31 seconds to shade

[9]Note that a volume does not have to be built which encompasses an entire environment, but could be built only around an area of interest.

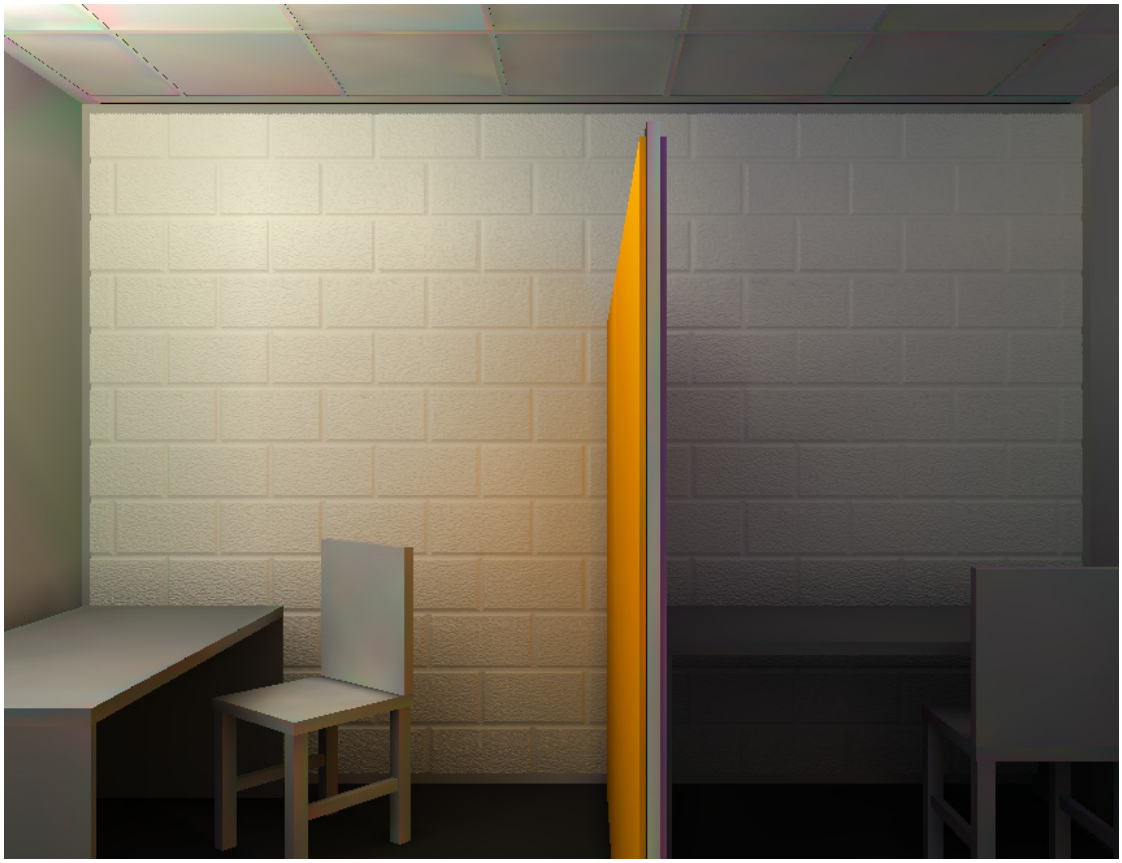


Figure 5.12: *A polygonal model of a cinderblock wall, illuminated from the volume.*

the polygons. Figures 5.12 and 5.13 show two views of the wall. By using this method, arbitrarily complex geometry can be placed in a scene[10]. Note that the polygonal texture, unlike bump-mapping, generates true geometric perturbations and will thus have a correct appearance at oblique viewing angles.

By placing new geometry in a scene after the global-illumination has been com-

[10]Since objects illuminated from the volume do not cast shadows, the geometric simplification method as we use it tends to work best for rendering objects whose lack of a shadow will not be missed. For instance, the lack of self-shadowing on the cinder block wall is not noticeably apparent, but if a different texture with a higher profile was used, the lack of shadows could be significant.

