OPTIMISING DISCONTINUITY MESHING RADIOSITY

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
IN THE FACULTY OF SCIENCE

By

Neil Gatenby
Department of Computer Science

January 1995

✓ Th19342
(DLAXE)
85720118

Contents

List of Figures

Abstract

UNIVERSITY OF MANCHESTER

ABSTRACT OF THESIS submitted by **Neil Gatenby** for the Degree of Doctor of Philosophy in Computer Science and entitled **Optimising Discontinuity Meshing Radiosity**

Month and Year of Submission: January 1995

Radiosity, in computer graphics, is a rapidly-expanding area of research. From its meagre beginnings in 1984, modelling an empty room with N patches, using $\mathcal{O}(N^2)$ storage and time, it has progressed to a stage where highly complex scenes can be modelled, fully accounting for occlusion, with only linear storage and time costs. Energy transfer is evaluated hierarchically, basis functions can be high order polynomials, and discontinuities can be accurately modelled.

This thesis presents a review of such algorithms, focussing on those methods which address the problems of discontinuities in the radiosity function. A number of problems with existing methods, are identified, and a number of optimisations and alternatives are presented.

Declaration

Copyright & Ownership

Dedication

This one's for MD and Pops, not forgetting J., and never forgetting Katy.

Acknowledgements

I would like to express my sincere thanks to my supervisors, Prof. Frank Sumner and Mr. Terry Hewitt. This thesis would not have been possible without their time, help and patience. Terry, in particular, has devoted much discussion and time to my research; his insights have been invaluable on more than one occasion.

I would also like to thank my partner, Jan, for putting up with me and for her skill in ducking early drafts as they flew across the room! My office mate, Lin Fenqiang, has been particularly helpful: always being available to discuss ideas, and share a joke. Thanks also go to a number of members of staff at the Computer Graphics Unit: these include Tony Arnold, Andrew Grant, Steve Larkin, Helen Morphet and Yien Kwok — all of whom helped me in various ways at various stages, whilst researching this work. Thanks also to Dave Hutchinson, for being a useful sounding board. The rest of the CGU posse are too numerous to mention, but they know who they are, and I thank them for a number of lunchtime adventures.

My friends; Lynn, Rebs-the-hip, Russ, Greg, Guy, Helen and Alison all deserve thanks for the times when they have been supportive.

Chapter 1

Introduction

Computer generated images, which an observer might mistake for a photograph of a *real* scene, are becoming more and more commonplace in the developed world. The borders between what is computer generated, and what is not, are becoming more and more blurred: the people developing the algorithms, used to generate the images, have the benefit of more than 20 years of computer graphics literature to peruse, and access to state of the art hardware which is cheaper, bigger and faster than ever before. No surprise then, that the images being produced are attaining the goal of those producing them: realism.

1.1 Physically correct *versus* Perceptually acceptable

In order to reach a goal, one must first define it: this thesis concerns itself with the generation of realistic images, using computers. What is meant by *realistic* in this context? How does one decide which image generation method produces the more realistic image? The remainder of this section addresses these issues.

'Photorealism' is a word much used in computer graphics literature, and less often qualified. Literally, a photorealistic image is one which a person might mistake for a photograph of a real scene. The question arises: is it possible, using a computer, to generate a photorealistic image of a scene[1], given only a detailed description of the scene, but given no access to a camera, or the scene itself (which may be no more than some plans on a designer's desk)? Indeed, are 'photorealistic images' quite what the people generating the images are striving for? Is a 'photorealistic computer generated image' indistinguishable from a photograph of the scene being modelled? For a large number of applications, the answer to these last two questions is 'no'! What is often required is an image which only:

- gives the observer so many perceptual cues that they instantly recognise all of the objects making up the scene, as well as their relative sizes, orientations, etc., and
- minimises those features in the image which might trigger perceptual responses that tell the observer that the image is computer generated.

These requirements only address the issue of *perception*; no mention of *accuracy* has been made: how closely does the modelled light distribution correspond to the actual distribution in the real world? [5, 69] Indeed, how should one measure

[1]For the purposes of this thesis, a "scene" is a description of a collection of "objects", some of which will (typically) be emitting light. The gas occupying the space around the objects is assumed to play no part in the way in which the objects become illuminated.

this 'closeness'? [43] Suppose, for example, that some scene is being modelled using two different rendering systems, each producing an image using the same view. Given any point in the scene, the first rendering system can very accurately estimate how much light, of a given wavelength, is leaving the point in a given direction. The second rendering system, is unable to provide any such information. Suppose further, that when a large sample of people were asked to compare the two images, and say which was computer generated, and which was not, the majority chose the first system's image as being computer generated, and the second system's image as being the 'real thing'. Which image is the more realistic?

For the purposes of this thesis, where *physical accuracy* is a major concern, the preferred image is the first one. For the purposes of film and TV special effects, or product advertising, where human perceptual issues are paramount, the preferred image would be the second one.

It should be noted, however, that concentrating on physical accuracy, rather than on what is perceptually pleasing, does not mean that the resulting images will appear unrealistic. Quite the contrary: where the physically-based approach differs from the perceptually-based approach is in its *route* to visual realism – reached *via* attempting to model what is actually happening to the light in between it leaving a light source, and arriving at the eye. If the physical model is accurate enough, the image will appear realistic. The more accurate the modelling, the more realistic the image. Indeed, one could assert that the physically-based approach is the more *certain* route to realism.

1.2 Modelling reality

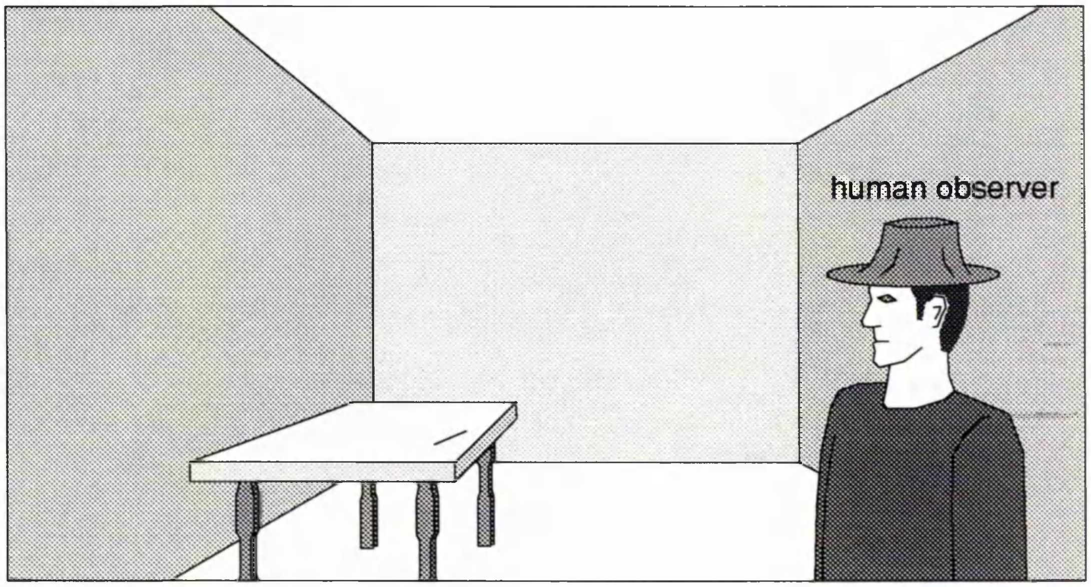
So, *physical accuracy* is a major concern in this thesis: primary applications of the work described here include lighting, heating[2] and architectural design. These are real-world applications, requiring real-world physical units to work with. For example, when an architect is designing a building, they would like to make use of a computer system which can tell them about the light distribution across the desktops (say) in a room: are they well-lit by daylight? Are extra light fittings needed? Are the windows too small, or poorly positioned? What difference does it make when a particular wall is moved? When a particular column is added? In summertime? In winter? How does the situation change when the walls are painted? When the carpet is changed? ... Any number of combinations can be tried out on the computer. As long as the simulation is physically accurate, the result should be a well-designed building, whose lighting is ideally suited to those who use it.

Exactly how does one proceed, when creating a *physically accurate* model of illumination in the real world, using a computer?

Modelling is achieved by storing a computer representation of the shape, position and orientation of the surfaces that make up the scene, together with their emittance and reflectance properties (figure 1.1). This representation, hereinafter referred to as the *scene model*, is repeatedly referenced to determine quantities such as inter-object visibility, separation, and orientation, as well as various surface properties. These values, in turn, are inserted into a mathematical model

[2]A change of wavelengths is all that is needed.

(i) The scene being modelled



(ii) The model of the scene

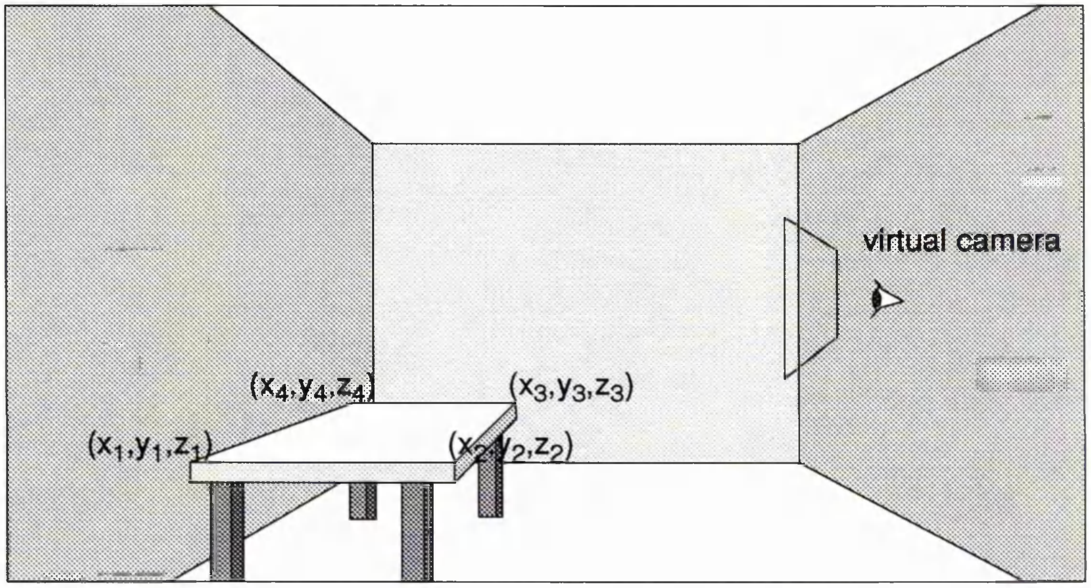


Figure 1.1: Some differences between the real scene, and the scene model

(the *transport model*) which defines how light interacts with the scene. The transport model takes information gleaned from the scene model and returns information about how various parts of the scene illuminate other parts. Typically, this process iterates until enough information to create an image, of the desired accuracy, has been found. Often, this is carried out for a number [74, 48] of different wavelengths of visible light, so that colour images can be generated.

The complexity of the scene geometry, and the intricacy of the emittance and reflectance information, are the factors which determine how difficult the problem will be to solve. The physical limits of the computer, together with time limitations that the user has, mean that it is only possible to obtain illumination information for a finite set of points in the scene. The physical limits of the display device, means that it is only *necessary* to have a solution for a finite set of points. Limitations such as these, make simplifications usual in the scene and transport models. Figure 1.1 illustrates some typical simplifications:

The observer

- In the real world, there is an observer: several feet tall, usually with binocular vision, and generally not very reflective! Their very presence will affect the light distribution in the scene.
- In the computer model, the observer is replaced by a virtual camera, which has no affect whatsoever on the modelled light distribution, and typically only provides one view of the scene.

Object detail

[3]Although this is hardly an issue for radiosity algorithms.

- Some objects in the scene being modelled may have very complex shape (e.g., the table legs).
- It may be that the modeller is not interested in modelling small details in the scene (e.g., table leg shape, pencil on table) and these may be simplified or ignored in the model.

Reflectance properties

- In the real scene, surfaces may well change their reflectance properties as a function of position, incoming and outgoing directions, wavelength and/or temperature.
- In the model, it is unlikely that all of these will variables be accounted for.

To conclude; some simplifications are unavoidable, some are desirable, and some will have a more significant effect on the results than others. The subsequent sections detail the development of a transport model for computer-generated images, bearing these limitations in mind.

1.3 Physically-based rendering

In order to model the steady-state[4] light distribution in a scene Γ, one must first construct a transport model which: not only handles how the light sources in Γ illuminate the other objects (*direct* illumination); but also how inter-object reflections and/or transmissions (*indirect* illumination) affect the final distribution.

[4]Throughout this thesis, all quantities are assumed to be time-independent.

It is important to decide how many degrees of freedom are to be allowed in the model being constructed – do we want to know how much light, of any given wavelength, leaves any given point in Γ in any given direction? Or, will the *total* amount of light, leaving each object in Γ, regardless of direction, be sufficient? Or, should the model cater for something in between these two extremes? The answer is that the ideal model will be application and environment dependent: for radiation heat transfer applications [99, 103], the less accurate approach may well be sufficient, but for many computer graphics applications [100], something nearer the first approach will be needed. The 'environment dependency' refers to the hardware being used to run the model; it simply may not be practical, on some platforms, to run too intricate a model.

What follows, is a derivation of the *rendering equation*[5], first introduced to the computer graphics community by Kajiya in [61]. For now, few simplifying assumptions are made, only that the surfaces are opaque, and the solution is steady (i.e., is not varying with time).

1.3.1 Some nomenclature and units

As has already been stated, the resulting model must be able to provide the user with information which can be directly applied to real-world situations: a set of physical units is needed, which can describe all relevant quantities. One must be able to quantify:

[5]Also known as the *radiance equation* [98].

- how much light leaves the various light sources, and how this is spectrally and directionally distributed;
- how different surfaces reflect light incident upon them: how does this vary with position, incoming and outgoing directions, wavelength?

With these criteria in mind, this section introduces a number of definitions, terms and units which can be used to quantify the flow of light through the scene. Most of the terminology is taken from the illumination engineering community because, unlike the thermal radiation heat transfer community – which spawned the early radiosity work in computer graphics [43], they have standardised their terminology [89], and it is now becoming the norm to use it for computer graphics [98, 84]:

directional notation: When referring to the direction in which light is travelling, relative to a point $\mathbf{x}$ on some surface in the scene, a local spherical coordinate system is assumed. This system is centred at $\mathbf{x}$, and measures an angle θ between the direction and the surface normal at $\mathbf{x}$, and an angle ϕ between the direction and some fixed, arbitrary, surface tangent at $\mathbf{x}$ (see figure 1.2). The single bold character $\mathbf{\Theta}$ denotes the pair (θ, ϕ).

solid angle: The solid angle subtended by an object, at a point, is equivalent to the surface area of the radial projection of the object, onto the unit sphere about the point. Normally referred to by the symbol ω, it is measured in `steradians` (sr), and can be regarded as a sort of 'field-of-view' indicator.

radiant flux: Light is a form of electromagnetic radiation. Radiation is a flow of energy – a *flux*. The energy of a packet of light rays is referred to as its *radiant energy* (Q), and is measured in `joules` (J). The rate at which this flow of radiant energy changes with time, is known as the *radiant flux* ($\Phi = dQ/dt$), and is measured in `joules per second` (Js^{-1}), or `watts` (W). Radiant flux corresponds to *power*; the amount of energy per unit time, and is often referred to as the *radiant power* [77, 98].

radiant intensity: Whilst, at first sight (sic), radiant flux may seem like a suitable quantity for the transport model under construction, one should note that it has no dependence on viewing angle, which *should* be incorporated into the model. The quantity *radiant intensity* ($I = d\Phi/d\omega$) is a measure of the radiant flux per unit solid angle. This is measured in `watts per steradian` ($Wsr^{-1} = Js^{-1}sr^{-1}$) and represents the radiant flow from a point source, along a particular direction.

radiant exitance: The quantity used to measure how radiant flux varies as a function of surface area, is *radiant exitance* (M); measured in `watts per metre squared` (Wm^{-2}). This measures the radiant flux leaving a surface, per unit surface area. In heat transfer (and consequently computer graphics) nomenclature, this is referred to as *radiosity* ($B = d\Phi_{\text{out}}/dA$) – a term that shall be adopted for the remainder of this thesis.

irradiance: Whereas *radiosity* is a measure of radiant flux leaving a surface, *irradiance* ($E = d\Phi_{\text{in}}/dA$), also measured in `watts per metre squared` (Wm^{-2}), is a per-unit-area measure of the radiant flux *incident* on the surface.

radiance: Whilst radiant intensity details how radiant flux varies with viewing angle, and radiosity details how it varies with surface area, a term which covers how radiant flux varies with *both* these quantities is still needed. The

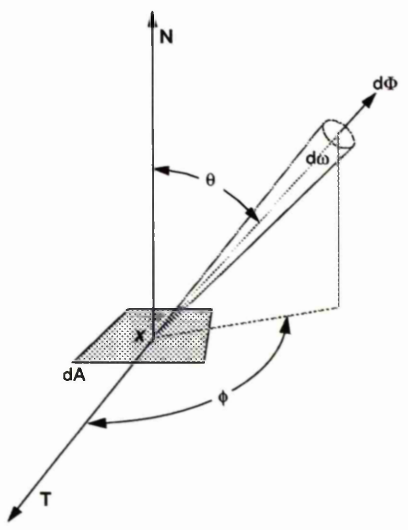


Figure 1.2: Terms used in the definition of *radiance*.

radiance

$$L = \frac{1}{\cos\theta}\frac{d^2\Phi}{dA d\omega} \tag{1.1}$$

along a specified direction $\mathbf{\Theta} = (\theta, \phi)$, is the radiant flux per projected surface area per unit solid angle, and is measured in **watts per metre squared per steradian** ($Wm^{-2}sr^{-1}$). See figure 1.2. Note also, that radiance is the radiant intensity per projected surface area ($L = \frac{1}{\cos\theta}\frac{dI}{dA}$). Radiance closely corresponds to the 'colour' of an object, as noted by a human observer, but is independent of the size of the object being viewed, and the distance to it. As such, it is ideal for use in the model under construction.

reflectance: To quote from [77]: "*Reflection* is the process by which electromagnetic flux, incident on a stationary surface or medium, leaves that surface or medium from the incident side without change in frequency. *Reflectance* is the fraction of the incident flux that is reflected."

