DISSERTATION

Photon Tracing for Complex Environments

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften unter der Leitung von

Universitätsprofessor Dr. Werner Purgathofer
Institut 186 für Computergraphik und Algorithmen

eingereicht an der Technischen Universität Wien
Technisch–Naturwissenschaftliche Fakultät

von

Dipl.–Ing. Alexander Wilkie
Matrikelnummer 9026293
Preßgasse 21/8, A–1040 Wien

Wien, am 15. April 2001

This thesis is dedicated to the memory of Dipl.–Ing. Richard Ströll. He was my great–uncle, and he can be said to have started the preference of the Austrian side of my family for becoming engineers.

After studying electrical engineering in the nineteen twenties at the Vienna University of Technology, he initially spent some time at Siemens, but then became one of the many victims of the Great Depression and was unemployed for several years. During the Second World War, he worked for the German aircraft company Heinkel, and was entrusted with the development of various aircraft components. After the war, he was held as a prisoner in the Soviet Union under very bad conditions; when he came back to Austria, his own mother did not recognize him at first. He never fully recovered from this ordeal, and lived from his invalid's pension afterwards.

I received my first mechanical toys as a Christmas present from him when I was three years old, shortly before his rather early death in 1973. I also remember being fascinated by various designs and models of the inventions he made during his retirement, and which I found in a box in the attic years later. These remnants, together with the family's reminiscences of him, substantially influenced my childhood plans to become an engineer myself – which eventually turned out to be just what happened.

Although it is from a field of science that is beyond anything that existed during his active working life, I do hope that he would have liked this thesis for its technical content.

Kurzfassung

Qualitativ hochwertige Bildsyntheseverfahren haben im Allgemeinen das Problem, daß sie bei Anwendung auf komplexe Szenen entweder zu lange Rechenzeiten benötigen, oder daß ihr Speicherplatzbedarf zu groß wird; bei manchen ansonsten durchaus brauchbaren Methoden treten auch beide Probleme zugleich auf.

Die Komplexität einer Szene definiert sich in diesem Zusammenhang normalerweise über die Anzahl der zu ihrer Modellierung benötigten Grundobjekte. Die Ressourcenanforderungen der im Rahmen dieser Arbeit betrachteten photorealistischen Bildsyntheseverfahren steigen mindestens linear mit der Anzahl der beteiligten Objekte, woraus sich ein direkter Zusammenhang zwischen der Szenenkomplexität und dem Laufzeitverhalten der Bilderzeugungsprogramme ergibt.

Daraus leitet sich als erster Ansatz die Forderung ab, möglichst nur solche Modellierungsmethoden zu verwenden, die mit verhältnismäßig wenigen Grundobjekten bereits imstande sind, Szenen ausreichend zu beschreiben. Konkret geht es darum, die in der Computergraphik–Industrie derzeit übliche Darstellungsweise durch große Mengen an Polygonen zu vermeiden, und effizientere Verfahren wie *Constructive Solid Geometry* (kurz CSG) oder regelbasierte Modellierung durch Automaten (wie etwa L–Systeme) zu verwenden.

Diese speichersparenden Methoden zur Szenenbeschreibung haben allerdings den Nachteil, daß sie aus verschiedenen Gründen bislang nicht zusammen mit einer bestimmten Klasse von hochwertigen Bildsyntheseverfahren, den sogenannten *photon tracing methods*, verwendet werden konnten.

Diese Verfahren berechnen eine Lösung der Lichtverteilung in einer Szene durch Nachvollziehung der Lichtausbreitung mittels stochastischer Methoden. Das Ergebnis dieser Simulation wird dann in einem zweiten Arbeitsschritt von einem beliebigen Blickpunkt aus dargestellt. Besonders wenn die Erzeugung ganzer Bildsequenzen – beispielsweise für Animationen – erforderlich ist, erspart man sich durch diese Teilung des Ablaufes in 2 Phasen, von denen die erste pro Szene nur einmal durchgeführt werden muß, einen wesentlichen Aufwand.

Allerdings kann der an sich nützliche Umstand, daß die Verteilung der Lichtenergie in der Szene in irgendeiner Form zwischen den beiden Arbeitsschritten zwischengespeichert werden muß, auch ein Problem darstellen. Das ist dann der Fall, wenn derartige Verfahren auf komplexe Szenen angewandt werden. Bei diesen wird der Speicherplatzbedarf extrem groß, und kann unter Umständen die Verwendung solcher Methoden überhaupt unmöglich machen.

Im Rahmen dieser Arbeit werden nun drei Ansätze aufgezeigt, solche *photon tracers* doch auch für komplexere Umgebungen nutzen zu können. Der erste präsentiert eine Möglichkeit, durch die direkte Verwendung von CSG–Modellen

die Szenenkomplexität an sich im Vorfeld der Berechnungen zu reduzieren.

Der zweite setzt am Problem des Speicheraufwandes für die Lichtverteilung an, und stellt ein Verfahren vor, das für einzelne komplexe Objekte lediglich eine lokale Näherung für die Beleuchtung speichert. Dadurch werden zwar Abweichungen von der Ideallösung erzeugt, aber der dadurch drastisch reduzierte Platzbedarf kann in manchen Fällen die Einbindung komplexer Objekte erst ermöglichen.

Der dritte erweitert das im zweiten Punkt vorgeschlagene Verfahren insoferne, als er die Anwendung dieser Näherungsverfahren auch auf durch Automaten lediglich implizit gegebene Objekte ermöglicht, was eine weitere entscheidende Reduktion des zur Durchführung von Beleuchtungsberechnungen erforderlichen Speicherplatzes möglich macht.

Im Anhang der Arbeit werden einige Aspekte der physikalisch korrekten photorealistischen Bildsynthese abgehandelt. Sie stehen zwar nicht in direktem Zusammenhang mit den im Hauptteil der Arbeit präsentierten Verbesserungen, sind aber insoferne für den Themenbereich relevant, als es einer der Vorteile von *photon tracing algorithms* ist, derartige Effekte in die Bildberechnungen einbeziehen zu können. Die entsprechenden Untersuchungen wurden ebenfalls im Zuge der Arbeit an dieser Dissertation vorgenommen.

Abstract

When used on complex scenes, many high quality image synthesis methods encounter problems. These can either be that their calculation times become unacceptable, or that they would require inordinate amounts of memory. Some – otherwise perfectly useful – methods even suffer from both problems at once.

In this context, scene complexity is directly related to the number of geometric primitives used to model the scene. The resource requirements of the image synthesis methods we consider in this work are linked to the number of involved primitives through an at least linear relationship, which leads to the obvious conclusion that a reduction in the number of primitives needed to model a scene is highly desirable.

An efficient way of meeting this goal is to use scene description methods with higher descriptive power than the current industry standard of polygonal models. Good choices in this area are the approach of *constructive solid geometry* and rule–based techiques such as *L–systems*. However, these memory–saving modeling methods have the disadvantage that so far it has not been possible to use them in conjunction with a particular, highly efficient class of global illumination methods, namely *photon tracing* algorithms.

These techniques calculate the distribution of light in a scene through a stochastic simulation of light propagation. The result of these calculations is stored, and used as a basis for actually displaying the scene in a second step. This two–step approach is highly useful and time–saving if sequences of images – e.g. for animations – are to be rendered, since the first photon tracing step is viewpoint–independent and only has to be performed once for each scene.

However, the useful property of retaining illumination information for later use can turn into a serious problem when complex scenes are considered; in that case the memory requirements can grow to such dimensions that use of photon tracing methods becomes totally unfeasible.

In this thesis we present three approaches which aim at making the use of photon tracers possible even for complex environments. The first technique aims at reducing the overall scene complexity by making the direct use of CSG models for lightmap–based photon tracers possible.

The second addresses the problem of large memory requirements for the storage of illumination information on complex objects by introducing a special type of approximative lightmap which is capable of averaging the indirect illumination on entire objects. While this introduces artifacts, the drastic reduction of needed memory can make the – otherwise impossible – inclusion of complex objects in a photon tracing simulation feasible for the first time.

The third extends the second proposal by making it possible to use these approximative data structures even on implicitly defined objects, such as those spec-

ified by L–systems.

In the appendix we cover several aspects of physically correct predictive rendering. While they are not directly related to the improvements presented in this thesis, they are of interest in this context because the capability of performing global illumination calculations where these effects are taken into consideration is one of the advantages of photon tracing methods. Also, the results which are presented in the appendix were obtained during the work on the main topic of the thesis.

Acknowledgements

I would like to thank Prof. Purgathofer for his comments and ideas regarding this thesis, and for initiating the highly productive environment at the Institute of Computer Graphics and Algorithms which made working there a pleasure.

My special thanks go to Dr. Robert F. Tobler, whose invaluable tuition during the first three quarters of my time at the institute made this work possible.

I would also like to thank The Master Eduard Gröller, Jan Prikryl, Anton L. Fuhrmann, Thomas Theußl and all others at the institute for their valuable scientific input, constructive criticisms and the good working climate.

Special thanks also go to my parents, and to all amongst my family and friends who had to bear with me during the sometimes rather tedious time when I wrote this thesis.

Contents

Chapter 1

Introduction

1.1 Historical notes

Realistic images

Since the time when neolithic painters first depicted animals on the walls of their caves, it has been the desire of artists to capture the appearance of the world that surrounds us in images. While the methods employed by painters may have changed over the centuries, this basic goal remained at least partially valid right up to recent times. Many examples could be given to illustrate the proficiency attained by artists of previous centuries; particularly pleasant ones would be the extremely accurate Dutch naturalists of the 16^{th} century, or the painters of the Italian Renaissance, who, as can be seen from the examples in figures 1.1 and 1.2, developed the capturing of a scene on canvas to a degree of perfection that has few equals.

It has to be noted, however, that even then the realism of an image was not necessarily a means unto itself: although stunningly realistic in every detail, the main objective of paintings like the religious scene in figure 1.2 was arguably not to convey an impression of a real scene in the same sense as a photograph of a historical event might, but to present the viewer with a carefully arranged setting that fits the subject at hand. Titian in particular was a master of using carefully chosen colour and placement of picture elements as a means to unify a composition that expressed a given topic; in a certain wider sense, the realism of the painting was only a by–product of his general artistic intentions.

Beginning with the 19^{th} century, rapid advances in both technology and artistic expression altered the world of pure art so radically that the hitherto tacitly assumed connection between realistic images and artistic expression was permanently lost; a striking example, which at the time marked a culmination of this development, and which dates from the early 20^{th} century, is shown in figure 1.3.

Figure 1.1: Jan Vermeer: The Geographer. 1668-1669, oil on canvas 52 x 45.5 cm, Stadelsches Kunstinstitut, Frankfurt am Main. A prime example of the photographically accurate painting style employed by the Dutch master painters of the 17th century.

But although it was now eschewed by the artists, and made obsolete in some areas through new technologies such as photography, the capability to produce convincingly realistic synthetic images still managed to grow into something which can be considered a technical skill in itself.

Practice of this skill was, apart from the fewer and fewer artists who still believed in truly realistic pictures as a means of expressing themselves, more or less confined to what one would nowadays refer to as visualization problems, mainly in the field of architecture and design. The degree of realism achieved was, in the tradition of academic painting from previous centuries, confined to what the graphic artists could infer through observation of similar scenes in reality and, what usually was the more limiting factor, the effort warranted by the project at hand.

Figure 1.2: Titian: Madonna with saints and members of the Pesaro family. 1519-26; Altar-painting: oil on canvas, 478 x 266 cm; Church of Santa Maria dei Frari, Venice.

New technical advances beyond the high level of realism already attained did not occur; the craftsmanship of those who were now referred to as *illustrators* rather than painters still rested on perspective drawing and well–established heuristics for the appearance of objects and scenes. Predictive images of altogether novel settings were rare, since the tools for their creation were lacking. And there – at least in principle – matters rested for a considerable time.

As in so many other areas of our lives, this situation was substantially altered by the advent of the current so–called information age, and the widespread availability of vast computing power that it brought to an unsuspecting general public. Since computers are by definition calculating machines, it was not long after their introduction before interest arose in using them to predict with mathematical ac-

Figure 1.3: Pablo Picasso: Three Musicians (Musiciens aux masques). Summer 1921, oil on canvas, 200.7 x 222.9 cm, The Museum of Modern Art, New York. Artistic expression beyond any realism in the conventional sense.

curacy the appearance of synthetic scenes. Unlikely contenders as they initially might have seemed, electronic brains (as they were incorrectly dubbed in the early days of computing) have eventually turned out to be a tool that is a highly useful complement to canvas and easel in the hands of modern artists and illustrators alike.

Development of computer-based image synthesis

The necessary theoretical background for computer–generated photorealistic images was available long before even the term was coined, since the properties of the human visual system, and the processes which govern the propagation and interaction of light with our surroundings, were already well understood before the first computer was ever built.

However, this research had primarily aimed at understanding and describing these processes. Due to the complexity of the problem, and the consequently huge mathematical workload involved, no–one had previously put much thought to actually using this knowledge to actively *calculate* what the appearance of a given synthetic scene would be to a human observer.

The comparatively humble calculating power, storage facilities and output possibilities of early machines caused computer graphics initially to evolve only very gradually. The major initial breakthroughs towards realism were the development of key methods such as recursive raytracing, radiosity techniques and the development of practicable realistic shading models; these mainly occurred in the period between 1970 and 1990. Concurrently almost all aspects of light transport in nature were researched; nowadays it is more or less safe to claim that – at least in principle – the problem of generating arbitrarily accurate photorealistic pictures of given scenes is solved.

As with so many other areas of research, the obvious problem with the above statement is, of course, the phrase “in principle”.

1.2 Problem domain of this thesis

As hinted above, the task of generating convincing and correct photorealistic images is, while understood in theory, far from being solved in practice. Certain comparatively simple, and unfortunately also inherently inaccurate, techniques, such as hardware–accelerated rendering and raytracing, have – within their limits – been developed to high degrees of reliability. However, most reasonably sophisticated and realistic techniques (such as radiosity or path tracing) that have been proposed over the past two decades still have problems that severely hinder their widespread acceptance and use in consumer level products. The problems they are prone to are very diverse, and range from unpredictable and/or unacceptable execution times to exorbitant memory requirements when used on nontrivial scenes.

Photon tracing

One of the more promising realistic approaches to calculating convincing images of scenes are the techniques that fall into the broad category of *photon tracing* algorithms. As opposed to ray–based methods such as bidirectional path tracing, and finite element methods such as normal radiosity, photon tracing attempts to reconstruct light transport in a scene through sampling the paths of randomly chosen and propagated simulated light particles. The interactions of these sample particles with their environment are recorded in some way, and later used to generate views of the scene.

The main advantage of this basic idea is that – since it is in effect a physically plausible simulation process – it has no principal restrictions with respect to the physical accuracy obtainable with it. It usually also leads to viewpoint–independent intermediate solutions of the illumination in a scene, from which

multiple views of the same scene can be generated much more quickly than if the underlying simulation had to be performed separately each time.

Although the potential benefits of such an approach are obvious, so are also the drawbacks: for single views of a scene they have to be compared against ray–based approaches which use much less memory while offering similar performance, it is difficult to contain the computational effort to those parts of the scene which actually contribute to the final image (a problem common to all global methods), the potentially very large memory requirements to store the illumination state of the scene (which are usually dependent on the scene size, and make these techniques unfeasible for very detailed and/or extensive settings), and the convergence problems associated with all stochastic methods.

1.3 Purpose and outline of this thesis

Our goal is to improve the ability of certain types of photon–based rendering methods to be used on complex scene descriptions. This is an application area which has been denied to these techniques so far, mainly due to the large memory requirements of the original algorithms.

Before presenting our proposed improvements, we first give a detailed overview of the state of the art in this general area, and of how the main types of photon tracers work in particular. We then outline two enhancements to existing techniques, and demonstrate their viability by testing them in various representative settings. We conclude the thesis by outlining future directions for related research.

Chapter 2

State of the Art in Rendering

In this chapter we briefly state the underlying problem of photorealistic image synthesis, and discuss some of the deterministic algorithms published so far that attempt to solve the problem. Since they are of central importance to our work, the stochastic approaches to the problem are discussed separately in chapter 3.

2.1 The Rendering Equation

In a landmark paper Jim Kajiya published a formal specification of the relationships which govern the process of light transport on a macroscopic scale [Kaj86]. Ultimately, all techiques that attempt to generate realistic images are – in one way or another – based on this *rendering equation*, which models the transfer of light between all the surfaces S in a scene.

$$L(x,\omega) = L_e(x,\omega) + \int_\Omega \rho_{bd}(x,\omega,\omega')L_i(x,\omega')\cos\theta' \, d\omega' \tag{2.1}$$

- $L(x,\omega)$ is the radiance (energy per unit time per unit projected area per unit solid angle) leaving point x in direction ω.
- $L_e(x,\omega)$ is the self emitted radiance at point x leaving in direction ω.
- Ω is the set of all directions ω' covering the hemisphere over point x.
- $\rho_{bd}(x,\omega,\omega')$ is the *bidirectional reflectance distribution function* (in short, *BRDF*) describing the reflective properties of the surface at point x. It is defined as the ratio of the reflected radiance in a given outgoing direction over the incoming flux density in another direction.
- $L_i(x,\omega')$ is the incoming radiance at point x from direction ω'. It is related to the outgoing radiance at the first surface point x' in direction ω' by:

$L_i(x,\omega') = g(x,x') \cdot L(x',-\omega')$, where $g(x,x')$ is a geometry term that equals $1/|x-x'|^2$ if x and x' are mutually visible and 0 if x and x' are occluded from each other.

- $\cos\theta'$ where θ' is the angle between ω' and the surface normal at point x, changes the integral to be over the projected solid angle.

The rendering equation is a Fredholm integro–differential equation of the second kind. Systems of this type cannot be solved analytically, so all global illumination rendering algorithms have to use approximations to the correct solution.

In the following section we present several suitable deterministic approximations that have been developed so far, and in the following section discuss their key properties.

2.2 Raytracing Methods

This large group of methods owes its popular name to a particular rather simple technique, namely raytracing; a more appropriate general term for them would probably be *ray–based methods*. They are originally based on an algorithm introduced by Appel [App68] that traces the path of imaginary light rays from the eye of the observer back into the scene. The point where this path intersects the picture plane gets its colour from the closest of the intersection points with the objects in the scene. Whitted [Whi80] extended this method to reflection and refraction rays, so that images of reflective and transparent objects can be rendered. This was the first recursive raytracing method.

In order to calculate the diffuse propagation of light, distributed raytracing was developed by Cook [CPC84]. A better name – which is actually slowly replacing the old term – for this method is *distribution raytracing*, since the distribution functions for various effects are used by a Monte Carlo integration method to calculate the illumination of a surface point on an object.

Path tracing [Kaj86] is an optimization of distributed raytracing. In each node of the recursive ray tree, only the most important ray (estimated by its influence on the final pixel intensity) is recursively followed. Together with suitable adaptive integration methods this comprises an improvement on the original Monte Carlo algorithm for raytracing.

The advantage of all these methods is the exact calculation of the non–diffuse propagation of light. Therefore reflective and transparent objects and so–called highlights, the reflections of light sources on surfaces, can be easily rendered. The calculation of the diffuse propagation of light is a theoretical possibility, but its prohibitive cost in terms of computation time limits the application of these methods.

For this reason a number of approximations have been introduced that try to speed up the calculation of global illumination with raytracing. In a method based on path tracing, developed by Ward et al. [WRC88], the calculation of the diffuse component of the traced ray is not done for each ray. Calculated illumination values are stored at the object surfaces and re–used for rays that intersect the same surface in the vicinity. The decision if a fresh calculation of the diffuse component has to be done is made by using an analysis of the resulting error.

Recent methods, like bidirectional path tracing [LW93a] try to combine raytracing, which could also be called *backward* raytracing, since the rays are followed from the eye backwards to their origin, with *forward* raytracing. By optimally combining the information gained by forward and backward raytracing [VG94a] [VG95] additional performance increases can be achieved; a particularly promising example of this is Metropolis light transport, which was proposed by Veach [VG97].

Also, Suykens et al. [SW99a] investigated the use of bidirectional path tracing in the context of multipass methods, and also the correct weighting of the various stages of multipass rendering [SW99b].

2.3 Deterministic Radiosity Methods

The classic techniques subsumed under the generic label "radiosity methods" simplify the global illumination problem by dealing only with its diffuse – or Lambertian – part. Under this assumption the radiosity $B(x)$ is the direction independent radiance at a surface point x. The bidirectional reflectance distribution function $\rho_{bd}(x,\omega,\omega')$ in this case reduces to the diffuse reflection factor $\rho(x)$ at point x and can be pulled out of the integral in equation (2.1):

$$B(x) = E(x) + \rho(x) \cdot \int_{\Omega} B(x') \cdot g(x,x') \cdot cos\theta' \, d\omega \tag{2.2}$$

- $B(x)$ denotes the radiosity and $E(x)$ the emission, both at point x. $E(x)$ is obviously only greater than 0 for light sources.
- x' denotes the first point visible from point x in direction ω.

The key property of this simplification is that it makes the precomputation of a viewpoint–independent solution to the light transport problem possible.

The computation of such a solution to the integral (2.2) is greatly facilitated if the scene of interest is not described in terms of continuous surfaces, but rather as a set of polygons (or other simple primitives) that are called *patches* in this context. These patches are in effect a discretization of the scene. If one further

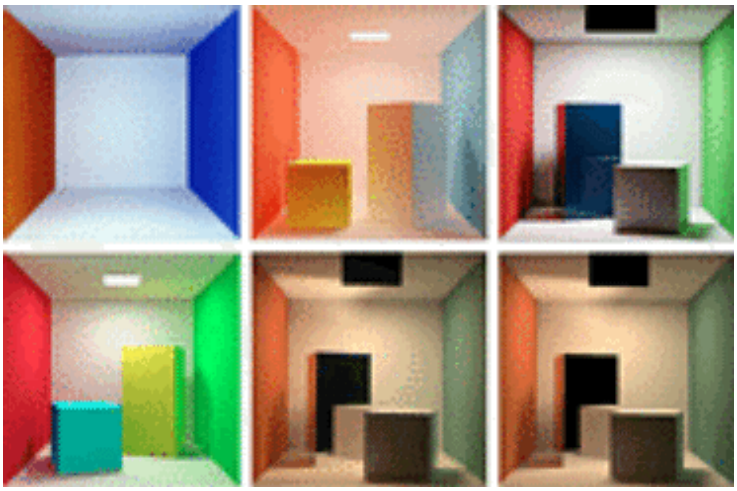


Figure 2.1: The path to the Cornell box. The images range from Cindy Goral's first plain radiosity box, to the final images from that particular program, which one could compared alongside the real objects without noticing any difference. Image by the Cornell Program of Computer Graphics.

assumes constant radiosity B_i over the entire surface of a patch P_i, the formula for the radiosity of a patch turns into:

$$B_i = E_i + \rho_i \cdot \sum_{j=1}^{n} F_{ij} \cdot B_j \tag{2.3}$$

- ρ_i denotes the reflection factor of patch P_i.
- F_{ij} is the *form factor* of patch P_j with respect to patch P_i. This form factor is the cosine-weighted integral of the geometry term $g(x,x')$ for all x on the patch P_i and all x' on patch P_j. It indicates the percentage of the "sky" above patch P_i that is covered by patch P_j; the value of F_{ij} is 0 if patch P_j is totally occluded from the viewpoint of patch P_i. The sum of all F_{ij} is 1 for patches in closed scenes.

Bringing equation (2.3) into the form

$$B_i - \rho_i \cdot \sum_{j=1, j\neq i}^{n} F_{ij} \cdot B_j = E_i \tag{2.4}$$

leads us to a system of equations (2.5) that can be solved for the B_i if the F_{ij} are known. The F_{ii} are always 0, since a patch does not contribute to its own

illumination. Also, the E_i are only $\neq 0$ if the patch in question is a light source.

$$\begin{pmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix} \tag{2.5}$$

Since $\sum F_{ij} \leq 1$ and $\rho_i \leq 1$, the matrix in equation (2.5) is diagonally dominant, and it can be proven that iteration methods like the Gauss-Seidel or Jacobi iterations converge to the true solution of the system. These iteration methods converge quickly for the radiosity equation system and are numerically stable. One step of the iteration

$$B_i^{k+1} = E_i + \rho_i \cdot \sum_{j=1}^{n} B_j^k \cdot F_{ij} \quad (i = 1,2,\ldots,n) \tag{2.6}$$

corresponds to updating the current estimates of *all* radiosities B_i by "gathering" the radiosities of all other patches P_j.