Figure 5.13: *A closer view of the cinderblock wall.*

puted, we are introducing inaccuracies into the illumination solution. However, if we are careful in choosing the simplified geometry, these inaccuracies will be small. A discussion of accuracy issues for geometric simplification can be found in [71].

Rendering Procedural Models

Many rendering algorithms produce complex procedural models while executing. The best known example is the software developed by Pixar [28]. In Pixar's software, geometric primitives are processed independently without reference to others, allowing the rendering of large scenes containing numerous highly-detailed complex objects. During rendering, each surface is composed with optional surface maps[11], and diced into "micropolygons" which are then shaded and scan-converted. Because each primitive is rendered separately, only local illumination techniques are used to shade the micropolygons.

An irradiance volume could be used to approximate the global illumination on each micropolygon as it is created. This method would require an initial rough rendering of a scene using a global-illumination algorithm to get the approximate light flow in the environment. An irradiance volume would then be built from this rough rendering and used to shade the micropolygons created during the final rendering.

[11] Such as texture, bump, and displacement maps.

5.3 Illuminating objects in picture-based environments

As discussed in Section 2.1.4, picture-based methods use sets of pictures of an environment, either rendered or taken through photographic means, to compose a scene. Currently, picture-based methods are only used to render static environments. However, Chen [21] notes that interactive rendered objects can be composited onto a reconstructed view using layering, alpha-blending, or z-buffer[12] techniques. As picture-based methods become more popular, applications containing non-static objects will likely increase in number. This section describes how an irradiance volume might be built and utilized to illuminate objects in an application of this kind.

Volume samples are taken at the same locations, or "nodes", of the plenoptic samples. Actual views of an environment only exist at these nodes; views at other locations than these nodes are either interpolated from the nodes or are ignored[13], depending on the application. If an application does support interpolation between nodes, additional volume samples can be constructed between nodes using an interpolated image panorama. However, using this process only makes sense if the interpolated panoramas result in volume samples that provide better accuracy than simply linearly interpolating between volume samples at the nodes (Figure 5.14).

[12] If the pictures composing an environment are obtained through rendering, depth information can be recorded for each pixel [57]. This is, of course, impractical for photographically captured images.

[13] Some applications constrain the position of the viewer to the node locations.

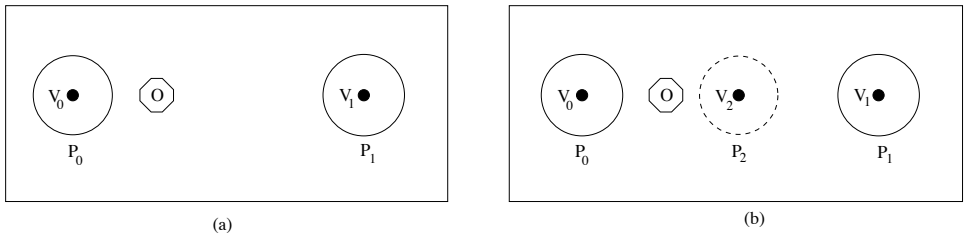


Figure 5.14: *P_0 and P_1 represent panoramic plenoptic samples. P_2 is a plenoptic sample interpolated from P_0 and P_1. V_0, V_1, and V_2 are irradiance volume samples computed from their respective panoramas. In (a), the illumination of object O is linearly interpolated from V_0 and V_1. In (b), the illumination is linearly interpolated from V_0 and V_2. If adding V_2 increases the accuracy of the illumination of O, the (application-dependent) plenoptic interpolation method can be used to calculate volume samples at non-node locations.*

The method used to acquire the radiance, $L(\mathbf{x}, \omega)$, is application-dependent, depending on such factors as the way the image data is stored and the shape of the plenoptic panorama[14]. Once the radiance is sampled, the irradiance is computed using the method described in Section 4.2.3. Querying the irradiance from the volume remains the same.