It is common practise, in computer graphics, to assume that no (monochromatic) light which is incident upon a stationary surface will leave that surface from the incident side *with* a change in frequency. This simplification means that each wavelength of interest can be modelled separately, considerably easing the computational difficulty. When referring to any of the terms introduced in this section on a 'per wavelength' basis, the word 'spectral' should be placed in front of the term to indicate that this is the case (*spectral radiant flux*, *spectral radiant intensity*, etc., ...). However, wherever these terms appear in this thesis, it is the spectral quantity that is being referred to – the reader should take it as being implicitly present.

bidirectional reflectance-distribution function: Whilst reflectance reveals *some* information about how a surface reflects light, it does not incorporate the directional dependence which is needed by the transport model under development. The bidirectional reflectance-distribution function (commonly abbreviated to *brdf* (f_r), units are sr^{-1}) is the reflected radiance along some outgoing direction, divided by the casual irradiance from some incoming direction.

$$f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) = \frac{dL_o(\mathbf{x}, \Theta_o, \lambda)}{L_i(\mathbf{x}, \Theta_i, \lambda) d\omega_i \cos\theta_i} \tag{1.2}$$

Further details of any unfamiliar terms can be found in figure 1.3 and table 1.1.

It is worth noting that the *brdf* satisfies the *Helmholtz reciprocity rule* [99], which asserts that swapping the incoming and outgoing directions will not affect the fraction of light being reflected:

$$f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) = f_r(\mathbf{x}, \Theta_o, \Theta_i, \lambda) \tag{1.3}$$

Quantity	Symbol	Units	Explanation
radiant energy	Q	J	energy carried by packet of rays.
radiant flux	Φ	W	rate of change of Q w.r.t. time.
radiant intensity	I	Wsr^{-1}	radiant flux per unit solid angle.
radiant exitance	M	Wm^{-2}	see radiosity.
radiosity	B	Wm^{-2}	Φ leaving surface, per unit area.
irradiance	E	Wm^{-2}	Φ reaching surface, per unit area.
radiance	L	$Wm^{-2}sr^{-1}$	Φ per projected surface area per unit solid angle.
brdf	f_r	sr^{-1}	reflected radiance along Θ_o over causal irradiance from Θ_i.

Table 1.1: Illumination Engineering nomenclature used throughout this thesis.

Numerous attempts at deriving a useful analytic formulation for the *brdf* (in terms of incoming/outgoing directions and material parameters) have been described in the computer graphics and optics literature over a number of years [114, 113, 117, 86, 12, 26, 60, 51, 126, 122, 68, 91]. Not all of these formulations are physically-based, and few have been applied to models which simulate *global* illumination. Others, however, have taken great care to observe issues of physical accuracy [51, 122, 68, 91]. Many of these models are discussed in [36].

1.4 The rendering equation

With this comprehensive nomenclature in mind, what can one assert about the radiance leaving a point $\mathbf{x}$ (in Γ), along the direction $\Theta_o = (\theta_o, \phi_o)$, due to light incident from the direction $\Theta_i = (\theta_i, \phi_i)$ only? From the definition of *brdf*, (1.2),

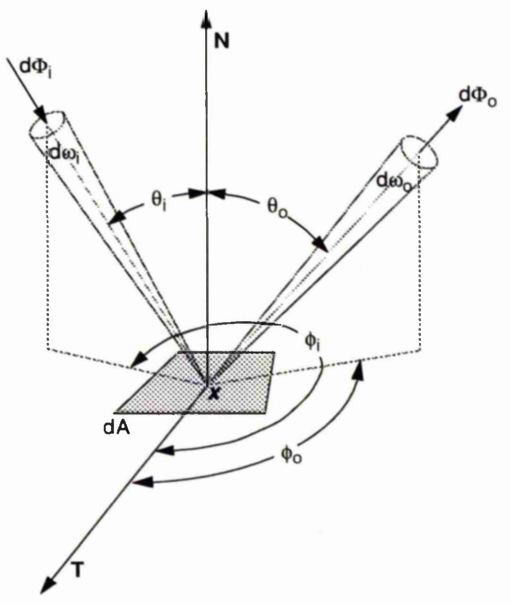


Figure 1.3: Terms used in the definition of *brdf*.

one can see:

$$dL_o(\mathbf{x}, \mathbf{\Theta}_o, \lambda) = f_r(\mathbf{x}, \mathbf{\Theta}_i, \mathbf{\Theta}_o, \lambda) L_i(\mathbf{x}, \mathbf{\Theta}_i, \lambda) \cos\theta_i d\omega_i \tag{1.4}$$

In order to account for contributions from *all* light incident at **x**, not just that which is incident from the direction $\mathbf{\Theta}_i$, one must integrate (1.4) over the hemisphere of incoming directions, Ω_i, above **x**:

$$L_o(\mathbf{x}, \mathbf{\Theta}_o, \lambda) = \int_{\Omega_i} f_r(\mathbf{x}, \mathbf{\Theta}_i, \mathbf{\Theta}_o, \lambda) L_i(\mathbf{x}, \mathbf{\Theta}_i, \lambda) \cos\theta_i d\omega_i \tag{1.5}$$

Adding in one extra term, L_e, to account for light being *emitted* from **x**, in the relevant direction, gives a simple version of the rendering equation:

$$L_o(\mathbf{x}, \mathbf{\Theta}_o, \lambda) = L_e(\mathbf{x}, \mathbf{\Theta}_o, \lambda) + \int_{\Omega_i} f_r(\mathbf{x}, \mathbf{\Theta}_i, \mathbf{\Theta}_o, \lambda) L_i(\mathbf{x}, \mathbf{\Theta}_i, \lambda) \cos\theta_i d\omega_i \tag{1.6}$$

Unfortunately, the equation is not very useful in this form: the aim is to solve for L_o, but the integral is still in terms of L_i – the radiance incident at **x**.

For the purposes of this thesis, the medium in which the scene sits is being regarded as a vacuum. This means that the radiance incident at $\mathbf{x}$, along the direction Θ_i, is equal to the radiance *leaving* some point $\mathbf{x}'$, in Γ, found by tracing a ray from $\mathbf{x}$ down the direction Θ_i:

$$L_i(\mathbf{x}, \Theta_i, \lambda) = L_o(\mathbf{x}', \Theta_i, \lambda) \tag{1.7}$$

Figure 1.4 illustrates this point, and shows how it is possible to change the integrand in (1.6) so that rather than considering the radiance arriving at $\mathbf{x}$ from all directions visible to it, one considers the radiance leaving all points $\mathbf{x}'$ visible to it, and travelling towards it. This change of integrand,

$$d\omega_i = \frac{\cos\theta' dA(\mathbf{x}')}{\|\mathbf{x} - \mathbf{x}'\|^2} \tag{1.8}$$

together with a change of integration limits, so that the integration now takes place over those points in Γ which can see $\mathbf{x}$ (the set $V(\mathbf{x}, \Gamma)$), results in:

$$L_o(\mathbf{x}, \Theta_o, \lambda) = L_e(\mathbf{x}, \Theta_o, \lambda) + \int_{V(\mathbf{x},\Gamma)} f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) L_o(\mathbf{x}', \Theta_i, \lambda) \frac{\cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} dA(\mathbf{x}') \tag{1.9}$$

The right-hand side of the equation is now in terms of L_o, as desired, but the integration limits are still odd (how does one determine which points lie in $V(\mathbf{x}, \Gamma)$?). The integration limits can be changed to cover the whole of Γ (a superset of $V(\mathbf{x}, \Gamma)$) and a visibility term $g(\mathbf{x}, \mathbf{x}')$ is introduced. This new term takes the value 1 if the straight line joining $\mathbf{x}$ and $\mathbf{x}'$ passes through no other part of Γ, and zero, otherwise. This leaves, the fundamental result of this section, the rendering equation:

$$L_o(\mathbf{x}, \Theta_o, \lambda) = L_e(\mathbf{x}, \Theta_o, \lambda) + \int_{\Gamma} g(\mathbf{x}, \mathbf{x}') f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) L_o(\mathbf{x}', \Theta_i, \lambda) \frac{\cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} dA(\mathbf{x}') \tag{1.10}$$

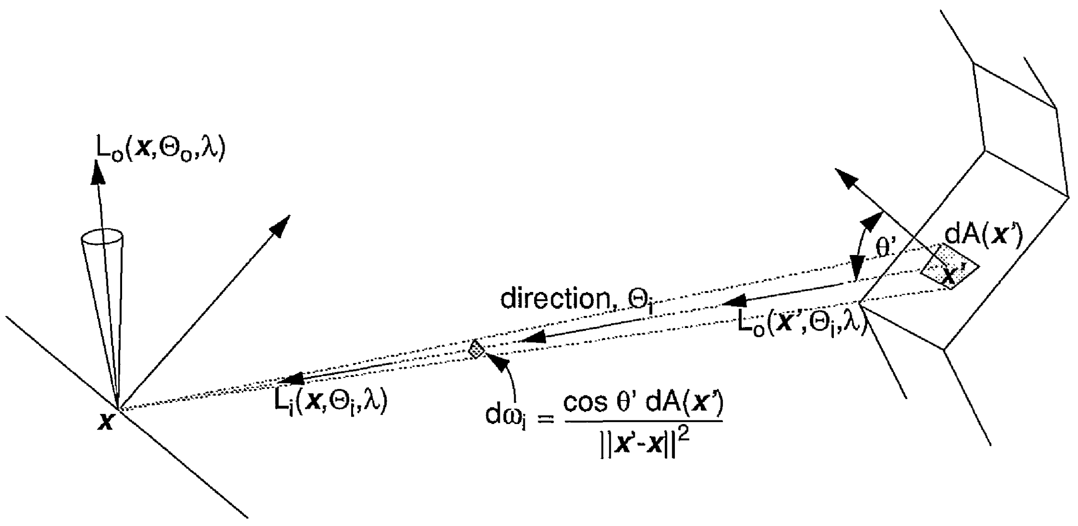

$L_o(x,\Theta_o,\lambda)$
direction, Θ_i
$L_o(x',\Theta_i,\lambda)$
θ'
$dA(x')$
x'
$L_i(x,\Theta_i,\lambda)$
$d\omega_i = \frac{\cos\theta' \, dA(x')}{\|x'-x\|^2}$
x

Figure 1.4: Geometrical considerations when changing the integrand in the rendering equation.

This equation expresses the radiance leaving any point in the scene, in any direction, in terms of the same quantity for all other points (and directions) in the scene. The recursive nature of the equation is clear: in order to find one value of L_o (for some point and direction), one must first find enough other values of L_o (for appropriate points and directions) to be able to evaluate the integral on the right hand side of (1.10). In order to find any of these other values, one must first .... (recursion).

1.4.1 A different parameterization

The rendering equation is often seen parameterized differently to (1.10) where, rather than thinking in terms of directions relative to the point x, the problem is formulated solely in terms of points on surfaces in Γ (figure 1.5):

$$L_o(\mathbf{x},\mathbf{x}'',\lambda) = L_e(\mathbf{x},\mathbf{x}'',\lambda)+\int_\Gamma g(\mathbf{x},\mathbf{x}')f_r(\mathbf{x}',\mathbf{x},\mathbf{x}'',\lambda)L_o(\mathbf{x}',\mathbf{x},\lambda)\frac{\cos\theta_i\cos\theta'}{\|\mathbf{x}-\mathbf{x}'\|^2}dA(\mathbf{x}') \tag{1.11}$$

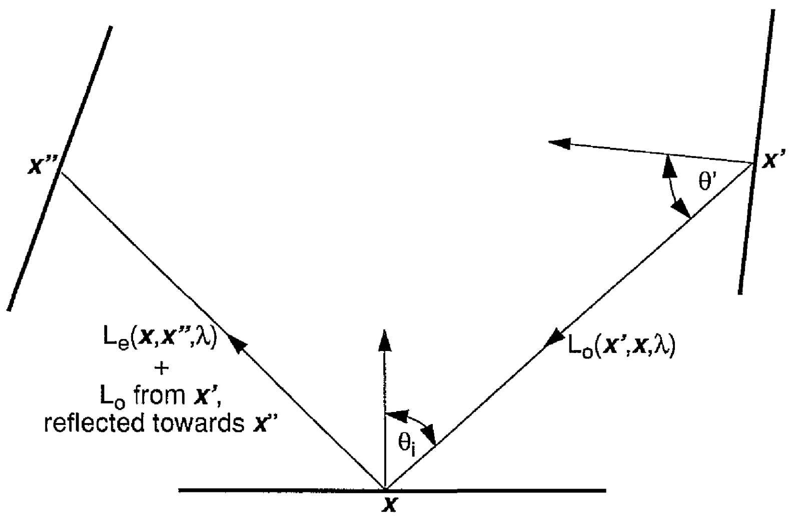


Figure 1.5: Understanding the spatial parameterization of the rendering equation.

Here, the radiance $L_o(\mathrm{x}, \mathrm{x}'', \lambda)$, leaving x in the direction of some other point x'', is written as the sum of:

- any emitted radiance leaving x which is travelling towards x'', and
- the radiance which has left points x' in Γ, only to be reflected at x towards x''.

Note that the *brdf* is now written in terms of three points, rather than one point and two directions. Schröder and Hanrahan [94] have coined the terms *directional* and *spatial* parameterizations for (1.10) and (1.11), respectively.

1.5 Algorithm classification

Recall (1.10):

$$L_o(\mathrm{x}, \Theta_o, \lambda) = L_e(\mathrm{x}, \Theta_o, \lambda) + \int_\Gamma g(\mathrm{x}, \mathrm{x}') f_r(\mathrm{x}, \Theta_i, \Theta_o, \lambda) L_o(\mathrm{x}', \Theta_i, \lambda) \frac{\cos\theta_i \cos\theta'}{\|\mathrm{x} - \mathrm{x}'\|^2} dA(\mathrm{x}')$$

Some of the surfaces in the scene, Γ, will be emitting light (the subset Γ_L), the remainder will not (the subset Γ_R). Expressing the rendering equation in terms of these two distinct subsets:

$$\begin{aligned}
L_o(\mathbf{x},\Theta_o,\lambda) &= L_e(\mathbf{x},\Theta_o,\lambda) + \\
&\int_{\Gamma_L} g(\mathbf{x},\mathbf{x}')f_r(\mathbf{x},\Theta_i,\Theta_o,\lambda)L_o(\mathbf{x}',\Theta_i,\lambda)\frac{\cos\theta_i\cos\theta'}{\|\mathbf{x}-\mathbf{x}'\|^2}dA(\mathbf{x}') + \\
&\int_{\Gamma_R} g(\mathbf{x},\mathbf{x}')f_r(\mathbf{x},\Theta_i,\Theta_o,\lambda)L_o(\mathbf{x}',\Theta_i,\lambda)\frac{\cos\theta_i\cos\theta'}{\|\mathbf{x}-\mathbf{x}'\|^2}dA(\mathbf{x}') \quad (1.12)
\end{aligned}$$

(1.13)

The integral over Γ_L represents the part of $L_o(\mathbf{x},\Theta_o,\lambda)$ which is due to light emitting surfaces only. The Γ_R integral details how the non-emitting surfaces contribute. The way in which different illumination algorithms handle these two integrals can be used to place them into two separate categories:

- Those algorithms which make no real effort to evaluate the Γ_R integral at all, and approximate the Γ_L integral by replacing the radiance term $L_o(\mathbf{x}',\Theta_i,\lambda)$ inside the integral with its emissive component only ($L_e(\mathbf{x}',\Theta_i,\lambda)$), are known as *local illumination* models.

- Those algorithms which make a genuine attempt to solve both of the integrals in (1.13), are known as *global illumination* models.

Local illumination models concern themselves only with how each light source illuminates each point of interest. Global illumination models recognise that not only will those surfaces visible to the light sources be directly illuminated, but also that the surfaces visible to *these* surfaces will be indirectly illuminated, and so on, *ad infinitum*. Local models are of no further interest here, their failure to account to for inter-object illumination means that they return inaccurate results in all but the most simple scenes[6]. They are, however, discussed in some detail in [36].

Illumination algorithms may be further classified, by examining how they *approach* the problem under consideration. Those algorithms that first establish which regions of the scene are of interest to the viewer, and then go on to establish how those regions are illuminated, are known as *image space* algorithms. Those algorithms which calculate how the light sources have illuminated each surface in the scene, *before* considering the viewer, are known as *object space* algorithms.

Image space algorithms may very well not find out *any* information about how large areas of a scene are illuminated; the areas being of little relevance to the particular view. Object space algorithms always have at least some information about how *every* point in the scene is lit.

This thesis is primarily concerned with global illumination, object space, models. Global illumination because *accuracy* is important, and object space because not all applications are solely concerned with image quality; many being equally interested in having data on exactly how the various surfaces in the scene were illuminated.

[6]A single convex object, for example.

1.6 Thesis outline

The rendering equation, derived in this first chapter, is the fundamental transport model upon which all global illumination algorithms are based. The remainder of this thesis devotes itself to examining various solution methods for equations of this type, with a particular emphasis being placed on methods which specifically account for *discontinuities* in the solution [54].

Chapter 2 examines some classical solutions of the rendering equation. 'Classical', in this sense, refers to algorithms which were published before the research for this thesis, began. Various radiosity and ray tracing algorithms are examined, compared and contrasted.

Chapter 3 reviews the theory and practise of higher order radiosity methods. That is, algorithms which are amenable to approximating the radiosity across a surface by a piecewise polynomial function, rather than simply a piecewise constant function. Chapter 3 covers Galerkin radiosity, collocation radiosity and wavelet radiosity.

Chapter 4 looks at the theory of discontinuities in the radiosity function being modelled, and examines which of these discontinuities should be regarded as being significant. A review is presented, of a number of algorithms which have accounted for such discontinuities in their meshing, solution and rendering phases.

Chapter 5 presents a number of optimisations for the methods described in chapter 4. These optimisations are the central premise of this thesis. Some results, demonstrating the effectiveness of the various improvements, are also presented.

Chapter 6 is the concluding chapter.

Appendix A presents a précis of binary space partitioning trees and shadow visibility binary space partitioning trees. An explanation of the winged-edge data structure is given in appendix B, and appendix C contains images and statistics from the implemented code.

Chapter 2

Solving the rendering equation: classical solutions

This chapter presents a review of classical solution methods for the rendering equation (1.10). For all but the simplest scenes, equations of this type have no known analytic solution, and numerical approximation methods are the norm. This chapter emphasises the radiosity method, since it is of most relevance to the remainder of the thesis, but competing solution strategies are also examined, and their various merits discussed.