The implementation of this basic approach requires that one solves several intermediate problems, which we briefly discuss in the following subsections.

2.3.1 Form Factor Calculation

In order to be able to solve the equation system (2.5) one has to determine the *form factors* F_{ij}. These factors encode the influence of the patches i and j on each other as a single coefficient, and are a geometrical, illumination–independent property of the scene geometry.

If one assumes patch P_j to have area A_j the projection of P_j to the unit hemisphere above patch P_i has area A_j' (see figure 2.3). If the projection area is small one can assume it to be identical to the projection area on the appropriate tangential plane to the unit hemisphere; this assumption makes mathematical treatment easier. With angle θ_j one obtains the expression

$$A_j' = \frac{A_j \cdot \cos\theta_j}{r^2} \tag{2.7}$$

where r is the distance between patches P_i and P_j. The influence of a patch is proportional to the cosine of the incidence angle (Lambertian reflection); the steeper, the higher the influence. This indicates that the area of A_j' projected to the hemisphere base is the form factor; since this base has area π we get

$$F_{A_j' \to P_i} = F_{ij} = \frac{A_j' \cdot \cos\theta_i}{\pi} \tag{2.8}$$

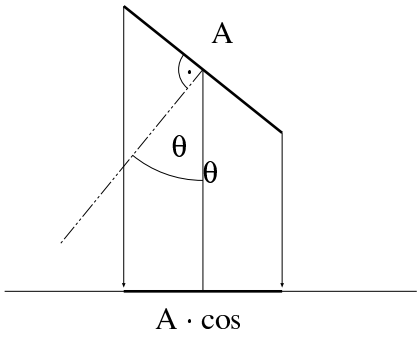


Figure 2.2: Projection of a patch

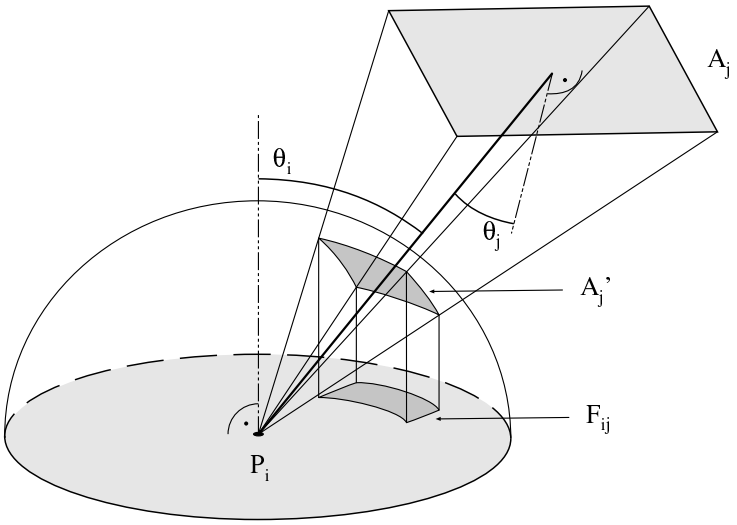


Figure 2.3: Participants in hemisphere form factor calculation

which leads to

$$F_{ij} = \frac{A_j \cdot \cos\theta_j \cdot \cos\theta_i}{r^2\pi} \tag{2.9}$$

as the expression for form factor influence; note that this does not take mutual occlusion between patches into account. Also, it is very costly because of the necessary calculation of 2 cosines and several multiplications.

2.3.2 Hemicube Methods

To address the problems one has with the determination of "genuine" form factors, namely their high computational cost and the difficult integration of mutual visibility information, an approximative method was developed by Cohen et al. [CG85]; one places a unit hemicube over the patch in question. The surrounding

scene is projected onto a *z-buffer* on the surfaces of this hemicube (see figure 2.4); each pixel keeps track of the nearest patch that is visible in its line of sight.

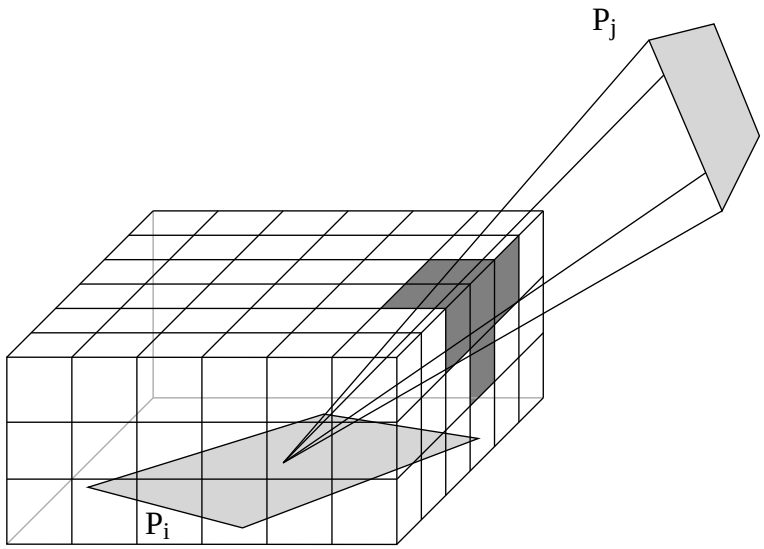


Figure 2.4: Hemicube

To compute a form factor F_{ij} one only has to sum up all P_j pixels, weighted with the pixel form factor, after projecting patch P_j onto the hemicube. Since one can precompute these pixel form factors for each size of hemicube used in the scene there is little runtime penalty to this.

The biggest problem with this approach is that it is virtually impossible to completely eliminate the aliasing artifacts that are caused by the comparatively coarse and scene–insensitive resolution of the hemicube pixel mesh. However, refined versions of the method like e.g. non–uniform meshing of the hemicube in order to take different pixel weights into account can be highly efficient.

2.3.3 Progressive Refinement

The basic method presented at the beginning of the section had a patch *gather* the influence of its surroundings during the iterative solution of the equation system associated with the scene. If one lets a patch *shoot* its contribution into the scene instead, the result is an iteration process that yields intermediate solutions of increasing accuracy. The idea of progressive refinement was first published by Cohen in [CCWG88]. The shooting approach lets the light sources actively distribute their light into the scene. Due to the *reciprocity theorem*

$$A_i \cdot F_{ij} = A_j \cdot F_{ji} \tag{2.10}$$

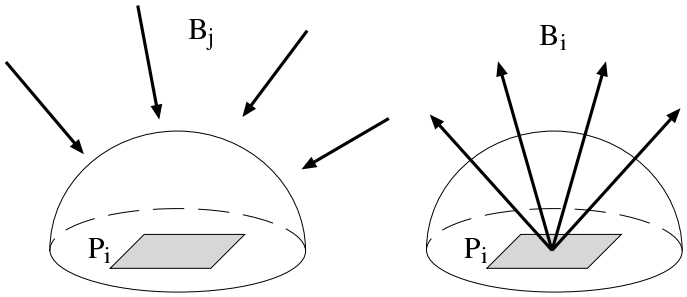


Figure 2.5: Gathering versus shooting

the shooting influence of patch P_i on patch P_j is given by

$$B_{ji} = \rho_j \cdot B_i \cdot F_{ji} = \rho_j \cdot B_i \cdot F_{ij} \cdot \frac{A_i}{A_j} \tag{2.11}$$

One step of the iteration therefore is

$$B_j = B_j + B_{ji} \tag{2.12}$$

for all j. For a performance comparison of gathering and shooting methods see table 2.1. In this table n is the number of polygons in the scene, and (for shooting only) s is the number of necessary iteration steps. Obviously shooting is faster than gathering if the number of iteration steps is smaller than the number of patches. This is very probable for normal radiosity scenes.

To speed up the convergence and the appearance of intermediate solutions one can use several enhancements, two of which are:

- For each iteration step the *brightest* patch (i.e. the one with the highest unshot radiosity) is selected as the shooter, since its potential influence on the scene is greatest.
- The unmodified algorithm leads to many polygons still being completely dark during the first passes of the algorithm. This can be overcome by initializing all polygons with a *default illumination* that is successively decreased as the calculation proceeds (an idea akin to the *ambient factor* mentioned earlier). The effect is that previews of the scene already look plausible much earlier during the simulation process.

2.3.4 Ray Traced Form Factors

One problem which we have not addressed so far is that the extrapolation of the computed overall patch illumination into the patch vertices, which is necessary for

	Gathering	Shooting
Complexity of F_{ij} calculation	O(n^2)	O($n \cdot s$)
Memory for F_{ij}	n^2	$n \cdot s$
Complexity of iteration step	O(n^2) (fewer steps)	O(n) (more steps)

Table 2.1: Comparison of plain gathering and shooting techniques with n patches each, and s iteration steps for shooting.

Gouraud shading of the result, tends to introduce a considerable error. One possible solution is to shoot the energy available to all other patch *vertices*, and to use raytracing to determine the correct visibility (see figure 2.6 for an illustration). To lessen aliasing, one can choose the *shooting point* on surface of P_i stochastically for each ray.

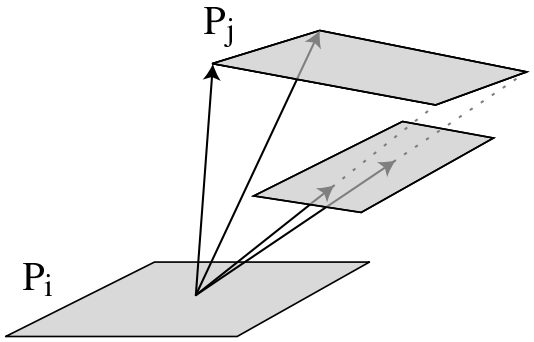


Figure 2.6: Shooting radiosity to patch vertices with patch occlusion

For this to work one needs patch–to–point form factors. When we discussed patch–to–patch shooting, we found equation (2.12) for one iteration step with equation (2.11) for the influence that patch P_i has on patch P_j. This actually is the influence of patch P_i with respect to the *point* at the center of patch P_j.

A good approximation of F_{ji} for small patches is

$$F_{ji} \approx \frac{A_i}{A_i + r^2\pi} \cdot \cos\theta_i \cdot \cos\theta_j \cdot \delta_{ij} \tag{2.13}$$

δ_{ij} is 1 if P_i and P_j have "visual contact", 0 if they are occluded from each other. Since the formula in equation (2.13) turns out to be too inaccurate for larger patches, one resorts to splitting those patches into m smaller pieces, which in turn leads to

$$F_{ji} \approx \sum_{k=1}^{m} \frac{\frac{A_i}{m}}{\frac{A_i}{m} + r^2\pi} \cdot \cos\theta_i \cdot \cos\theta_j \cdot \delta_{ij} \tag{2.14}$$

as an approximate expression for the form factor. The way a patch is split is up to the user; even irregular meshes are possible.

2.3.5 Substructuring

Since the illumination of an object can take any conceivable form, a fixed subdivision of its surfaces into patches unnecessarily constrains the accuracy of the solution. But fixed subdivisions are all one is likely to get from a mesh generator preprocessing step that knows nothing about the radiosity distribution in a scene.

A way to cope with this could be to generate a very dense mesh of all the surfaces in a scene in the hope that the increased accuracy would make further measures unnecessary. This would be wasteful in a number of ways; the worst effect is that, since the complexity of all radiosity calculations is at least $O(n^2)$, every additional patch costs dearly in terms of performance. Also, most of them are usually useless insofar as they contribute little or nothing to the accuracy of the solution.

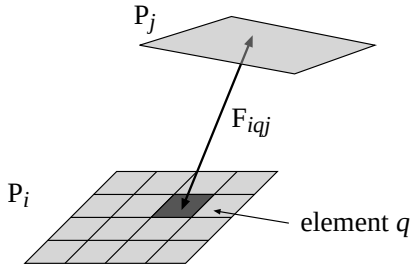


Figure 2.7: Element form factors

A more sensible approach is to retain large patches, but to split these into subsurfaces called *elements* (see figure 2.8). The difference from the "more patches" approach lies in the fact that one solves the radiosity system for the patches only; the elements are used as a better representation of the illumination on the patch.

Initially one computes the form factors F_{iqj} for all elements of a patch; q is the index of the element in question. For a patch P_i with k elements the overall form factor F_{ij} is computed according to

$$F_{ij} = \frac{1}{A_i} \sum_{q=1}^{k} F_{iqj} \cdot A_q \tag{2.15}$$

A conventional radiosity computation that uses elements proceeds as follows:

1. Calculation of the element form factors F_{iqj}

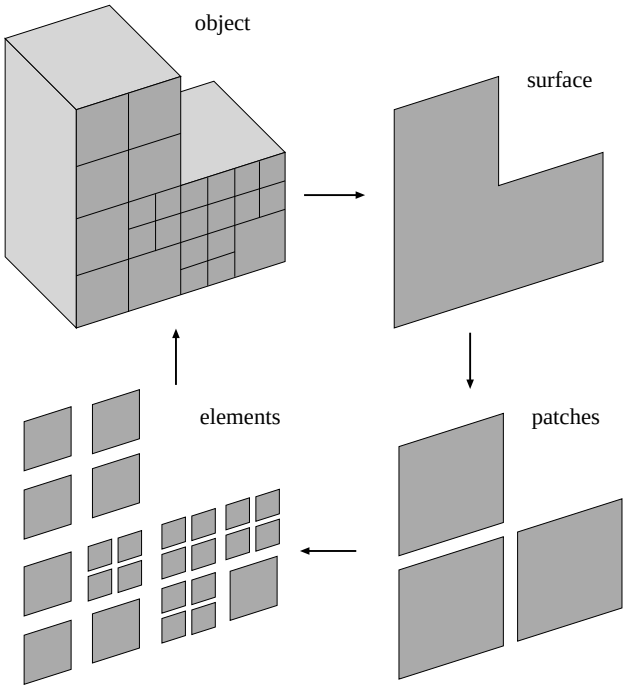


Figure 2.8: Substructuring of an object surface

2. Calculation of the patch form factors F_{ij}

3. Solving of equation system (2.5), in other words obtaining the B_i on the basis of the F_{ij}

4. Calculation of the element radiosities according to

$$B_{ij} = E_i + \rho_i \frac{1}{A_i} \sum_{j=1}^{n} F_{iqj} \cdot B_j \tag{2.16}$$

2.3.6 Hierarchical Radiosity

This approach, proposed by Hanrahan et al. [HSA91], aims at simplifying the determination of patch–to–patch form factors and is related to (and uses) the substructuring technique presented in the previous section. It relies on the fact that, as with the *n*–body problem in physics, it does not make sense for radiosity calculations to resolve all interdependencies between patches individually and with the highest possible accuracy. A sizeable proportion of these form factors is, to all intents and purposes irrelevant, due to both the accuracy limits of machine arithmetic and also diminutive size.

At the heart of the algorithm is a hierarchy of subdivided patches. At the beginning one starts with patches (referred to as *nodes*) that are comparatively large in size, for example whole walls and desktops; they are usually already subdivided to some shallow depth.

As the algorithm determines the interrelations with the rest of the environment for each patch by pairwise comparison with each of the other patches, it considers the error that would be made by using them at their current meshing depth and, if necessary, splits them further. One way of estimating this error is to compare the form factors between the patches; if it is larger than a certain threshold, one should probably rather use a finer subdivision to accommodate any variations in the illumination function. If the *link oracle* decides that one or both nodes in question should be split, the oracle is recursively called for each combination of the resulting subnodes. If the two nodes are usable in their present form they are *linked* together. A link represents a unidirectional relationship between patch elements.

After the form factors have been determined this way, a normal two-step radiosity solving algorithm is applied to the matrix obtained. HR techniques represent the current state of the art for this kind of solution process.

2.3.7 Galerkin Radiosity

Galerkin radiosity was first proposed in [Zat93]. It uses the Galerkin integral equation technique as a mathematical foundation; the main improvement over the previous techniques is that the surface illumination functions are approximated by polynomials, and not assumed to be constant.

For a discussion of this technique it is convenient to formulate the radiosity equation in parametric form, with (s,t) and (u,v) as the surface parameters of P_i and P_j, respectively:

$$B_i(s,t) = E_i(s,t) + \sum_j \iint K_{ij}(s,t,u,v) \cdot B_j(u,v)\, dv\, du \tag{2.17}$$

In this equation, all the complexity of surface interaction - the reflection coefficient $\rho_i(s,t)$, the form factor (including visibility) $F_{dA_i dA_j}(s,t,u,v)$ and the area $A_j(u,v)$ - is abstracted into a single *kernel function* $K_{ij}(s,t,u,v)$:

$$K_{ij}(s,t,u,v) = \rho_i(s,t) \cdot F_{dA_i dA_j}(s,t,u,v) \cdot A_j(u,v) \tag{2.18}$$

The area function $A_j(u,v)$ can be expanded in terms of the surface geometry $\vec{x}_j(u,v)$:

$$A_j(u,v) = \left|\left| \frac{\partial \vec{x}_j(u,v)}{\partial u} \times \frac{\partial \vec{x}_j(u,v)}{\partial v} \right|\right| \tag{2.19}$$

We now define the *inner product* $\langle f|g\rangle_W$ of two functions $f(s,t)$ and $g(s,t)$ with respect to some suitable weighting function $W(s,t)$ as

$$\langle f|g\rangle_W = \int_{-1}^{1}\int_{-1}^{1} f(s,t)\cdot g(s,t)\cdot W(s,t)\,ds\,dt \tag{2.20}$$

for parameters in the interval $[-1,1]$. Galerkin radiosity approximates the radiosity function $B_i(s,t)$ by a weighted sum of orthonormal basis functions $N_k(s,t)$. *Orthonormal* in this context means that the inner product of a function N_k with itself is always 1, and that the inner product of two distinct basis functions is always 0.

In his work Zatz used basis functions defined by the product of two one-dimensional Legendre or Jacobi polynomials in different variables.

Now we can approximate the radiosity function of a surface i using a set of orthonormal basis functions N_k and an associated weighting function W:

$$B_j(s,t) \approx \sum_k B_j^k \cdot N_k(s,t) \tag{2.21}$$

where the coefficients B_i^k are given by the inner product

$$B_i^k = \langle B_i|N_k\rangle_W \tag{2.22}$$

The Galerkin method determines the optimal approximation coefficients B_i^k for a set of orthonormal basis functions with respect to the exact parametric radiosity equation (2.17) and the Galerkin error metric.

The $B_i(s,t)$ term in equation (2.17) can be expanded in terms of the basis $N_l(u,v)$ by using equation (2.21):

$$B_i(s,t) = E_i(s,t) + \sum_j \sum_l \left[B_j^l \cdot \iint K_{ij}(s,t,u,v)\cdot N_l(u,v)\,du\,dv \right] \tag{2.23}$$

Since the coefficients B_j^l do not depend on u and v they can be moved outside the integral. Next we calculate the inner product of both sides with $N_k(s,t)$. By using bilinearity and equation(2.22) we can then write

$$B_i^k = E_i^k + \sum_j \sum_l \left[B_j^l \cdot \left\langle \iint K_{ij}(s,t,u,v)\cdot N_l(u,v)\,du\,dv \middle| N_k(s,t) \right\rangle \right] \tag{2.24}$$

The inner product depends on known information only: the kernel function $K_{ij}(s,t,u,v)$ is determined by the environment, $N_k(s,t)$ and $N_l(u,v)$ are the basis functions of choice. The actual evaluation of the inner product turns out to be

rather complex, since four integrals (two of the explicit, two from the definition of the inner product) have to be solved. This makes it an obvious candidate for numerical methods, e.g. Monte Carlo integration or special quadrature rules that are particularly well suited to the basis functions used [Ger95]. The inner product K_{ij}^{kl} in equation (2.24) is referred to as the *kernel matrix*:

$$K_{ij}^{kl} = \left\langle \int\int K_{ij}(s,t,u,v) \cdot N_l(u,v)\, du\, dv \middle| N_k(s,t) \right\rangle \tag{2.25}$$

With this notation the radiosity coefficients B_i^k can be obtained from the following set of equations:

$$B_i^k = E_i^k + \sum_j \sum_l B_j^l \cdot K_{ij}^{kl} \tag{2.26}$$

The kernel matrix puts radiosity functions on different surfaces into relation. K_{ij}^{kl}, B_i^k and E_i^k correspond to the form factor matrix, the radiosities and the emittance values in "classical" radiosity.

Since the surface indices i, j and the function indices k,l are independent of one another, equation (2.26) is a two-dimensional matrix equation just like the conventional radiosity equation (2.3). For the sake of conciseness and clarity the indices i and j can be replaced by new indices $p = k \cdot n + i$ and $q = l \cdot n + j$ where n denotes the number of surfaces in the scene. Equation (2.26) can then be written as

$$B_p = E_p + \sum_q B_q \cdot K_{pq} \tag{2.27}$$

This set of equations can be solved with any standard technique for large systems of linear equations.

In the paper where he first proposed Galerkin radiosity Zatz investigated basis functions based on Legendre and Jacobi polynomials. The first are easier to compute, but there is a problem with edge singularities near the common edge of mutually visible, adjacent surfaces. There the form factor $F_{dA_i dA_j}$ approaches infinity as a pole of order two because the distance between dA_i and dA_j in the denominator goes towards zero.

There are, however, a number of problems with this approach. The first and major problem of the Galerkin method is that the complexity of calculating the kernel matrix (with its coefficients K_{ij}^{kl}) goes up with $O(k^4)$ if k is the order of the polynomials which are used to represent the radiosity function. In addition to that, higher order polynomials tend to exhibit an artifact called "ringing" which is known from image processing: hard borders between illuminated and non illuminated areas on the same patch will produce ghost borders in the vicinity.

2.3.8 Deterministic Radiosity Methods in Practice

Robustness

Due to the fact that deterministic radiosity methods are by far the oldest existing techniques for solving the global illumination problem, and because of their usually comparatively simple algorithms, their behaviour has been studied in great detail and is very well understood. The resulting robustness and – within their limits – dependability has led to their adoption as an enhancement in numerous commercial software packages, and – apart from the Radiance system by Greg Ward– virtually all global illumination software in production use nowadays uses some variant of the abovementioned approaches.

Realism

When the visual quality of their results is considered, all the techiques presented in the previous section suffer from the same main drawback, namely the initial assumption that all contributions except the perfectly diffuse term are ignored. Scenes which truly exhibit such characterstics are rare in practice; this relegates radiosity methods to those areas where approximative solutions of diffuse versions of a scene are acceptable substitutes. In such areas they perform well, and are quite widely used.

A typical example of this would be architectural lighting simulation software, which is usually not concerned with the generation of truly photorealistic images, but with the delivering of an illumination estimation that – within certain thresholds – can be considered to be reliable.

However, if one wants to include arbitrary surface characteristics, transparent or specular objects and effects like dispersion or polarization in an image, one has to look elsewhere for global illumination techniques that permit their accomodation.

Complexity

Using any of the of the abovementioned methods on highly complex scenes is usually problematic, but manageable. A substantial number of techniques has been published for improving the characteristics of radiosity methods for such scenes.

Rushmeier et al. [RPV93] suggested the use of geometric simplification (GSII) in order to reduce computation time for form–factor based radiosity. They eliminate small, isolated patches and replace clusters of objects with simple, optically equivalent boxes, which are used for the radiosity calculations. Their approach is an extension of a progressive multi–pass rendering method proposed by Chen et

Figure 2.9: This famous scene, which at the time it was created was the most complex radiosity solution obtained up to then, is intended as an example of how convincing a "conventional" radiosity image can look if the setting is right. The fact that no illumination except that between perfectly diffuse surfaces and direct contributions from lightsources are represented, is definitely *not* the first thing that one notices about this scene. Image by the Cornell Program of Computer Graphics.

al. [CRMT91]. The authors develop a theoretical basis in order to determine when the use of GSII is appropriate in a scene.

2.4 Combined Methods

Standard raytracing and radiosity methods have complementary strong and weak points, and therefore a number of combined methods have been developed in order to render images with global diffuse illumination and exact reflection and transparency calculations. In a number of cases these methods use techniques from both raytracing and radiosity methods, and therefore they cannot be easily said to be either of the two.

A lot of these algorithms use a two phase strategy: in the first phase the diffuse interreflection is calculated and in the second phase a raytracing algorithm is used to render the effects arising from non–diffuse reflection. Thus these methods sacrifice the viewpoint–independent solution for a more realistic image from the chosen viewpoint.

An extension of the radiosity algorithm has been proposed by Immel [ICG86]. The non–diffuse light propagation is calculated with a rasterized cube derived from the hemicube that is placed around the center of gravity of each polygon. Each cell on the surface of this cube holds energy values of incoming and outgoing light. The outgoing light energy is calculated using the bidirectional reflection function of the polygon. The second phase consists of gathering the values for outgoing energy for all those directions that point to the viewpoint.