In order to build an irradiance volume in a picture-based environment, a practical environmental issue which must be resolved is the dynamic range of the pictures. Large dynamic range compressions typically occur when natural scenes are imaged onto film and videotape. In addition, these images may also lose further dynamic

[14]In order to get a true plenoptic sample at a point, data must be gathered over the entire sphere of directions. In practice, cylindrical panoramas are often used instead to avoid problems inherent to spheres, such as the distortion and difficulty of stitching images together at the poles. When sampling radiance from a cylindrical panorama, a default value needs to be assigned to the samples that are taken in directions which point out the top or bottom of the cylinder.

range when they are transferred into a digital format for computer use. Compression of the dynamic range has the effect of causing the irradiance volume to lose intensity detail. The dynamic-range problem may be somewhat alleviated in the future as high-dynamic range cameras are developed. Currently, regions of images corresponding to high-intensity features may have to be numerically enhanced either manually or with some image-processing algorithm.

5.4 Light Transfer Exploration

Rendering researchers and lighting engineers both attempt to understand the flow of light in the environment. Since light is only seen where it reflects from surfaces, and runs invisibly through the environment, it is difficult to understand this light flow. In addition, it is often difficult to understand where the light striking a surface originates.

Because the irradiance volume caches information about light flow in the volume and can be interactively queried, it is useful as a pedagogical tool. For example, by pre-computing separate volumes for the direct and indirect irradiance, an application can allow a user to see the influence of direct lighting, indirect lighting, or combined lighting on an object. Figure 5.15 shows direct and indirect only volumes computed in the environment shown in Figure 4.9.

Before an irradiance volume is built in an environment, a set of surfaces could be selected that are to be ignored while building the volume. During the radiance gathering stage (Section 4.2.2), rays hitting these surfaces return zero or some other

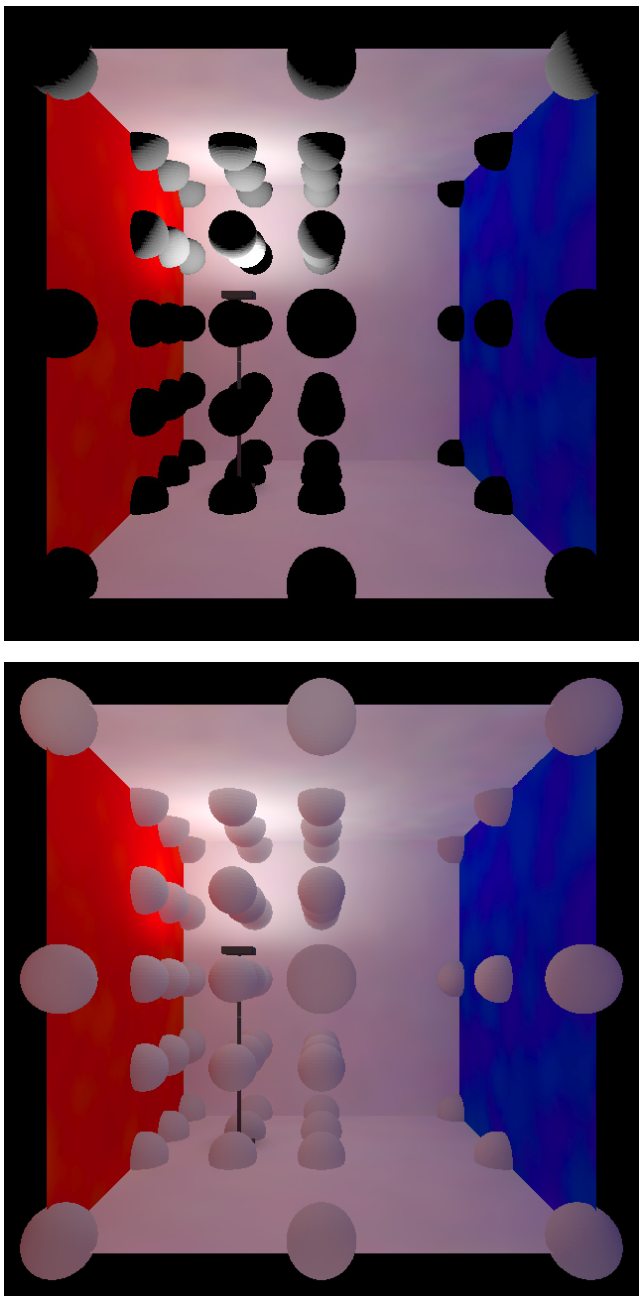


Figure 5.15: *The top image shows a volume computed with direct lighting only and the bottom image a volume computed with indirect lighting only.*