2.1 Radiosity from radiance

It is clear that the rendering equation (1.10) would be a lot simpler to solve if the directional dependence was taken out of the *brdf*. Not all applications will have surfaces whose reflectance properties are so complicated. For example, many surfaces will have reflectance characteristics which do not vary as a function of ϕ (figure 1.3) In other applications, reflectance properties may be closely approximated by a much simpler model than precisely the *brdf*: if the height of surface irregularities is of the order of a wavelength or more, the reflectance is mostly diffuse — if the irregularities are small relative to a wavelength, most of the light is reflected specularly [10].

Whatever the reasons are, and some of them are historical (the simpler problem was solved first), the following steps detail how the rendering equation (1.10) changes, when all surfaces are assumed to be ideal Lambertian emitters and reflectors.

2.1.1 Ideal Lambertian surfaces

Lambertian surfaces, by definition, obey Lambert's Law [64], which states that 'the luminous intensity in any direction from any element of area of the surface is equal to the intensity of the element seen in the direction of the normal to the surface multiplied by the cosine of the angle between the normal and the direction considered'. A change of units, and the introduction of projected area, rewords this assertion: *The reflected radiance does not vary as a function of outgoing direction, $\Theta_o = (\theta_o, \phi_o)$, from no matter direction the element is irradiated* (figure 2.1).

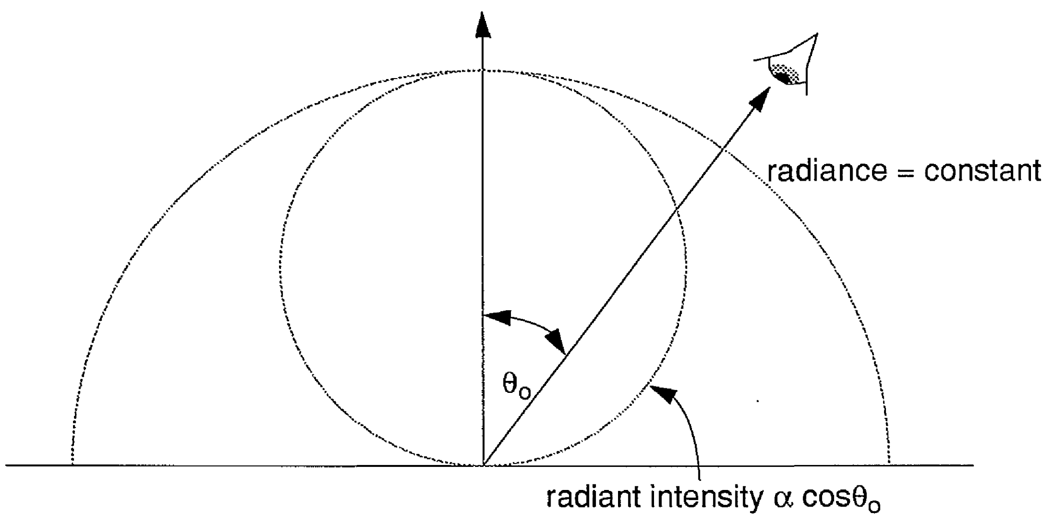


Figure 2.1: An ideal Lambertian surface.

Constant *brdf*

Recall (1.5), which expresses the reflected radiance in terms of the incident irradiance and the *brdf*:

$$L_o(\mathbf{x}, \Theta_o, \lambda) = \int_{\Omega_i} f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) L_i(\mathbf{x}, \Theta_i, \lambda) \cos\theta_i d\omega_i$$

The surface now being considered is ideal Lambertian, so the left-hand side is known to be independent of the outgoing direction, Θ_o. The only term on right-hand side which has any dependence on Θ_o, is the *brdf*; so this too must be independent of outgoing direction:

$$L_o(\mathbf{x}, \lambda) = \int_{\Omega_i} f_r(\mathbf{x}, \Theta_i, \lambda) L_i(\mathbf{x}, \Theta_i, \lambda) \cos\theta_i d\omega_i \tag{2.1}$$

Recall, however, that the Helmholtz reciprocity rule (1.3) asserts that one can swap the incoming and outgoing directions, and the value of the *brdf* will remain the same. It follows that, for an ideal Lambertian surface, the *brdf* is independent of both incoming and outgoing directions:

$$f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) = f_{r,d}(\mathbf{x}, \lambda) \tag{2.2}$$

2.1.2 The radiosity equation

The fraction of incident radiance from a given incoming direction that is reflected *anywhere*, is the *directional hemispherical reflectance*(ρ_d) [77], and is given by[1]:

$$\rho_d(\mathbf{x}, \Theta_i, \lambda) = \int_{2\pi} f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) \cos\theta_o \, d\omega_o \tag{2.3}$$

But, for an ideal Lambertian surface, $f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) = f_{r,d}(\mathbf{x}, \lambda)$, so:

$$\begin{aligned} \rho_d(\mathbf{x}, \Theta_i, \lambda) &= f_{r,d}(\mathbf{x}, \lambda) \int_{2\pi} \cos\theta_o \, d\omega_o \\ &= f_{r,d}(\mathbf{x}, \lambda) \int_0^{2\pi} \int_0^{\pi/2} \cos\theta_o \sin\theta_o \, d\theta_o \, d\phi_o \\ &= \pi f_{r,d}(\mathbf{x}, \lambda) \end{aligned} \tag{2.4}$$

Recall the rendering equation (1.10):

$$L_o(\mathbf{x}, \Theta_o, \lambda) = L_e(\mathbf{x}, \Theta_o, \lambda) + \int_\Gamma g(\mathbf{x}, \mathbf{x}') f_r(\mathbf{x}, \Theta_i, \Theta_o, \lambda) L_o(\mathbf{x}', \Theta_i, \lambda) \frac{\cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} dA(\mathbf{x}')$$

If the surfaces now being dealt with are all ideal Lambertian, then the rendering equation loses its dependence on outgoing directions, and the *brdf* can be replaced by directional hemispherical reflectance:

$$L_o(\mathbf{x}, \lambda) = L_e(\mathbf{x}, \lambda) + \rho_d(\mathbf{x}, \lambda) \int_\Gamma g(\mathbf{x}, \mathbf{x}') L_o(\mathbf{x}', \lambda) \frac{\cos\theta_i \cos\theta'}{\pi \|\mathbf{x} - \mathbf{x}'\|^2} dA(\mathbf{x}') \tag{2.5}$$

Now that the radiance no longer varies as a function of outgoing direction from a surface, it seems a strange unit to maintain. Consider the following, recalling definition (1.1):

$$L = \frac{1}{\cos\theta} \frac{d^2\Phi}{dA d\omega}$$

[1]The notation '$\int_{2\pi}$' indicates integration over a hemisphere of directions.

$$\begin{aligned} L &= \frac{1}{\cos\theta}\frac{d}{d\omega}\left(\frac{d\Phi}{dA}\right) \\ \frac{d\Phi}{dA} &= \int_{2\pi} L\cos\theta\, d\omega \end{aligned} \tag{2.6}$$

The left-hand side of (2.6) is the radiosity (B) expressed in terms of the radiant flux, Φ. For an ideal Lambertian surface, the integral on the right-hand side can be trivially evaluated, because the radiance is isotropic:

$$B = L\int_{2\pi}\cos\theta d\omega = \pi L \tag{2.7}$$

Carrying out a similar integration for all the terms in (2.5):

$$B_o(\mathbf{x},\lambda) = B_e(\mathbf{x},\lambda) + \rho_d(\mathbf{x},\lambda)\int_\Gamma g(\mathbf{x},\mathbf{x}')B_o(\mathbf{x}',\lambda)\frac{\cos\theta_i\cos\theta'}{\pi\|\mathbf{x}-\mathbf{x}'\|^2}dA(\mathbf{x}') \tag{2.8}$$

Equations (1.10), and (2.8) are both *linear Fredholm integral equations of the second kind*. These are discussed in greater detail in chapter 3; suffice to say here that they are equations in which the unknown function appears both inside and outside an integral, and that except for a few special cases [103, 93], there is no known analytic solution to problems of this type; numerical solution methods, however, abound [28, 6, 56]. In classical radiosity [58, 43], the problem is simplified, by imposing the additional assumption that the scene can be split into a finite number (N) of regions (*patches*) across each of which the radiosity only varies as a function of wavelength (figure 2.2). With this assumption in operation, integrating (2.8) over the j^{th} patch (containing the point $\mathbf{x}$) gives:

$$A_jB_j(\lambda) = A_jE_j(\lambda) + \rho_{d,j}(\lambda)\sum_{k=1}^{N}B_k(\lambda)\int_{A_j}\int_{A_k}g(\mathbf{x},\mathbf{x}')\frac{\cos\theta_i\cos\theta'}{\pi\|\mathbf{x}-\mathbf{x}'\|^2}dA(\mathbf{x}')dA(\mathbf{x}) \tag{2.9}$$

Where $B_j(\lambda)$ is the radiosity across the j^{th} patch, $E_j(\lambda)$ is the radiosity being *emitted* from this patch, and the integral over the whole scene has been written as a sum of integrals over each patch, k, in the scene.

If the *form factor*[2] F_{jk} is introduced:

$$F_{jk} = \frac{1}{A_j} \int_{A_j} \int_{A_k} g(\mathbf{x}, \mathbf{x}') \frac{\cos\theta_i \cos\theta'}{\pi \|\mathbf{x} - \mathbf{x}'\|^2} dA(\mathbf{x}') dA(\mathbf{x}) \tag{2.10}$$

... then (2.9) simplifies even further:

$$B_j(\lambda) = E_j(\lambda) + \rho_{d,j}(\lambda) \sum_{k=1}^{N} B_k(\lambda) F_{jk} \tag{2.11}$$

The form factor F_{jk} is the fraction of energy leaving the j^{th} patch which impinges *directly* on the k^{th} patch[3]. Numerous methods [36] exist for evaluating the N^2 form factors that appear in equations (2.11), but it is clear that once the form factors are known, there remains a linear of system of N equations, which can be solved using any number of methods to find the N unknowns, B_j.

It is an interesting aside to note that whilst (2.11) has been the mainstay of classical radiosity [58, 43, 22], its derivation has typically involved taking a more simplistic view of the situation from the start, and (2.11) is reached without resorting to integral equations (see, for example [36]).

2.2 Constant radiosity algorithms

This section presents a review of computer graphics algorithms which are based upon (2.11). Algorithms of this type, where the radiosity is approximated as a

[2]Also known as *angle factor* [103], or *configuration factor* [99]

[3]The form factor definition described by (2.10) is valid only for patches across which the radiosity does not vary as a function of position.

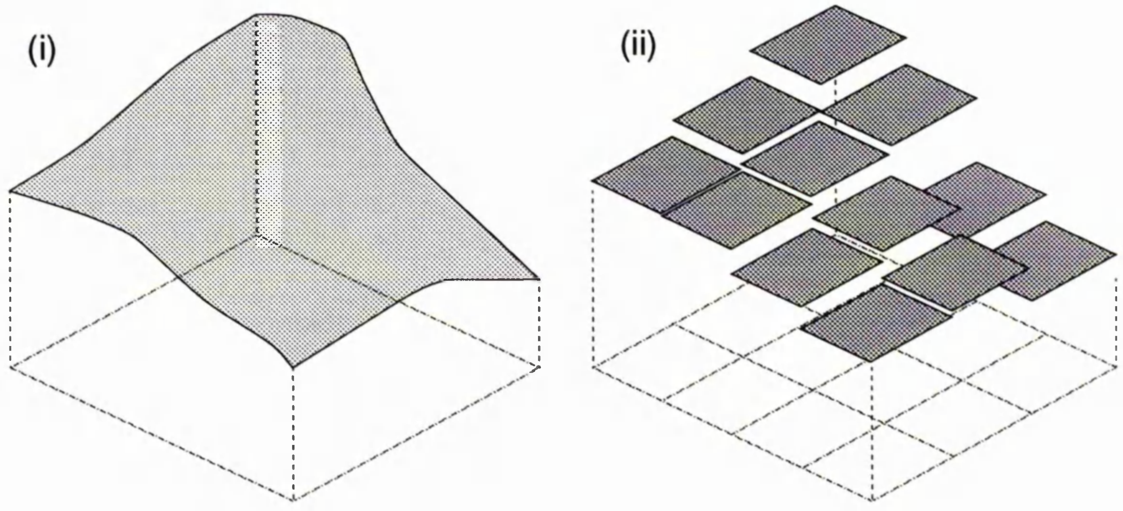


Figure 2.2: Approximating radiosity as a piecewise constant function.

piecewise constant (step) function, were first introduced to the computer graphics community by Goral *et al* in [43]. Whilst this early method did not account for inter-object occlusion (the problem was only solved for a simple cubic room), more complex algorithms were quick to follow:

2.2.1 Form factors

The problem encountered by Goral *et al*, with regard to inter-object occlusion, was their inability to evaluate the form factor between a pair of patches which were partially occluded. Goral *et al* applied Stoke's theorem [27] to the double area integral in (2.10)[4] in order to convert it to a double *contour* integral, thereby making the evaluation tractable (the patches all being planar quadrilaterals). If the two patches had been partially occluded, application of Stokes' theorem [103] would not have been possible: discontinuities in the kernel, caused by the occlusion, invalidate its application [65].

[4]with $g(\mathbf{x}, \mathbf{x}') \equiv 1$

Goral's paper was quickly followed by Cohen and Greenberg's classic *hemi-cube* paper [22], which offered a fast numerical approximation to (2.10), accurate provided that [8]:

- the patches are small compared to their separation distance,
- the patches are not *partially* occluded, and
- a suitable level of numerical precision is in use.

The hemi-cube has been the dominant method of form factor evaluation for some time, even given its drawbacks [8] and a number of competing methods introduced both before and since [81, 72, 8, 101, 120, 104, 38, 93]. Form factor evaluation methods are discussed in depth in [36]. The pertinent point here is that, even with inter-object occlusion, the desired form factors *can* be found; with *no* shortage of evaluation methods to choose from.

2.2.2 Full matrix radiosity

Recall (2.11). This describes how the j^{th} patch is illuminated by all the other patches in the scene: patch j *gathers* light from the patches around it. Notice from (2.11), that in order to see how the whole scene illuminates one patch, an entire row $\{F_{jk}\}_{k=1,\ldots,N}$ of the form factor matrix is needed. In order to evaluate the radiosity of every patch, the whole form factor matrix must first be evaluated. Consequently, algorithms of this type are known as *full matrix* (FM) radiosity methods: N^2 form factors are evaluated before any images are generated.

The question remains; given the form factor matrix, how does one utilise (2.11) to actually obtain a solution, and thence an image?

Re-arranging (2.11), with a view to solving for B_j:

$$B_j(\lambda) - \rho_{d,j}(\lambda) \sum_{k=1}^{N} B_k(\lambda) F_{jk} = E_j(\lambda)$$

Now, taking the wavelength dependency as being implicit, and re-writing in vector/matrix notation:

$$\begin{pmatrix} (1-\rho_{d,1}F_{11}) & -\rho_{d,1}F_{12} & \cdots & -\rho_{d,1}F_{1N} \\ -\rho_{d,2}F_{21} & (1-\rho_{d,2}F_{22}) & \cdots & -\rho_{d,2}F_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_{d,N}F_{N1} & -\rho_{d,N}F_{N2} & \cdots & (1-\rho_{d,N}F_{NN}) \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{pmatrix} \tag{2.12}$$

This system of equations can either be solved *directly*, which is typically an $\mathcal{O}(N^3)$ problem, or iteratively, in which case convergence can be achieved in $\mathcal{O}(N^2)$ steps. One particular iterative method, Gauss-Seidel iteration [33], has been especially favoured as a solution strategy for (2.12). The strong diagonal dominance of the matrix, typically ensuring rapid convergence. The coding simplicity of the Gauss-Seidel method, has also ensured its continued popularity within the graphics community [125, 24].

Note that, if rendered, a solution of (2.11), would appear noticeably 'blocky'. Such a solution is, after all, a representation of the radiosity as a piecewise constant function. Given small enough patches, such a solution can always be made arbitrarily accurate. Typically, however, this is not the case, and the visually displeasing 'blocky' appearance is avoided by extrapolating patch radiosities to patch vertices, and then using linear interpolation [46] across patches, when rendering [43]. This approach does not increase the *accuracy* of the solution, but it does remove the discontinuities in *value* from the rendered image — which human observers find so noticeable [7].

Patches and Elements

Discretizing the original scene geometry into N flat patches is discussed in greater detail in later chapters. Typically, the discretization is based on the programmer's experience of such issues (e.g., lots of patches where shadows are expected, few where they are not), or on the *shape* of the surface being discretized, in the case of curved, or otherwise-complicated, surfaces. It is only in recent years that consideration has been given to exactly *where* these patches might best be situated for computational efficiency [14, 54, 70].

The notions of *accuracy* and *arbitrarily positioned patches* seem to be at odds with one another: Figure 2.3 shows a surface across which the radiosity is represented by four patches, and across which the true radiosity is varying rapidly. The use of only four patches results in the expected 'blocky' appearance. Also shown, is the same surface represented by 64 patches – with the expected improvement in appearance. Unfortunately, solving (2.11) is an $\mathcal{O}(N^2)$ problem, so increasing the number of patches in this way, has a dramatic effect on the complexity of the problem.

This situation was somewhat improved, with the introduction of *elements* [23] – which patches can be divided up into. An element is some sub-region of a

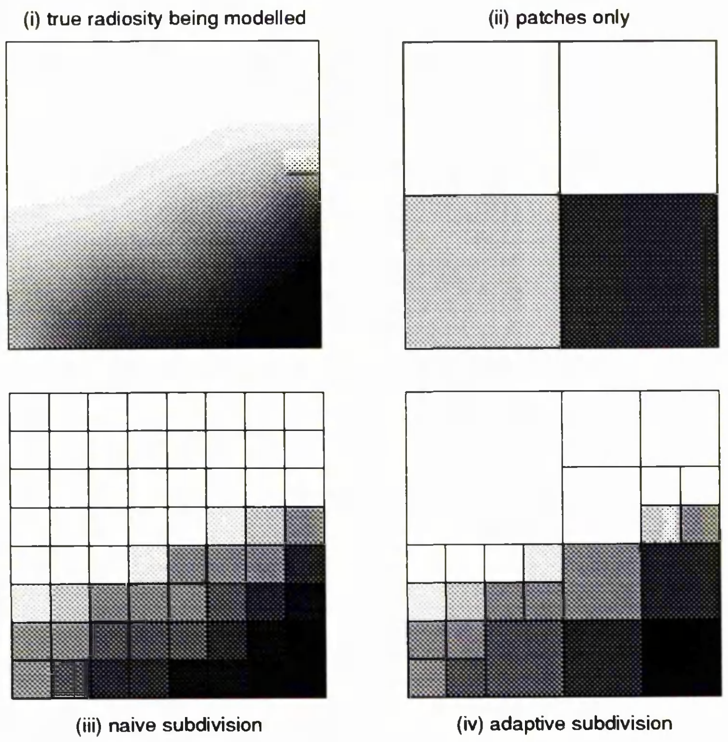


Figure 2.3: Patches subdivided into elements.

patch. Elements *receive* energy the same way patches do, by accumulating all the contributions from the patches in the scene, but they *distribute* their energy via their parent patch, whose radiosity is the area-weighted average of its element children. Consider how the first two subdivisions shown in figure 2.3, would influence the radiosity across other surfaces in the scene. Cohen *et al* [23] assert that it makes little difference to most other surfaces whether the contribution of the subdivided surface is taken patch by patch (4 calculations), or element by element (64 calculations, in this example). The cheaper option results in some impressive images and run-times, with little error.