A two pass method based on a similar radiosity algorithm which allows the propagation of diffuse light via perfect mirrors was introduced by Wallace [WCG87]. The second pass, a method derived from raytracing uses a *z*–buffer to integrate the area around the reflection ray or refraction ray. This allows rendering of highlights arising from area light sources.

An improved calculation of the form factors was proposed by Sillion [SP89]: he uses raytracing and circumvents the aliasing–problems of the hemicube. This method is also limited to planar polygons.

Heckbert [Hec90a] uses a two pass method that uses a forward ray tracer to calculate diffuse illumination values in the first phase and store them in a data structure derived from texture mapping. A texture coordinate system is assigned to each surface of each object and a quadtree in this coordinate system is used to store illumination values. Another quadtree is used to assign different weights to illumination rays, in order to adaptively refine the distribution of light energy.

Some of these combined methods make it possible to render effects like caustics: multiple indirect illumination of some objects by other reflecting or transparent objects (e.g. the light pattern at the shadow side of an illuminated drinking glass). A number of these methods completely separate the calculation of ideal diffuse illumination and exactly reflected and transmitted light. General reflection functions, which would allow intermediate kinds of light propagation, are often ignored.

During the last few years it has become standard practice in the radiosity community to use a raytracing backend for the final image generation. A lot of the newer radiosity methods could therefore also be called combined methods, but in most of these methods the final raytracing step just obtains the previously calculated values for diffuse illumination and uses these instead of shooting rays towards each light source.

Chapter 3

Stochastic Global Illumination Methods

There are a number of methods and algorithms which together comprise the basis of the rendering algorithms discussed in this work, and which we eventually aim to apply to more complex scenes than was hitherto possible. The aim of this chapter is to provide the reader with a brief overview of these techniques and related areas.

3.1 Motivation

As can be inferred from section 2.1, the generation of truly realistic images amounts to the solving of the complex integro–differential equation given there. Attempts to solve it analytically are bound to fail, at first glance mainly because the equation expands into an infinite cascade, which in theory transforms the problem into the solution of an equation with infinite dimensionality. The physical counterpart to this mathematical formalism is the propagation of light in a scene via – potentially inifinitely many – interreflections and bounces; each propagation step corresponds to a recursion level in the cascade.

In practice, the fact that light is *not* interreflected infinitely many times reduces the problem to an integral of finite dimensionality. However, the dimension of this integral still remains extremely high even for simple nontrivial scenes. This, together with the usually very complex structure of the integrand, still renders analytical treatment of the problem more or less pointless.

3.2 Monte Carlo Integration

As it turns out, deterministic numeric integration techniques – such as the finite element approach used for conventional radiosity – can only yield crudely approx-

imative solutions to such a problem.

An inherently more elegant approach are stochastic integration methods, which are also referred to as *Monte Carlo* methods. They are proven techniques for solving of high–dimensional integrals, and actually predate the whole field of computer graphics. Early applications for which these approaches were used were neutron transport problems [SG69] and thermal heat transfer [SH81].

Since their first practical use originated in the rather secretive nuclear physics research communities during the cold war era, researchers from both superpowers of the time independently developed very similar techniques. Due to the fact that a sizeable amount of the research that was performed for these programmes is still classified, the exact history of their development is not yet quite clear. Also, the intriguing possibility that substantial improvements on known algorithms still wait to be declassified is not entirely to be discounted, although it is rather improbable.

The rendering techniques which depend on these stochastic approaches can be broadly divided into two main groups, namely those which are used to generate single images *from a particular viewpoint*, and those which – similar to conventional radiosity computations – yield *viewpoint–independent* solutions of the illumination in a scene.

The main advantages and drawbacks of these two types of approach are somewhat complementary, so practical implementations sometimes are hybrid methods which attempt to take the best from both worlds.

3.3 Viewpoint Dependent Rendering Techniques

Since they do not have to rely on any additional data structures apart from the scene description, and they do not impose any restrictions on the nature of light propagation and the objects involved, these are generally the most flexible, and potentially the most powerful rendering methods known.

However, while considerable advances have been made in this area, the main drawbacks of this group are very high computational cost (to the extent of rendering them impracticable for this sole reason), and the possibility of prominently visible variance artifacts in the images. Also, every image has similarly high rendering cost, so these techniques are not well suited for animation purposes.

Although this familiy of techniques is not the focus of this thesis, we will give a brief overview over some important representatives here.

3.3.1 Distribution Raytracing

A naive first approach to stochastic rendering as proposed by Cook et al. [CPC98] casts rays into the scene in exactly the same way as normal raytracing; the key

extension is that at each surface intersection, the ray is propagated according to the probability distribution of the surface BRDF at that point. Also, some kind of stochastic area lightsource sampling is usually employed.

While this version of raytracing sill lacks global illumination, it is already capable of rendering soft shadows and arbitrary surface characteristics. It can also very easily accommodate effects such as depth of field and motion blur. Rendering times are generally high, and the obtained solution can still exhibit variance problems.

3.3.2 Path Tracing

In the same paper in which he formulated the rendering equation [Kaj86], Kajiya also used what was later termed a *path tracer* to compute his results.

The basic idea behind this technique is to cast rays into the scene according to the probability distribution of the involved surfaces in a fashion similar to distribution raytracing. The key difference is that, instead of normal light and shadow ray calculations, a given path is terminated after a random number of surface interactions, and one attempts to connect the last point in the path directly to a lightsource. The outcome of this operation contributes to the pixel in question; typically, dozens of such rays are cast for each pixel in order to gather a useable estimate of the pixel colour. In this way global illumination information is gathered, and all conceivable types of surfaces and objects can be rendered.

However, there are problems, mainly with respect to performance: while this method is unbiased and eventually converges to the correct solution, it is also very inefficient, since most of the paths which are cast into the scene contribute only very little to the final image. Typical reasons for this can be if the final connection to the lightsource is blocked, or the path involves sequences of surfaces with low reflectivity.

Since the most expensive operation in any ray–based renderer is the actual casting and tracing of rays, very long rendering times and visible noise artifacts which are very hard to remove, are typical hallmarks of naive path tracers.

3.3.3 Bidirectional Path Tracing

A first substantial improvement over basic path tracing was *bidirectional* path tracing, which was independently developed by Lafortune and Willems [LW94] [LW93b] and Veach and Guibas [VG95] [VG94b].

Simply put, one of the worst problems with naive path tracers is that for non-trivial geometrical setups with a large number of occluders (especially objects near or around the lightsources, e.g. lampshades), it is comparatively hard to acquire paths which contribute significantly to the pixel estimate when all one is doing is

to cast rays which start at the eyepoint. As mentioned before, for each useful path a large number of paths with blocked light connection are cast in such a case.

This behaviour can be significantly improved if the raycasting scheme is altered to trace *two* stochastically distributed rays into the scene, one starting at the eyepoint (as in a normal path tracer), and the second one starting at one of the lightsources. At each intersection, the lightsource contributions are evaluated. The final connection step is then made between the endpoints of these two paths; in most typical scenes this greatly increases the chance of obtaining a useful path.

While this approach does not limit the generality of the path tracing concept, and significantly reduces rendering times, bidirectional path tracers still need very long rendering times, and are also still prone to problems concerning visible variance artifacts.

3.3.4 Metropolis Light Transport

With the introduction of *Metropolis light transport* [VG97], Veach and Guibas proposed a highly sophisticated variant of path tracing which for certain scenes offers large performance gains over normal bidirectional path tracing.

Since raycasting operations are the most computationally expensive part of a rendering system, any reduction in the number of traced paths is a desirable goal. More specifically, even with a bidirectional path tracer it is still highly desirable to further reduce the number of those paths which do not contribute to the final image.

The key idea of Metropolis light transport is to *re–use* existing paths which have already proven to be valid connections. This has to be done in such a way that the simulation is still unbiased, which is basically achieved by applying *permutations in path space* (also referred to as *mutations*) to existing paths. Colloquially speaking this can be described as trying to get the most out of a path that has already proven to be useful, which is done by changing it “a little bit” – as much as one can get away with without breaking the simulation process.

The art behind doing this in practice is that any deterministic choice of mutated paths would introduce a bias into the Monte Carlo process, so the actual mutation strategies have to be chosen very carefully.

The basic strategy are *bidirectional mutations*, which are responsible for making large changes to the path. A given path segment can be replaced by an entirely different subpath; amongst other things this can also alter the length of the path. *Perturbations*, the act of changing the direction of a path by small amounts, are very useful for the rendering of caustics. The final example which is given by Veach and Guibas are *lens subpath mutations*, which serve to stratify the samples over the image plane; here only the lens subpath of a given path is altered.

Recently, investigations by Szirmay–Kalos et al. [SKDP99] have shed more light on the general properties of MLT. The fact that one has let the simulation process run for some time before it has "settled down" enough for mutation strategies to be applicable introduces a start–up bias that can adversely affect the performance for some types of scenes.

3.4 Viewpoint Independent Rendering Techniques

Techniques from this second group are generally capable of generating image sequences of a given scene – e.g. for animations – with comparatively little additional effort, but always include a – potentially very time–consuming – initial processing step, and also have considerable additional storage requirements beyond the space needed for the basic scene description. Due to the fact that the obtained illumination solution has to be stored in some form, these approaches are also prone to any artifacts which this storage process implies.

Since we propose to extend precisely such methods for operation on complex scenes in the remainder of this thesis, we describe them in more detail than the techniques from the first group.

3.4.1 Basic Monte Carlo Radiosity

Contrary to the inherently limited finite element approach of conventional radiosity methods discussed in section 2.3, the term "Monte Carlo radiosity" usually denotes a physically plausible simulation of the energy transfer in a scene. Rays are stochastically shot from emitting patches into the environment, and each ray represents one photon or light ray that carries a portion of the total emission in the scene. This portion is transferred to the nearest patch hit by the ray. The actual weight by which the photon is counted at this patch is dependent on the way the energy is further propagated by the algorithm.

Although the basic idea behind them is similar, several different flavours of Monte Carlo radiosity methods with sometimes considerably differing performance characteristics exist. The discrimination criteria are:

- *the simulation of photons*: some algorithms simulate the complete path of a single photon, with reflections and refractions, before going on to the next photon. Other algorithms carry each photon only from one surface to the next, store the energy at the receiving surface, and continue the simulation from there in the next iteration step.

- *the simulation of the photon interaction with each surface*: it is possible to continue the path of a photon based on the probability given by BRDF

if the surface, or photons with partial energy can be propagated after the interaction with a surface. The former are sometimes referred to as *Russian roulette* methods, while the latter are known as *fractional* photon tracers.

There are a number of additional schemes which are theoretically possible; common to them all is that they need to count the energy deposited at each surface in a different way in order to obtain an unbiased estimate.

The most important stochastic radiosity algorithms can all be derived from one of the following original algorithms:

3.4.2 Shirley's Algorithm [Shi90]

This algorithm is based on Progressive Refinement Radiosity, but the form factors for patches are approximated by stochastic shooting of rays between them.

3.4.3 Particle Tracing

In this method proposed by Pattanaik and Mudur [PM92] rays representing particles (also referred to as *photons*) are emitted (or *shot*) from the light sources; the number shot is proportional to the contribution of the emitter to the total emission in the scene. This ensures that every particle carries a similar amount of *flux energy* Φ. The ray origins for these *photons* are normally uniformly distributed over

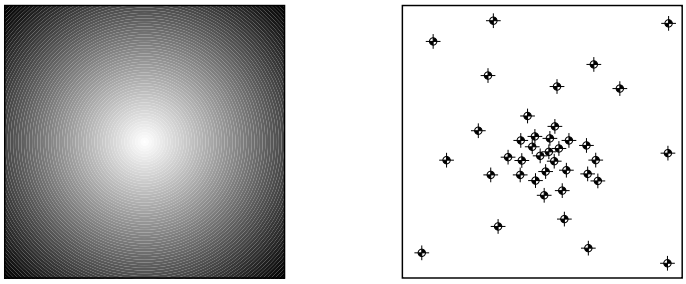


Figure 3.1: Relation between a non-uniform emission function and the starting locations for rays shot from a patch

the surface of the emitting patch. It is also possible to model varying brightness of light sources by applying an appropriate weighting function to the starting locations (see illustration 3.1); areas where many photons originate are brighter than those with low particle emittance (note that this is only possible when particles have the same weight). Once under way, the particles travelling through the environment mimic the behaviour of real light particles insofar as they are diffusely or specularly reflected, refracted and finally absorbed in interaction with surfaces.

The decision *which* interaction is to be taken is made stochastically according to the BRDF of the surfaces. This means that particles travel to their eventual destination without their *content* (i.e. the "energy packet" containing radiosity information) being split (although their content is modified to accommodate effects like colour bleeding). Any changes in direction are modelled through the *probability* of the corresponding interaction.

3.4.4 The Stochastic Ray Method

In this method introduced by Neumann et al. [NPT+95] an iteration step consists of stochastically distributing the energy of all patches simultaneously through the environment by only one reflection step. An average over many such iterations yields an almost converged solution; however, due to memory and efficiency constraints it is virtually impossible to keep track of all steps. Neumann therefore proposed an approach where the result so far and the outcome of the last shooting step are combined in such a way that the importance of the latter decreases as the simulation proceeds. Expressed with B^i as the solution in step i, R_s^n as the shooting result for step n, and τ_i as the weighting factor for step i ,this reads

$$B^n = \tau_n B^{n-1} + (1-\tau_n) R_s^n \tag{3.1}$$

A suitable sequence for the weighting factor τ is for instance the harmonic series $\frac{1}{n}$. One drawback of this method is that the illumination energy spreads very slowly through the scene; this can be overcome by adding a preprocessing step that scatters large portions of the initial energy through the scene. Shirley [Shi91] shows that the number of rays needed to compute a solution of given accuracy with Monte Carlo radiosity is linear to the number of patches in the scene, as long as all patches in the scene are of similar size. The accuracy of Monte Carlo solutions is measured by the variance of the radiosity in the scene. All Monte Carlo type algorithms (not only the ones for radiosity), share the property that the variance of a solution has the complexity $O(1/\sqrt{N})$, for N being the number of samples taken (or in Monte Carlo radiosity, the number of rays shot).

3.4.5 Stochastic Galerkin Radiosity

In most stochastic radiosity methods, one of the problems that have to be solved is the reconstruction of the radiosity function over a surface based on a set of random samples across this surface. The samples can be viewed as particles or photons that have reached the surface at the current stage of the simulation, and the *photon density function* encodes the radiosity function. Thus the reconstruction of the radiosity function can be viewed as a density-estimation problem. Shirley et al.

[SWH+95] proposed a solution which is based on the idea of splatting, i.e. the photons leave their energy not only in the form of delta-spikes, but in the form of a kernel function that describes the deposition of the photon's energy on the surface.

A completely different approach can be derived from the Galerkin method [Zat93]. Feda [Fed96] has shown that the Galerkin method can be adapted to stochastic methods using a Monte Carlo evaluation of the kernel integral. This section will give a short description of the method.

The radiosity function $B_j(u,v)$ of a surface j can be projected into the space of the basis functions $\{N_k\}$ by calculating the corresponding coefficients B_j^k, which are given by the corresponding inner products. If the basis functions are defined over the parameter interval $[a,b] \times [c,d]$ the inner products are given by:

$$\langle B_j | N_k \rangle_W = \int_a^b \int_c^d B_j(u,v) \cdot N_k(u,v) \cdot W(u,v)\, dv\, du \tag{3.2}$$

This integral can be solved numerically using Monte Carlo quadrature in the following way:

$$\langle B_j | N_k \rangle_W \approx \frac{(b-a)\cdot(d-c)}{m} \sum_{i=1}^{m} B_j(\xi_i) \cdot N_k(\xi_i) \cdot W(\xi_i) \tag{3.3}$$

where the ξ_i are uniformly distributed over the parameter interval. This is a very simple, and therefore very inefficient type of Monte Carlo integration.

This equation cannot be directly used to calculate the coefficients for the basis function, since the radiosity function $B_j(\xi_i)$ is not known in advance. The density of the sampling points, as provided by the radiosity simulation, corresponds to the radiosity function. This fact can be used to modify the method: equation (3.3) is based on a uniform sampling strategy. It is, however, very easily changed to non-uniform sampling, by introducing a probability density function (*PDF*) f_j. For non-uniform sampling we therefore get:

$$\langle B_j | N_k \rangle_W \approx \frac{1}{m_j} \sum_{i=1}^{m_j} \frac{B_j(\xi_i) \cdot N_k(\xi_i) \cdot W(\xi_i)}{f_j(\xi_i)} \tag{3.4}$$

where the samples ξ_i are distributed according to f_j, for short: $\xi_i \sim f_j$. The energy flux function is transformed to the *PDF* f_j by normalization:

$$f_j(u,v) = \frac{B_j(u,v) \cdot A_j(u,v)}{\int_a^b \int_c^d B_j(s,t) \cdot A_j(s,t)\, dt\, ds} \approx \frac{B_j(u,v) \cdot A_j(u,v)}{m_j \cdot \Phi} \tag{3.5}$$

Here m_j is the number of photons that are reflected and emitted on the surface j, and Φ is the energy flux carried by each photon. Note that the denominator of

the first formulation of equation (3.5) is the total outgoing energy, which can be written as the energy carried by all outgoing photons. Combining equations (3.4) and (3.5) we find that the inner product can be evaluated by

$$\langle B_j | N_k \rangle_W \approx \Phi \cdot \sum_{i=1}^{m_j} \frac{N_k(\xi_i) \cdot W(\xi_i)}{A_j(\xi_i)} \qquad \text{where } \xi_i \sim f_j \tag{3.6}$$

The stochastic evaluation of the Galerkin integral is therefore reduced to a weighted sum of the basis functions evaluated at the points where photons are emitted or reflected.

3.4.6 The Global Lines Method

Something that is common to all previously mentioned techniques is that they can be considered *local* algorithms insofar as the propagation of photons along their paths is primarily dependent on the local geometry encountered by the particle during its travels.

Mateu Sbert et al. [SPNP96] proposed an entirely different approach that – instead of tracing photon paths from the lightsources – casts *global lines* through a scene; these traverse the scene from end to end, and all intersections with any object they encounter is recorded. All mutual light transfers are then performed between the scene intersection points that lie on these randomly chosen paths.

While the global lines technique is quite efficient by itself, it performs best when it is preceded by a shooting step where the lightsource energy is initially distributed through the scene.

Although the performance of the algorithm is quite impressive, it does suffer from a few drawbacks, such as that in its original form it is only applicable to diffuse scenes, and that the most efficient way to cast rays in arbitrary scenes is not quite clear.

3.5 Methods of Recording Photon Hits

Apart from the exact type of Monte Carlo radiosity algorithm used, the second key characteristic of any implementation which is based on the techniques outlined in the previous section is how the illumination function of the scene in question is stored – and reconstructed – from the data gathered during the simulation.

There are two basic options for storing the interactions that occur during a photon tracing simulation. The first records every single interaction separately; this preserves all the information gained during the simulation (and hence – when done properly – ensures maximal fidelity of the results), but needs large amounts

of storage space and has potential performance problems during the reconstruction process.

The second possibility is to use a discretized functional representation of the illumination function on surfaces or in volumes, and to update this representation according to the gathered photon hits. The main advantages here are the controllable amount of storage used for the illumination representation, and the somewhat better predictability of when a convergent solution has been obtained. In the following sections we describe the two approaches in more detail.

3.5.1 Photon Maps

The concept of *photon maps*, where the incidences of the individual photons on surfaces in the scene are stored in a global data structure with all their information – such as incoming direction, location and intensity – was first proposed by Jensen [Jen96, Jen97].

While the initial recording of the photon hits is a conceptually simple task – albeit one with the minor disadvantage of needing huge amounts of memory if large numbers of photons are traced – the challenging part of this technique is the reconstruction of the illumination function from these randomly placed point samples of the illumination function.

In order to compute the radiance L_r at a given point x in the scene, one basically has to locate the n photons nearest to x and add up their influences. While this would be next to impossible if a brute force approach were used, the search for these n photons can be done efficiently if the photons are stored in a balanced kD–tree. The balancing of the kD–tree, which can be quite computationally intensive, has to be done only once – after the photon tracing pass, which originally leaves totally unordered photon hits in a global list, is over.

Performance Considerations

A key advantage of photon maps over the lightmap approach discussed in the next section is that, since photon hits are stored in global coordinates, the method is theoretically totally independent of the object representations used for the scene. Specifically, as pointed out by Jensen [JC98], photon maps naturally extend to cover participating media, and require no properties of the geometric primitives in a scene other than that they can be intersected or traversed by a ray.

In practice, photon maps are not used alone during the rendering pass; for specular and highly glossy surfaces Monte Carlo sampling is used, and direct illumination is calculated using shadow rays. Active density control during photon map acquisition was demonstrated by Suykens [SW00].

Usability in Complex Scenes

Jensen [Jen97] points out that his method, while it is (unlike most other photon tracing methods) not particularly prone to variance problems due to undersampling, has the disadvantage that it tends to use large amounts of memory even for comparatively small scenes (cf. the memory usage statistics given in the paper), making it a less than optimal choice for complex scenes.

3.5.2 Photon Tracing using Lightmaps

The second major approach to storing the samples gathered during the shooting pass is to use *photon lightmaps* that cover the faces of the geometric primitives in the scene (see figure 3.2) and store the energy deposited by photon hits there.

We use the term "photon lightmaps" to reflect the usage of these data structures in a photon tracing environment; the concept of such radiosity textures was introduced in a bidirectional raytracing context under the name of *Rexes* by Heckbert in [Hec90b]. While the data structures that we use for storing irradiance are similar, we employ a different, more efficient method of determining the actual indirect illumination. Any kind of functional representation can be used for the recording

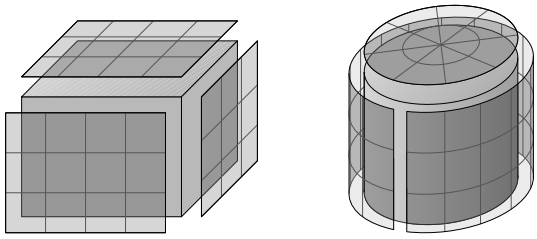


Figure 3.2: Photon lightmaps applied to the faces of basic geometric primitives.

of photon hits on these lightmaps; although higher–order bases have been used (e.g. by Zatz [Zat93]; the problem is also discussed by Bekaert [Bek99]), the most commonly implemented solutions use constant and bilinear representations (see figure 3.3 for a comparison). This is mainly because the number of hits needed for a stable estimate rises sharply with the order of representation, but also because issues such as how exactly a high–quality interpolation of the lightmap elements can be performed, turn out to be unexpectedly thorny for higher order representations. Photon lightmaps can be applied to the faces of basic geometric primitives much in the same way as texture maps, as shown in figure 3.2. While the "texels" on these lightmaps are separate (patch–like) "buckets" into which photons fall, the real object geometry is used during the simulation and the final rendering pass. Also, the whole lightmap is considered a coherent entity (as opposed to a

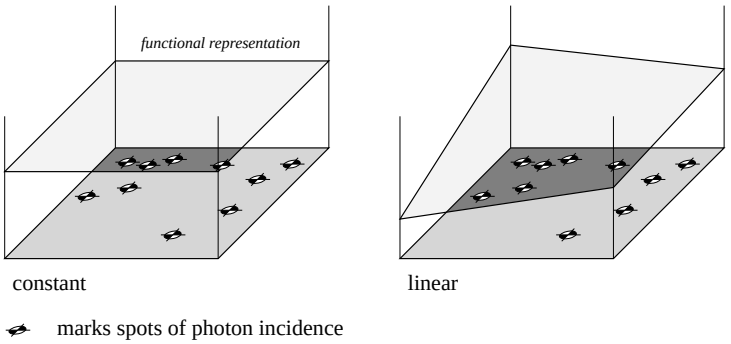


Figure 3.3: Constant and linear representation of the same set of photon hits on a surface. Higher–order representations work in a similar way.

sum of independent patches): a fact that is highly useful for interpolation purposes during the final rendering pass.