constant for the radiance value. The effects of ignoring the selected surfaces can be seen by building two volumes, one which ignores the chosen surfaces and one which is constructed normally. By switching between the volumes, visual differences of objects lit by the volumes can be compared. This procedure could provide useful information about how various surfaces affect the light flow through an environment.

Another example in which an irradiance volume could be used is an interior design application. Consider the task of furniture placement in an office environment. A model of a desk could be interactively moved around a pre-rendered scene of an office using an application similar to the one discussed in Section 5.1. The approximate irradiance across the surface of the desk could be displayed both in numerical and visual form to the user. The application would warn the user if the desk was moved into areas where it was insufficiently or overly illuminated. When a good location is found, the scene could be rendered with the desk by a separate program to obtain the exact lighting.

Analyzing Global Illumination Algorithms

In order to decrease the computational complexity of many global illumination algorithms, techniques have been developed that approximate portions of the global illumination calculation. Examples of such approximations include clustering methods [80], and patch-and-element radiosity [23]. The irradiance volume could prove useful in analyzing these types of algorithms by enabling the interactive exploration of their resulting spatial characteristics. Several methods could be compared with one another or with a reference irradiance volume.

To compare global illumination algorithms, the irradiance values that comprise an irradiance volume would be computed using those algorithms rather than using the method presented in Section 4.2. For example, to compare two clustering methods, two volumes would be built, with each volume obtaining its irradiance from the environment by using one of the clustering approaches. The two volumes could then be interactively compared by illuminating objects from the volumes or by comparing the irradiance spheres of the volumes themselves. In addition to switching between different volumes for comparison, separate objects in an environment can also be illuminated by different volumes concurrently, giving a useful side-by-side comparison.

Chapter 6

Summary and Conclusion

During the past two decades, computer graphics researchers have developed global illumination algorithms in an attempt to accurately model light-energy transfer in an environment. Even though the realistic effects that these algorithms provide are often spectacular, the computational expense is frequently too great for many applications. This thesis takes a different approach. Instead of striving for greater accuracy at a large computational expense, we present a new method, the *irradiance volume*, which provides a *reasonable approximation with high computational performance.* Thus, global illumination effects can be obtained by applications where the use of global illumination techniques was previously impractical.

Chapter 3 describes the basic radiometric quantities radiance and irradiance. The radiance distribution and irradiance distribution functions are discussed and their behavior is examined. A volumetric approximation to the irradiance distribution function, the irradiance volume, is presented.

Chapter 4 describes how an irradiance volume is built within an environment. Volume sampling strategies are discussed and methods for efficiently obtaining the radiance and approximating the irradiance distribution function are presented. A method of querying the volume is described and the data structures used to represent the volume are outlined. Finally, an extension to the irradiance volume method for non-diffuse surfaces is presented.

Chapter 5 describes how the irradiance volume method might be used in several classes of applications. The volume is shown in use to shade dynamic objects and complex polygonal textures. Methods to use the irradiance volume in picture-based systems are discussed and ways in which the volume might be used for light-flow visualization are described.

6.1 Future Work

6.1.1 Sampling Strategy

One of most important avenues for further exploration is the study of the sampling strategies and rates and how they affect the accuracy of the volumetric approximation. Currently, these rates are chosen interactively until a satisfactory result is obtained.