If the N patches are split into a total of M elements, then the algorithm proceeds as follows:

1. Find the MN element-to-patch form factors.

2. Use area-weighted averaging to find the N^2 patch-to-patch form factors.
3. Solve for the patch radiosities.
4. Use the patch radiosities and the element-to-patch form factors to find the element radiosities.
5. Render, using the elements.

This is an $\mathcal{O}(N(M+N))$ problem, which is much better than $\mathcal{O}(M^2)$ for $M \gg N$.

The method can be further improved, by building a hierarchy of elements, where some initial subdivision is used to reach a fairly coarse solution. This solution is then examined, and patches or elements across which the radiosity seems to be varying significantly are subdivided further. This is known as *adaptive subdivision* [23]. Whilst it is by no means obvious what subdivision criterion should best be applied [118], the method can lead to pictures as realistic as those generated using simple subdivision, but for far less cost (figure 2.3.iii). It is interesting to note, that whilst adaptive subdivision results in a *hierarchy* of elements, contained within their parent patch, it is only the *leaf* elements of this hierarchy which take place in energy transfer. This situation is improved upon in [50].

2.2.3 Progressive refinement radiosity

The radiosity methods described in the preceding sections, resulted in some images which were unsurpassed, at that time, in terms of accuracy and realism. However, the method still had two major flaws: storage costs and time. Before one spends $\mathcal{O}(N^2)$ time solving the linear system of equations (2.11), one must

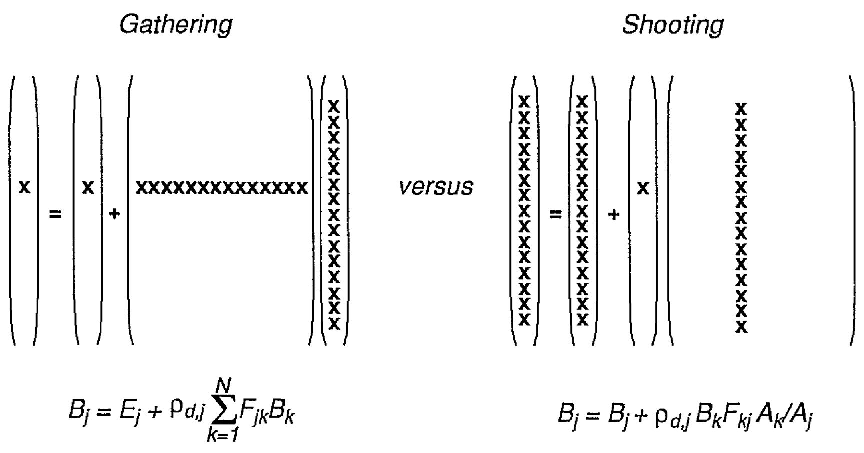

Gathering
Shooting
versus
$B_j = E_j + \rho_{d,j} \sum_{k=1}^{N} F_{jk} B_k$
$B_j = B_j + \rho_{d,j} B_k F_{kj} A_k / A_j$

Figure 2.4: A schematic comparison of gathering and shooting (after [21]).

first spend $\mathcal{O}(N^2)$ time evaluating the form factor matrix, and devote $\mathcal{O}(N^2)$ storage to keeping those values in the machine. These costs soon become prohibitive, for useful scenes, on even the largest computers.

Fortunately, a reformulation of the problem — again due to Cohen *et al* [21] — overcomes some of these difficulties. Using this new formulation, images are seen to appear quickly on the screen and, if left long enough, can converge to usefully accurate solutions in $\mathcal{O}(N)$ time. Recall (2.11):

$$B_j = E_j + \rho_{d,j} \sum_{k=1}^{N} B_k F_{jk}$$

It is clear that the contribution to the j^{th} patch, due to the k^{th} patch *only*, is the amount:

$$\rho_{d,j} B_k F_{jk} \qquad (2.13)$$

This means that if only the k^{th} column of the form factor matrix $(\{F_{jk}\}_{j=1,\dots,N})$ is evaluated, then one can find the contribution of patch k to the whole scene, in only $\mathcal{O}(N)$ calculations. Whereas in full matrix radiosity, a *gathering* process takes place, this new approach — *progressive refinement* (PR) radiosity [21] —

can be regarded as patch k *shooting* its energy out into the scene (figure 2.4).

A flaw with the method, as it has been described thus far, is that form factor evaluation methods will return a *row* of the form factor matrix in a single iteration, not a column. This problem can be circumvented by evaluating the k^{th} *row* $\{F_{kj}\}_{j=1,\ldots,N}$ of the form factor matrix, and applying the form factor reciprocity relation [36] ($A_j F_{jk} = A_k F_{kj}$) to evaluate the requisite column. The amount (2.13) now becomes:

$$\rho_{d,j} B_k F_{kj} A_k / A_j$$

It is simple enough to ensure that patch k is an important emitter; the patches can be ordered according to $A_j B_j$. The result of this single shoot can now be rendered, giving an image of the scene, locally illuminated by the patch k.

In order to achieve some *graceful* process, by which the first image is seen, as successive shoots take place, to *smoothly* progress to a converged solution, Cohen *et al* [21] calculate the contribution due to the *increase* in B_k, since patch k last shot:

$$\Delta B_j \mathrel{+}= \rho_{d,j} \Delta B_k F_{kj} A_k / A_j \tag{2.14}$$

Here ΔB_j is the *unshot* radiosity of patch j; the radiosity gathered at patch j, which has not yet been shot.

Most PR radiosity systems evaluate form factors on the fly, only storing one column of the form factor matrix at a time, so storage costs are kept linear too.

Application of the reciprocity relation to a row of form factors, which are all numerical approximations, has its own costs in terms of accuracy: PR methods generally do not converge to quite the same solution as their costlier, but more accurate, FM counterparts. Careful evaluation of the row entries can help alleviate this problem [8].

Patch-to-vertex form factors

A significant improvement was made to PR radiosity, when patch-to-vertex form factors were incorporated into the algorithm [120]. Previously, having established the radiosity values for each patch/element, these values are extrapolated to the vertices in the patch/element data structure, and linear interpolation is then used to smooth shade the patches. Extrapolating data, however, introduces errors; so the vertex radiosities which are used to render the scene all too often lead to irritating anomalies such as light leaks and shadow leaks [7].

In the approach outlined by Wallace *et al* [120], they evaluate the contribution of the current shooting patch to every patch *vertex* in the scene, and then use this (accurate) data to interpolate across patches. This approach has remained popular, and has carried on into the work described here, in that the contribution of the current shooting patch is evaluated on a vertex-by-vertex basis: no extrapolation takes place. More details are presented in chapters 4 and 5.

2.2.4 Hierarchical radiosity

Hierarchical radiosity [49, 50] was the first of a glut of innovative algorithms, which have flooded the radiosity community within the last few years, and have done much to improve the method's accuracy and performance.

Hierarchical radiosity was inspired by efficient solution methods developed for the *N-body problem*. In the N-body problem, there are N particles, each exerting a gravitational/electromagnetic force on all the $N-1$ other particles: the problem is to compute the total force on each particle. There are a number of similarities between the N-body problem and radiosity:

- In both problems, there is a total of $N(N-1)/2$ pairwise interactions.
- Gravitational and/or electromagnetic forces both fall off according to $1/r^2$. The same fall off rate is true of form factors.
- Gravitational forces are equal and opposite. The form factor reciprocity relationship details a similar situation for form factors.

Whilst the two problems are not without their differences, it became apparent to Hanrahan [49], at least, that recent speed-up methods for solving the N-body problem, might be usefully applied to radiosity. The two main lessons learnt being:

- Numerical form factor evaluation methods are subject to error. Consequently, one need only attempt to model light transport to within some given precision.
- The light leaving a cluster of elements, reaching some distant point, can be often be calculated, to within the given precision, by a *single* term which represents the contribution of the cluster taken as a whole.

To begin with, consider only the case where there is no inter-object occlusion:

Hierarchical radiosity methods begin by looking at each pair of patches (j, k) in the scene, and making low-cost estimates on the upper-bounds of the form factors F_{jk} and F_{kj}. If both of these estimates are smaller than some pre-defined F_ε, then the two patches are *linked*, to indicate that the calculated energy transfer between them, will be within the desired level of accuracy. If one of the estimates is larger than F_ε, then the patch with the larger form factor (patch j, if $F_{jk} > F_{kj}$, say) is subdivided[5] and the procedure recurses until either the F_ε condition is met, or the subpatches become too small. Subpatches which become too small to subdivide (their area is less than some predefined A_ε) are linked regardless. This stops infinite subdivision around, say, the corners of a room.

The result of this initial *refining and linking* process, is that patches in the scene will now not only have a hierarchy of subpatches, as per [23], but each patch j will have a link, somewhere in its hierarchy, to every part of every other patch k in the scene, for which $F_{jk} > 0$ (all other patches, in the unoccluded case: figure 2.5).

Careful error analysis [49, 50] reveals that this method provides an *automatic* means of calculating, to within a fixed error tolerance, the form factor between two unoccluded patches. The method ensures that the form factor evaluation method (which is known to be subject to numerical error) is only applied where it is going to be accurate. If the algorithm deems a transfer falls outside of the desired error tolerance, it will force it to take place lower down the hierarchy. At the same time, interactions are encouraged to take place as high up the hierarchy

[5]Unless the patch has already been subdivided, when evaluating its interaction with some patch other than k.

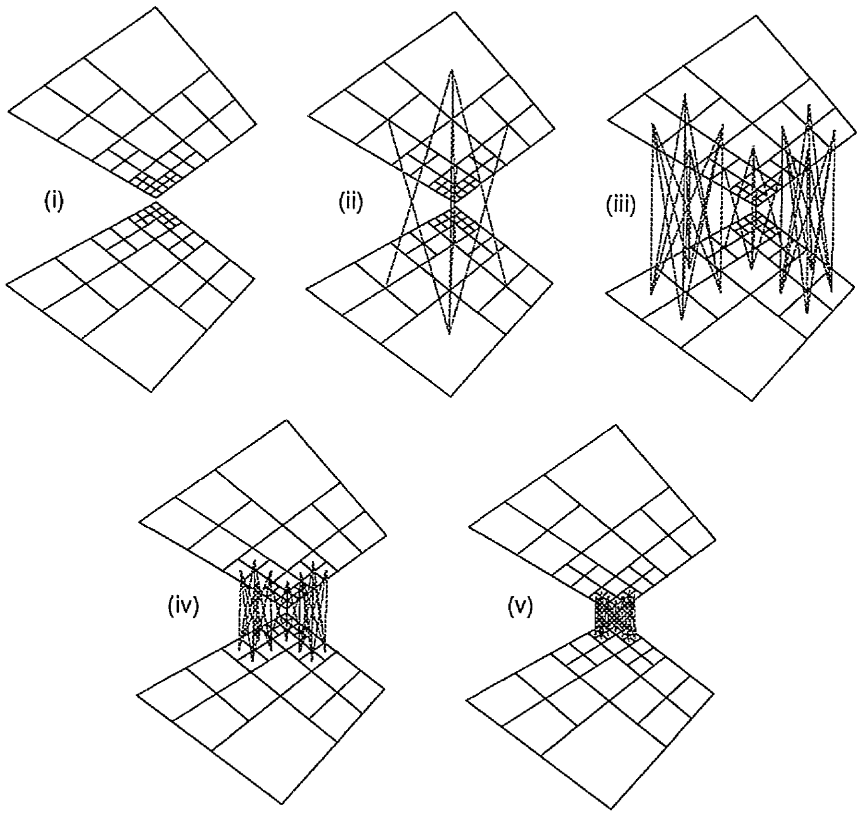


Figure 2.5: Possible hierarchical subdivision of two patches (after [49]).

as possible.

Rather than having a form factor matrix, detailing how different patches will transfer energy to one another, hierarchical radiosity results in a data structure which details exactly how different parts of the scene interact with one another. Furthermore, Hanrahan *et al* [49] assert that each patch will only interact with a constant number of other patches, making a total of $\mathcal{O}(N)$ interactions — effectively representing the form factor matrix as a collection of $\mathcal{O}(N)$ *blocks*; where each block describes how one sub-region of a patch exchanges energy with another.

Once the patches have been refined relative to one another, and the links created, it is possible to tour all the links of one patch j, and add in the contribution of each link patch k using:

$$B_j += \rho_{d,j} F_{jk} B_k$$

It is important to note, however, that patch j is not linked to *all* of the scene via its links, it is linked to the *rest* of the scene via its subpatch links; so this gathering process must be repeated for all subpatches of j, at every level of granularity, before moving onto the next iteration.

Because each subpatch is only linked to some subset of the scene, the energy gathered by each subpatch does not represent the *total* energy incident on the region represented by it. Consider the top left subpatch in figure 2.6.iii — this region receives some of its energy via its parent patch's links, and the rest via its own links. In order to store the subpatch energy collected by the parent *with* the subpatch (and the energy collected by the subpatch with the parent) it is necessary to tour the hierarchy (figure 2.6), *pushing* the energy of parent patches down to their children, and *pulling* the element contributions up to their parents (using area-weighted averaging).

The question of occlusion, in hierarchical radiosity, is now addressed. Hanrahan *et al* [50] simply cast a constant number of rays between patches to get a visibility estimate $V_e \in [0,1]$. This is then multiplied by the unoccluded form factor to get the desired form factor estimate. It is easy to see how a number of methods[6] could be applied to speed up this part of the calculation – Hanrahan *et al* used BSP trees.

[6]BSP trees [112], octrees [40], shaft culling [47], etc. [42]

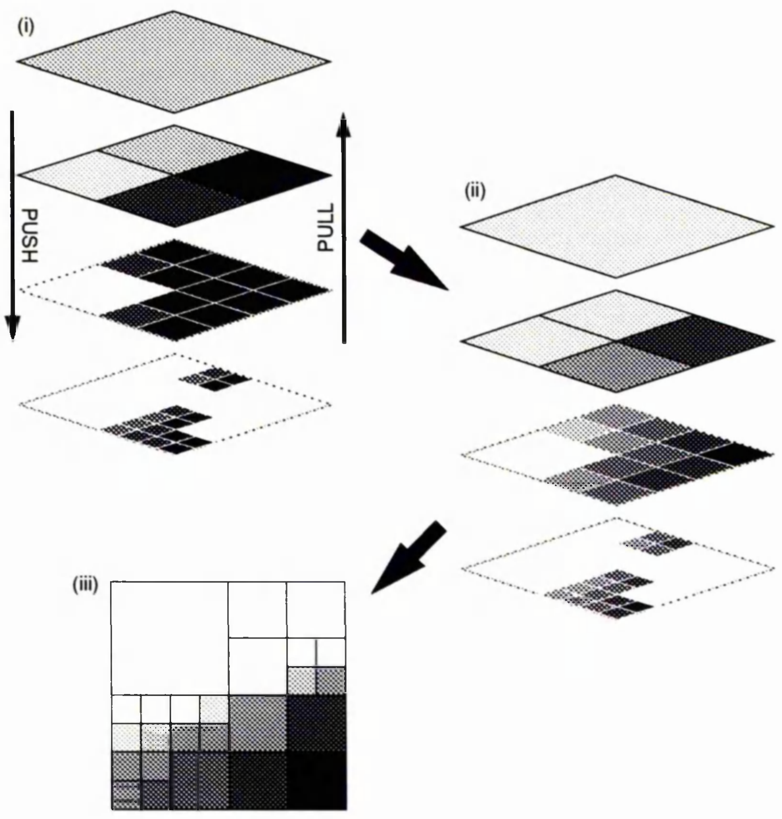


Figure 2.6: Hierarchical radiosity: *push* and *pull*

Multi-gridding, BF-refinement and importance. Hierarchical radiosity can be further enhanced by incorporating *multi-gridding* into the algorithm [50]. Multi-gridding is a technique borrowed from the finite element method [88], and is a process whereby a converged solution is found for a coarse discretization of the scene, and is then used as a starting point for the costlier iterations which result when the mesh is refined further. A solution for a coarse mesh can be found quickly. A solution for a finer mesh, using the coarse mesh solution as its starting point, will also be efficient. In turn, *this* solution can be used as the starting point for an even finer mesh, and so on, until the desired level of accuracy is reached. This proves a much cheaper approach than solving for the finest level from scratch, and has the added advantage that the various solutions can be viewed as and when they become available.

Hierarchical radiosity requires little modification to incorporate multi-gridding — in order to iterate, simply refine using smaller and smaller values of F_ε, deleting those links which were present at the previous iteration, but where now refinement is called for.

With multi-gridding, a series of solutions is available, and so-called *BF-refinement* can be incorporated into the solution process [50]. With BF-refinement, the decision as to whether to link or refine, is not based upon the form factor estimate (F), but upon the product of F with the current radiosity estimate (B). Effectively, transport is only evaluated accurately where reasonable amounts of energy are involved; the transport of small amounts of energy is evaluated at a coarser level. The solution method has now been tailored so that shooting and gathering are all but indistinguishable: all interactions involve roughly the same amount of energy ($BF \in [B_\varepsilon F_\varepsilon/4, B_\varepsilon F_\varepsilon]$), making their ordering according to brightness somewhat superfluous.