It has to be noted that the regular subdivision of the lightmaps shown in figure 3.2 is in no way mandatory; one can use any meshing and/or an adaptive hierarchical approach, as originally demonstrated by Heckbert [Hec90b] or (adapted for photon tracing) by Tobler et al. [TWFP97] for this task. The radiosity of a

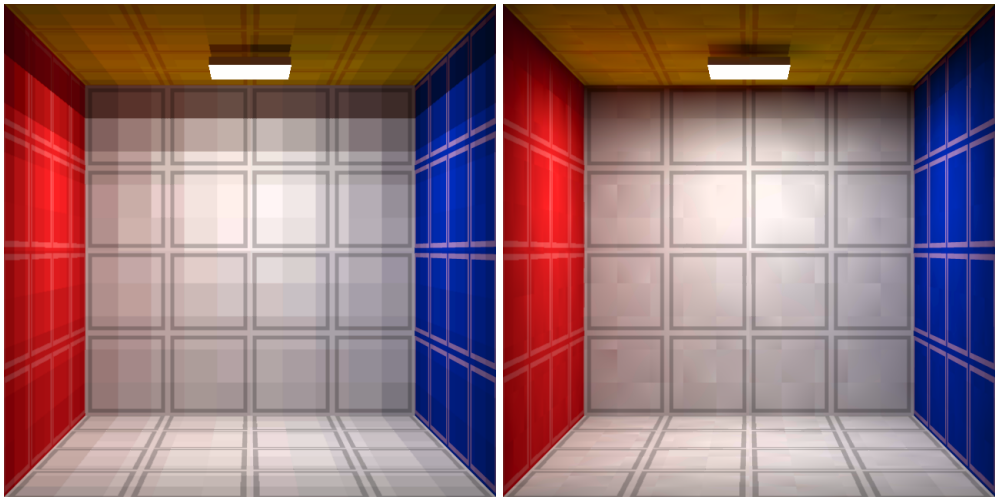


Figure 3.4: Uninterpolated radiosity solutions using constant and linear lightmap texels. The arguably more convincing appearance of the linear solution is offset by the fact that it requires more photon hits, and is harder to interpolate properly.

planar lightmap texel j is (at any time during the simulation)

$$B_j = \frac{1}{A_j} \cdot m_j \cdot \Phi \tag{3.7}$$

where m_j is the number of particles received by texel j, A_j its surface area and Φ the energy carried by one photon.

This shows the necessity that during the setup phase of the simulation the surface area has to be computed for each lightmap texel; a requirement that incurs a considerable execution time penalty if curved and arbitrarily transformed objects are used, especially in the presence of CSG intersections that may clip off part of the texels as discussed in Wilkie et al. [WTP98]; especially the latter problem is discussed in detail in chapter 5.

The surface area of a texel is also the reason why it is necessary to use a "good" surface tesselation with no singularities for the lightmaps. For example a normal, single patch globe–like texture mapping on a sphere is not suitable for texel storage because of the singularities at the poles; in our system we e.g. use tesselation into 8 triangles for spheres; this configuration avoids singularities at the patch joints.

Performance Considerations

Similar to Jensen's Photon Maps, photon lightmaps are normally not used alone – their advantages are best put to use in the context of a multipass renderer that e.g. uses area light source sampling to determine the direct illumination at a surface point, and the information in the lightmaps for all other contributions. The only modification one has to make to the photon tracing algorithm to accommodate for this is, that in this case the lightmaps start recording photon hits only after their first bounce from a surface.

Compared with Jensen's photon maps, lightmaps are – due to their discretization of the illumination function on a surface – inherently less capable of resolving fine illumination details, such as those which one can find e.g. in caustic patterns. Their main strength lies in the fact that in normal, mainly diffuse and slowly varying illumination constellations on large surfaces (i.e. a typical architectural interior scene), they use far less memory, and are also slightly faster to process during the final rendering pass.

Due to these properties, we deem lightmaps to be worthy of further investigation. Even though they are in some cases – notably caustics and specular reflection patterns– not capable of generating results of the same high quality as photon maps with comparable effort, their significantly lower memory footprint makes them the better choice as a starting point for adaptations to complex environments.

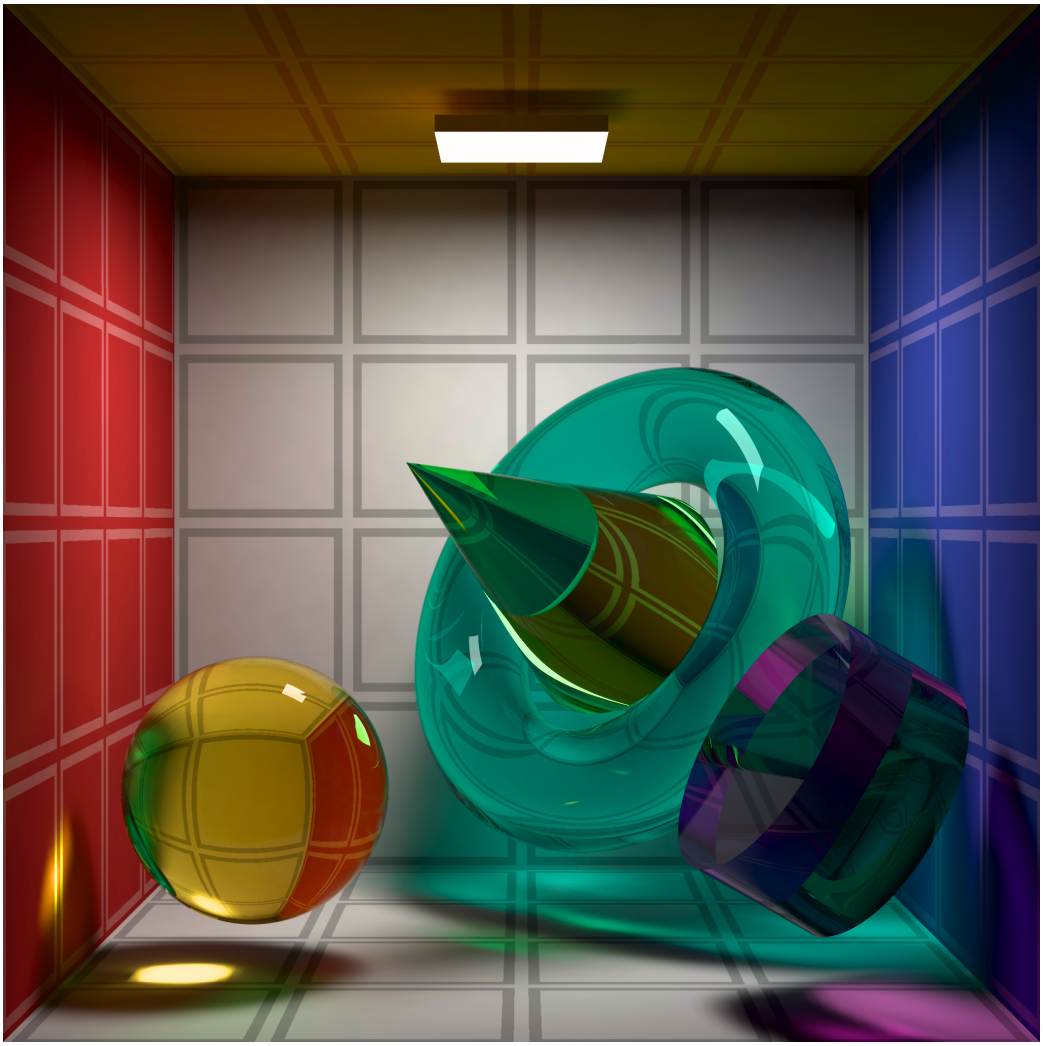


Figure 3.5: Caustic patterns, such as the bright and dark areas caused by the reflection from, and refraction through the transparent objects in this scene, are a phenomenon that is more efficiently traceable by a renderer that uses photon maps than with lightmaps. However, if the lightmap resolution is set high enough, equivalent images are possible, as this image demonstrates.

Usability in Complex Scenes

Lightmaps in their original form take badly to complex scenes for three main reasons. The data structures that record photon hits are maintained locally for each primitive in the scene, leading to more or less fixed memory requirements per geometric primitive. While this is practicable for Cornell box environments, it does not work for e.g. a forest with a huge number of needles on one tree alone. Also, in order to perform a meaningful reconstruction of the illumination on a surface, a certain number of photon hits has to be recorded on each photon lightmap. This is clearly impractical for complex scenes, especially since the shooting of photons becomes more expensive as the number of participating objects increases. Finally, the cost of performing exact surface area calculations is prohibitive for large scenes.

It has to be noted that one has to address all these three issues (area estimation, memory consumption and convergence speed) simultaneously if one wants to make lightmap–based photon tracing suitable for use in complex environments; every single one of these problems would render it impracticable for the purpose. In chapter 5 we take a first step towards this goal, and expand further on this in the following chapters.

Chapter 4

Modeling complex scenes

In this chapter we aim to provide an overview of the techniques used for efficiently specifying scene descriptions. Their main goal – to keep memory usage within reasonable bounds – implies that as few geometric primitives as possible should be used in the process, and that implicit and/or rule–based definitions of objects ought to be employed whenever possible.

The current preoccupation of the computer graphics industry with hardware–accelerated rendering has led to a situation where triangles and polygons, which are actually inappropriate for an exact representation of most scenes, are by far the most commonly used geometric primitives. Because of their low expressivity, they usually have to be employed in comparatively vast quantities if visual fidelity is to be maintained. This inherently raises scene complexities in a way which would actually be avoidable.

Most of the techniques outlined in this chapter are not as frequently used as they ought to be, since they attempt to move beyond this industry standard and use more advanced concepts for representing objects. This does not diminish the fact that they are important tools for reducing scene complexity, which in turn greatly eases the burden on global illumination techniques.

4.1 Constructive solid geometry

Constructive Solid Geometry (CSG) is one of the most efficient object representations for ray tracing. All kinds of solids with a well–defined boundary and interior/exterior classification can be combined by three Boolean operators to build up complex scenes.

Each scene is defined by a binary expression consisting of CSG operators and primitives. The operator $\cup$ creates the union of two volumes, operator $\cap$ their intersection and the operator $\setminus$ subtracts the volume of the second operand

from the volume of the first. Transformations are either associated with primitive objects or incorporated into CSG expressions as unary operators.

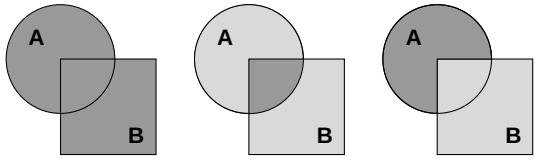


Figure 4.1: 2D examples of the three CSG operators. From left to right union, intersection and subtraction are shown, with the result shown in dark grey. Note that, unlike the other two, subtraction is not a symmetrical operation.

4.1.1 Raytracing CSG Models

Boolean operations of simple objects (CSG models) were originally introduced to the ray tracing world by Goldstein and Nagel [GN71].

During modeling, such expressions are represented as binary trees, called CSG trees or *scene graphs*, and which are capable of being traversed by the ray intersection procedure. The elements of these trees are referred to as *nodes*, and during operations which involve a traversal of the tree a *traversal state* is maintained. During a traversal, the state holds the currently active properties, such as the current material, surface type or 3D transformations; they are put on this stack–like data structure as the corresponding *attribute nodes* are passed on the way down in the graph, and removed on the way up.

As a simplifying preparation to the actual raytracing of a scene, the transformation matrices – which are normally distributed over the entire scene graph – are usually pushed down and stored in the leaves of a CSG tree, so that a ray has to be mapped from world space into primitive object space only once. This significantly saves computation time during rendering.

All intersections of a ray with primitives in the leaves of a CSG tree are collected in a list, which is usually referred to as *hit list* in the context of a raytracer. The Boolean operators can then be performed directly on the intersection intervals of this ray hitlist. The three–dimensional problem of combining primitive volumes is thereby reduced to the one–dimensional problem of combining intervals of a ray [Rot82].

An alternative data structure for the representation of CSG expressions is a directed acyclic graph (DAG), where primitive objects are stored only once, a feature that was called object instancing in [RW80]. This implies that transformations are stored as internal nodes of a CSG DAG, which have only one successor.

This structure saves memory if the description of the primitive is large, but when used in a ray tracing context it has the drawback that a ray has to be transformed more than once on its way through the graph to a primitive node.

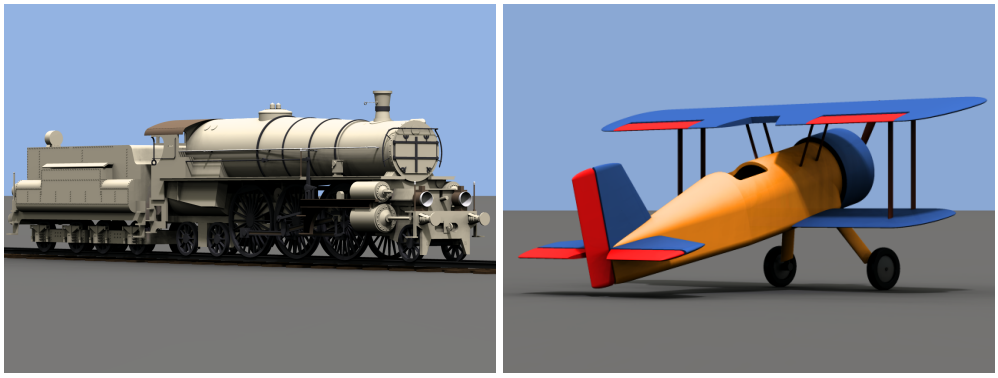


Figure 4.2: These two images are a good examples of where CSG is a superior modeling paradigm compared with polygon–based approaches. The highly detailed type 310 steam locomotive model contains only 3744 geometric primitives (more than a third of which are paraboloids for the individual rivets), while less than 380 suffice for the biplane (the majority are engine parts). Since there is a large number of curved surfaces in each model one would need orders of magnitude more polygons to adequately describe them in similar quality. (steam locomotive model by Katharina Weislein and Dr. Wolfgang Freund)

4.1.2 CSG Models in Practice

While CSG models are a natural match for ray–based rendering methods, other techniques have difficulties with them. Direct real–time rendering (e.g. using OpenGL) of CSG models is impossible due to the lack of high–level geometric objects in most rendering hardware and low–level APIs. Although several efficient schemes have been published that utilize the information in the *z*–buffer of a graphics accelerator to properly display CSG intersections, native CSG is still very rare. CSG models also have to be tesselated before use, which is both wasteful with respect to storage space, and discards the high–level information about the involved primitives.

Apart from these real–time rendering issues, CSG models are also plagued by the problem that degenerate intersection cases can occur through through careless modeling. An example of this would be coplanar faces on objects; hit lists from such configurations are not properly resolvable.

These problems, together with the inherent preference of industry for polygons, have led to a situation where CSG is usually offered as an option in modeling programs, but is normally not used for the actual object representations during rendering, where polygonal tesselations of CSG intersections are preferred.

4.2 Nontraditional Object Descriptions

While CSG is a step in the right direction if one aims at reducing the number of geometric primitives needed to describe a scene, the possibilities of all conventional, primitive–based declarative modeling paradigms are somewhat limited when it comes to representations of natural objects.

Describing things like clouds, terrain or plants requires techniques which transcend the "exact object blueprint" approach which is normally used. During the past two decades, two main classes of such non–traditional object modeling techniques have been developed, although the dividing line between them is sometimes a bit blurred.

4.2.1 Fractals

Fractals, the first of these two groups, were actually already discovered around the turn of the last century by mathematicians working in set theory, but lack of computing power meant that – apart from a few very easy specimens such as the Koch curve – they could not be properly displayed before the advent of computer graphics.

The term *fractal* derives from the fact that it is possible to define objects – usually as the convergent limit of an iterative process – that have a fractional topological dimension. While this characterization is not particularly intuitive, it is one of the few common denominators in a whole range of techniques, which can be used to generate objects with appearances that range from nature–like things such as clouds, to merely strange shapes like Menger sponges, and downright bizarre objects like 3D quaternion fractals.

Since the actual fractal is usually a limit surface or shape that is the end result of an infinite iteration process, most practical applications use approximations to them, which are obtained by terminating the iterations after a certain number of steps.

A distinguishing feature which applies to most types of fractal with varying degrees of intensity and obviousness is *self–similarity*. While the famous Mandelbrot set contains infinitely many slightly modified copies of itself in all sizes, fractal terrains realize this property through statistically similar appearance at arbitrary magnifications – no matter how close one zooms in, the viewed portion of

the surface is always as wrinkled as the whole area.

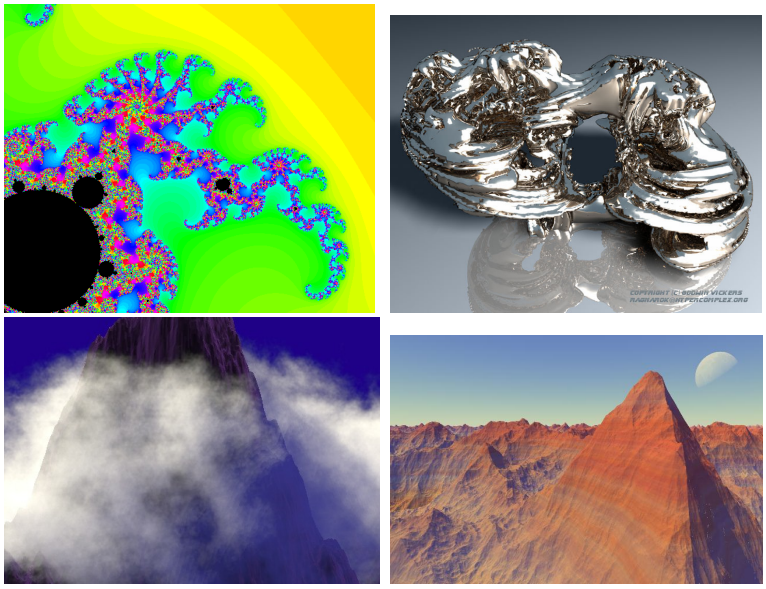


Figure 4.3: Representatives of various fractal types. In clockwise order, starting from the top left: a closeup of the Mandelbrot set, a 3D rendering of a quaternion fractal, a fractal landscape and a fractal cloud simulation. (Quaternion image by Godwin Vickers, landscape images by Ken Musgrave)

The tasks for which fractals are nowadays commonly used as modeling tools in computer graphics are the description of clouds (through a class of methods which is usually referred to as *plasma fractals*), and terrain generation. The latter are usually iterative and recursive displacement methods that operate on terrain grids. For added realism it is not uncommon to add post–processing steps such as erosion models to contour meshes which were generated by such fractal processes.

4.2.2 L–Systems

While fractal methods are very useful tools for efficient generation of several object types, such as clouds or terrain, they fall seriously short in some other areas, particularly the description of plants and other organized structures.

An obvious property which speaks against the use of fractals for plants is that – apart from some rare exceptions such as broccoli – plants are not self–similar across scale ranges. They also exhibit an ordered growth structure which makes the use of rule–based systems which simulate the underlying processes to some degree a much more feasible proposition.

Similar to fractals, the main technique which has proven useful in this context does not originate from the field of computer graphics. Aristid Lindenmayer originally introduced the concept of *parallel rewriting systems* (which were later renamed L–systems after their inventor) to biology as a simulation for cellular interactions [Lin68].

However, the basic idea of representing state as a set of symbols which are recursively modifed by repeated application of certain rules has proven to be universal enough to be adapted for uses outside biology. Several different versions of the same principle have been developed; we briefly discuss some of them.

D0L–Systems

The simplest type of these L–systems are *deterministic*, and are commonly referred to as *D0L–systems*. As all other versions, they are rewriting systems which operate on strings of *symbols* (which are chosen from a predefined *alphabet*), and iteratively generate output strings by application of their *rewriting rules* (also called *productions*) to these strings. The output strings are then recursively used as inputs of the next iteration; the step which generates this new output is called *derivation*. An initial string, the *axiom*, must also be defined as starting point for the iteration.

0L–systems are also known as *context–free L–systems*. They can formally be defined by a triplet $L :< V, \omega, P >$, where V is the used alphabet (a finite set of symbols), ω is the axiom (a string from V), and P is the applicable finite set of substitution rules, the productions.

As an example we demonstrate the simple system $L :< \{a,b\}, b, \{a \rightarrow ab, b \rightarrow a\} >$ in "action". In the first derivation step the axiom b is replaced by a according to the second production. In the second step the production $a \rightarrow ab$ is applied, which yields ab. The third step replaces both symbols of the string *in parallel* using the appropriate productions (those which take a and b as input, respectively), which leads to the result aba.

In general, the derivation of a string s_2 from a string s_1 is denoted by $s_1 \Rightarrow s_2$. The result s of the n^{th} derivation step can be written as $s = h^n(\omega)$. In our example the derivation sequence is $b \Rightarrow a \Rightarrow ab \Rightarrow aba \Rightarrow abaab \Rightarrow \ldots$, and e.g. $h^5(b) \Rightarrow abaababa$.

The Turtle Interpretation

As defined in the previous section, D0L–systems are not particularly useful for computer graphics yet, since they only generate strings, and not objects. Prusinkiewicz proposed a graphical interpretation for the string symbols [Pru86]

of such systems that is based on the *turtle graphics* of the LISP–like programming language LOGO, which was popular on home computers of the time.

The turtle is a stateful drawing cursor which can be moved around the drawing area by a simple command set. The commands are F (move one step of length d and draw a line), f (the same, only without drawing a line), $+$ (turn clockwise by an angle of δ, and $-$ (same, only counterclockwise). An example of such a command sequence is given in figure 4.4.

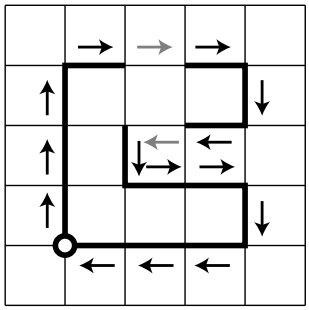


Figure 4.4: A simple shape encoded with turtle commands. The step length d is 1, and the turning angle δ is 90 degrees. The command sequence to generate this output would be $FFF-Ffff-F-Ff+F+FF-F-FFF$.

One shortcoming of the command set in the quoted form is that it is impossible to generate branches. This functionality can be added to the system by adding two new commands in the form of opening and closing brackets ([and]). These push and pop the state of the turtle onto and from the stack, respectively. These *bracketed* (D)0L–systems were introduced by Prusinkiewicz [Pru87]; figure 4.5 deomstrates an example.

Stochastic L–Systems

Even though we are now able to generate branching structures with L–system turtle graphics, the output of these automata leaves quite a lot to be desired when it is compared with real plants. One observation is that, although their basic plan is similar, hardly any two plants are ever quite alike.

This leads us to the proposition of Eichhorst et al. [ES80] to introduce randomness into the productions of a system. In particular this means that we can now specify multiple productions with a given predecessor, and assign a probability to each of them (the sum of these probabilities has to be one). Formally, a stochastic L–system is now a quadruplet $L :< V, \omega, P, \pi >$; the new component $\pi : P \rightarrow (0,1]$ contains the probabilities for each production.

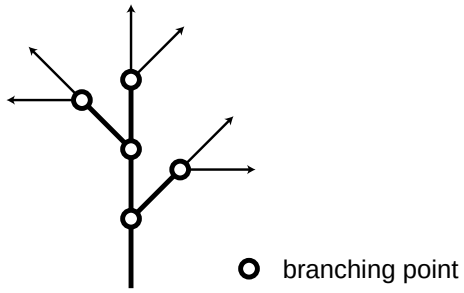


Figure 4.5: A simple branching shape encoded with turtle commands. The step length d is 1, and the turning angle δ is 45 degrees. The command sequence to generate this structure is $F[-F[-F]F]F[+F[+F]F]F[-F]F$, and the segments with the arrows would be those where a further recursion would add elements.

This implies that, during a single rewriting step, multiple occurrences of the same symbol can – according to the probabilites of the possible productions – be replaced by the result of different rules. If only one rule exists for a given predecessor, its probability has to be one.

Context–Sensitive L–Systems

An even finer way of controlling the growth of structures is to take the *context* of a given symbol (i.e. its neighbours) into account before rules are applied to it. This is different from just applying a rule to the group formed by the symbol and its neighbours, since the context cannot be modified by the production; it can only influence the selection of an appropriate rule.

The number of context symbols left and right of the main symbol which are used is arbitrary and can be chosen by the user. In practice, it is apparently rare to use more than one or two context symbols; such L–systems are referred to as 1L and 2L–systems, respectively (this also explains the notation 0L for context–free systems).

Parametric L–Systems

However, the possibilites offered by 2L–systems are still limited insofar as the ruleset has no way of influencing the behaviour of the turtle beyond the simple drawing commands listed earlier, which correspond to the insertion of geometric primitives in 3D. While this suffices for the creation of fractals and artificial branching structures, the large number of factors which govern the development of real plants makes such a restricted system an unlikely contender for a good simulation.