Spatial Sampling

As discussed in Chapter 3, most of the serious errors in spatial interpolation occur across visibility events. Although using a bilevel grid helps to alleviate this problem

by reducing the size of the areas in which interpolation does not work well, it does not eliminate the problem. The accuracy of our implementation could be improved by some type of three-dimensional discontinuity meshing algorithm but this would require a much more difficult implementation. This would not seem to be a useful approach unless the inaccuracy of simpler data structures proved to be a problem for a particular application.

Instead of sampling on a regular grid, an adaptive sampling scheme might prove useful. A initial regular grid would be produced, and if the eight samples surrounding each cell differed by a specified tolerance, the cell would be subdivided into sub-cells. This process would continue until some stopping criteria is met. However, as with any adaptive sampling strategy, areas with discontinuities may still be missed. It is also not clear what metric should be used to compare two sample spheres.

Directional Sampling

As mentioned in Section 4.2.2 the directional sampling resolution needed to obtain a good approximation is heavily environment-dependent. Since it is impractical to increase the directional sampling rate so that every surface in an environment is sampled, a better method must be used. Some form of informed sampling seems to be the most promising method. Before the volume is built in an environment, light sources and other "important" surfaces could be flagged. The radiance sampling routine would make sure that the flagged surfaces are sampled, ensuring their contribution.

We currently use a constant directional resolution for each point sample in a volume. Higher efficiency would most likely result from sampling at a different resolution at each spatial sample location, depending on some environmental metric at that point.

6.1.2 Querying the Volume

Queries are made to the volume with an associated position and direction. The direction is mapped back to (u, v) space to determine which bin on the sample sphere contains the irradiance value corresponding to that direction (Section 4.3). All query directions which map back to a particular bin will return the same irradiance. Smoother shading on an object could be obtained by interpolating the irradiance between neighboring bins (Figure 6.1). However, performing these interpolations will reduce the performance of volume queries.

6.1.3 Display Issues

Self Occlusion of Shaded Objects

As discussed in Section 5.1.3, objects which have large self-occlusions may be incorrectly shaded from the volume. One possible approximate solution to this problem is to pre-process the objects which are to be shaded. The amount of self-occlusion seen from each vertex of the object could be determined, resulting in a visibility ratio, indicating how much of the hemisphere above that vertex was visible. This ratio could be used in the shading calculation, perhaps as an attenuation factor to darken

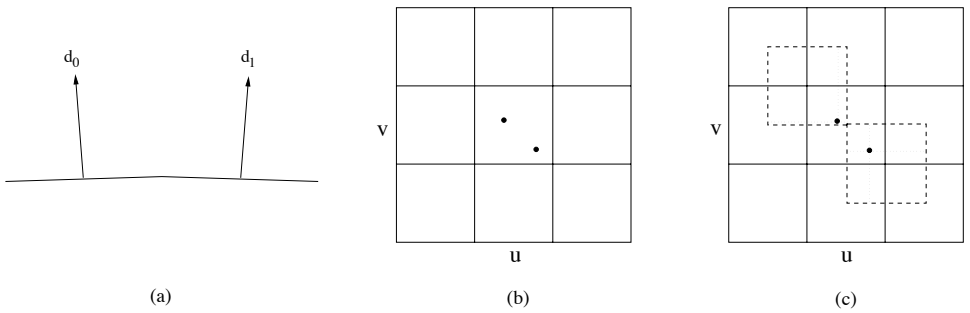


Figure 6.1: *Interpolating between bins for better shading. Consider two query directions, d_0 and d_1, shown in panel (a). Both directions map to (u, v) coordinates contained in the same bin, resulting in the same irradiance value being assigned (panel (b)). At a greater computational cost, bilinear interpolation can be used to interpolate the irradiance from neighboring bins (panel (c)).*

areas in self-shadow. This method, of course, does not account for inter-object light reflection.

Shadows

When shading objects in an environment with the irradiance volume, the lack of shadows cast on the environment by the object may be distracting and detract from the visual quality of the scene. Graphics systems which support hardware shadowing can be used to produce shadows from direct sources. Hardware texture mapping can also be used to create direct shadows at interactive rates. Segal *et al.* [73] describes a texturing method to create shadows from point light sources and Heckbert *et al.* [46] describes a texturing method for producing shadows from area

light sources.