Another effective criterion for refining the links in the radiosity method has been described by Smits *et al* [102], where the notion of *importance* is introduced. Whereas the radiosity of a patch details how favourably it is positioned with respect to the various light sources, and the rest of the scene; a patch's *importance* details how favourably it is positioned with respect to the current view, and the rest of the scene. Mathematically, importance is the *dual* [84] of radiosity. Essentially, those interactions which prove important to the particular view take place further down the hierarchy, as do those which involve large amounts of energy. Interactions which match both of these criteria take place at the finest level of refinement. The method does not only solve for the radiosity of each patch, but also for its importance, so that it can be incorporated into the refinement criterion of the multi-gridding method. What results, is an algorithm which devotes as little effort as possible to finding out how 'out of sight' surfaces are lit, whilst maintaining accurate information for those that appear in the current view, or which are important contributors to visible surfaces. The time and memory savings over simple BF-refinement are impressive [102].

To conclude, hierarchical radiosity is notable because not only does it take great care to ensure that its results satisfy some predefined error tolerance, but also because it reaches those results in linear time, with linear storage costs.

Constant radiosity: some closing remarks

The way in which the radiosity method has progressed, since its introduction to the computer graphics community, has impressed many. Starting as an $\mathcal{O}(N^2)$ algorithm, only capable of handling scenes without occlusion, it has progressed to hierarchical radiosity — an algorithm capable of rendering accurate, realistic images, of complex scenes with occlusions, in only $\mathcal{O}(N)$ time.

Throughout the development of the radiosity method, the algorithm's only competition for the title of *best global illumination algorithm* have been the various ray tracing approaches. These are outlined in the following section, and compared and contrasted with the radiosity method in the closing section of the chapter.

2.3 Ray tracing

Traditionally, in the computer graphics community, ray tracing has taken a back seat to radiosity in terms of attempting to take a physically accurate (as opposed to perceptually pleasing) approach to realistic rendering. This is less true of modern ray tracers [61, 123, 98], where physical accuracy is carefully accounted for, and impressive images result.

2.3.1 Whitted ray tracing

Whilst ray tracing was first introduced by Appel [3] in 1968, the method was not extended to account for any kind of global illumination effects until Whitted's algorithm [127] was introduced in 1980.

Whitted, like many others [124], models the *brdf* of a surface by splitting it into two parts: one which represents the diffuse (uniform) part of the reflected light, and another which represents the specular (mirror-like) part:

$$f_r(\mathbf{x}, \Theta_o, \Theta_i, \lambda) = \rho_d(\mathbf{x}, \lambda) + \rho_s(\mathbf{x}, \Theta_o, \Theta_i, \lambda)$$

Ray tracing relies on classical optics' ray model of light transport: rays of light leave the light sources, travel in straight lines except when they interact with the scene, and eventually some of them reach the eye of the observer. Unfortunately, tracing rays from the light source, in a large range of directions, in the hope that enough rays to make a useful image eventually reach the virtual camera, stretches the limits of even the most powerful computers.

Whitted's cheaper approach [127] (now known as *light-backwards* ray tracing) traces *eye rays* from the eye of the observer[7] back into the scene. A recursive routine is used, whereby having found the impact point on the first surface hit by a ray, the algorithm generates a reflected *specular* ray and a refracted *transmitted* ray, which are then recursively traced through the environment until either:

- a wholly diffuse, opaque surface is encountered,
- some maximum depth of recursion is reached, or
- the ray misses every surface.

As well as spawning a specular and a transmitted ray, those rays which hit a surface with $\rho_d > 0$, also generate *shadow rays*. Shadow rays are cast between the impact point and the point light sources, to establish which sources have an unoccluded view of the point. Those which have a clear view, contribute to the radiance leaving this point. The direct contribution, from the sources, is diffusely reflected at the impact point. This is, in fact, the only account made for diffuse reflection in the algorithm — diffuse surfaces do *not* illuminate other diffuse surfaces; only light sources illuminate diffuse surfaces (figure 2.7). The radiance returned for each impact point is a linear combination of these three contributions (specular, transmitted and direct diffuse).

Heckbert [52] introduced a useful notation for categorising the different paths that light can take in between leaving a light (L), and arriving at the eye (E). Photons will either go directly to the eye, or bounce off any number of diffuse (D) and specular (S) surfaces on their way there. The set given by the regular

[7]i.e., the virtual camera.

expression[8] $L(D|S)^*E$ describes all such possible paths, and is useful when classifying different global illumination algorithms. Whitted ray tracing accounts for only the $L(D?)S^*E$ paths. A simple *ambient* contribution, designed to account for these shortcomings, can be included when evaluating each contribution.

Effectively, when illuminated *directly*, the surfaces behave like ideal Lambertian surfaces. When illuminated *indirectly* the surfaces behave like perfect mirrors. Simply put, what results is an algorithm which shows diffusely-shaded objects, and reflections of diffusely-shaded objects.

A detailed review of Whitted ray tracing is given in [36]. The pertinent points here are that it is physically unsound, and it only attempts to gather global illumination information for surfaces visible in a particular view.

2.3.2 Distribution ray tracing

Distribution ray tracing[9] [25] is a term used to describe those ray tracing algorithms which use Monte Carlo[10] integration methods to evaluate the integral term found in the rendering equation (1.10). For each pixel making up the image, a number of rays are traced from the eye, through the pixel, back into the scene; the way in which each of these rays (and the rays they spawn) are reflected, when they hit a surface in the scene, is controlled by a probability distribution function [62] which effectively describes the *shape* of the surface's *brdf*.

[8]The regular expression A^* describes 0 or more occurrences of the character A (e.g., A, $AA, \ldots, AAAAA, \ldots$). The regular expression '$A|B$' indicates 'either A or B', whereas $A?$ refers to 0 or 1 instances of A [67].

[9]Originally called *distributed ray tracing*, the name has been changed to avoid confusion with algorithms describing ray tracing on distributed memory parallel computers.

[10]More usually, quasi-Monte Carlo [128].

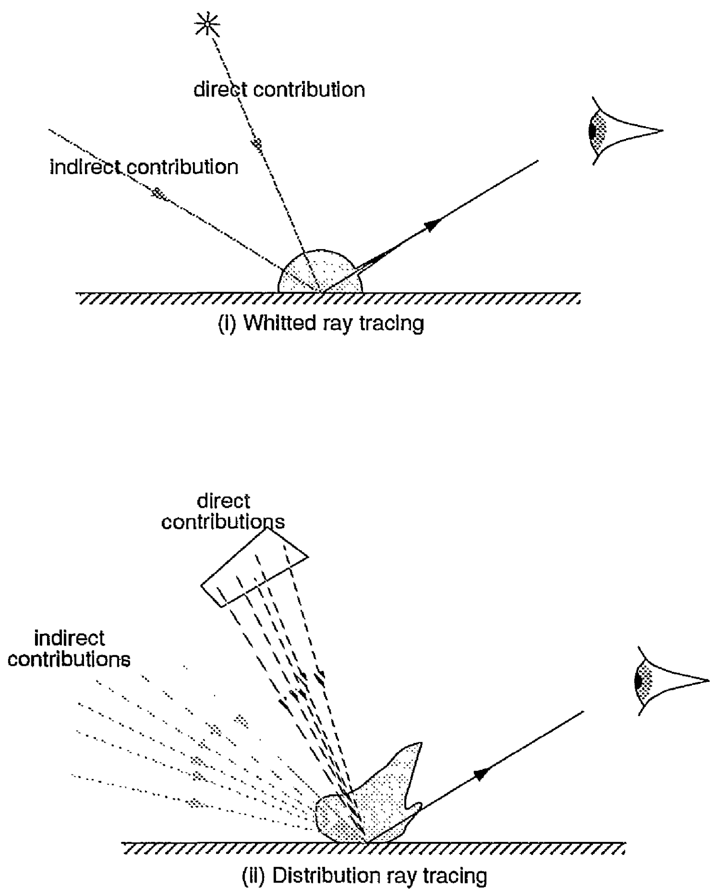


Figure 2.7: Whitted ray tracing *vs* Distribution ray tracing.

Ideally, for each ray/surface intersection, one should sample the whole hemisphere of directions above the impact point, to establish how the rest of the scene illuminates this point. In distribution ray tracing, the sample points are chosen by distributing the rays around the reflection direction, according to the surfaces reflection properties. Typically, the whole hemisphere is *not* sampled (figure 2.7).

Ray tracing has expensive overheads – each ray has the $\mathcal{O}(N)$ problem of finding which surface in the scene it will hit first[11] – if each impact point were to spawn a large number of rays to sample its reflection hemisphere, then the algorithm would quickly grind to a halt: this is the reason Whitted rejected the approach [127]. In distribution ray tracing, each dimension being sampled is only allotted one

[11]This cost can be lowered with hierarchical data storage.

ray: the large number of eye rays cast then ensure that each of the dimensions are sensibly sampled. Care is taken not to resample the same part of the same dimension more than once.

Distribution ray tracing was introduced primarily as an anti-aliasing technique: Cook *et al*'s original algorithm [25] only handled *glossy* reflection and transmission — no attempt was made to model diffuse interreflection. A number of follow-up papers addressed the question 'how many rays are enough?' [66, 29, 87, 75, 83], but none of these methods are modelling any light paths other than those modelled by Whitted ($L(D?)S^*E$). Instead, they concern themselves with *which* $L(D?)S^*E$ paths will result in a cheap, accurate solution. The difficulty for a ray tracer, with diffuse interreflection, comes with the sampling of *large* solid angles, and one of the recursion stopping conditions:

- In order to model glossy surfaces, one need only sample some reasonably small solid angle around the mirror and refraction directions — accounting for diffuse interreflection involves sampling solid angles which may one or two orders of magnitude larger than this.
- Most ray tracers will not spawn reflected/transmitted rays when a parent ray hits a purely diffuse surface: modelling diffuse interreflection would remove this stopping condition, considerably increasing the depth of the average ray tree [127, 36] in the image.

2.3.3 Ray tracing and diffuse interreflection

The first attempt to solve the rendering equation in its entirety, was made by Kajiya in 1986 [61]. Kajiya overcame the computational hurdle of repeatedly sampling large solid angles, by combining a modified distribution ray tracer with the elegant application of some Monte Carlo variance reduction techniques. The latter allowed the relevant integrals to be solved satisfactorily, with fewer samples than would have been possible using a naïve method.

Kajiya's modified ray tracer differs from others in that when a ray hits a surface, rather than spawning a specular ray and a transmitted ray, only *one* ray is generated and traced through the scene. This new approach, dubbed *path tracing* by Kajiya, does not propagate as expensively as conventional distribution ray tracing, even though it carries out the extra task of sampling indirect diffuse illumination (figure 2.8). To proceed, each surface stores its diffuse, specular and transmission coefficients (ρ_d, ρ_s and ρ_t). These are then used to ensure that proportionate amounts of work are put into sampling each of the dimensions corresponding to the three coefficients. Effectively, for each ray/surface intersection, the new algorithm generates a uniform random variable (T) in the range $[0, \rho_d + \rho_s + \rho_t]$, then:

- if $T \in [0, \rho_d]$, a diffuse ray is spawned,
- if $T \in [\rho_d, \rho_d + \rho_s]$, a specular ray is spawned,
- otherwise, a transmitted ray is spawned.

Note that the new algorithm is integrating the same dimensions as conventional

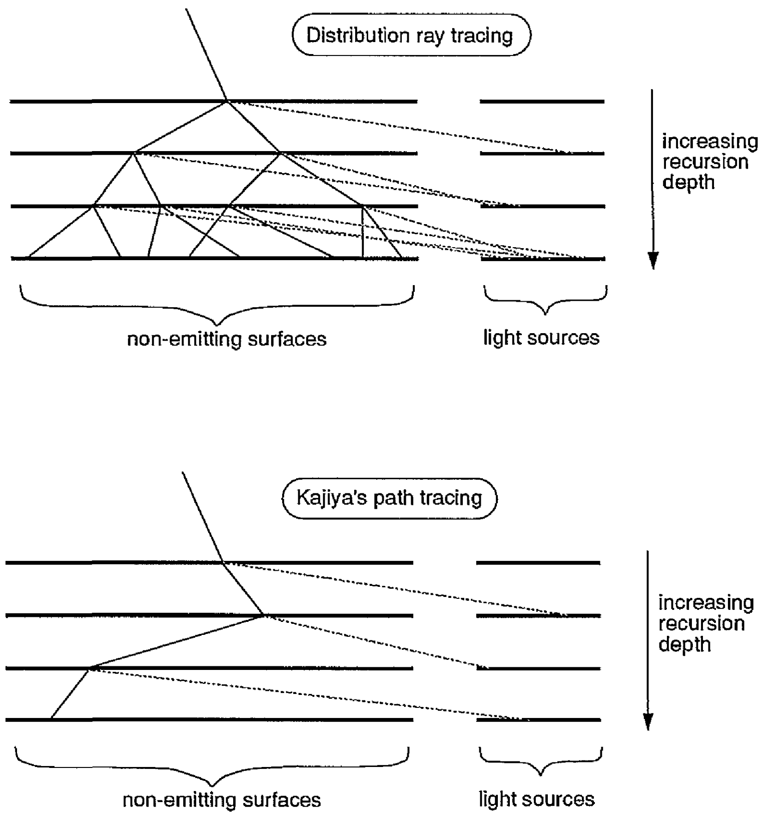


Figure 2.8: Number of rays cast: ray tracing *vs* path tracing. (After Kajiya)

distribution ray tracing[12], but is being far more conservative with its method of spawning samples (figure 2.8). The large number of eye rays cast, and the discrete nature of the solution, ensure that (all else being equal) the two different sampling methods will return the same results. Fortunately, all else is not equal: Kajiya's method (correctly) integrates one extra dimension, by generating rays which sample *diffuse* interreflection (DI). The slowly varying nature of indirect diffuse illumination should help to ensure that its important features are efficiently captured by the hierarchical integration approach adopted by Kajiya. Whilst the method results in impressive images, and is certainly an elegant piece of work in itself, the computational costs are still large: Kajiya gives the rendering time for one simple scene as being more than 20 CPU-hours[13].

[12]i.e., specular reflection & transmission
[13]On an IBM 3081

A more successful attempt at modelling DI using ray tracing, was introduced by Ward *et al* [123]. Here, it was recognised that ray tracing's poor performance in accounting for DI was due to its insistence on solving for this component on a pixel by pixel basis: indirect diffuse illumination is the *least* local, and most slowly varying, of the all the components that need to be accounted for. By evaluating it on a pixel by pixel basis, a lot of work is being needlessly repeated.

Ward *et al* [123] separate how the different lighting components are calculated, choosing to evaluate the indirect diffuse part globally (i.e., view independently), and then storing the information with the scene for inclusion in the conventional pass. DI is not being hastily evaluated on the fly, from scratch, for each pixel: the pertinent information is found as and when it is needed, and stored for possible later use in an octree data structure.

Ward *et al* also introduce an estimate for the gradient of the *indirect diffuse* component of the radiance across a surface, based on scene geometry. A small gradient is implied by no nearby objects, a large gradient is assumed near object boundaries, and sharp corners, where the function usually changes rapidly. Now, if the indirect diffuse component is needed at some point P, and values have already been stored at nearby points, then the value at P can be taken as a weighted average of the neighbouring values, using the inverse of the nearby gradient estimates to weight the different contributions. Indeed, the weights can be used to decide exactly which points are neighbours of P. Whilst a simple radiance interpolation scheme was used in [123], this is improved upon in [121].

A similar approach to modelling indirect diffuse illumination, has been adopted by Shirley [97, 98], and others [63]. These approaches take advantage of the fact that indirect diffuse illumination changes slowly, and model it using a modified radiosity approach — which requires only a coarse discretization of the scene to capture its slowly varying features. The radiosity algorithm used, is a modified progressive refinement method, whereby having shot from all the light sources, the accumulated radiosity of each patch is set to 0, whilst their unshot radiosities are left alone. The algorithm then progresses to a solution, as usual. This results in the *indirect* diffuse component being stored in the patches, for later inclusion in the ray tracing step.

Bi-directional ray tracing [4, 16, 52] is an otherwise conventional ray tracing algorithm which incorporates a *light-forwards* pre-process designed to establish how the various surfaces are indirectly diffusely illuminated. As its name suggests, light-forwards ray tracing involves spraying rays out from the light sources, into the scene. Each ray has associated with it some portion ($E(\lambda)$) of the light source's energy; as the ray makes it way through the scene, it deposits a fraction ($\rho_d(\lambda)E(\lambda)$) of this energy onto each surface it meets (except for the first bounce, which is not *indirect*). The ray continues around the scene until its associated energy falls below some threshold. The numerous energy deposits on the various surfaces are stored in *illumination maps*, which are stored with the surface: these are essentially just texture maps which store illumination information [4]. Once this information is stored in the illumination maps, it can be retrieved in a conventional light-backwards ray tracing pass, as per [97, 63].

2.4 Closing remarks: radiosity *vs* ray tracing

As described thus far, there are some very striking differences between the constant radiosity algorithms, and the ray tracing methods:

- Whereas the ray tracing algorithms have trouble modelling diffuse interreflection (due to their point-sampling approach), the radiosity algorithms (with their area sampling approach) model little else.
- The radiosity algorithms discretize the scene into a number of *flat*[14] patches, while the ray tracing algorithms are content with a large range of object primitives.
- Radiosity is an object-space approach, whereas ray tracing is an image-space approach.

This last point does not apply to all of the ray tracing algorithms described: some researchers have recognised that indirect diffuse illumination is a very slowly varying quantity — once light is reflected diffusely, any detail that might have present in the incident light is lost (figure 2.9). The consequent problems, for ray tracers, have already been made clear; the various solutions principally having involved taking an object-space approach to solving for this component.

Now, whereas ray tracing methods have to take special steps to account for indirect diffuse illumination, radiosity methods are ideally suited to modelling this component: they model, in fact, diffuse illumination *only*. None of the radiosity methods described thus far have made any attempt to account for specular

[14]The patches being flat is only a consequence of the form factor evaluation method, not the radiosity method *per se*.