A major improvement can be realized by allowing the system to modify the parameters of turtle movement according to the current state. These *parameterized L–systems*, which were introduced by Prusinkiewicz et al. [PLH88], are far more general and sophisticated as they finally enable the user to describe various repetitive objects like fractal terrain, linear fractals, plants or seashells. Furthermore, they potentially allow mutual influences of the visual appearance of these objects, and interdependency of their geometry.

These systems operate on strings of *modules*, which consist of a symbol and a finite set of associated parameters. If V is the alphabet of the system, then $V \times R^*$ defines a module; R^* is the set of all finite sets of parameter values. A module can be written as $M(a_0,\ldots,a_n)$, with $M \in V$ and $a_0,\ldots,a_n \in R^*$.

Formally, a stochastic pL–system can be written as a quadruplet $L :< V, E\omega, P >$. The differences to the previous definitions are the inclusion of E – the formal set of parameters – and changes in scope – $\omega \in (V \times R^*)^+$, and $P \in (V \times E^*) \times L(E) \times (V \times A(E))^*$. $(V \times R^*)^+$ is the set of non–empty module strings, $L(E)$ is the set of all valid logical expressions, and $A(E)$ is the set of all valid arithmetic expressions that can be specified using parameters from E.

Productions are now written in the form of *predecessor* : *condition* $\rightarrow$ *successor*, as in e.g. $F(t) : t > 0 \rightarrow F(t-1)F(4)$. A production p can only be applied to a module m if the symbol of m and the predecessor symbol of p match, the number of parameters of the predecessor and m match and the condition in p evaluates to true.

As defined here, pL–systems are deterministic, but they can also express the same structures as stochastic L–systems if calls to a random function are permitted in production rules (both in the condition and succesor parts). This more flexible approach obviates the need for explicit choosing probabilites π for the individual rules.

$$
\begin{aligned}
&\omega : X(6) && && (4.1)\\
&p_1 : X(l) && : l = 0 && \rightarrow generateTetrahedron()\\
&p_2 : X(l) && : l > 0 && \rightarrow T_1X(l-1) + T_2X(l-1) + T_3X(l-1) + T_4X(l-1)
\end{aligned}
$$

The ruleset shown above is a simple example to illustrate these concepts. It is the pL–system for the generation of a Sierpinski tetrahedron; the DCSG counterpart for this system is shown in figure 4.7. By specifying $X(6)$ as the axiom ω, we determine that this system will generate a Sierpinski tetrahedron of recursion level 6. Productions p_1 and p_2 represent a conditional branch: if $l > 0$, then l is decremented by one and the transformations T_n are applied to the four recursively emitted X. The T_n are responsible for the shrinking and translating of the sub–tetrahedra at the corners. .

4.3 Directed Cyclic Scene Graphs

Gervautz and Traxler [GT96] proposed to use Directed Cyclic Scene Graphs (DCSGs) based on pL–systems. The result is an object instancing technique for procedurally defined complex scenes that is well suited for representation of complex natural scenes.

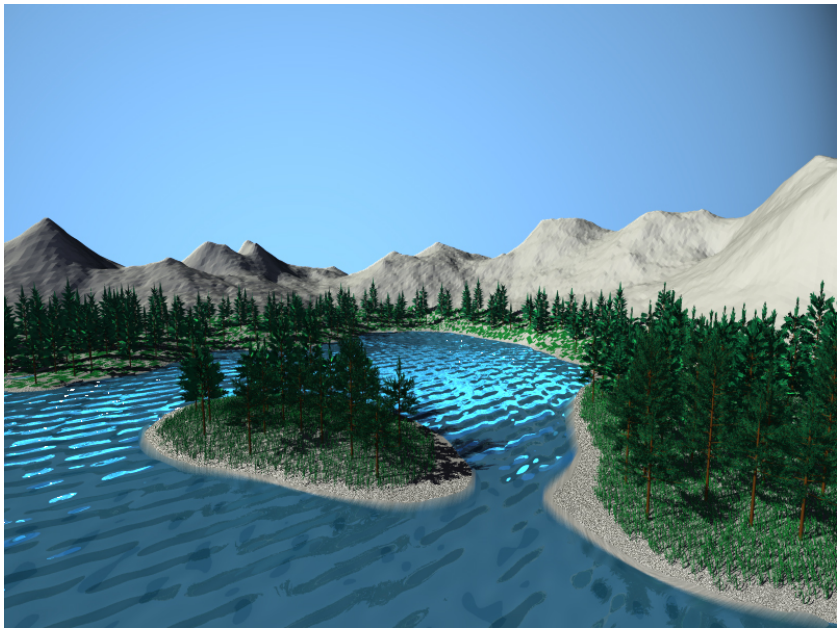


Figure 4.6: DCGs were used to model (through automatic tree placement) and represent this scene. It was rendered on a machine with 64 MB of RAM, which was not enough to keep the entire scene in memory. Image by Dr. Christoph Traxler.

PL–DCSGs are a powerful extension to the DCSGs used by Hart and de-Fanti [HD91] for ray tracing of linear fractals. Traxler and Gervautz [TG97] also demonstrated the combination and cooperation of different pL–systems to simulate the spread of fauna on a fractal terrain according to natural constraints (i.e. considering sea level, timber line and south slopes). This allows the rendering of huge scenes consisting of a vast amount of primitives represented by only a small set of interlinked DCSGs; an example of their work can be seen in figure 4.6.

4.3.1 PL-systems for CSG Expressions

Because CSG expressions can be seen as strings, it is possible to derive them from a pL–system. It is an essential aspect of pL–systems that the geometry of

the generated objects evolves from the derivation sequence. For this reason it is advantageous to specify transformations as parameterized unary operators within the CSG expressions.

As pointed out in detail by Gervautz et al. [GT96], one has to be careful when designing a pL–system that is supposed to generate a valid CSG object. The main problem is that the derivation sequence cannot be stopped arbitrarily as when using pL–systems, where the turtle ignores modules that do not belong to its command set; the result of the derivation process has to be a set of well–formed CSG expressions.

This has two consequences: first, rules can only be applied to modules (generating rules), and second, at least one rule which finally substitutes all variables with a string of terminals (terminating rules) must exist for every module.

4.3.2 Translation of pL-systems into Cyclic CSG Graphs

The first step is to interpret the right hand side of each rule (the string following the derivation symbol $\rightarrow$) as a normal CSG expression, and to instance any modules in these expressions as special scene graph nodes which fit into the scheme described in section 4.1.1. In this way we can build a normal CSG tree with cyclic elements – which correspond to branching rules – out of each right hand side. After that, the rules are encoded in another special type of node. We will discuss the working of these nodes in turn.

For the following explanations it is necessary to recall that in our rendering system, the current state of a pL–system is stored by the raytracing scene graph traversal state, which normally holds such information as the current transformation matrix; this information has to be extended to include the currently active rules and the current variable bindings.

4.3.3 Value Nodes

Transformations and other scene graph nodes which are dependent on numeric values have to have the capability to optionally evaluate *value nodes* instead of hardcoded numeric values. In the case of pL–system rules, value nodes that are capable of performing the needed calculations – such as increment and decrement of branching angles and indices, and performing logical evaluationsof conditions – are used.

It is worth noting that value nodes are also desirable in a rendering system for reasons other than support for pL–systems; e.g. shading language constructs are also dependent on similar features, and also require special value nodes (such as colour interpolation) that normally have little relevance to pL–systems.

4.3.4 Reference Nodes

Reference nodes provide the equivalent to pointers within a cyclic graph in our system. However, they are *not* actual pointers, but rather simply provide the unique string which identifies their targets. This string is then used to obtain a reference from the name bindings in the traversal state.

In this way our implementation of cyclic scene graphs is technically still a DAG, while providing full DCG functionality; this made the implementation easier, since we could rely on the tested DAG code.

4.3.5 Assignment Nodes

These nodes store the relationships between parameters (or *variables*) and actual values. The values can be either numeric, or the result of value nodes. Parameters are identified by their names, i.e. a unique string, and the task of these nodes is to push their content onto the traversal stack when they are passed on the way down, and to pop it off again on the way up.

4.3.6 Rule Nodes

These nodes contain the actual rules which govern the behaviour of a pL–system. They also push their contents onto the rule stack when passed, and remove their contribution again when it is no longer needed.

In order to provide a compact, yet instructive, example of these concepts, we show the section of a cyclic scene graph that generates a classical strictly self–similar fractal, the Sierpinski tetrahedron, in Figure 4.7. DCSGs for actual natural objects like e.g. trees operate on exactly the same principles, but are more complicated and less suited as demonstration objects. If one were e.g. to include the detailed findings of Weber and Penn [WP95] about realistic tree models in a DCSG, the result would be a very realistic tree, but the scene graph would be much harder to decipher.

4.4 Raytracing of Cyclic CSG Graphs

Raytracing is done as with any normal scene by traversing the CSG–graph in a recursive way. As in conventional raytracing with CSG–trees, the ray is intersected with all primitive objects that are found to be within the possible range of the ray (e.g. through bounding box tests), and the gathered intersection information is combined afterwards by Boolean operations in the operator nodes.

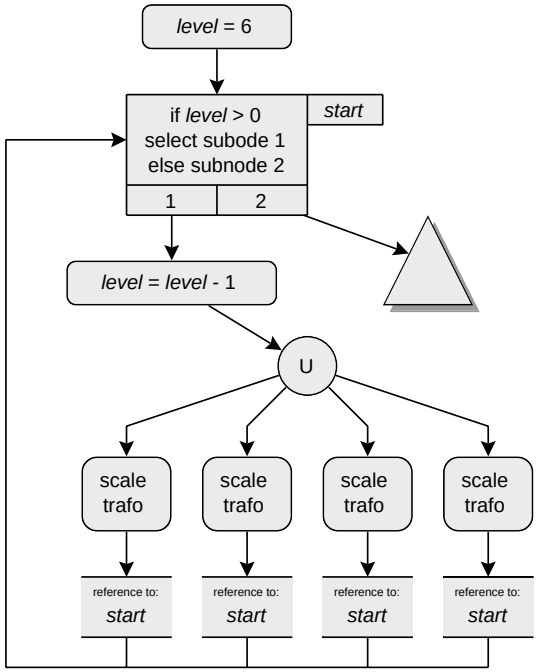


Figure 4.7: Representation of a Sierpinski tetrahedron of recursion level 6 as a cyclic CSG graph. The level of recursion is determined by the variable assignment $level = 6$ at the beginning of the DCSG. The triangle symbol denotes a tetrahederon primitive, while the circles are CSG OR operators. The actual cyclic references are not made through pointers, but rather through symbol name strings like *start*.

If DCSG elements are present, the only differences to the standard DAG raycasting algorithm lie in the way these special nodes work; most of this has already been explained in the previous section.

One area which requires special attention is bounding boxes and the efficiency of intersection tests. In the beginning, all DCGs are unfolded once during the scene setup phase in order to determine their bounding boxes. The generated objects are not retained, and only the obtained bounding information is stored. During the actual raycasting, only those objects are instantiated whose bounding volumes are intersected by the ray; in this way, only a small percentage of the scene has to be kept in memeory at a given time.

However, this constant discarding of just instantiated objects is usually wasteful, because neighbouring rays tend to intersect the same objects. It is therefore advisable to maintain a cache of recently expanded DCG objects, and to replace the least recently used instance in the cache once a new instance has to be created.

This scheme can yield considerable performance gains, but it has to be tuned well in order to be effective. A too small cache actually performs worse than none at all due to the administrative overhead incurred, while large caches are wasteful on memory and yield only small performance gains because there is usually only a limited amount of ray–to–ray coherence which is exploitable in a scene.

Chapter 5

Photon Tracing for CSG Solids

5.1 Motivation

After having presented the necessary background in the preceding chapters, we now present in this and the next two chapters three successive improvements to the basic photon tracing algorithm which greatly improve their usability on complex scenes.

As can be seen from section 2.3, a major stumbling–block for using conventional viewpoint–independent global illumination algorithms on complex scenes is the fact that these techniques require the scene to be discretized into planar patches before any light transport simulation is performed.

Even for scenes that do not contain curved surfaces, this discretization greatly increases the number of objects that participate in the calculations. This in turn greatly reduces the size and complexity of the scenes for which it is possible to compute a solution, since the limits of the used algorithms – in this case with respect to the number of objects that can be processed – naturally remain the same.

5.1.1 Avoiding Scene Tesselation

An obvious point for improvement here is to ensure that no tesselation of the scene is necessary in the first place – this solves the issues conventional algorithms face concerning both complexity and applicability to scenes with nonplanar objects.

Because they break the one–to–one correspondence betweeen geometrical primitives in a scene and the data structures used to compute and store the illumination solution, both lightmaps and photon maps offer themselves as solutions in this context; in both cases there is no longer a need for an explicit discretization of the involved objects.

This of course means that it is possible to retain the original, untesselated scene description during the entire process of image synthesis, which for typical scenes amounts to a huge decrease in scene complexity. This in turn makes global illumination calculations feasible for scenes that would be well beyond the capabilities of the involved algorithms, if they had to be treated in tesselated form.

This led us to the conclusion that the use of photon maps and lightmaps in scenes which are modelled using comparatively few high–order primitives by methodologies such as CSG, is a feasible first step towards obtaining global illumination solutions for more complex scenes.

Since both are just different storage methods for the illumination information gathered during a photon tracing simulation (as described in the previous chapter), the main question is how well they can be used on models that are specified using CSG.

5.2 CSG Objects and Photon Tracing

Since photon tracers model the propagation of photons through a scene by using ray intersection methods that take the exact geometry of any involved CSG solid into account, we are – as a by–product of the raycasting process – already provided with photon hits on the surfaces of the primitives that the CSG solids consist of. This enables us to correctly reconstruct the illumination function on the exposed surface parts of these primitives.

5.2.1 Lightmaps and CSG

Since they are in effect a special kind of “texture” applied to the objects, lightmaps that are attached to the faces of geometric primitives remain in place and functional when CSG operations are applied to these primitives. The parts which are not totally covered or removed by other objects take part in the simulation as recipients of photon energies in just the way they would if no CSG operations had been applied to the primitive to which they are attached. This is a fundamental advantage of photon–tracing algorithms over form–factor based approaches.

The texels on these lightmaps fall into two categories: those which are intersected by an object boundary of some sort, and those which are not. The illumination representation on the latter is correct without further modifications (this includes texels which are completely removed by CSG operations; they simply remain inert). Properly treating the former case requires some extra measures to be taken.

The artifacts possible due to complex CSG intersection geometry can be supressed by *splitting* the texel in question. Figure 5.1 shows some of the cases

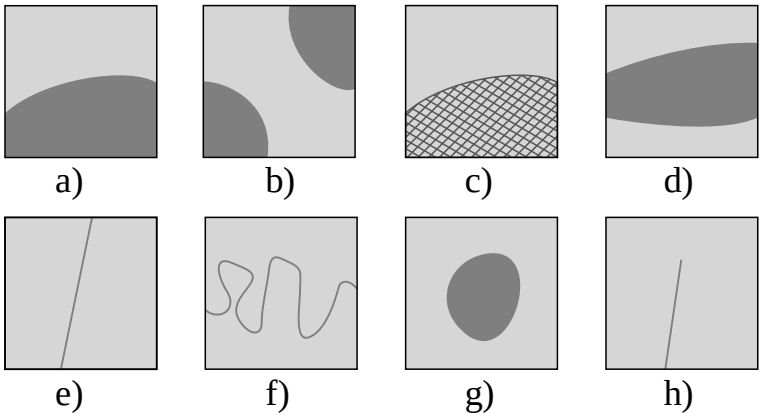


Figure 5.1: Cases of CSG intersections seen from the viewpoint of lightmap texels: a) and b) One or more opaque objects cover part of the texel surface. c) Part of the surface is covered by a non–opaque object. d) An opaque object divides the texel into two disjoint regions. e) Thin objects such as polygons do not cover any texel space but cause a split. f) Very irregular dividing lines between texel split regions can lead to problems with the accuracy of the shadow–mask edges and (in this case) the affiliation quad–tree (see the section on texel splitting for an explanation). g) and h) Both require some kind of triangulation to be performed if low–order bases are used to represent the illumination.

that can occur where a texel *has* to be split into several independent illumination representations (that we refer to as *sub–texels*) in order to avoid shadow and light leaks.

Since there is no limit to the complexity of the CSG operations that affect the area of one particular texel, any practical solution has to be suited to handling an unlimited number of arbitrarily shaped splits, with the possibility of employing some heuristic in order to cull unnecessary splits that would not impair the accuracy of the solution beyond some given degree (as suggested by Rossignac and Voelcker [RV89]).

For split texels both *area estimation* and *updating the functional representations* becomes more complex due to the fact that those parts of a texel that are covered or subtracted by other primitives have to be ignored. Fortunately, the additional information needed is already computed during the splitting phase and only has to be suitably evaluated.

5.2.2 Splitting of Texels

In order to determine whether a texel has to be split at all (and if so, into how many pieces), we need an algorithm that is able to detect and outline any geometrically

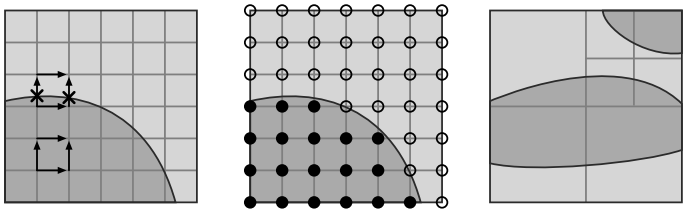


Figure 5.2: Texel split detection and management: a) For each sampling grid square rays are cast along the edges to detect object intersections. b) Sub–texel affiliation information used for area computations is stored on a per–grid–element basis. c) A case where a quad–tree data structure has to be maintained to disambiguate affiliation of ray intersections with respect to both the light and dark grey texel areas, which both belong to only one object each (see the section on identifying texels for an explanation). Note that the quad–tree subdivision has only to go to a level where an unambiguous traversal affiliation list can be maintained.

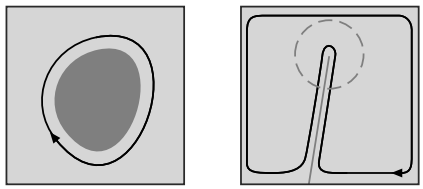
disjoint regions within some given surface area on a CSG primitive.

The solution to finding geometric intersections is to cast sampling rays on the surface of the texel. We assume that the meshing of the lightmap provides us with reasonably planar texels; this is no additional meshing constraint since any significant texel curvature would (in most cases) impair the efficiency and validity of the illumination representation, and therefore has to be avoided by the lightmap mesher anyway.

Gathering the Data Needed to Decide

In order to supply the data needed to identify coherent regions we propose to use a marching algorithm for 2D *sampling grid* elements (analogous to Lorensen and Cline's *marching cubes* [LC87] in 3D) on the surface of a texel. Since all information about the objects is available, one can resolve the 2D equivalents of the edge–tracing ambiguities mentioned (for instance) by Nielson and Hamann in [NH91]. Both the *location information* (where, that is within which primitive, the sampling grid elements lie with respect to the CSG operations performed) and the coordinates of any intersection points on grid element borders are retained for processing by the actual splitting algorithm.

While this approach in its simplest incarnation can miss intersections with very thin or oddly shaped objects (e.g. if they lie totally within mesh elements), it can be made very thorough by using an adaptive extension in the spirit of ACSGM [TGP96] and others.



a) b)

Figure 5.3: Detecting cases that require triangulation (in this case of the light gray areas) by examining the topology of the sub–texel borders: a) Closed loop inside the sub–texel. b) Large negative angle (marked by a dashed circle).

Identifying and Creating Sub–Texels

The information gathered during the marching step is first used to determine the number of sub–texels needed by counting the number of disjoint regions. If any object intersections were found within the texel we use a sweep–line algorithm that identifies coherent regions on the basis of sampling grid vertex and intersection point data. During this sweep we also construct a polygonal approximation of the sub–texel borders as a by–product.

Cases like Figure 5.1 f) and 5.1 g) can now be detected by scanning for closed loops inside the sub–texels or hard corners in the borders of those sub–texels that would warrant splitting as shown in Figure 5.3 (the definition of “hard corner” depends on the desired quality of the representation). If any indicators are found the offending sub–texel is split using some fitting triangulation.

After having determined the number of sub–texels we then make use of the location information at the sampling grid vertices. It is used to compute the *fraction of the texel surface* covered by the individual sub–texels. The grid information is also used to define *shadow masks* (as suggested by Zatz in [Zat93]) for each sub–texel; all parts of the texel that a sub–texel is not responsible for add to the mask. Since each sub–texel maintains an illumination representation that would normally cover the entire texel, this masking is necessary to remove hidden areas correctly, and to avoid interpolation errors at rendering time (see Figure 5.4 for an illustration). Due to the arbitrary shape of the sub–texel boundaries it is not feasible to represent them directly as unclipped patches without masking.

For instance, in Figure 5.2 b) the black sampling dots are used to define the shadow mask for the light grey area. To improve accuracy the object intersection points on the grid lines can be used in addition. It has to be noted that, given a correctly split texel, imperfect shadow masks are the only possible source of artifacts in the illumination representation within the limits defined by the implicit meshing and the functional representation.

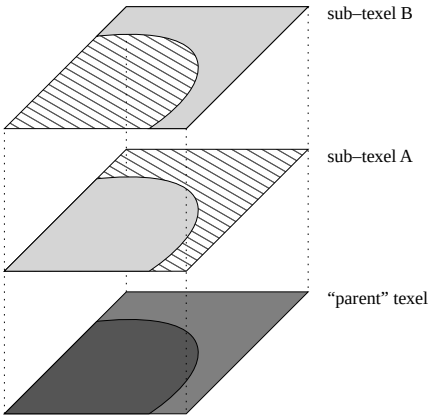


Figure 5.4: Sum of sub–texels that together covers the whole area of their parent texel. Each sub–texel uses a functional representation to store the illumination that ranges over the entire texel space. Only the shaded areas are valid; the rest is removed by shadow masks. The shadow masks would ideally coincide with the hatched areas.

We then proceed to construct a mapping data structure that stores the *traversal affiliations* for the various regions of the texel. Since a texel typically has a very small number of sub–texels, a linear list usually suffices for this task. The traversal affiliation determines which sub–texel to choose based on the material and geometric primitive last traversed by a photon or ray and, if this is not unambiguous by itself, the coordinates of the hit on the texel. The most recently traversed material and primitive are information provided as a by–product of the ray intersection computations involved in both photon tracing (i.e. during the radiosity simulation) and ray–tracing (i.e. at rendering time). Relying on this information makes the decision which sub–texel to choose immune against artifacts resulting from object boundary aliasing.

For the example shown in Figure 5.4 the affiliation mapping would consist of two entries: if the last traversed material of a ray was “dark grey”, choose sub–texel A, otherwise choose sub–texel B.

Only for texels that have ambiguous intersection geometries (that is, where the material last traversed is not a sufficient criterion for sub–texel selection by itself) we have to construct a quad–tree data structure as shown in Figure 5.2 c) that allows us to select the appropriate sub–texel by using additional information about the coordinates which interest us on the texel. As can be seen from Figure 5.2 c), the splitting of the quad–tree only has to go to a level where each leaf of the tree has an unambiguous traversal affiliation list. Note that use of a quad–tree for this purpose is not essential; other similar data structures could also be used.

Memory and Performance Considerations

Unsplit texels (which in "normal" scenes constitute the vast majority) do not cause any splitting overhead and only their functional representations are stored. Normally, most split texels require the maintenance of a short list of traversal affiliations in addition to their possibly numerous sub–texel illumination representations. For the (normally few) remaining "hard cases" an additional overhead is caused by the quad–tree used to disambiguate texel references.

For scenes with an average geometric complexity, the memory footprint of CSG–enabled lightmaps is not significantly higher than that of ordinary functional radiosity lightmaps.

Most additional computations (the classification and splitting of texels) occur during the set–up phase of the simulation. The run–time penalty is limited to the additional steps needed to choose a sub–texel when a ray intersects with split texels. No run–time penalty is incurred for ray intersections with unsplit texels, so that the performance loss of both the radiosity simulation and the ray–tracing remains within acceptable limits.