Calculating shadows from indirect sources is difficult because every surface in an environment is a potential source. Typically however, only a small percentage of surfaces in an environment account for noticeable indirect shadowing, e.g. the reflector in a lamp. These few important indirect sources can be reclassified as emitters and thus handled by conventional direct lighting methods.

6.1.4 Irradiance Volume Applications

The volume methods presented are appropriate for applications in which visual appearance is more important than numerical accuracy. The two test applications (Sections 5.1 and 5.2) produced good results. For the irradiance volume to be useful in a wider array of applications, such as a lighting-design system, more studies must be performed on ways to both measure and improve the accuracy of the method.

6.2 Conclusion

This thesis presented an exploratory study on the feasibility and effectiveness of sampling and storing irradiance in a spatial volume. Constructing a volumetric approximation of the irradiance distribution function within a space enabled the approximate irradiance at any point and direction within that space to be quickly queried. This allowed the reconstruction of believable approximations to the illumination in situations which overwhelm traditional global illumination algorithms, such as semi-dynamic environments. The irradiance volume procedures may also

prove useful as a research tool for testing the validity and accuracy of new global illumination algorithms.

Bibliography

[1] E. H. Adelson and J.R. Bergen. The plenoptic function and the elements of early vision. In Michael Landy and J. Anthony Movshon, editors, *Computation Models of Visual Processing*. MIT Press, Cambridge, MA, 1991.

[2] John M. Airey, John H. Rohlf, and Frederick P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics*, 24(2):41–50, March 1990.

[3] Kurt Akeley. Realityengine graphics. *Computer Graphics*, pages 109–116, August 1993. ACM Siggraph '93 Conference Proceedings.

[4] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[5] James Arvo. The irradiance jacobian for partially occluded polyhedral surfaces. *Computer Graphics*, 28(3), July 1994. ACM Siggraph '94 Conference Proceedings.

[6] James Arvo. *Analytic Methods for Simulated Light Transport*. PhD thesis, Yale University, December 1995.

[7] James Arvo and David B. Kirk. Fast ray tracing by ray classification. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 55–64, July 1987.

[8] James Arvo, Kenneth Torrance, and Brian Smits. A framework for the analysis of error in global illumination algorithms. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 75–84. ACM SIGGRAPH, ACM Press, July 1994.

[9] P. Atherton, K. Weiler, and D. Greenberg. Polygon shadow generation. *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3):275–281, August 1978.

[10] L. Aupperle and Pat Hanrahan. A hierarchical illumination algorithm for surfaces with glossy reflection. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 155–162, 1993.

[11] Sig Badt, Jr. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4(3):123–132, September 1988.

[12] Daniel R. Baum, John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. The back-buffer algorithm: An extension of the radiosity method to dynamic environments. *The Visual Computer*, 2(5):298–306, September 1986.

[13] L. D. Bergman, H. Fuchs, E. Grant, and S. Spach. Image rendering by adaptive refinement. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 29–37, August 1986.

[14] James F. Blinn. Models of light reflection for computer synthesized pictures. *Computer Graphics*, 11(2):192–198, July 1977.

[15] OpenGL Architecture Review Board. *OpenGL Reference Manual*. Addison-Wesley Publishing Company, first edition, 1992.

[16] L. S. Brotman and N. I. Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications*, 4(10):71–81, October 1984.

[17] Alan G. Chalmers and Derek J. Paddon. Parallel processing of progressive refinement radiosity methods. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, pages 149–159, New York, 1994. Springer-Verlag.

[18] J. Chapman, T. W. Calvert, and J. Dill. Exploiting temporal coherence in ray tracing. In *Proceedings of Graphics Interface '90*, pages 196–204, May 1990.

[19] J. Chapman, T. W. Calvert, and J. Dill. Spatio-temporal coherence in ray tracing. In *Proceedings of Graphics Interface '91*, pages 101–108, June 1991.

[20] Shenchang Eric Chen. Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 135–144, August 1990.