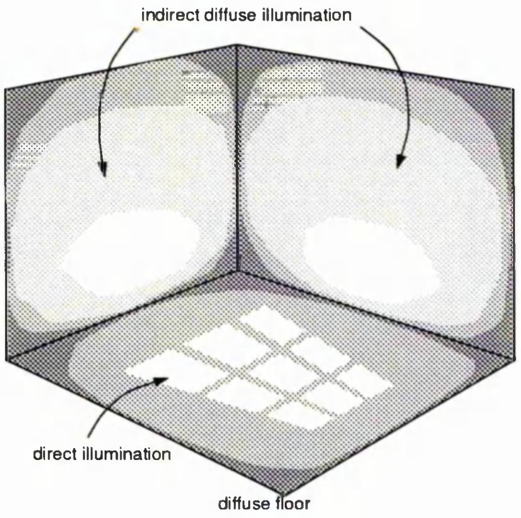


Figure 2.9: Detail that is present in the incident light is lost after diffuse reflection — making indirect diffuse illumination, a particularly slowly-varying quantity.

transport. Many such methods [59, 119, 96, 101] are reviewed in [36]. With the exception of Immel's algorithm [59] all of these approaches recognise the impracticality of trying to solve the rendering equation globally[15], and only attempt to account for the specular reflections which reach the eye of the observer. This is accomplished by *extending* form factors so that light which leaves one diffuse surface, and arrives at another via a number of specular reflections, is accounted for in the form factor. Having evaluated the extended form factors, the radiosity equation can be solved in a *first pass*, and a distributed ray tracing (i.e., image space) second pass can then capture the desired specular effects. It has been pointed out [123] that this still does not account for all possible light paths (e.g., $LSSDE$), but more comprehensive extensions can overcome even this [17].

Essentially, the ray tracing algorithms, which are primarily image-space approaches, find that they have to incorporate an object-space pass if they are to correctly handle the light transport in the scene. The converse is true of

[15]More recently, the wavelet method, described in the next chapter, has been extended to do exactly this [94, 85].

radiosity algorithms.

The remainder of this thesis concerns itself with the accurate solution of the rendering equation in diffuse environments. Care is taken to ensure that direct lighting effects (figure 2.9) are accurately modelled, and that the constant radiosity assumption [110] is not allowed to adversely affect the accuracy of the solution. Concentrating on diffuse surfaces only, in this way, can be justified with a number of reasons. In order to find useful real-world applications for the type of work described in this thesis, an object-space algorithm seems essential; implying radiosity. Also, recent work relating finite element methods to radiosity (described in the next chapter) have opened up new and interesting fields of research, which bear close investigation. A final justification, is that limiting oneself to diffuse transport only, in the first instance, does not prevent one incorporating specular transport at a later stage — considering diffuse-only scenes should simply be regarded as a good starting point.

Chapter 3

Higher order radiosity methods

The previous chapter described a collection of radiosity methods, all of which relied on the premise that the radiosity across a surface can be represented as a piecewise constant function. This premise, in turn, necessitated the use of (inappropriate[1]) interpolation in order to achieve images which appeared perceptually acceptable. This chapter describes algorithms which recognise the need for a consistent, higher order, radiosity representation, if accuracy is to be achieved. Such a representation would be used throughout the solution and rendering phases.

In order to meet accuracy demands, the *constant radiosity algorithms* of chapter 2, used a large number of patches in regions where the radiosity was varying rapidly. The radiosity algorithms of this chapter approximate the true radiosity using (piecewise) higher order polynomials, rather than simple step functions. The computational complexity of handling a large number of constant functions,

[1]Using linear interpolation on a solution which is piecewise constant, is inconsistent: a hack.

is replaced by the computational complexity of handling a smaller number of higher order functions. The relative merits of the two approaches are discussed.

Before considering the specifics of higher order radiosity methods, some background theory on the solution of integral equations, is presented as a framework:

3.1 Integral Equations

Much of the following discussion, on integral equations and associated solution methods, deals with functions of a single variable: whilst this typically will *not* include the radiosity across some surface in a 3D scene, the results *do* generalise to functions of more than one variable [54]. The reader is referred to [28] or [6] for a fuller discussion of the issues covered in this section. For a computer graphics-specific discussion, see [54] or [55].

An *integral equation* is one in which the unknown function appears inside an integral. When the limits of the integration are definite, the equation is known as a *Fredholm* integral equation. Further, where the equation is linear in the unknown function, it is called a *linear* Fredholm integral equation. Linear Fredholm integral equations take three distinct forms:

$$\begin{aligned} y(s) &= \int_a^b K(s,t)x(t)dt \\ x(s) &= y(s) + \int_a^b K(s,t)x(t)dt \\ x(s) &= \int_a^b K(s,t)x(t)dt \qquad (3.1) \end{aligned}$$

These are known as linear Fredholm integral equations of the *first*, *second*, and *third* kinds, respectively. The functions $y(s)$ and $K(s,t)$ are both known; $x(s)$ is the unknown. The function $y(s)$ is known as the *driving term*, the function $K(s,t)$ is known as the *kernel* [28]. The region $[a,b]$ is the *domain* under consideration.

Integral equations are often written in terms of integral *operators*. For example, a linear Fredholm integral equation of the second kind, of which the rendering equation (1.10) is an example, might be written:

$$x = y + \mathcal{K}x \tag{3.2}$$

Here, the dependence of the unknown function x on some parameter (s) is taken as being implicit, and the integral operator $\mathcal{K}$ is defined by:

$$(\mathcal{K}x)(s) = \int_a^b K(s,t)x(t)dt \tag{3.3}$$

When a second kind integral equation is written in the form of (3.2), it is easy to see how, if $\mathcal{I}$ is the identity operator, a solution might take the form:

$$x = (\mathcal{I} - \mathcal{K})^{-1}y \tag{3.4}$$

If, in some sense, $||\mathcal{K}|| < 1$, then the intuitive solution

$$x = y + \mathcal{K}y + \mathcal{K}^2y + \cdots + \mathcal{K}^iy + \cdots \tag{3.5}$$

seems to follow from (3.4).

Why 'intuitive'? Well, in the case of the rendering equation, this corresponds to the concept of light leaving the sources (y), bouncing off the surfaces in the scene $(\mathcal{K}y)$, then this light bouncing again $(\mathcal{K}^2y)$, and again $(\mathcal{K}^3y)$, *ad inf.* So, the light incident at any point in the scene consists of a direct illumination part, a once-reflected part, a twice-reflected part, ... Also, (3.5) seems to follow from (3.4) because (3.5) *looks* very much like the binomial expansion has been applied to (3.4).

This thesis is concerned only with functions and kernels which are in some sense *well behaved*. The only functions, $x(s)$, that will be considered, are such that:

$$\int_a^b |x(s)|^2 ds < \infty$$

In the case of radiosity, we talk about functions of *finite energy*. The set of all such functions define the function space $\mathcal{L}^2(a,b)$. The function $x(s)$ is known as an $\mathcal{L}^2$ function. Suitable definitions of 'equality' and a 'null function' can be combined with the set of $\mathcal{L}^2$ functions to form a complete linear vector space [28].

Similarly, this discussion will only consider kernels $K(s,t)$ which are such that:

$$\int_a^b \int_a^b |K(s,t)|^2 ds dt < \infty$$

The L_2 *norm* of an $\mathcal{L}^2$ function $x(s)$ is given by:

$$\|x\|_2 = \left\{ \int_a^b |x(t)|^2 \right\}^{1/2}$$

In turn, this function norm[2] can be used to define a norm for the integral operator $\mathcal{K}$ which maps the $\mathcal{L}^2$ function x onto the $\mathcal{L}^2$ function $\mathcal{K}x$:

$$\|\mathcal{K}\| = \sup_{x \in \mathcal{L}^2, x \neq 0} \frac{\|\mathcal{K}x\|}{\|x\|} \tag{3.6}$$

It has been shown [28], that when $\|\mathcal{K}\| < 1$, the *Neumann series* (3.5) does in fact provide an exact solution to the integral equation (3.2):

$$x = y + \sum_{i=1}^{\infty} \mathcal{K}^i y \tag{3.7}$$

[2]The subscript 2 is taken as being implicit, in this thesis.

Kajiya [61] has noted the similarity between the Neumann series expansion and the iterative solution methods common to full matrix (FM) radiosity methods (section 2.2.2). Several iterations being equivalent to summing early terms in expansion (3.7). It is easy to visualise how, in the case of the radiosity equation, the norm of the kernel *is* less than unity: here, the kernel describes how light is reflected off the surfaces in the scene, and the conservation of energy applies – so no more light can be reflected than was incident (i.e., $||\mathcal{K}x|| \leq ||x||$). In real scenes, the reflected energy will always be strictly less than the incident energy, i.e., $||\mathcal{K}x|| < ||x||$. Inserting this information into definition (3.6), it becomes clear that $||\mathcal{K}|| < 1$ for the radiosity kernel; thereby validating the application of the Neumann series [28].

The following definition, whose relevance will become more apparent in the next section, is for the *inner product* of two $\mathcal{L}^2$ functions x and y over the domain $[a, b]$:

$$< x, y >= \int_a^b x(t)y(t)dt \tag{3.8}$$

Two (non-zero) functions x and y are said to be *orthogonal* when $< x, y >= 0$.

3.2 Method of weighted residuals

Recall (3.2):

$$x = y + \mathcal{K}x$$

The aim now is to solve (3.2) 'as best we can' (in some sense) by *restricting* the solution to a familiar function space — one which contains functions whose behaviour

- comes close to the complexity estimated of the *true* solution function, and
- is well understood.

In general, the exact solution lies in an infinite-dimensional (Hilbert) function space. The more complex the restricted subspace, the closer, one presumes, this approach will get to the true solution.

As yet, little effort has been made to define the notion of *closeness*, as it has been used here; a clearer outline is given now.

3.2.1 The approximation function

The quantity

$$\tilde{x}(s) = \sum_{i=1}^{n} w_i N_i(s) \tag{3.9}$$

is referred to as a *trial function* [32] for the equation in question (3.2). The n functions $\{N_i\}_{i=1}^{i=n}$ are known *basis functions* which, together, define the span of the space that the trial function has been restricted to. The weights $\{w_i\}_{i=1}^{i=n}$ are not yet known; they are the scalars by which the basis functions are weighted to ensure that the trial function $\tilde{x}$ is as close as possible to the exact solution, x.

Algorithms which adopt this approach are known as *projection methods* — the idea being that the trial function is the projection of the exact solution into the chosen, finite-dimensional, function space.

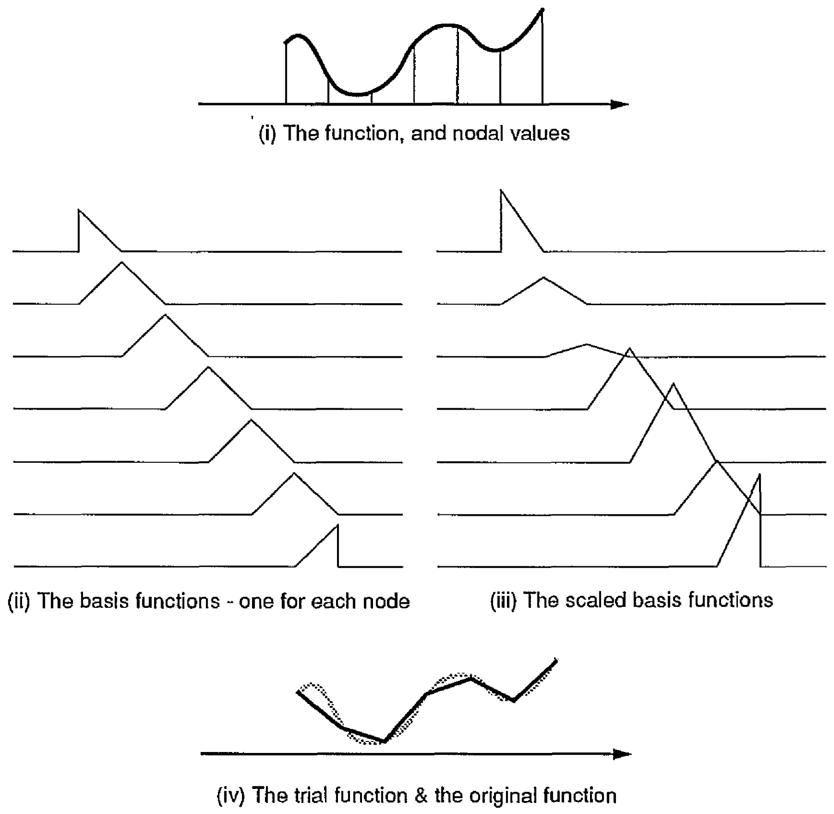

(i) The function, and nodal values
(ii) The basis functions - one for each node
(iii) The scaled basis functions
(iv) The trial function & the original function

Figure 3.1: Building a trial function from linear basis functions (after [24]).

A choice must now be made about which basis functions are to be used — effectively choosing the function space that the solution is to be restricted to. Some approaches use basis functions with *global support* (i.e., non-zero anywhere) [32]. Another approach is to utilise basis functions with only *local support* — i.e., those which are zero everywhere, except in some small region of the domain. It is these last functions, which have gained huge popularity through their use in finite element methods [32, 6, 82], that are considered here. Not only are they simple to visualise, they are also computationally robust and easily generated, even for the most complex geometric shapes.

A one-dimensional example of a trial function is shown in figure 3.1. Here, the basis functions each have *local support* and are linear. The function is sampled at seven points, known as *nodes*: the weights $\{w_i\}_{i=1}^{i=n}$ are found, in this case, by evaluating the function at each of the nodes. The domain has been split up into a number of *elements*, which are the regions used to limit the support of the basis functions. As can be seen from figure 3.1, each basis function is associated with a single node, and a basis function is non-zero only on those elements which are adjacent to its corresponding node. Notice also, that each basis function is zero when evaluated at any node other than its 'own'.

A more complicated case, for a two-dimensional domain[3], is shown in figure 3.2. Again, the basis functions are linear, but now the elements are quadrilaterals, having nodes at their vertices. Again, there is one basis function for each node, and each basis function is zero when evaluated at any node other than its own.

In general, the nodes are positioned at various points within, and on the boundaries of, the elements. In turn, the elements are chosen according to the geometric complexity of the surface being modelled (their shape), and the expected complexity of the function being modelled (their order: the number of nodes per element). Whilst figure 3.2 shows only the linear case, cubic basis functions (say) across a quadrilateral element would require at least 12 nodes per element (many being shared with neighbouring elements) [131]. The greater the accuracy required, the smaller the elements, and the more nodes per element. It is easy to see how such a set-up might be used to model radiosity varying across a surface.

In both of the examples mentioned thus far (figures 3.1 and 3.2) whenever a basis function has been evaluated at a node other than its own, it has been zero. This

[3]Any parameters (s, t) or domain limits (a, b) can now be regarded as 2D vectors. So the parameter s can be regarded as the ordered pair $s = (s_1, s_2)$, where s_1 and s_2 are both reals. The domain $[a, b]$ becomes $[a_1, b_1] \times [a_2, b_2]$, the integral $\int_a^b ds$ should be regarded as $\int_{a_1}^{b_1} \int_{a_2}^{b_2} ds_1 ds_2$, and so on. Similarly for higher dimensions. Since little is gained by expansions of this type, the notation will remain unchanged, in the main.

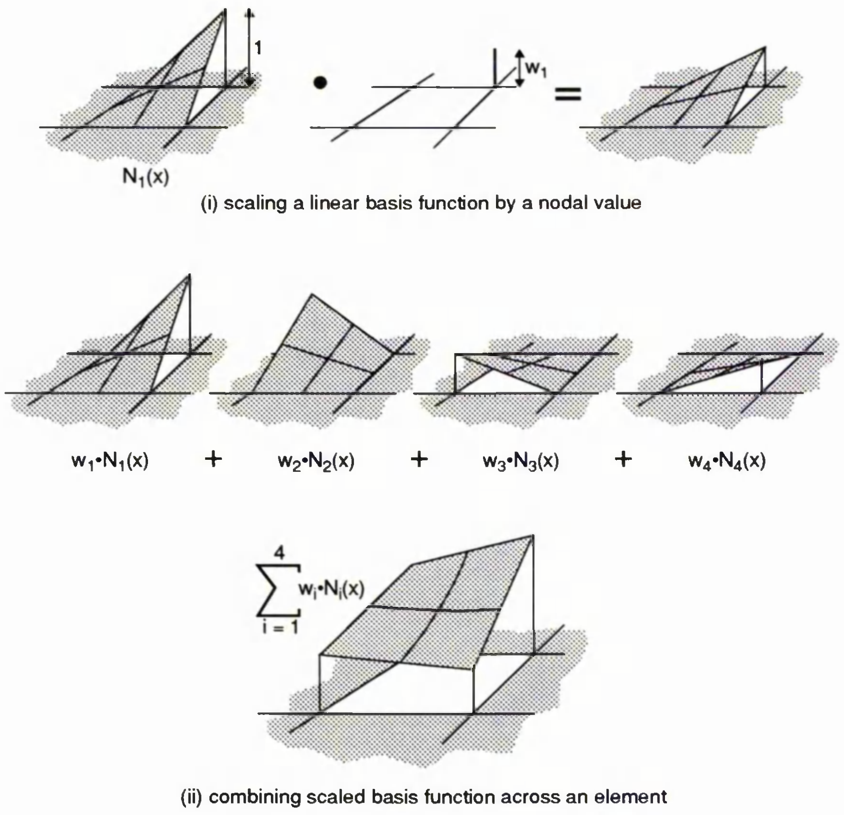

1
w_1
$N_1(x)$
(i) scaling a linear basis function by a nodal value
$w_1 \cdot N_1(x)$ + $w_2 \cdot N_2(x)$ + $w_3 \cdot N_3(x)$ + $w_4 \cdot N_4(x)$
$\sum_{i=1}^{4} w_i \cdot N_i(x)$
(ii) combining scaled basis function across an element

Figure 3.2: Building a 2D trial function from linear basis functions (after [24]).

typical of basis functions with local support [131, 57].

Having settled on a set of basis functions, and evaluated the weights corresponding to each one, what has effectively been finalised is *how* the trial function interpolates. In order to find out exactly *which values* it interpolates, one must evaluate the trial function at each node ($\{s_i\}_{i=1}^{i=n}$). It is for this reason (computational ease) that basis functions are usually chosen where $N_j(s_i) = \delta_{ij}$ — the Kronecker delta, for in this case we have simply: $\tilde{x}(s_i) = w_i$.