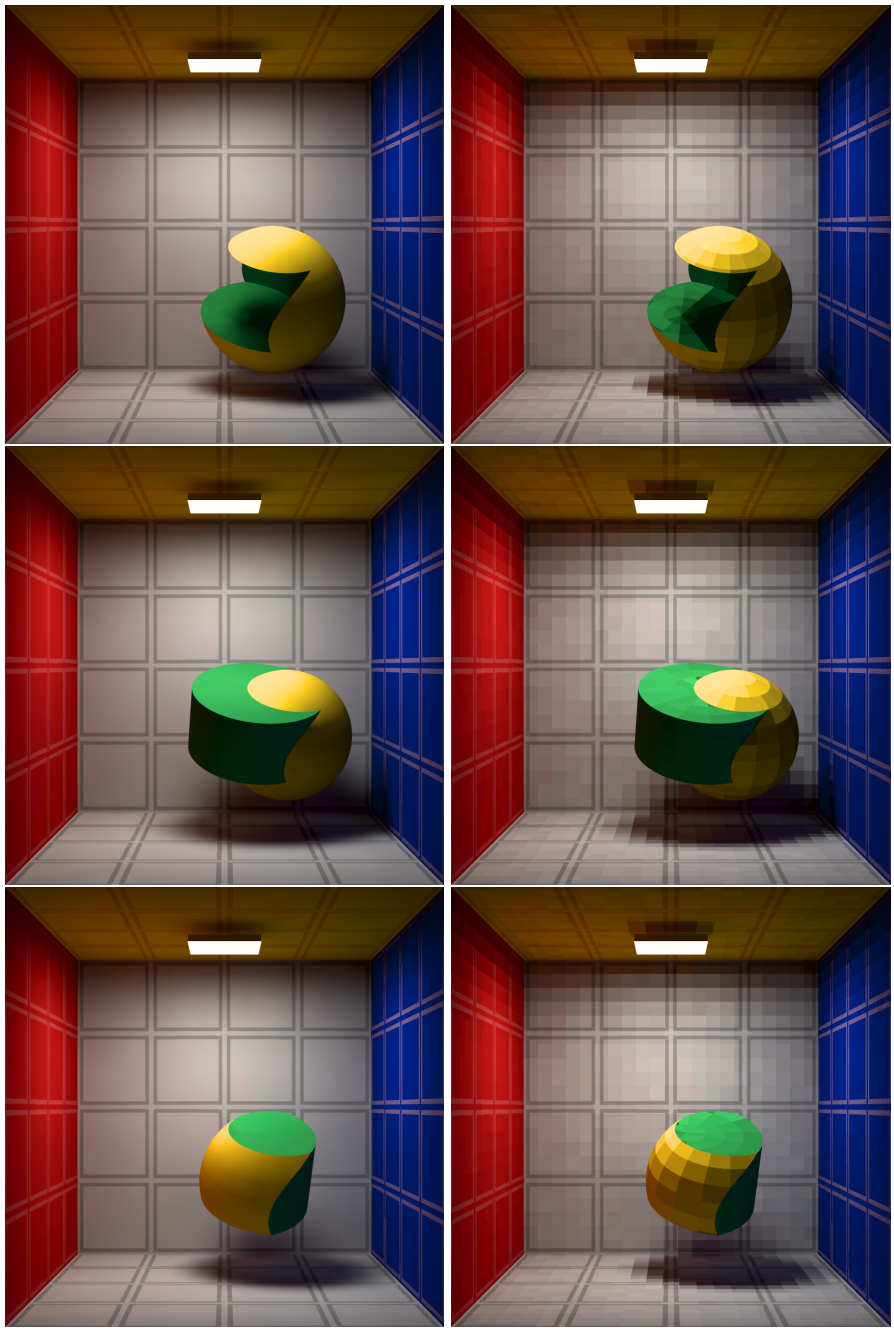


Figure 5.5: Interpolated and noninterpolated rendering of a simple test case: a cylinder and a sphere combined with the three basic CSG operators.

Chapter 6

Approximative Lightmaps

In chapter 5 we took a first step towards being able to use more complex models in a global illumination environment. This is achieved by no longer having to tesselate the scene description even for lightmap–based photon tracing algorithms, which use considerably less memory than their photon map counterparts, and are therefore – in spite of their higher algorithmic complexity – better suited for large environments.

While this certainly constitutes a big improvement over prior techniques where photon tracing was performed on a tesselated scene, the gains possible from this approach are still a drop in an ocean if one considers *truly* complex scenes, such as e.g. forests.

On the contrary, the untesselated CSG approach discussed in chapter 5 can in certain cases turn out to behave considerably *worse* than a conventional technique. For instance, a workable tesselation of a tree model might implicitly reduce the complexity of the model by e.g. substituting individual intricately modelled leaves with single polygons for the duration of the simulation, whereas the CSG approach would be stuck with the much more (and probably unnecessarily) complicated original model.

6.1 Approximative Illumination Solutions

This observation leads us to our working hypothesis for the second improvement we want to introduce: that the key to obtaining usable global illumination algorithms for complex scenes lies in the use of suitable approximative techniques, both from the perspective of memory requirements and calculation times.

A qualitative argument to further this view is that, as a general rule, it is fairly safe to assume that the more complex a scene gets, the less likely an observer is to notice individual errors in its illumination.

This, of course, only holds true as long as these errors do not lead to individual prominent artifacts, or systematically distort the appearance of the scene. The complexity of *perceptually–driven rendering methods* – the science of trading accuracy for performance gains, while preferably keeping the errors incurred to imperceptible levels – is testimony to the fact that deciding *which* tradeoffs one can safely make is a question which often poses a worse problem than the illumination situation they are applied to.

However, when faced with the problem of getting global illumination algorithms to work on highly complex scenes, we do not follow the goal of perceptually driven methods, although the setting is very similar. Our focus is slightly shifted insofar as certain implicit errors can be deemed an acceptable price for getting any results at all for large scenes, whereas perceptually driven approaches usually try to maximize performance without compromising the quality of the results.

6.1.1 Approximative Photon Tracing

In a photon tracing environment, the obvious point to address when one wants to trade accuracy for lower memory usage and higher speed, are the data structures used for storage of the gathered illumination information, and not the photon tracing process itself.

Photon maps offer little possibilities in this respect, but already have the inherent advantage that, since they are to a high degree independent of the scene description they are used on, they always at least fail as gracefully as possible in the event of too having received too few photon hits for a meaningful result on complex geometry, and do not distort the overall simulation.

In the case of lightmaps, the one–to–one correspondence between object surfaces and lightmaps is wasteful in all those circumstances where the exact illumination on these surfaces is not of particular importance as long as the overall appearance of the scene remains intact. Typical examples would be trees when observed from a distance, where the exact illumination on individual leaves is usually not important, as long as the whole tree has correct brightness.

This leads us to the conclusion that an approximative version of lightmaps would be a key improvement to photon tracing algorithms [WTP00a].

6.2 Orientation Lightmaps

The logical step towards such an approximative type of illumination storage structure is to disassociate the lightmaps from actual objects, and to modify them so that they simultaneously cover whole groups of objects.

In order to achieve this, we propose a lightmap–like data structure that averages all incoming irradiance for complex objects based on the *surface normal* of the photon hit point. We call this an *orientation lightmap* (OL for short) due to the surface normal dependent way it stores the illumination of the object it "covers". The proposal is in a way akin to the work of Rushmeier [RPV93] in that, in respect of photon storage, it performs an implicit geometric simplification for the object it is assigned to. However, due to the markedly different nature of photon tracing, this is also where the similarity ends.

We average the photon hits based on normal direction, rather than the more obvious direction of incidence, because the latter is not possible for simple lightmaps. The reconstruction of the illumination at a given surface point according to equation 3.7 requires knowledge of the surface area associated with a given lightmap texel. There is unfortunately no meaningful way to average the surface area of an object based on the directions of photon incidence. However, since it is possible to average the area of an object according to surface normals, we chose to use this approach instead.

Topologically, an orientation lightmap can be thought of as a spherical lightmap that surrounds the object of interest. As shown in figure 6.1, the place where incoming irradiance is stored depends on the photon hit normal. Consequently, evaluation of the irradiance for any surface point during subsequent rendering passes is based on its surface normal only; all points on the underlying object with the same surface normal have the same irradiance.

They can be seen as a local, directionally dependent term which is determined through the photon tracing simulation. Orientation lightmaps serve the same purpose as the environment maps proposed by Reinhard et al. [RTJ94]; the main difference is that the "content" of OLs is generated during the photon tracing phase of rendering, and does not have to be acquired manually.

While this technique obviously can lead to potentially huge errors in the illumination of an object (in that light energy is "spread" across nonconvex objects), we contend that this is still a useful approach for a wide variety of cases. The limit case for which orientation lightmaps yield the same results as normal lightmaps is when both are applied to spheres; the effect of applying OLs to a simple nonconvex object (a torus) can be seen from figure 6.2.

6.2.1 Properties

Orientation lightmaps do not lose or generate energy during the photon tracing pass. They maintain the overall appearance of the object they "cover" well if the object is reasonably isotropic, and are very fast to insert in a scene graph — much faster than normal lightmaps, since they do not have to perform an exact area calculation for each component of the object in question (see subsection 6.3 for

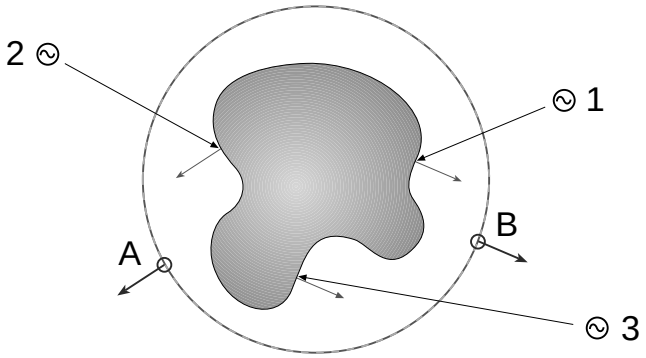


Figure 6.1: How an orientation lightmap, applied to a generic nonconvex object, stores photon hits. The photons 1, 2 and 3 all hit different points of the surface, but 1 and 3 are subsumed in the lightmap entry B since their surface normals are the same.

details). Due to the fact that they are usually responsible for larger objects, they gather large numbers of photon hits, which in turn yields an irradiance estimate with lower variance.

Also, orientation lightmaps can be freely mixed with normal lightmaps in one photon tracing simulation, meaning that some objects can carry normal and others orientation lightmaps, just as the desired accuracy of the solution requires. It is at the discretion of the user or rendering application to insert such lightmaps instead of normal ones wherever it is deemed appropriate to do so.

If the objects in question are suitable, it is also possible to use OLs hierarchically (see figure 6.3 for an example). It is even possible to use OLs above the "genuine" lightmaps in a scene. In this case they serve as a backup: they are used to reconstruct the illumination for those objects that they cover where the variance of the normal lightmaps is too high.

6.2.2 Applicability

The main area of application for our method is the rendering of complex objects that either will not come under close scrutiny by the observer, or that are simply too complex for the normal lightmap–per–primitive approach to work on a given setup.

Generally, they are not a technique that one would use if it can be avoided, but are rather intended for those circumstances where photon tracing would not work in its original form. In such cases they can make a rendering possible that would otherwise fail, sometimes even without a noticeable degradation in quality.

Figure 6.2: Comparison of normal and orientation lightmaps: Normal (left) and orientation (middle and right) lightmaps applied to a tilted torus. The two images on the left were rendered using the two–pass renderer described in the text; the image on the right is similar to the middle image, except that a photon–tracing–only setup was used to better demonstrate the artifacts of OL averaging. Overall, the images show the error incurred by direction-dependent averaging of the illumination on a simple nonconvex object to be rather small.

They are also useful if one wants to save on rendering time by actively reducing the accuracy of the solution in those parts of the scene which are guaranteed to be viewed only at greater distances, or not at all.

However, they are *not* an optimal choice for certain types of objects, such as those with only a small number of surface normal directions (such as e.g. a cube or similar polygonal objects). While OLs still function correctly within their limits in such cases, they cause a very strong averaging effect which is normally not desirable.

6.2.3 Photon Tracing with OLs

When performing photon tracing calculations for a scene (which in this case we assume to be represented by a directed graph), the first step is to insert the lightmap data structures which will hold the received photon samples into the graph. As shown in figure 6.4, this can be done virtually anywhere in the scene graph.

At this time the necessary meshing of the lightmaps into texels is determined and the appropriate data structures are allocated. During this phase the lightmap also computes the area of the underlying object(s); for orientation lightmaps we use the stochastic algorithm that we present in subsection 6.3 in order to gain acceptable performance. The area value computed for a lightmap texel on an orientation lightmap is an estimate of that part of the surface area of the underlying object, which has surface normals that point in the direction the orientation

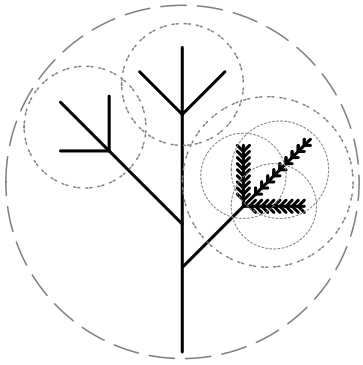


Figure 6.3: An example of the hierarchical application of orientation lightmaps, in this case on the branch of a tree. In such a setup the lowest level of the hierarchy which has received a significant amount of photon hits is used for display purposes.

lightmap texel covers. Once the lightmaps are in place, the actual photon tracing pass is performed. It is important to note that in the case of objects that are covered by an OL the real object geometry is still used for photon–object intersections; only the storage of illumination is done on the OLs. In this way the OLs do not alter the flux of photons in the scene in any way.

Once the tracing is complete, the gathered irradiance values on the lightmaps (normal and orientation) are interpolated. The final step in the rendering process is a raytracing pass that uses the information in the lightmaps to determine the illumination of the objects in the scene.

6.3 Stochastic Area Estimation

For the concept of orientation lightmaps to be feasible it is essential that an efficient area estimation method, that does not use the brute force approach of explicitly calculating the surface area of all geometric primitives in a complex object, is used.

We use a stochastic area estimation method that determines the surface area of an object by evaluating a certain number of sampling points on its surface. There is no restriction on the geometry of the object, other than that it has to be made up of parts that have (u,v)–parameterisable faces or consist of patches with this property — for objects modeled using CSG or B–rep, this is typically the case.

We first discuss the proposed method for the case of a single patch, and then show its extension for compound objects.

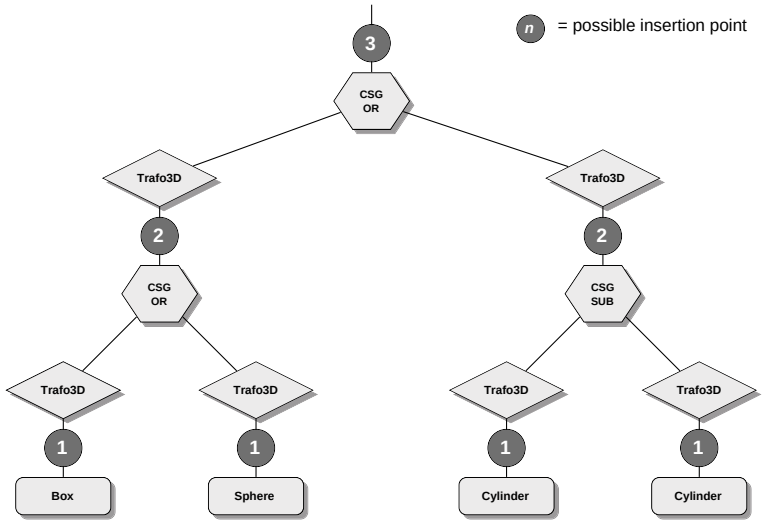


Figure 6.4: Possible insertion points for orientation lightmaps in a scene graph. They can be placed virtually anywhere above geometric objects where the stochastic area estimation would work.

6.3.1 Area of a Single Patch

On a surface patch with (u,v)–parameterisation the area function $A(u,v)$ can be expanded in terms of the surface geometry $\vec{x}(u,v)$ as follows:

$$A(u,v) = \left|\left| \frac{\partial \vec{x}(u,v)}{\partial u} \times \frac{\partial \vec{x}(u,v)}{\partial v} \right|\right| \tag{6.1}$$

This means that evaluation of the integral

$$A_{patch} = \iint A(u,v)\, du\, dv \tag{6.2}$$

yields the surface area of patch when the double integration is performed over the entire (u,v) parameter range. This evaluation can be carried out numerically by generating n random points p_k ($k = 1 \ldots n$) on the surface of the patch and numerically differentiating the surface at these points (see figure 6.5 for a schematic). The area of each of these samples is

$$A_k = \frac{||du \times dv||}{||u|| \cdot ||v||} \tag{6.3}$$

where u and v are the "differential" tangent vectors in u and v direction — in effect the ε for the numeric differentiation. If the samples are distributed evenly across

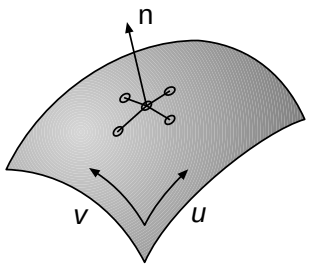


Figure 6.5: A possibility for numeric differentiation at a sample point on a patch: In the vicinity of a randomly chosen sampling point, four additional points are selected and used to construct four crossproducts, the average of which is used as the normal vector and area sample.

the (u,v) parameter space of the patch, an approximation of the entire patch area can be computed using

$$A_{patch} \approx A_{uv} \cdot \frac{1}{n} \cdot \sum_{k=1}^{n} A_k \tag{6.4}$$

where A_{uv} is the area of the (u,v) region over which the patch is defined, n is the number of samples per patch and A_k are the area values of the individual samples. The crossproduct $du \times dv$, which has to be calculated during this process, is the normal vector needed to determine which texel of the OL the sample is added to.

Each sampling point contributes to the surface area of the normal direction it represents; the samples are added to the area estimate of the OL texel that "contains" their direction (the area estimate is inititalised to zero for all directions at the start of the estimation process). This eventually yields a valid area estimate for each texel in the OL. After the area estimation, OL texels that cover normal directions that are not present in the object will still have an area of zero, but since they are never used during the simulation, this does not constitute a problem.

6.3.2 Area of a Compound Object

For determining the area of a compound object consisting of m parts one generates s samples, which are distributed evenly over the m parts (irrespective of their relative sizes, since this property is not known at this time). It has to be noted that s can be (even considerably) smaller than m in some cases; for objects which are made up of large numbers of similar parts, we typically do not have to sample every single one of them.

This yields n_i sampling points (where $\sum_{i=1}^{m} n_i = s$) for each part of the compound object, where the process outlined for single patches is applied. All one

has to do is to incorporate the information gathered from these samples on all the object parts into the OL that “surrounds” the complex object in the same way as if the samples were from a single patch.

Chapter 7

Lightmaps for Cyclic Scene Graphs

The introduction of orientation lightmaps in chapter 6 enables us to use an approximative version of the lightmap approach to photon tracing on highly complex objects.

While this is already a large step towards making photon tracing possible for large scenes, the fact that orientation lightmaps still have an explicit link to the objects which they cover – by having to be placed in the scene graph directly above them as shown in figure 6.4 – hinders their usage on objects modelled with one of the most memory–saving modeling techniques known, namely the rule–based object definitions which we outlined in section 4.3.

This denies us the use of orientation lightmaps on exactly the kinds of object which would benefit the most from their availability. This is the problem which we address in the following sections.

7.1 Rule–Defined Objects and Global Illumination

Raytracing and interactive display using a *z*–buffer are the rendering techniques which are so far normally used in conjunction with DCSGs. However, for high–quality realistic image synthesis these methods are obviously not sufficient. As typical examples, the results published in the paper of Traxler and Gervautz [TG97] show that the forest canopies are far too dark, because the mutual illumination and translucency of plant members are ignored with plain raytracing (to say nothing of *z*–buffer renderings).

On the other hand, certain view–dependent features like specularity and reflections are not overly important for such scenes, which again suggests the possibility of using approximative methods.

From these observations we derived our motivation to apply our approximative photon–tracing based global illumination rendering techniques to DCSGs for

pL–systems without losing their key property – their unmatched low memory consumption.

7.2 Lightmaps and DCSGs

The non–explicit nature of a scene graph that contains DCSGs makes the straightforward attachment of lightmaps (or similar global illumination data structures) to the objects they have to cover impossible. Doing this would require unrolling of the cyclic graphs, and destroy the low memory footprint property of rule–based object descriptions.

Also, even if one used a reference indexing scheme – like e.g. the one we describe for orientation lightmaps in section 7.4 – for the inclusion of "ordinary" lightmaps in DCSGs, the usually huge number of primitive objects in a DCSG object would render such an approach impracticable.

What has to be done is to reduce the number of global illumination data structures used in the rendering process, without reducing the complexity of the DCSG object. This is where orientation lightmaps come into play.

7.2.1 Combining Orientation Lightmaps and DCSGs

Due to the fact that they can cover multiple objects, and that they do not rely on individual properties of these objects, orientation lightmaps are an excellent possibilty to let DCSG objects be used in photon tracing rendering, if one is content with only an approximative solution for their illumination.

It has to be noted that the following technique applies to the use of OLs in the context of *any* rule based modeling environment that is based on cyclic extensions to a scene graph (such as Open L–systems [MP96] or context–sensitive L–systems [PJM94]); we just used generic pL–systems for our reference implementation, but the approach is general enough to be of common use.

7.3 Orientation Lightmaps for Entire DCSGs

The insertion of OLs over DCSG objects is in no way different from lightmap insertion over "normal" geometric objects. Since the data needed to instantiate an orientation lightmap can be gathered from traversals of the DCSG, no flattening of the scene graph has to be performed. The only distinction from insertion over a normal section of the scene graph is the slight execution time penalty for cyclic graph traversal with its slower symbolic references.

However, for most applications a single orientation lightmap around the entire object generated by a DCSG will be a too coarse approximation, so it would be highly desirable if one were able to insert OLs above parts of such an object, but without having to unroll the cyclic graph.

7.4 Orientation Lightmaps for Parts of a DCSG

Inserting OLs *inside* a DCSG object is actually quite easy. Since the individual objects generated by a DCSG are non–pertinent, one has to separate the OL instances from their point of usage. The key idea here is to use a storage node directly above the DCSG that maintains all OLs that belong in the cyclic part of the graph, and that the cyclic part of the graph only contains stubs at appropriate locations that refer to instances in this OL list. An illustration of this principle can be seen in Figure 7.1.

The main questions in this context are where one ought to place such OL insertion stubs in the DCSG, and how the correspondence of OL instances in the storage node and OL stubs can be established efficiently.

7.4.1 Choice of Insertion Points

In our system, the user determines possible insertion points for OLs within the DCSG structure manually during the modeling phase. Since the use of OLs within a DCSG entails both the dangers of excess memory consumption (if too many OLs are inserted) or of an overly inaccurate solution (if too few are used), and modeling with L–systems is a very intricate task to begin with, the insertion is left at the discretion of the scene designer.

Normally, one uses rather few, strategically placed OLs in a DCSG at spots that are more or less predefined by the nature of the modelled object, so the additional work for the scene designer stays within bounds. In the case of trees, it is e.g. intuitively advisable to have separate OLs for the interior of the plant (i.e. the trunk and the main branches) and for sections of the foliage. However, it is also conceivable that appropriate heuristics for an automated insertion could be devised e.g. along the lines of the abovementioned generic rules for trees, but that problem is beyond the scope of this thesis.

A potential problem when inserting OL stubs into a repetitive DCSG (manually or otherwise) is that one normally does not want to insert OLs above a symbol at *every* recursion of a DCSG, and that suitable insertion points are initally lacking. An example of this would be OL insertion in the Sierpinski tetrahedron; inspection of the scene graph in Figure 4.7 reveals that in the unmodified graph

there are no insertion points except for the terminal symbol (the tetrahedron primitive), the selection node at every level of recursion, or the entire DCSG available. The solution is to use a selective, recursion–level dependent insertion through a simple conditional mechanism as shown in Figure 7.1.

7.4.2 Correspondence between OLs and their Stubs

During traversals of the cyclic part of the scene graph, a reference to the lightmap storage node above the DCSG is stored in the traversal state. Any operations on or beneath an orientation lightmap stub in the DCSG that need to access the currently relevant OL instance (such as photon hits during the tracing phase, or illumination computations during raytracing) have to query the storage node for a reference to "their" lightmap.

The key used for search and retrieval is the unique address of the OL stub within the traversal of the cyclic graph section. This address is easily generated by noting which branch was taken at all nodes in the DCSG that have more than one subnode. This string of decisions (in our case, the numbers of the selected subnodes) that leads to a particular node is unique within the DCSG, and is automatically concatenated as a by–product of graph traversal.