[21] Shenchang Eric Chen. Quicktime VR - an image-based approach to virtual environment navigation. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, Computer Graphics Proceedings, Annual Conference Series, pages 29–38, July 1995. ACM Siggraph '95 Conference Proceedings.

[22] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, August 1993.

[23] 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.

[24] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 75–84, August 1988.

[25] Michael F. Cohen and Donald P. Greenberg. The Hemi-Cube: A radiosity solution for complex environments. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 31–40, August 1985.

[26] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7–24, January 1982.

[27] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.

[28] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering archtecture. *Computer Graphics*, 21(4):95–102, July 1987. ACM Siggraph '87 Conference Proceedings.

[29] Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (SIGGRAPH '77 Proceedings)*, 11(2):242–248, July 1977.

[30] J. D. Foley, A. van Dam, Steven K. Feiner, and John F. Hughes. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley Publishing Company, second edition, 1990.

[31] David A. Forsyth, Chien Yang, and Kim Teo. Efficient radiosity in dynamic environments. In *Fifth Eurographics Workshop on Rendering*, pages 313–323, Darmstadt, Germany, June 1994.

[32] Thomas A. Funkhouser and Carlo H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics*, pages 247–254, August 1993. ACM Siggraph '93 Conference Proceedings.

[33] David W. George, Francois X. Sillion, and Donald P. Greenberg. Radiosity redistribution for dynamic environments. *IEEE Computer Graphics and Applications*, 10(4):26–34, July 1990.

[34] Reid Gershbein, Peter Schröder, and Pat Hanrahan. Textures and radiosity: Controlling emission and reflection with texture maps. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 51–58. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[35] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[36] Andrew S. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications*, 8(2):60–70, March 1988.

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

[38] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):212–22, July 1984.

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

[40] Ned Greene. Efficient approximation of skylight using depth projections. *Photorealistic Volume Modeling and Rendering Techniques*, 1993. ACM Siggraph '91 Course Notes 27.

[41] Ned Greene, M. Kass, and Gavin Miller. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993.

[42] R. A. Hall and D. P. Greenberg. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications*, 3:10–20, November 1983.

[43] Pat Hanrahan and David Salzman. A rapid hierarchical radiosity algorithm for unoccluded environments. In *Proceedings Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 151–71, Rennes, France, June 1990.

[44] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 197–206, July 1991.

[45] Xiao D. He, Kenneth E. Torrance, Francois X. Sillion, and Donald P. Greenberg. A comprehensive physical model for light reflection. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 175–186, July 1991.

[46] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. In *Submitted to Eurographics Workshop on Rendering*, 1996.

[47] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 133–142, August 1986.

[48] Henrik Wann Jensen. Importance driven path tracing using the photon map. In *Rendering Techniques '95*. Springer-Verlag/Wien, 1995.

[49] David A. Jevans. Object space temporal coherence for ray tracing. In *Proceedings of Graphics Interface '92*, pages 176–183, May 1992.

[50] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 165–174, July 1984.

[51] Douglas S. Kay. Transparency, refraction, and ray tracing for computer synthesized images. Master's thesis, Cornell U., January 1979.

[52] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, August 1986.

[53] Eric Lafortune and Yves D. Willems. A 5d tree to reduce the variance of monte carlo ray tracing. In *Rendering Techniques '95*. Springer-Verlag/Wien, 1995.

[54] Daniel Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(2):25–39, November 1992.

[55] Daniel Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 199–208, 1993.

[56] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 95–102, April 1995.

[57] Nelson Max and Keiichi Ohsaki. Rendering trees from precomputed z-buffer views. In *Rendering Techniques '95 - Proceedings of the Eurographics Workshop*, pages 74–81, Dublin, Ireland, June 1995.

[58] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, Computer Graphics Proceedings, Annual Conference Series, pages 39–46, July 1995. ACM Siggraph '95 Conference Proceedings.