3.2.2 Error considerations

Now, inserting the trial function (3.9) into the integral equation (3.2) gives:

$$\begin{aligned} \tilde{x} &\approx y + \mathcal{K}\tilde{x} \\ 0 &\approx \tilde{x} - \mathcal{K}\tilde{x} - y \end{aligned} \tag{3.10}$$

The amount by which (3.10) falls short of zero, at any given point, defines the *residual function* r:

$$r = \tilde{x} - \mathcal{K}\tilde{x} - y \tag{3.11}$$

Ideally, the weights $\{w_i\}_{i=1}^{i=n}$ should be chosen so that the residual function is identically zero. Alas, whilst the trial function is restricted to a particular finite-dimensional function space, this is not true of the exact solution, and therefore neither is it true of the residual. So, realistically, the weights should be chosen so that the residual is as small as possible everywhere. In fact, they are chosen so that the residual is forced to zero in some *average* sense [32]:

A number of independent *weighting functions* $\{W_i(s)\}_{i=1}^{i=n}$ can be chosen, and the residual minimised with respect to these functions by setting

$$< W_i, r >= 0, \qquad \forall i \in \{1, \ldots, n\} \tag{3.12}$$

Inserting the residual definition (3.11) into (3.12) gives, for each $i \in \{1, \ldots, n\}$:

$$\begin{aligned} 0 &= < W_i, \tilde{x} - \mathcal{K}\tilde{x} - y > \\ &= < W_i, \sum_{j=1}^{n} w_j N_j - \sum_{j=1}^{n} w_j \mathcal{K} N_j - y > \\ &= \sum_{j=1}^{n} w_j < W_i, N_j > - \sum_{j=1}^{n} w_j < W_i, \mathcal{K} N_j > - < W_i, y > \end{aligned} \tag{3.13}$$

Let us define the $n \times n$ matrices $\mathbf{M}$ and $\mathbf{K}$, whose i^{th} row, j^{th} column, entries are given by:

$$M_{ij} =< W_i, N_j >= \int_a^b W_i(s) N_j(s) ds \tag{3.14}$$

and,

$$K_{ij} =< W_i, \mathcal{K} N_j >= \int_a^b W_i(s) \int_a^b K(s,t) N_j(t) dt ds \tag{3.15}$$

respectively.

The matrix $\mathbf{M}$ is the *mass*, or *stiffness*, matrix. The matrix $\mathbf{K}$ is the *discretized kernel* matrix.

If we also define two column vectors $\mathbf{w}$ and $\mathbf{y}$, whose i^{th} row entries are given by w_i and, $y_i =< W_i, y >$ respectively, then (3.13) can be rewritten in matrix/vector form:

$$(\mathbf{M} - \mathbf{K})\mathbf{w} = \mathbf{y} \tag{3.16}$$

The $n \times n$ matrix $(\mathbf{M} - \mathbf{K})$ is known as the *generalised stiffness matrix* – its entries can all be found by utilising known information in some numerical quadrature method. Similarly for the vector $\mathbf{y}$. This leaves only the vector $\mathbf{w}$ unknown, and it becomes apparent why exactly n weighting functions were chosen: the system (3.16) represents a linear system of n equations in the n unknowns $\{w_i\}_{i=1}^{i=n}$. Solving this system gives weights which put the trial function very close to the exact solution, in some average sense.

Exactly which weighting functions are chosen depends on the specifics of the problem at hand, and the effort one is willing to devote to reaching a solution. Two common choices are found in the *collocation method* and the *Galerkin method*.

Both of which are now reviewed.

3.2.3 Collocation method

Suppose the n nodes are positioned at $s = s_1, s_2, \ldots, s_n$. In the collocation method, the n weighting functions are:

$$W_i(s) = \begin{cases} 1 & \text{if } s = s_i \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

Inserting these weighting functions into (3.12), forces the condition $r(s_i) = 0$. So this choice of basis function amounts to ensuring that the residual is zero at each of the nodes.

How do the collocation weighting functions affect the components in the linear system of equations (3.16)? The mass matrix is now given by:

$$M_{ij} =< W_i, N_j >= N_j(s_i) \tag{3.18}$$

Clearly, these entries are trivial to find – one need only evaluate a set of known functions at a set of known points. Typically, $N_j(s_i) = \delta_{ij}$, so the mass matrix will be the identity matrix (as is the case in classical radiosity (2.12)).

From (3.15) and (3.17), the discretized kernel matrix is given by:

$$K_{ij} =< W_i, \mathcal{K}N_j >= (\mathcal{K}N_j)(s_i) = \int_a^b K(s_i, t) N_j(t) dt \tag{3.19}$$

In order to evaluate each of these entries, an integral must be evaluated. Typically, this will involve some form of numerical quadrature procedure [28]. However, the basis function $N_j(t)$, having only local support, will be zero for much of

$t \in [a, b]$, so the problem is not as daunting as it might first appear.

The i^{th} component of the vector $\mathbf{y}$ is simply $y(s_i)$ which, again, is simply evaluating a known function at a known point.

Once the generalised stiffness matrix $(\mathbf{M} - \mathbf{K})$, and vector $(\mathbf{y})$, corresponding to the driving term, have been evaluated, it remains only to solve the linear system of equations (3.16) for the weights w_i. These can then be used to build the approximate solution $\tilde{x}$ (3.9) which is close to the exact solution in that the residual is guaranteed to vanish at each of the nodes.

Collocation radiosity

In order to apply this theory to solve for the radiosity across a surface, regard the function $x(s)$ as representing the radiosity across the surfaces that make up the environment. The driving term $y(s)$ represents emitted radiosity, and the kernel function is given by (figure 3.3):

$$K(s, t) = \rho_d(s) g(s, t) \frac{\cos \theta_s(t) \cos \theta_t(s)}{\pi r^2} \tag{3.20}$$

Consider the following approach. The environment is split up into n elements

$$\{[l_i, u_i]\}_{i=1}^{i=n} = \{[a, u_1], [l_2, u_2], \ldots, [l_n, b]\},$$

each containing a single node $s_i \in (l_i, u_i)$.

Now, if the basis functions N_j in (3.9) are chosen to be constant:

$$N_j(s) = \begin{cases} 1 & \text{if } s \in [l_j, u_j] \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

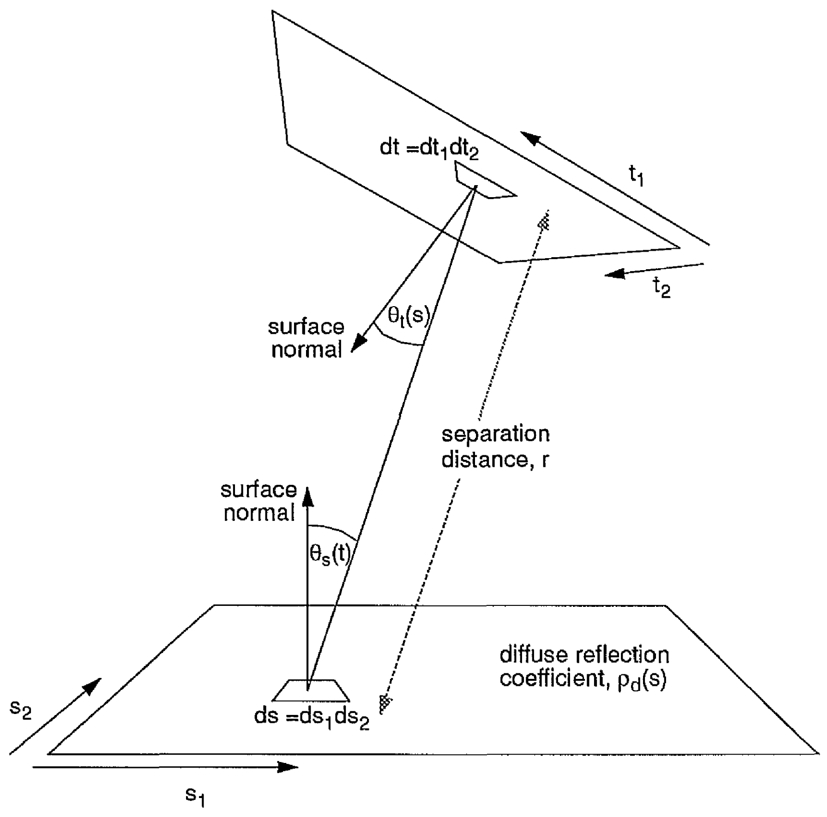


Figure 3.3: Patch-to-Patch form factor geometry, in parametric terms.

then the mass matrix (3.18) is the identity matrix ($M_{ij} = \delta_{ij}$), the discretized kernel matrix $\mathbf{K}$ is given by (3.19):

$$K_{ij} = \int_{l_j}^{u_j} K(s_i, t) dt$$

and the vector $\mathbf{y}$ is evaluated using $y_i = y(s_i)$.

Forcing the residual function to be zero at each node results in (3.16), the n constraints:

$$\sum_{j=1}^{n} w_j(M_{ij} - K_{ij}) = y_i$$

$$w_i = y_i + \sum_{j=1}^{n} w_j K_{ij}$$

$$w_i = y_i + \sum_{j=1}^{n} w_j \int_{l_j}^{u_j} K(s_i, t) dt$$

$$w_i = y_i + \rho_d(s_i) \sum_{j=1}^{n} w_j \int_{l_j}^{u_j} g(s,t) \frac{\cos\theta_{s_i}(t)\cos\theta_t(s_i)}{\pi r^2} dt \qquad (3.22)$$

The combination of choosing simple basis functions, combined with even simpler weighting functions, has brought us back to the radiosity equation (2.11) of chapter 2. There, the proximity assumption, combined with the visibility assumption, led to a system of equations identical to (3.22). The integral in (3.22) is the form factor as it evaluated by most numerical algorithms [22], the weights w_i are the patch radiosities, and the y_i are the patch emittances.

The first application of the collocation method (*per se*) to solve for the radiosity distribution in a scene, was on a 2-dimensional *flatland radiosity* environment in [54, 55]. Max *et al* [73, 116] have subsequently used the collocation method for 3D radiosity, with linear basis functions defined over triangular elements — effectively taking the classical radiosity method [22, 23] one step forward, by using Gouraud shading [46] from the solution phase onwards; thereby superseding its (otherwise inappropriate) use in the rendering phase.

3.2.4 Galerkin method

The Galerkin method is another commonly-used weighted residual method. Here, the weighting functions are chosen to be the basis functions which span the function space from which the trial function is chosen (i.e., $W_i = N_i$). With this choice of weighting function, the mass matrix becomes:

$$M_{ij} =< N_i, N_j >= \int_a^b N_i(s)N_j(s)ds \tag{3.23}$$

The discretized kernel matrix is given by:

$$K_{ij} =< N_i, \mathcal{K}N_j >= \int_a^b N_i(s) \int_a^b K(s,t)N_j(t)dtds \tag{3.24}$$

And the homogeneous vector, $\mathbf{y}$ by:

$$y_i = < N_i, y > = \int_a^b N_i(s) y(s) ds \tag{3.25}$$

Notice that whilst (3.24) must now be evaluated for n^2 different pairs of basis functions, these functions all have only local support, and therefore the integral (3.24) need only be evaluated over a fraction of $[a, b] \times [a, b]$.

Again, (3.16) holds, and it is by solving this linear system for the scalar weights w_i that the trial function which 'best' satisfies (3.2) is found. In order to evaluate (3.23), (3.24) and (3.25) one must invariably resort to numerical quadrature methods, although (computationally intensive) closed-form solutions are available, for constant basis functions [93].

If the basis functions are chosen so that

$$< N_i, N_j > = \delta_{ij} \tag{3.26}$$

— the basis functions form an *orthonormal set*.

With orthonormal basis functions, the mass matrix for the Galerkin method becomes the identity matrix (regardless of the order of the basis functions), with its obvious cost benefits. Details of different orthonormal basis sets are of little interest here, suffice to say that a number are available — (normalised) Legendre and Jacobi polynomials are two sets which can satisfy (3.26) for problems of arbitrary (finite) dimension [129].

It is interesting to compare (3.19) and (3.24) — the discretized kernels ($\mathbf{K}$) from the collocation and Galerkin methods, respectively. One striking feature is the double integral in the Galerkin kernel matrix compared to only a single integral for the collocation method. This superior sampling of the domain, ensures the Galerkin method achieves are *surer* representation of the integral operator $\mathcal{K}$, than its collocation counterpart. Clearly, the increased accuracy is accompanied by a consequent increase in computational expense.

An analogy

The following scenario is offered as a 3-space analogy for what the Galerkin method is trying to do in function space.

One end of a spring is fixed at a point above a table. A heavy weight is fixed to the other end of the spring. This weight lies on the table. The table is of such a size, and other physical factors are such that, no matter where on the table the weight is placed, it remains there. No matter where the weight is placed, the spring is stretched beyond its natural length.

For reasons which are beyond our ken, the people who live on the flatland world of the table top, wish to minimise the energy stored in the spring. This energy is directly proportional to amount by which the spring is stretched beyond its natural length [109]. In order to minimise the energy, the people on the table top move the weight to lie directly beneath the point where the spring is fixed.

The flatland table top world is spanned by the unit vectors $\mathbf{i}$ and $\mathbf{j}$. The extension of the spring is described by the vector $\mathbf{x}$. The people merely set $\mathbf{x} \cdot \mathbf{i}$ and $\mathbf{x} \cdot \mathbf{j}$ to zero, and their goal was achieved as best at it could be, while being restricted to the table top. Essentially, the scalar (inner) products of the quantity to be minimised, with the basis vectors, were forced to zero. This is exactly

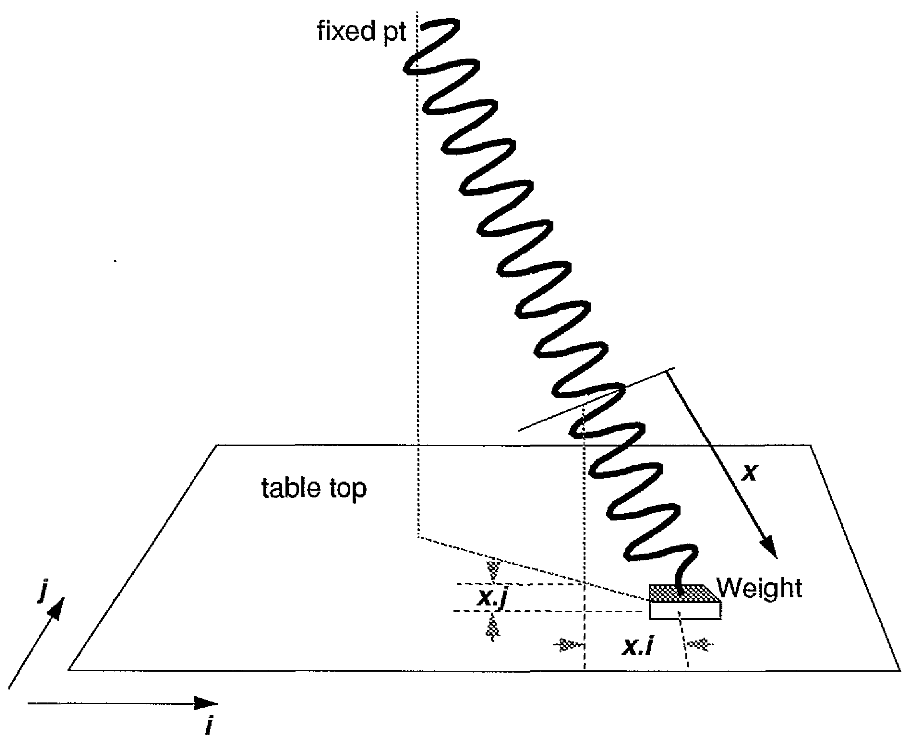


Figure 3.4: A 3-space analogy for the Galerkin method.

what is happening in the Galerkin method. Even though the function to be minimised (the residual) is more complicated than the chosen function space can accommodate, its inner product, with each basis function in the chosen space, is forced to zero. This minimises the function as best we can.

Galerkin radiosity

In the computer graphics community, Heckbert [54, 55], drawing on a wealth of radiation heat transfer research, was the first to propose using the Galerkin method for modelling radiosity. As with collocation radiosity, Heckbert only implemented the method in a 2-dimensional *flatland* world.

Subsequently, the method has been used to solve for the radiosity distribution across a set of bi-parametric surfaces, comprising a 3-dimensional scene,

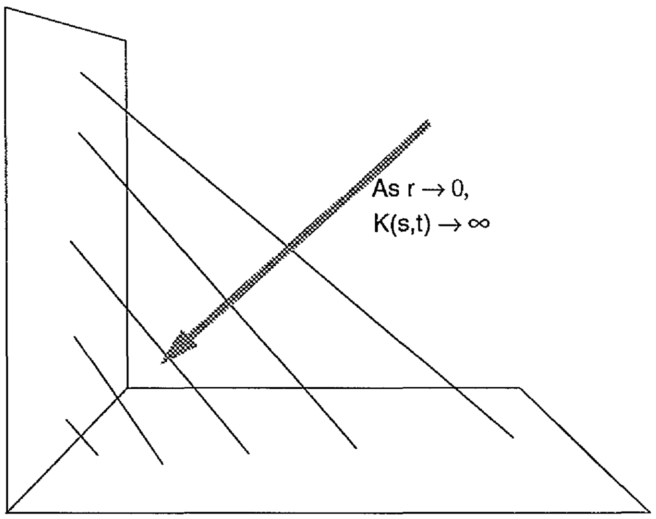


Figure 3.5: A singularity in the radiosity kernel: As $r \to 0$, $K(s,t) \to \infty$.

by Zatz [129, 130]. Zatz reports some impressively accurate results, whilst at the same time drawing attention to some problems inherent in applying the method to the radiosity equation.

By using basis functions of high enough order, Zatz concludes that radiosity transfer can be calculated with arbitrary accuracy between most pairs of surfaces. Problems arise, however, when the two surfaces meet along an edge or, less often, at a point. In these cases, the radiosity kernel (3.20) approaches infinity as a pole of order two as the area elements on the adjacent surfaces approach one another (figure 3.5). Such singularities in the kernel, can drastically affect the convergence of a quadrature rule being used to evaluate an integral involving the kernel — specifically, the K_{ij} of (3.24).

Thus far, we have only considered inner products of the form (3.8):

$$< x, y >= \int_a^b x(t)y(t)dt$$

Zatz overcomes the singularity problem by using a *weighted* inner product

$$< x, y >_{\mathcal{W}} = \int_a^b x(s)y(s)\mathcal{W}(s)ds \qquad (3.27)$$

whose weighting function $\mathcal{W}$ is specifically chosen to cancel out the pole.

A drawback of the method is that the weighting function $\mathcal{W}$, which is chosen to have a zero of multiplicity two along the problem edge, actually behaves like this along *all* edges, and so tends to make surfaces look overly dark near their edges, and bright in the middle.