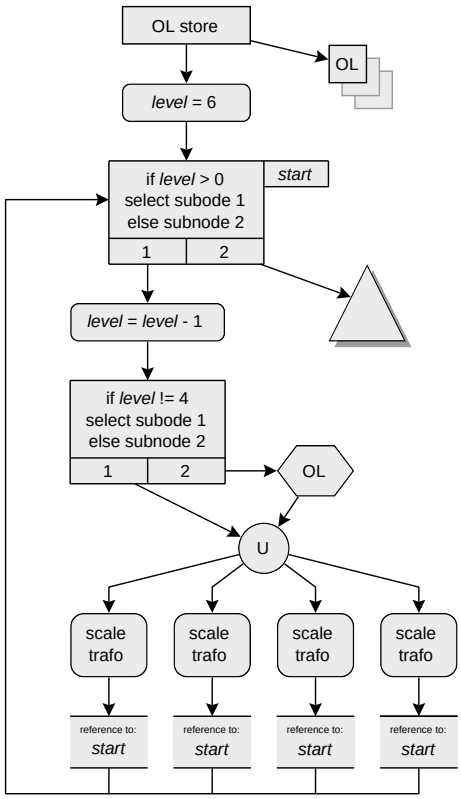


Figure 7.1: Inserting orientation lightmaps into a DCSG at a specific level of recursion: in this example, an OL stub node (symbolized by a hexagon) is inserted only at recursion level 4 into the Sierpinski tetrahedron example from Figure 4.7. The scene graph traversal state maintains a reference to the OL store node it encountered before entering the cyclic graph, and the approporiate OL instance is selected by the stub at rendering time.

Chapter 8

Results

In this chapter we present in a unified form the results of using the techniques we introduced in chapters 5, 6 and 7. We implemented all proposed methods as an extension to the Stochastic Galerkin Radiosity system in the Advanced Rendering Toolkit ART under development at the Iinstitute of Computer Graphics of the Vienna University of Technology.

8.1 Photon Tracing for CSG Models

In accordance with our assumptions, the only tangible execution–time penalties for using the CSG–aware lightmaps we implemented in ART according to the description given in chapter 5 were incurred at start–up time. Apart from that, we experienced no significant slowdowns during the simulations themselves. Also, the memory footprint of the lightmaps was not increased significantly. As a very simple specimen, figure 8.1 shows the three CSG operations from figure 5.5 in one picture, while the left part of figure 8.2 demonstrates a more complex example.

8.2 Orientation Lightmaps

We implemented the orientation lightmaps classes as derivatives of the normal photon lightmaps already in use. The main differences of the OL classes are in the setup methods (i.e. the area estimation code outlined in section 6.3). The renderer uses a two–pass method which utilises area light source sampling for the calculation of direct illumination, and the information in the lightmaps for all other contributions.

The results we have obtained are promising. For comparisons, we used a CSG model of a toy locomotive as a case where it is still easily possible to compute both an exact solution and an OL approximation; the results are shown in figure

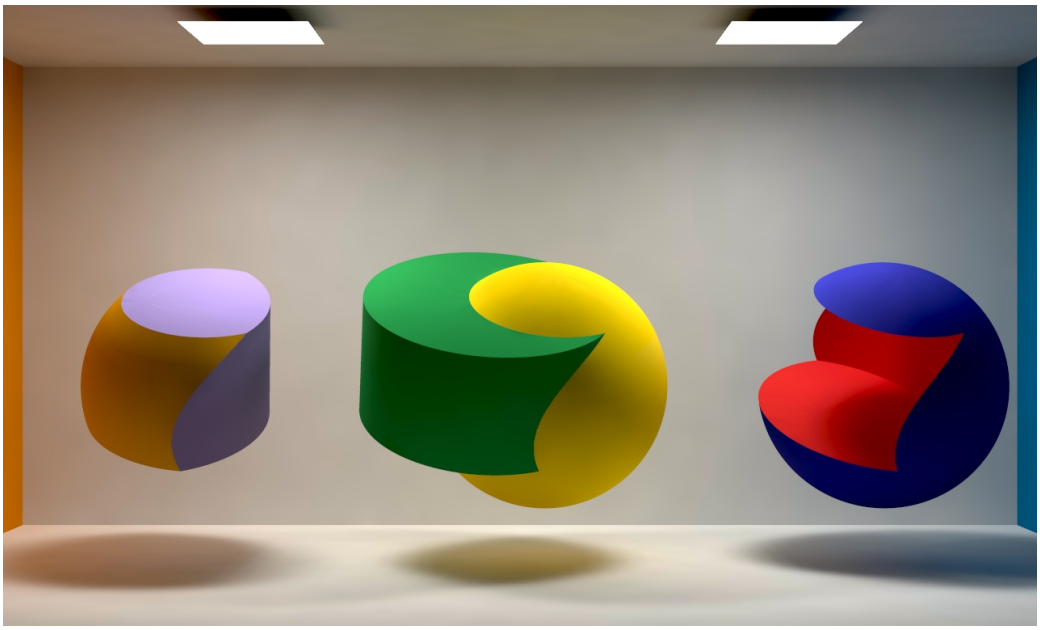


Figure 8.1: The three basic CSG operators demonstrated in a radiosity setting.

8.2. The advantage of the OL approach with respect to area computation time is evident (600 vs. 5 seconds on a PPro/200), and the artifacts incurred by their application are, while noticeable in comparison with the exact solution, subtle enough to demonstrate the usefulness of the method on complex, nonconvex objects. Differences are most noticeable on the wheels, especially on the large one beneath the “drivers cabin”: as to be expected, this part of the engine, which is only illuminated by indirect light, exhibits a lack of self–shadowing.

The two other test cases we present are a sphereflake and a tree model. The sphereflake is modeled to recursion level 4 — it consists of 7381 randomly coloured spheres. Figure 8.3 shows the difference between covering the entire object by a single OL, or covering the first–generation “children” with their own OLs. The overall appearance of both solutions is convincing, which indicates that in some cases a single OL can be sufficient even for objects that consist of many different components, especially in non–critical illumination situations.

The tree in figure 8.4 was manually modelled along the lines of a L–system; the entire object consists of roughly 120,000 CSG primitives. It is covered by a hierarchy of OLs; each major part of the tree (such as the branches) has its own OL. Area estimation for this object took about 180 seconds of CPU time on a Pentium II/450 with 20000 area samples per OL. The visual appearance of the tree is reasonably good, since the overal brightness is correct, and the spreading of illumination across the entire tree is prevented by the use of multiple OLs.

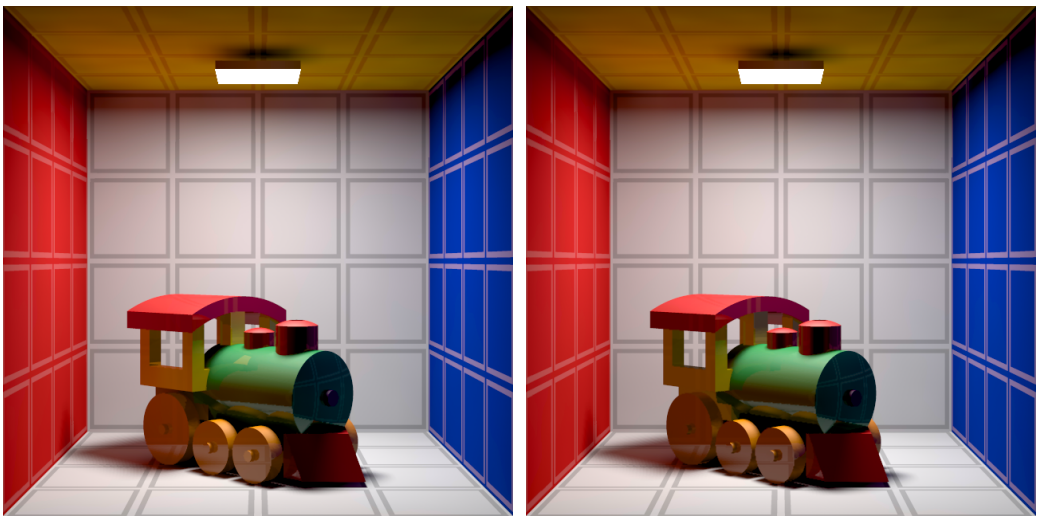


Figure 8.2: Comparison of normal and orientation lightmaps: a CSG model of a locomotive rendered using conventional photon maps (left) and orientation lightmaps (right).

8.3 Orientation Lightmaps for Cyclic Graphs

The first objects we rendered in order to test our implementation of DCSGs were classical DCSG shapes, the Menger sponge and Sierpinski tetrahedron. In what was, after initial consternation, a perfectly understandable result, both objects provided rather unsatisfactory results when viewed in a radiosity setting, but not because our implementation was to blame. Both objects are self–similar fractals and as such exhibit an inner structure that “swallows” considerable amounts of light, which in turn is disadvantageous to the use of orientation lightmaps. However, as our experiments with other objects suggest, this effect is really only a problem with self–similar objects.

The sympoidal tree shown in Figure 8.5 proved to be a more rewarding test case. We applied orientation lightmaps to each of the branches, which yielded about 20 OLs for an object that, when flattened, would consist of roughly 40000 primitives. The cyclic CSG graph, including all transformation, colour and material nodes that are not essential for the structure of the plant, is made up of just roughly 200 nodes. We observed raycasting operations on the DCSG tree to be about 20-30% slower than on equivalent instantiated objects. With 100k photons shot and 2000 area samples per OL taken, the rendering at a resolution of 640 by 640 pixels took 700 seconds. Memory usage of the 20 OLs and the raycasting DCSG unrolling cache was minimal compared with what an unrolled version of

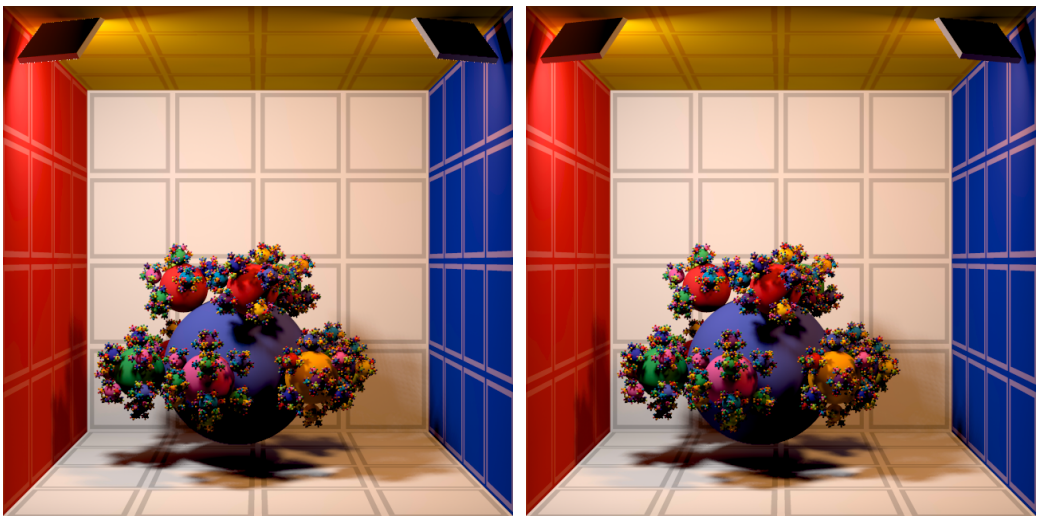


Figure 8.3: Comparison of orientation lightmap usage: a sphereflake rendered using one OL for the entire object (left) and one OL for each level of the sub-flake (right). The overall appearance of the two solutions is similar, although the solution with several OLs is naturally more accurate.

the graph would have used.

Figure 8.4: Making complex models usable for photon tracing: an explicitly modeled tree, which consists of roughly 120000 CSG primitives and is covered by a hierarchy of orientation lightmaps.

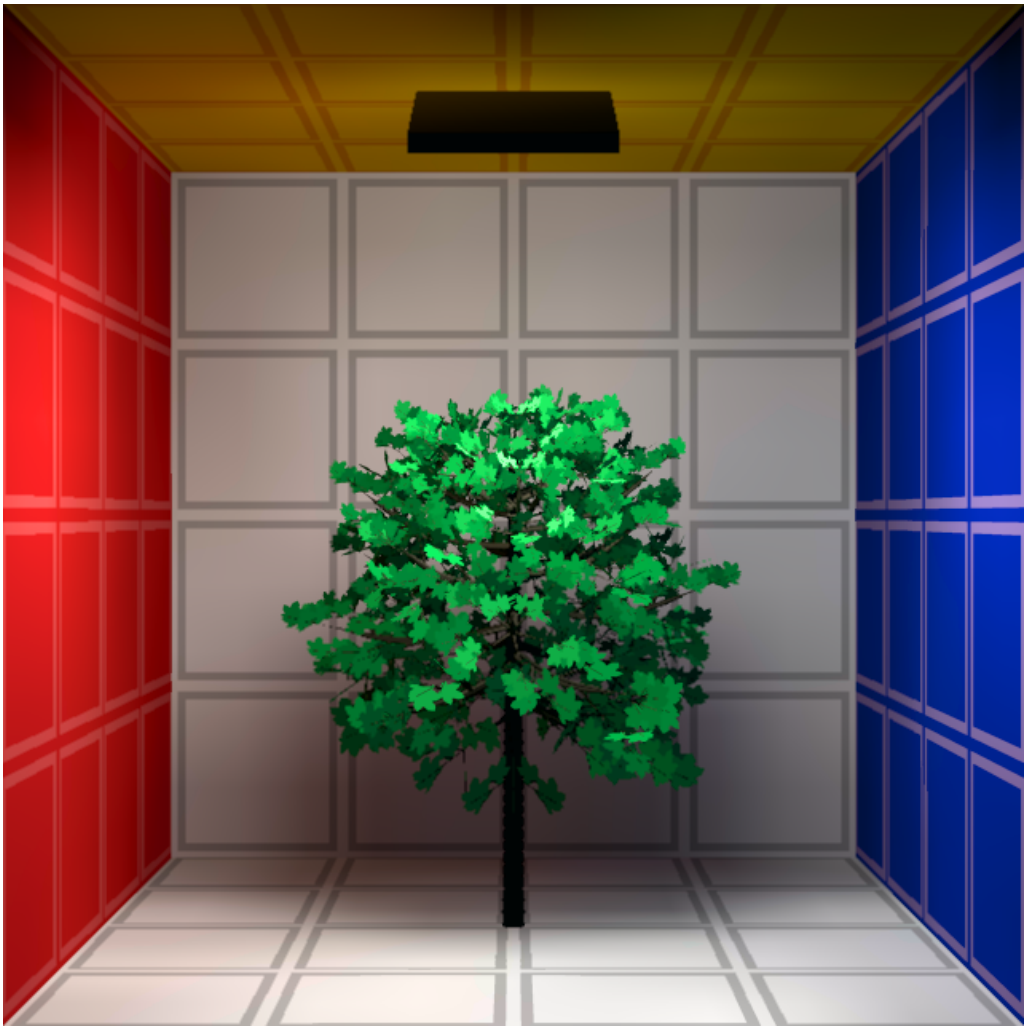


Figure 8.5: Orientation lightmaps on rule–defined objects. A simple tree generated by two rulesets: one for growing the main stem, the other for growing branches at the appropriate locations and fitting shape, with twigs and leaves attached. The tree generator ruleset is, apart from the more realistic translucent maple–like leaves, similar to one of the examples used by Traxler and Gervautz [TG97]. Note that the multitude of different leaf orientations, together with the fact that multiple OLs were used (one for each branch), leads to a visual sensation that is free of discernible OL–induced artifacts.

Chapter 9

Conclusion

We presented three improvements that enable us to use a particular, physically plausible and viewpoint–independent global illumination algorithm – namely lightmap–based stochastic photon radiosity – on far more complex scenes than was possible beforehand.

1. In chapter 5, we introduced an approach that enables lightmap–based methods to operate directly on objects modelled through CSG operations, and to maintain illumination representations on them that offer a high degree of accuracy due to the exact ray–CSG object intersections used to classify photon hits.

2. In chapter 6, we presented an approximative version of lightmaps called *orientation lightmaps*. They break the problematic one–to–one correspondence between objects and lightmaps in complex scenes, and can be simultaneously used on multiple objects. They average the photon hits on these objects, but do not alter the energy flow in the scene during the simulation process in any way.

3. Based on these orientation lightmaps, we presented a straightforward way of combining rule–based object generation and photon tracing in a way that appropriately exploits the advantages of both in order to make viewpoint–independent global ilumination possible for procedurally defined objects in chapter 7.

In addition to making the use of DCSGs in conjunction with photon radiosity possible for the first time, our extensions efficiently address the problems with respect to complex scenes that we identified with the original photon tracing algorithm at the end of section 3.5.2:

1. *Memory consumption:* By comparison with the best photon tracing method for normal scenes so far, the photon maps of Jensen [Jen96, Jen97], our combined methods offer the advantage of using far less memory; with increasing scene complexity this difference even increases in favour of our method.

2. *Variance of the solution:* By averaging all hits on a complex object orientation lightmaps manage to obtain a sufficiently stable solution from a reasonable number of photon hits, irrespective of the number of geometric primitives in the object, while incurring artifacts that are in most cases apparently not particularly prominent.

3. *Area estimation:* We presented a stochastic area sampling algorithm that is optimally suited for use in conjunction with orientation lightmaps, and that has excellent performance characteristics even for complex compound objects.

The downside to the orientation lightmap part of our approach is that the solution one obtains by their use is practically always locally inaccurate to some degree. However, since it is an add–on to the conventional lightmap method, it leaves the control of to which extent it is being employed, and hence what error is incurred, to the scene designer.

When seen in context, we deem the artifacts incurred to be a fair price for being able to render highly complex scenes with a physically based global illumination model without needing huge system resources, especially since one has the possibility of using the proposed orientation lightmaps only in perceptually less important parts of a scene if extremely high accuracy is desired.

Appendix A

Selected Physical Aspects of Rendering

In order to underline the importance of the stochastic global illumination techniques we aim to extend in this thesis, especially when compared with the various deterministic approaches from chapter 2, this appendix gives a brief overview of three aspects of photorealistic graphics which are impossible to integrate into a viewpoint–independent global illumination calculation by means other than Monte Carlo rendering. These are dispersion of light in transparent objects, and the inclusion of polarization information and fluorescence effects in the image synthesis process.

A.1 Dispersion in Dielectric Materials

Dispersion occurs where polychromatic light is split into its spectral components on a refractive material boundary, due to the fact that the index of refraction in transparent materials is dependent on the wavelength of the incident light. To complicate matters, this dependency on wavelength is non-linear and related to material constants that have to be measured in experiments.

A.1.1 Sellmeier Coefficients

The most widely used method of specifying the dispersion curve for materials in the visual range is to use the so–called *Sellmeier approximation*. Several basically similar forms exist that differ only in the number of empirical constants in structurally similar equations. The number of these constants usually depends on the measurement process by which the data for the approximation is obtained.

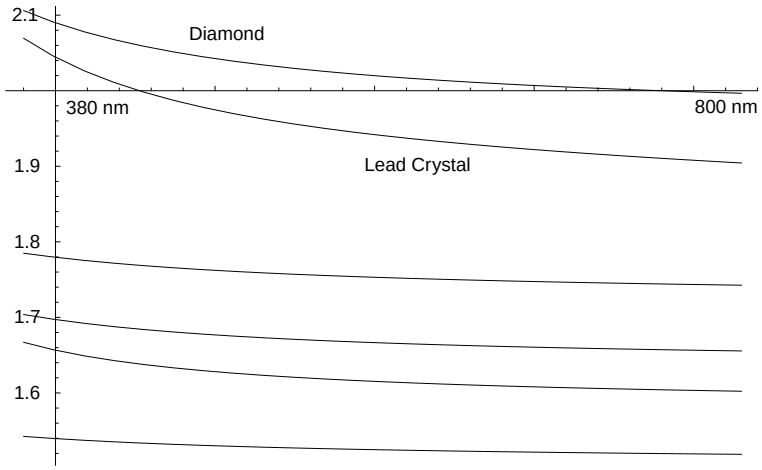


Figure A.1: Refractive indices for several materials. From top: diamond, lead crystal and several other glass types. Notice the varying amount of dispersion and non–linearity for different materials.

A typical example is the glass catalog of the company Schott Glaswerke, which is one of the worldwide leading suppliers of technical glass. In the catalog the technical data of the several hundred types of glass that the company sells is listed, and for specifying dispersion the form

$$n^2(\lambda) - 1 = \frac{B_1\lambda^2}{\lambda^2 - C_1} + \frac{B_2\lambda^2}{\lambda^2 - C_2} + \frac{B_3\lambda^2}{\lambda^2 - C_3} \tag{A.1}$$

is used, where n is the index of refraction at wavelength λ. The catalog lists coefficient values of B_n and C_n for the different glass types (ranging from normal window glass to highly dispersive lead crystal). In this particular case one can compute the index of refraction for wavelengths from ultraviolet to far infrared with a relative error of less than 1.0E-5 from just six coefficients per glass type. This makes the catalog a valuable source for accurate dispersion data, especially since it can be downloaded from the company website free of charge and contains specimens of all the main basic glass types (i.e. flints, crowns, lead crystal aso.).

There are also numerous other sources of similar freely available material measurements where one can obtain measurements of genuine materials other than glass (e.g. diamonds and other gemstones), both on the web and in book form. @BookMinnaert:1954:LCO, author = ”M. Minnaert”, title = ”Light and Color in the Open Air”, year = ”1954”, publisher = ”Dover”,

A.1.2 Photon Tracing Dispersion

The inclusion of dispersion in a rendering system brings about an effect which "standard" deterministic raycasting cannot handle: that a previously sharply defined primary ray suddenly fans out across a solid angle [WTP00b]. However, such an effect is readily traceable by spectral versions of Monte Carlo photon tracing methods.

If spectra are represented by n non–overlapping spectral bands, a photon tracer, on encountering a refraction with dispersion, shoots n new photons, each with only the colour contribution of channel n different from zero, in the direction corresponding to the average wavelength of the band, and additionally jitters this average wavelength by as much as half the width of the spectral band in order to uniformly cover the entire spectrum.

Due to the spreading of each contribution across one neigbouring band, the maximum fan–out in this case is 2, which in conjunction with multiple interreflections could lead to a considerable slowdown compared with the deterministic version, especially when nested interreflections are viewed. However, in practice we found the rendering times to be only up to two times slower than in the deterministic case, indicating that the increase in fan–out affects only the first refraction.

If no jittering is performed, the resulting reduction to n frequencies, and hence n discrete angles of refraction, leads to geometric aliasing, which can pose a serious problem when dispersion effects are closely viewed, and when noticeable dispersion caustics are present. However, for images that exhibit only small dispersion effects, like e.g. coloured fringes in glasses, the deterministic approach, which uses slightly less CPU time, is perfectly valid.

The fan–out on subsequent refractions is 1 in this case, since for each band a monochromatic photon, which cannot be split any further, is propagated. The maximum increase in computation time is n–fold for each photon that enters a dispersive medium.

A.2 Polarization Effects

Polarization has received particularly little attention in the rendering community because – while of course being essential for specially contrived setups that e.g. contain polarizing filters – it seemingly does not contribute very prominent effects to the appearance of an average scene. This misconception is in part fostered by the fact that the human eye is normally not capable of distinguishing polarized from unpolarized light[1].