[59] Stefan Müller and Frank Schöffel. Fast radiosity repropagation for interactive virtual environments using a shadow-form-factor-list. In *Fifth Eurographics Workshop on Rendering*, pages 325–342, Darmstadt, Germany, June 1994.

[60] Koichi Murakami and Katsuhiko Hirota. Incremental ray tracing. In *Proceedings Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 15–29, Rennes, France, June 1990.

[61] Jeffrey Nimeroff, Juile Dorsey, and Holly Rushmeier. A framework for global illumination in animated environments. In *Rendering Techniques '95 - Proceedings of the Eurographics Workshop*, pages 92–103, Dublin, Ireland, June 1995.

[62] Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 23–30, July 1985.

[63] James Painter and Kenneth Sloan. Antialiased ray tracing by adaptive progressive refinement. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 281–288, July 1989.

[64] Christopher Patmore. Illumination of dense foliage models. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 63–72. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

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

[66] Rudolph W. Preisendorfer. *Radiative Transfer on Discrete Spaces*. Pergamon, NY, 1965.

[67] Claude Puech, Francois Sillion, and Christophe Vedel. Improving interaction with radiosity-based lighting simulation programs. *Computer Graphics*, 24(2):51–57, March 1990.

[68] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 283–291, July 1987.

[69] Erik Reinhard, Lucas U. Tijssen, and Frederik W. Jansen. Environment mapping for efficient sampling of the diffuse interreflection. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pages 410–422, June 1994.

[70] H. Rushmeier, G. Ward, C. Piatko, P. Sanders, and B. Rust. Comparing real and synthetic images: Some ideas about metrics. In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

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

[72] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 293–302, July 1987.

[73] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul E. Haeberli. Fast shadows and lighting effects using texture mapping. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 249–252, July 1992.

[74] Carlo H. Séquin and Eliot K. Smyrl. Parameterized ray tracing. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 307–314, July 1989.

[75] Erin S. Shaw. Hierarchical radiosity for dynamic environments. Master's thesis, Cornell U., August 1994.

[76] Peter Shirley and Kenneth Chiu. Notes on adaptive quadrature on the hemisphere. Technical Report 411, Department of Computer Science,Indiana University, July 1994.

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

[78] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfer*. Hemisphere, Washington, third edition, 1992.

[79] Francois X. Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 335–344, July 1989.

[80] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 435–442. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[81] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 61–69, July 1991.

[82] K. E. Torrance and E. M. Sparrow. Polarization, directional distribution, and off-specular peak phenomena in light reflected from roughened surfaces. *Journal of Optical Society of America*, 56(7), 1966.

[83] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society of America*, 57(9), 1967.

[84] John R. Wallace. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. Master's thesis, Program of Computer Graphics, Cornell Univ., January 1988. longer version of paper.

[85] Leonard R. Wanger, James A. Ferwerda, and Donald P. Greenberg. Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, 12(3):44–58, May 1992.

[86] Gregory J. Ward. The RADIANCE lighting simulation and rendering system. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 459–472. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[87] Gregory J. Ward. Making global illumination user-friendly. In *Rendering Techniques '95 - Proceedings of the Eurographics Workshop*, pages 104–114, Dublin, Ireland, June 1995.

[88] Gregory J. Ward and Paul Heckbert. Irradiance gradients. *Third Eurographics Workshop on Rendering*, pages 85–98, May 1992.

[89] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 85–92, August 1988.

[90] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984.

[91] T. Whitted. An improved illumination model for shaded display. *Computer Graphics*, 13(3):1–14, August 1979.

[92] Lawrence B. Wolff and David J. Kurlander. Ray tracing with polarization parameters. *IEEE Computer Graphics and Applications*, 10(6):44–55, November 1990.

[93] Andrew Woo. Ray tracing polygons using spatial subdivision. In *Proceedings of Graphics Interface '92*, pages 184–191, May 1992.

[94] Ying Yuan, Tosiyasu L. Kunii, Naota Inamato, and Lining Sun. Gemstone fire: Adaptive dispersive ray tracing of polyhedrons. *The Visual Computer*, 4(5):259–70, November 1988.