[1]Contrary to common belief trained observers *can* distinguish polarized from unpolarized light with the naked eye. Named after its discoverer, the effect is known as *Haidinger's brush* and is

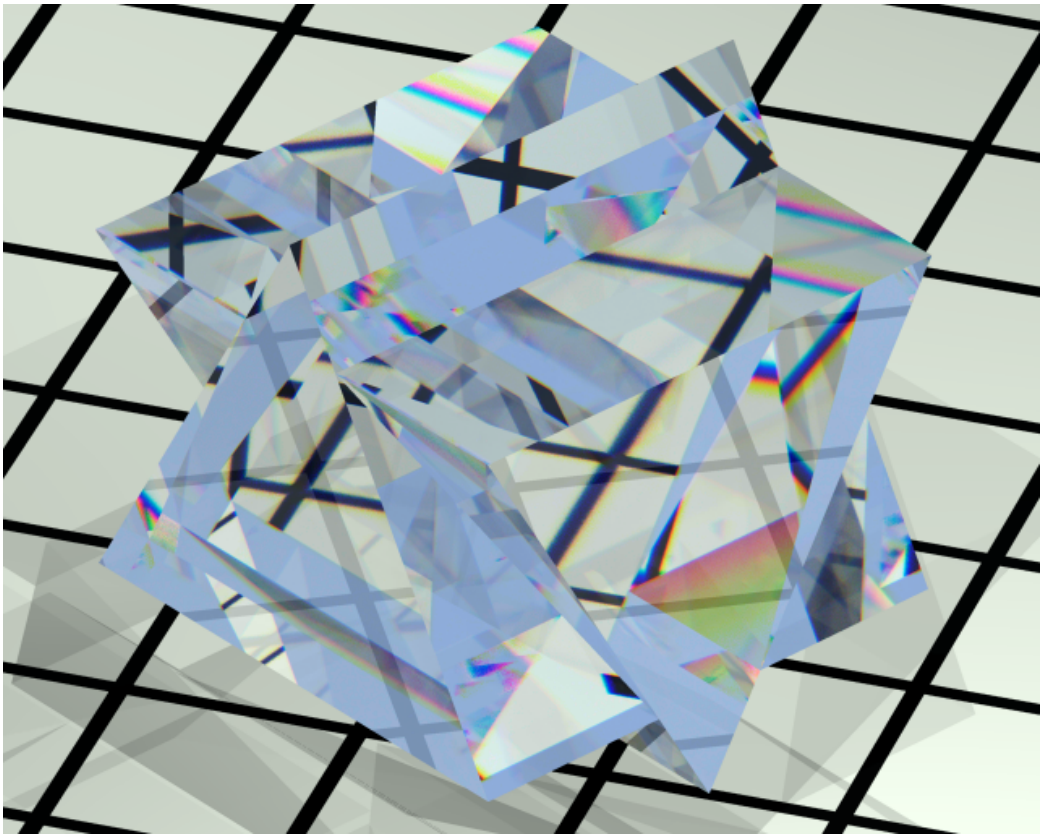


Figure A.2: Dispersion effects on a double prism, calculated with a stochastic raytracer.

One of the main areas where it in fact does make a substantial difference are outdoor scenes; this is due to the usually quite strong polarization of skylight, as one can find documented in G. P. Können's book [Kön85] about polarized light in nature. But since such scenes are currently still problematical for photorealistic renderers for a number of other, more obvious reasons (e.g. scene complexity and related global illumination issues), this has not been given a lot of attention yet. Other known effects which depend on polarization support are certain darkening or discolourization patterns in metal objects and their reflections, and the darkening of certain facets in transparent objects such as crystals.

described by Minnaert in his book about light in outdoor surroundings [Min54].

A.2.1 Polarized Light

While for a large number of purposes it is sufficient to describe light as an electromagnetic wave of a certain frequency that travels linearly through space as a discrete ray (or a set of such rays), closer experimental examination reveals that such a wavetrain also oscillates in a plane perpendicular to its propagation. The exact description of this phenomenon requires more that just the notion of radiant intensity, which the conventional representation of light provides.

The nature of this oscillation can be seen from the microscopic description of polarization, which closely follows that given by Shumaker [Shu77]. We consider a single steadily radiating oscillator (the light source) at a distant point of the negative Z–axis, and imagine that we can record the electric field[2] present at the origin due to this oscillator. Except at distances from the light source of a few wavelengths or less, the Z component of the electric field will be negligible and the field will lie in the X–Y plane. The X and Y field components will be of the form

$$\begin{aligned} E_x &= V_x \cdot (2\pi \cdot \nu \cdot t + \delta_x) \quad [\mathrm{V} \cdot \mathrm{m}^{-1}] \\ E_y &= V_y \cdot (2\pi \cdot \nu \cdot t + \delta_y) \end{aligned} \tag{A.2}$$

where V_x and V_y are the amplitudes $[\mathrm{V} \cdot \mathrm{m}^{-1}]$, ν is the frequency [Hz], δ_x and δ_y are the phases [rad] of the electromagnetic wavetrain, and t is the time [s]. Figure A.3 illustrates how this electric field vector E changes over time for four typical configurations.

Causes of Light Polarization

Apart from skylight, it is comparatively rare for light to be emitted in polarized form. In most cases, polarized light is the result of interaction with transmitting media or surfaces. The correct simulation of such processes is at the core of predictive rendering, so a short overview of this topic recommends itself.

The simplest case is that of light interacting with an optically smooth surface. This scenario can be adequately described by the *Fresnel equations*, which are solutions to Maxwell's wave equations for light wavefronts. They have been used in computer graphics at least since Cook and Torrance proposed their famous reflectance model [CT81], and most applications use them in a form which is simplified in one way or another.

[2] The electric and magnetic field vectors are perpendicular to each other and to the propagation of the radiation. The discussion could equally well be based on the magnetic field; which of the two is used is not important.

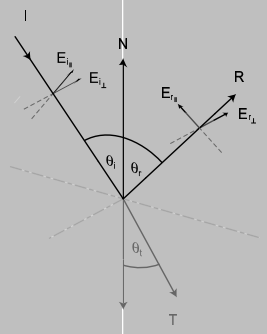


Figure A.3: **Left:** Four examples of the patterns traced out by the tip of the electric field vector in the X–Y plane: a) shows light which is linearly polarized in the vertical direction; the horizontal component E_x is always zero. b) is a more general version of linear polarization where the axis of polarization is tilted by an angle of α from horizontal, and c) shows right circular polarized light. The fourth example d) shows elliptically polarized light, which is the general case of equation (A.2). (Image redrawn from Shumaker [Shu77]) **Right:** Geometry of a ray–surface intersection with an optically smooth phase boundary between two substances, as described by the equation set (A.3). A transmitted ray T only occurs in when two dielectric media interface; in this case, all energy that is not reflected is refracted, i.e. $T = I - R$. The E–vectors for the transmitted ray $\mathrm{E}_{\mathrm{t}_{\|}}$ and $\mathrm{E}_{\mathrm{t}_{\perp}}$ have been omitted for better picture clarity. The $(\mathrm{E}_{\|}, \mathrm{E}_{\perp})$ components here correspond to the (x,y) components in the drawing on the left.

Fresnel Terms

In their full form (the derivation of which can e.g. be found in [SH92]), they consist of *two* pairs of equations. According to the reflection geometry in figure A.3, the first pair determines the proportion of incident light which is reflected separately for the x and y components of the incident wavetrain. This relationship is commonly known, and can be found in numerous computer graphics textbooks.

The second pair, which is much harder to find in computer graphics literature, descibes the *retardance* that the incident light is subjected to, which is the relative phase shift that the vertical and horizontal components of the wavetrain undergo during reflection. In figure A.4 we show the results for two typical materials: one conductor, a class of materials which has a complex index of refraction and is always opaque, and one dielectric, which in pure form is usually transparent, and has a real–valued index of refraction.

We quote the Fresnel equations for a dielectric–complex interface. This is the general case, since only one of two media at an interface can be conductive (and hence opaque), and a dielectric–dielectric interface with two real–valued indices

of refraction can also be described by this formalism.

$$
\begin{aligned}
F_{\perp}(\theta,\eta) &= \frac{a^2+b^2-2a\cos\theta+\cos^2\theta}{a^2+b^2+2a\cos\theta+\cos^2\theta} \\
F_{\parallel}(\theta,\eta) &= \frac{a^2+b^2-2a\sin\theta\tan\theta+\sin^2\theta\tan^2\theta}{a^2+b^2+2a\sin\theta\tan\theta+\sin^2\theta\tan^2\theta}F_{\perp}(\theta,\eta) \\
\tan\delta_{\perp} &= \frac{2\cos\theta}{\cos^2\theta-a^2-b^2} \\
\tan\delta_{\parallel} &= \frac{2b\cos\theta[(n^2-k^2)b-2nka]}{(n^2+k^2)^2\cos^2\theta-a^2-b^2} \\
&with \\
2a^2 &= \sqrt{(n^2-k^2-\sin^2\theta)^2+4n^2k^2}+n^2-k^2-\sin^2\theta \\
2b^2 &= \sqrt{(n^2-k^2-\sin^2\theta)^2+4n^2k^2}-n^2+k^2+\sin^2\theta
\end{aligned}
\tag{A.3}
$$

$F_{\parallel}$ is the reflectance component parallel to the plane of incidence, and $F_{\perp}$ that normal to it. Under the assumption that one is only interested in the radiant intensity of the reflected light, this can be simplified to the commonly used average reflectance $F_{average} = (F_{\perp}+F_{\parallel})/2$. $\delta_{\perp}$ and $\delta_{\parallel}$ are the retardance factors of the two wavetrain components.

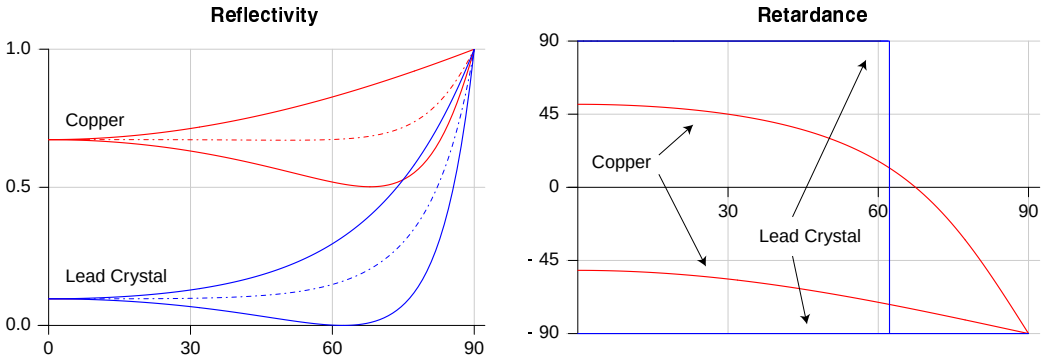


Figure A.4: Fresnel reflectivities $F_{\parallel}$, $F_{\perp}$ and $F_{average}$ (dashed lines), as well as parallel and perpendicular retardance values for copper (red) and lead crystal (blue) at 560nm. As a conductor, copper has a complex index of refraction, does not polarize incident light very strongly at Brewster's angle and exhibits a gradual shift of retardance over the entire range of incident angles. For lead crystal, with its real–valued index of refraction of about 1.9, total polarization of incident light occurs at about 62°. Above this angle, no change in the phase relation of incident light occurs (both retardance components are at $-90°$), while below Brewster's angle a phase difference of 180° is introduced.

A.3 Fluorescence

While the polarization of light at a phase boundary is a comparatively macroscopic phenomenon, fluorescence is caused by processes within the pigment molecules that are responsible for the colour of an object. Due to both lack of space, and the fact that an actual explanation of these processes is not necessary to properly implement support for it in a rendering system, we will not go into details about its causes.

The results of the phenomenon – which are what we try to simulate – are quite straightforward to describe: the characteristic property of fluorescent materials is that they re–emit portions of the incident light at different, lower wavelengths within an extremely short time (typically 10^{-8} seconds).

Instead of the reflectance spectra used for normal pigments, describing such a material requires knowledge of its *re–radiation matrix*, which encodes the energy transfer between different wavelengths. Such *bispectral* reflectance measurements are rather hard to come by; while "normal" spectrophotometers are becoming more and more common, the bispectral versions of such devices are by comparison very rare and in an experimental stage. Figure A.5 shows three visualizations of a sample bispectral reflectance dataset. Manual design of such

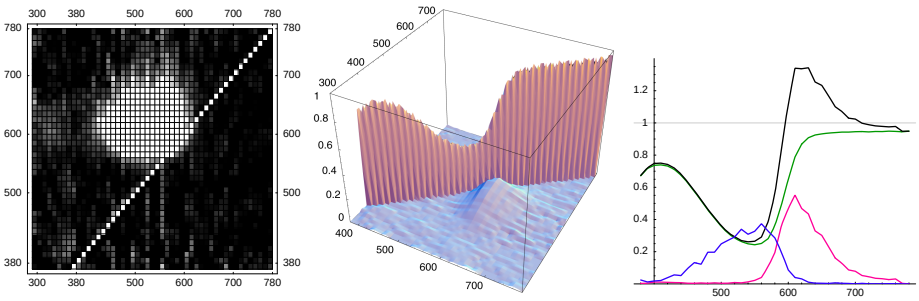


Figure A.5: Bispectral reflectivity measurements of pink fluorescent 3M Post-It® notes. The re–radiation matrix is shown for excitation wavelengths between 300nm and 780nm, and emission wavelengths from 380nm to 780nm, as 2D density plot and 3D graph. In the 3D view the off–axis contribution had to be exaggerated in order to be properly visible, and in both plots measurement noise is evident. The rightmost graph shows the nonfluorescent reflection spectrum (the main diagonal of the re–radiation matrix, shown in green), the energy absorbed at higher wavelegths (blue), the energy re–radiated at lower wavelengths (red) and the resulting "reflection" spectrum (black). Note that the resulting spectrum is well over 1.0 in some areas. Data courtesy of Labsphere Inc.

re–radiation matrices is much harder than explicit derivation of reflection spectra;

while the latter is already not particularly easy, their effect is by comparison still quite predictable. Also, it is easy to maintain the energy balance of normal reflection spectra by ensuring that no component is greater than one; for a re–radiation matrix this translates to the more difficult condition that the integral over the area must not exceed one.

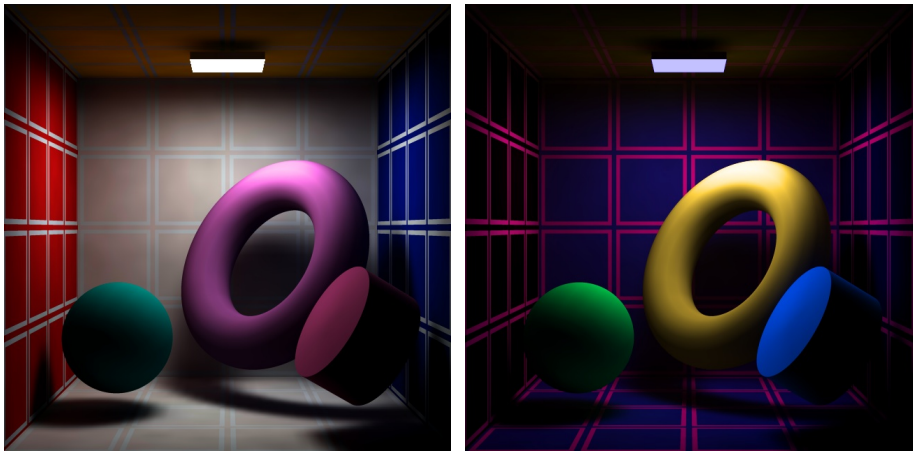


Figure A.6: A typical photon tracing radiosity box, lit by D65 and UV blacklight. The objects and the walls are partly coloured with empirically modelled fluorescent pigments. It has to be noted that these pigments were tailored for maximal fluorescence effect, and not physical plausibility. While pigments with such a bright and markedly monochromatic fluorescence effect exist, some of these do not look realistic under daylight. The aubergine-coloured torus and cylinder are the most problematic in this respect, since such extreme colour changes between blacklight and normal illumination are not common in real life.

Bibliography

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[Bek99] Philippe Bekaert. *Hierarchical and Stochastic Algorithms for Radiosity*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, 1999.

[CCWG88] 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.

[CG85] Michael F. Cohen and Donald P. Greenberg. The Hemi-Cube: A radiosity solution for complex environments. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):31–40, August 1985.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 137–45, July 1984.

[CPC98] R. Cook, T. Porter, and L. Carpenter. Distributed raytracing. In Rosalee Wolfe, editor, *Significant Seminal Papers of Computer Graphics: Pioneering Efforts that shaped the Field*, pages 77–86, N.Y., 1998. ACM Press.

[CRMT91] Shenchang Eric Chen, Holly E. Rushmeier, Gavin Miller, and Douglas Turner. A progressive multi-pass method for global illumination. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 165–174, July 1991.

[CT81] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *Computer graphics, Aug 1981*, 15(3):307–316, 1981.

[ES80] Peter Eichhorst and Walter J. Savitch. Growth functions of stochastic Lindenmayer systems. *Information and Control*, 45(3):217–228, June 1980.

[Fed96] Martin Feda. A Monte Carlo approach for Galerkin radiosity. to appear in *The Visual Computer*, 12, 1996.

[Ger95] R. Gershbein. A study of integration methods for couplings of galerkin radiosity systems. In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

[GN71] Robert A. Goldstein and Roger Nagel. 3-D visual simulation. *Simulation*, 16(1):25–31, January 1971.

[GT96] Michael Gervautz and Christoph Traxler. Representation and realistic rendering of natural phenomena with cyclic CSG graphs. *The Visual Computer*, 12(2):62–71, 1996. ISSN 0178-2789.

[HD91] John C. Hart and Thomas A. DeFanti. Efficient anti-aliased rendering of 3D linear fractals. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 91–100, July 1991.

[Hec90a] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 145–154, August 1990.

[Hec90b] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. volume 24, pages 145–154, August 1990.

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):197–206, July 1991.

[ICG86] 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.

[JC98] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 311–320. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 21–30, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[Jen97] Henrik Wann Jensen. Rendering caustics on non-lambertian surfaces. *Computer Graphics Forum*, 16(1):57–64, 1997. ISSN 0167-7055.

[Kaj86] James T. Kajiya. The rendering equation. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):143–150, August 1986.

[Kön85] G. P. Können. *Polarized Light in Nature*. Cambridge University Press, 1985.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[Lin68] A. Lindenmayer. Mathematical models for cellular interactions in development, I & II. *Journal of Theoretical Biology*, 18:280–315, 1968.
Lindenmayer's original articles on L-Systems.

[LW93a] Eric Lafortune and Yves D. Willems. Bi-directional path tracing. pages 145–153, Alvor, Portugal, December 1993.

[LW93b] Eric P. Lafortune and Yves D. Willems. Bi-directional Path Tracing. In H. P. Santo, editor, *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, December 1993.

[LW94] E. P. Lafortune and Y. D. Willems. A theoretical framework for physically based rendering. *Computer Graphics Forum*, 13(2):97–107, June 1994.

[Min54] M. Minnaert. *Light and Color in the Open Air*. Dover, 1954.

[MP96] Radomír Měch and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 397–410. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[NH91] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In *Visualization '91*, pages 83–91, 1991.

[NPT+95] László Neumann, Werner Purgathofer, Robert F. Tobler, Attila Neumann, Pavol Eliáš, Martin Feda, and Xavier Pueyo. The stochastic ray method for radiosity. In P. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95*, pages 206–218. Eurographics, Springer-Verlag, June 1995.

[PJM94] Przemyslaw Prusinkiewicz, Mark James, and Radomiŕ Měch. Synthetic topiary. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 351–358. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[PLH88] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental models of herbaceous plants for computer imagery purposes. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 141–150, August 1988.

[PM92] S. N. Pattanaik and S. P. Mudur. Computation of global illumination by monte carlo simulation of the particle model of light. *Third Eurographics Workshop on Rendering*, pages 71–83, May 1992.

[Pru86] Przemyslaw Prusinkiewicz. Graphical applications of L-systems. In M. Green, editor, *Proceedings of Graphics Interface '86*, pages 247–253, May 1986.

[Pru87] Przemyslaw Prusinkiewicz. Applications of L-Systems to Computer Imagery. In Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg, and Azriel Rosenfeld, editors, *Proc. 3rd Int. Workshop on Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 534–548. Springer-Verlag, 1987.

[Rot82] S. D. Roth. Ray casting for modelling solids. *Computer Graphics and Image Processing*, 18:109–144, February 1982.

[RPV93] Holly Rushmeier, Charles Patterson, and Aravindan Veerasamy. Geometric simplifications for indirect illumination calculations. In *Graphics Interface '93*. Canadian Human–Computer Communication Society, May 1993.

[RTJ94] Erik Reinhard, Lucas U. Tijssen, and Frederik W. Jansen. Environment mapping for efficient sampling of the diffuse interreflection. In *Photorealistic Rendering Techniques*, pages 410–422. Springer-Verlag, June 1994.

[RV89] Jaroslaw R. Rossignac and Herbert B. Voelcker. Active zones in CSG for accelerating boundary evaluation, redundancy elimination, interference detection, and shading algorithms. *ACM Transactions on Graphics*, 8(1):51–87, 1989.

[RW80] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. volume 14, pages 110–116, July 1980.

[SG69] J. Spanier and E. Gelbard. *Monte Carlo Principles and Neutron Transport Problems*. Addison Wesley Publishing Company, 1969.

[SH81] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corp., Washington, DC, 1981.

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

[Shi90] Peter Shirley. A ray tracing method for illumination calculation in diffuse-specular scenes. In *Proceedings of Graphics Interface '90*, pages 205–212, May 1990.

[Shi91] P. Shirley. Time complexity of monte carlo radiosity. In Werner Purgathofer, editor, *Eurographics '91*, pages 459–465. North-Holland, September 1991.

[Shu77] John B. Shumaker. Distribution of optical radiation with respect to polarization. In Fred E. Nicodemus, editor, *Self–Study Manual on Optical Radiation Measurements, Part 1: Concepts*. Optical Physics Division, Institute for Basic Standards, National Bureau of Standards, Washington, D.C., June 1977.

[SKDP99] L. Szirmay-Kalos, P. Dornbach, and W. Purgathofer. On the start-up bias problem of metropolis rendering. In *WSCG '99 (Seventh International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media)*, pages 273–280, Plzen-Borey, Czech Republic, February 1999. University of West Bohemia.

[SP89] 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.

[SPNP96] Mateu Sbert, Xavier Pueyo, Lazlo Neumann, and Werner Purgathofer. Global multipath monte carlo algorithms for radiosity. *The Visual Computer*, 12(2):47–61, 1996. ISSN 0178-2789.

[SW99a] F. Suykens and Y. D. Willems. Combining bidirectional path tracing and multipass rendering. In *WSCG '99 (Seventh International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media)*, pages 265–272, Plzen-Borey, Czech Republic, February 1999. University of West Bohemia.

[SW99b] Frank Suykens and Yves D. Willems. Weighted multipass methods for global illumination. In P. Brunet and R. Scopigno, editors, *Computer Graphics Forum (Eurographics '99)*, volume 18(3), pages 209–220. The Eurographics Association and Blackwell Publishers, 1999.

[SW00] Frank Suykens and Yves D. Willems. Density control for photon maps. In B. Peroche and H. Rushmeier, editors, *Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering)*, pages 23–34, New York, NY, 2000. Springer Wien.

[SWH+95] Peter Shirley, Bretton Wade, Philip Hubbard, David Zareski, Bruce Walter, and Donald P. Greenberg. Global illumination via density estimation. In P. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95*, pages 219–230. Eurographics, Springer-Verlag, June 1995.

[TG97] Christoph Traxler and Michael Gervautz. Efficient ray tracing of complex natural scenes. World Scientific Publishers, 1997.

[TGP96] Robert F. Tobler, Thomas M. Galla, and Werner Purgathofer. Acsgm — an adaptive csg meshing algorithm. In *CSG 96 —Set-theoretic Solid Modelling Techniques and Applications*, pages 17–31. Information Geometers, 1996.

[TWFP97] Robert F. Tobler, Alexander Wilkie, Martin Feda, and Werner Purgathofer. A hierarchical subdivision algorithm for stochastic radiosity methods. In Julie Dorsey and Philipp Slusallek, editors, *Euro-*

graphics Rendering Workshop 1997, pages 193–204, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.

[VG94a] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Fifth Eurographics Workshop on Rendering*, pages 147–162, Darmstadt, Germany, June 1994.

[VG94b] Eric Veach and Leonidas Guibas. Bidirectional Estimators for Light Transport. In *Fifth Eurographics Workshop on Rendering*, pages 147–162, Darmstadt, Germany, June 1994.

[VG95] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. *Computer Graphics*, 29(Annual Conference Series):419–428, November 1995.

[VG97] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 65–76. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[WCG87] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 311–320, July 1987.

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

[WP95] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 119–128. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[WRC88] 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.

[WTP98] Alexander Wilkie, Robert F. Tobler, and Werner Purgathofer. Photon radiosity lightmaps for CSG solids. In *CSG 98 - Set-theoretic Solid Modelling: Techniques and Applications*, Ammerdown, England, April 1998. Information Geometers.

[WTP00a] A. Wilkie, R. F. Tobler, and W. Purgathofer. Orientation lightmaps for photon tracing in complex environments. In *Proceedings of the Conference on Computer Graphics International (CGI-00)*, pages 279–286, Los Alamitos, CA, June 19–24 2000. IEEE.

[WTP00b] A. Wilkie, R. F. Tobler, and W. Purgathofer. Raytracing of dispersion effects in transparent materials. In *Proceedings of the Winter School of Computer Graphics (WSCG00)*, Plzen, Czech Republic, February 2000.

[Zat93] Harold R. Zatz. Galerkin radiosity: A higher order solution method for global illumination. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 213–220, 1993.