When the chosen basis functions are the Legendre polynomials, the *un-weighted* inner product (3.8) is used, and singularities can cause convergence problems. When the (more computationally expensive) Jacobi polynomials are used, the weighted inner product (3.27) is used, the singularity is circumvented, but instead of convergence problems, 'dark edge' problems result. In order to balance the pros and cons of the two approaches, Zatz [129] uses a hybrid approach, whereby energy transfer between surfaces is evaluated using Legendre basis functions, unless the surfaces share a common edge. In this case, the problem is couched in terms of Jacobi polynomials, the singular transfer is evaluated, and the results converted back into Legendre bases.

Another problem Zatz encountered, was with discontinuities in the radiosity kernel, caused by occlusion. Whilst the true radiosity across a surface may exhibit discontinuities (of various orders [54]) in and around shadow regions, mimicking this behaviour with a linear combination of continuous basis functions, is not possible. When one attempts such a projection, a phenomenon known as Gibbs ringing [129] becomes evident. The shadow region appears as one might expect it to, but the area around the shadow exhibits *ripples* which are clearly erroneous. The higher the order of the basis functions being used, the less objectionable the error becomes, but it never goes away completely.

Zatz circumvented the problem by taking a step backwards, and removing the visibility term from the radiosity kernel, so that it could be handled separately. Energy interchange between surfaces takes place without considering occlusion, then the result is weighted by a *shadow mask*, defined over the receiving surface. The shadow mask is effectively a texture map defined over the receiver, taking values in the range $[0, 1]$ — 0 for umbra regions, 1 for lit regions, with intermediate values inside the penumbra. This approach improves the appearance of images, but does so using a particularly crude technique.

The next section describes an extension of the higher order algorithms presented thus far. These new *wavelet* algorithms attempt to incorporate the positive aspects of the higher order methods, whilst trying to steer clear of some of their pitfalls.

3.3 Wavelet radiosity

In the previous chapter, it was shown how hierarchical radiosity (HR) outperformed other constant radiosity algorithms: even though the environment may be split up into a large number of small elements, HR exploits the fact that if these elements are grouped hierarchically, energy interchange can be accurately evaluated by allowing, where appropriate, *groups* of elements to interact with other groups — thereby reducing an $\mathcal{O}(N^2)$ problem to $\mathcal{O}(N)$. Similarly, the preceding sections of this chapter have shown that treating the radiosity function as being piecewise polynomial, rather than piecewise constant, can also lead to improvements. *Wavelet radiosity* [45, 92] unifies these two approaches, allowing not only piecewise polynomial basis functions of order ≥ 0, but also a hierarchical treatment of the way in which the basis functions interact. Furthermore, the (hierarchical) bases are specifically constructed so that the discretized kernel matrix, which details how energy is exchanged between different basis functions, contains many negligible entries — allowing particularly fast solution methods.

3.3.1 Function projections

As with all the radiosity methods examined thus far, the aim is to find a function $x(s)$, which lies (inaccessibly) in Hilbert space. A more accessible function space V_n, spanned by the known basis functions $\{N_i\}_{i=1}^{i=n}$, is considered instead, with $x(s)$ being approximated by the linear combination:

$$P_{V_n}x(s) = \hat{x}(s) = \sum_{i=1}^{n} x_i N_i(s) \tag{3.28}$$

The approximate function $\hat{x}(s)$ is the *projection* of $x(s)$ into the basis set $\{N_i\}_{i=1}^{i=n}$, the operator which achieves the projection is P_{V_n}.

If $< x - \hat{x}, N_i >= 0$ for all basis functions N_i, then $\hat{x}(s)$ is an *orthogonal projection* of $x(s)$ into $\{N_i\}_{i=1}^{i=n}$. If the chosen basis functions form an orthonormal set, then this orthogonal projection relation can be used to find the unknown coefficients x_i:

$$< x - \hat{x}, N_i >= 0$$

$$< x, N_i > - \sum_{j=1}^{n} x_j < N_j, N_i >= 0$$

$$x_i =< x, N_i >$$

$$P_{V_n} x(s) = \hat{x}(s) = \sum_{i=1}^{n} < x, N_i > N_i(s) \qquad (3.29)$$

In the case of radiosity, the integral equation to be solved is:

$$x(s) = y(s) + \int_a^b K(s,t)x(t)dt \qquad (3.30)$$

Where $x(s)$ represents the final radiosity and $y(s)$ the emitted radiosity.

The equation

$$\hat{x}(s) = \hat{y}(s) + \sum_i < \int_a^b K(s,t)\hat{x}(t)dt, N_i(s) > N_i(s) \qquad (3.31)$$

is known as the *related integral equation*. Notice how the kernel has been allowed to operate on the approximate function $\hat{x}(s)$; an operation which will typically *not* leave the result in our chosen finite dimensional function space V_n, so this has been reprojected back into V_n, as per (3.29).

Solving (3.31) for the unknowns $x_i =< x, N_i >$ can be done as soon as the discrete kernel coefficients

$$K_{ij} = \int_a^b N_i(s) \int_a^b K(s,t) N_j(t) dt ds \qquad (3.32)$$

have been found. Notice that (3.32) is exactly (3.24), and that ensuring the projection $< x - \hat{x}, N_i >$ is orthogonal for every basis function N_i has simply resulted in a different derivation of the Galerkin method. The projection method derivation has been included here, because the notion of projecting a function into different basis sets is an important one, which can lead to computational savings.

All that remains, is to choose a suitable set of basis functions, evaluate the K_{ij}, and thence solve for the x_i. Notice that many different sets of basis functions can span the same function space — simply deciding to use n (orthogonal) polynomials of degree $0, \ldots, n-1$, is a long way short of uniquely identifying a suitable basis set. Different bases may represent a given function more efficiently than others. One family of basis sets, specifically constructed to yield efficient representations of a function, by exploiting any *smooth* sections it might have, are the wavelet bases:

3.3.2 Wavelet bases

Wavelet bases [11, 2] are a relatively new tool, even in the signal processing field, where they originated. Wavelet bases form hierarchical basis sets which, unlike the hierarchical bases discussed thus far [49, 50] (figure 3.6), do not simply represent each basis function as being the *average* of its children on the level below. Instead, the basis functions at each level in the hierarchy record what detail is lost between this level and the finer level, below.

To begin with, consider the *Haar* wavelet basis. Given two box functions:

$$\phi_{1,0}(s) = \begin{cases} 1 & \text{if } s \in [s_1, s_2] \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_{1,1}(s) = \begin{cases} 1 & \text{if } s \in (s_2, s_3] \\ 0 & \text{otherwise} \end{cases} \tag{3.33}$$

It is clear that these two functions span some simple function space, and that

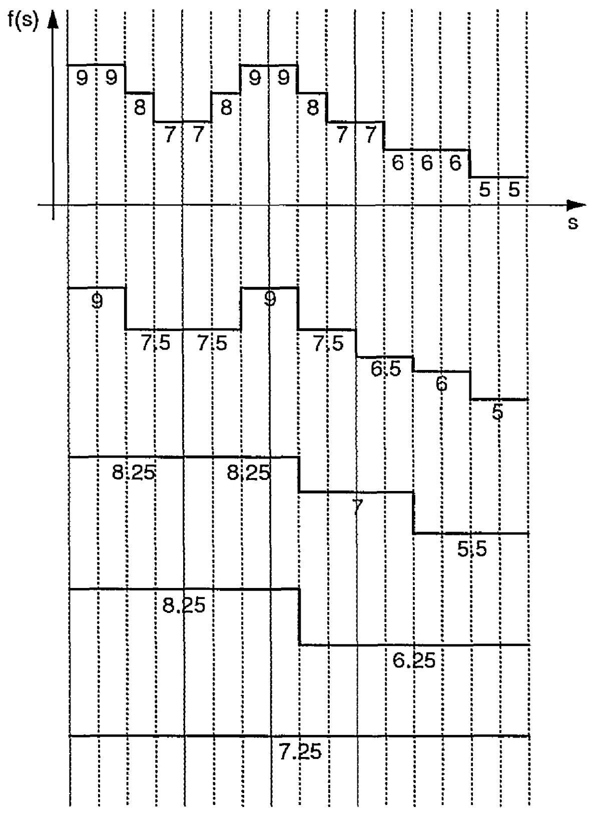


Figure 3.6: Standard hierarchical basis. A function is shown at five levels in a hierarchy. The values of the basis functions, which simply average their children's values, are shown.

any function $x(s)$ lying in this space can be written as a linear combination:

$$x(s) = \alpha\phi_{1,0}(s) + \beta\phi_{1,1}(s) \tag{3.34}$$

It is also clear that (3.34) can be manipulated to give:

$$x(s) = \frac{\alpha+\beta}{2}(\phi_{1,0}(s) + \phi_{1,1}(s)) + \frac{\beta-\alpha}{2}(\phi_{1,1}(s) - \phi_{1,0}(s)) \tag{3.35}$$

Introducing $\phi_0(s) = \phi_{1,0}(s)+\phi_{1,1}(s)$ and $\psi_{0,0}(s) = \phi_{1,1}(s)-\phi_{1,0}(s)$ it becomes clear that $\phi_0(s)$ and $\psi_{0,0}(s)$ together, span exactly the same space as $\{\phi_{1,0}(s), \phi_{1,1}(s)\}$. Whilst this observation may not seem significant in itself, and the subscripts may seem overly complex, the point to note is that the original box basis set has been replaced by one function (ϕ_0) which represents the *average* of the function over the interval, and another ($\psi_{0,0}$) which expresses how the function *differs* from the average.

The construction of the Haar basis (figure 3.7) begins with $n = 2^L$ (finest level) box functions $\{\phi_{L,j}(s)\}_{j=0}^{j=2^L-1}$ which span the chosen finite dimensional function space V_n. As described in the last paragraph, a *pair* of these functions $\phi_{L,2j}$, $\phi_{L,2j+1}$ can be replaced by a $\phi_{L-1,j}$ 'average' function and a $\psi_{L-1,j}$ 'difference' function. Pairwise replacement of *all* of the $\phi_{L,j}$ in this way results in $n/2$ $\phi_{L-1,j}$ functions (similar to the $\phi_{L,j}$, but twice as wide) and $n/2$ $\psi_{L-1,j}$ functions, which record how the $\phi_{L,j}$ differ from the $\phi_{L-1,j}$. In exactly the same way, the function space spanned by the $n/2$ $\phi_{L-1,j}$ functions can now be replaced by $n/4$ ϕ functions and $n/4$ ψ functions. This process recurses, until we are left with a single function ϕ_0 which represents the average of the function over the whole domain, and a hierarchy of functions $\{\psi_{i,j}\}_{i=0,j=0}^{i=L-1,j=2^i-1}$ which record the detail lost between the

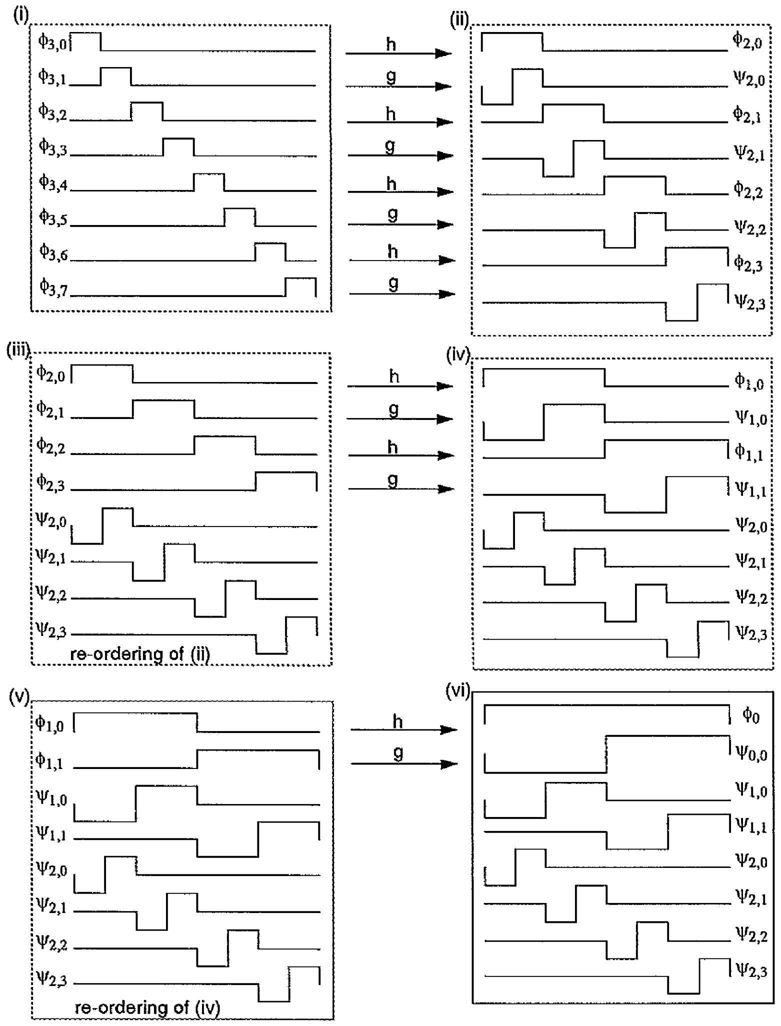


Figure 3.7: The (recursive) construction of the Haar basis set.

various levels of the hierarchy. Together, these n functions span the original space V_n, and are known as the *standard* Haar basis set.

The $\phi_{i,j}$ are known as *smooth* functions, whereas the $\psi_{i,j}$ are referred to as *detail* functions.

The *non-standard* Haar basis consists of the standard Haar basis, together with all of the $\phi_{i,j}$ functions ($i = 1, \ldots, L-1$) which were discarded and replaced by detail functions when creating the standard basis. Non-standard basis sets are discussed in section 3.3.5.

$$\begin{array}{ccccccc}
 & & & \phi_0 & & & \\
 & & & \psi_{0,0} & & & \\
 & & \psi_{1,0} & & \psi_{1,1} & & \\
 & \psi_{2,0} & \psi_{2,1} & & \psi_{2,2} & \psi_{2,3} & \\
 & & \cdots & & \cdots & & \\
\psi_{L-1,0} & \psi_{L-1,1} & & \cdots & & \psi_{L-1,2^{L-1}-2} & \psi_{L-1,2^{L-1}-1}
\end{array}$$

Figure 3.8: The standard wavelet basis can be regarded as a pyramid.

The Haar basis is the simplest of a whole family of wavelet bases, all of which are constructed in an analogous fashion — starting with a finest level that consists of a single smooth function, translated $n-1$ times, to give the $n = 2^L$ functions $\phi_{L,j}$ (figure 3.7.i). The functions in this finest level are linearly combined to give the $n/2$ smooth and $n/2$ detail functions at level $L-1$. This process recurses, as in the construction of the Haar basis. Finally, these bases consist of a single, coarsest level, smooth function $\phi_0(s)$, together with a pyramid of detail functions $\psi_{i,j}(s)$ (figure 3.8).

The smooth functions from one level (i) uniquely define the smooth and detail functions at the next most coarse level ($i-1$):

$$\begin{aligned}
\phi_{i-1,j} &= \sum_k h_{k-2j}\phi_{i,k} \\
\psi_{i-1,j} &= \sum_k g_{k-2j}\phi_{i,k}
\end{aligned} \tag{3.36}$$

The sequences h and g, which give the ϕ and ψ functions, respectively, can be thought of low-pass and high-pass filters, respectively. The relationship (3.36) is known as the *two-scale relationship* for the basis in question.

The two-scale relationship is constructed to ensure that *any* function in the non-standard basis can be expressed as a single function (defined over $[0,1]$) suitably scaled and translated. That is, one function (ϕ) for the smooth functions, and one (ψ) for the detail functions:

$$\phi_{i,j}(s) = 2^{i/2}\phi(2^i s - j)$$
$$\psi_{i,j}(s) = 2^{i/2}\psi(2^i s - j) \tag{3.37}$$

This means that $\phi_{i-1,j}$ is identical to $\phi_{i,j}$ except it is twice as wide, and $1/\sqrt{2}$ times as tall — proportions which ensure that the inner products $< \phi_{i,j}, \phi_{i,k} >$ remain independent of i. In the Haar basis, the original smooth function (ϕ_0 in figure 3.7) is a simple box function, and the original detail function ($\psi_{0,0}$ in figure 3.7) is the difference of two adjacent box functions.

3.3.3 Projections into wavelet bases

The upshot of all this theory, is that not only can an arbitrary function $x(s)$ be projected into an L-level wavelet basis using

$$P_L x(s) = \hat{x}(s) = < x, \phi_0 > \phi_0(s) + \sum_{i=0}^{L-1} \sum_{j=0}^{2^i-1} < x, \psi_{i,j} > \psi_{i,j}(s) \tag{3.38}$$

but also, that costly quadrature routines need not be employed to evaluate the inner products which appear. Once the inner products $< x, \phi_{L,j} >$ have been found by solving the related integral (3.31) with $N_i = \phi_{L,i}$, then the two-scale relationship (3.36) can be used to evaluate the coefficients corresponding to the detail functions further up the hierarchy.

A *pyramid* algorithm will, given the n coefficients $< x, \phi_{L,j} >$, apply the two-scale relationship to these coefficients, taking $O(n)$ steps to find the $n/2$ coefficients $< x, \phi_{L-1,j} >$ together with the $n/2$ coefficients $< x, \psi_{L-1,j} >$. The algorithm can then take the $< x, \phi_{L-1,j} >$ coefficients and re-apply the two-scale relationship.

In this way, a pyramid of inner products is returned by the algorithm in $O(n + \frac{n}{2} + \frac{n}{4} + \cdots + 1) = O(n)$ steps.

Reversing the process is also possible. Here, the two-scale relationship is utilised to return the coefficients corresponding to the smooth functions of level i, given only the coefficients corresponding to the smooth and detail functions of level $i - 1$.

Because a standard wavelet basis stores *detail* functions in its hierarchy, if a function is well approximated, over some part of its domain, by the smooth function $\phi_{i,j}$ (say), then the coefficient corresponding to the detail function $\psi_{i,j}$, which records how the function differs from the smooth function, will have to be very small. If sufficiently small coefficients are ignored (i.e., set to zero) then functions which are suitably smooth can be represented by approximate wavelet projections, with few non-zero coefficients — a saving which can prove most valuable (figure 3.9).

Clearly, the more near-zero coefficients in (3.38), the cheaper is the approximation. Will the cost of the projection[4] vary for different wavelet bases? The answer is yes: consider the inner product $< x, \psi_{i,j} >$, one of the coefficients from (3.38):

$$< x, \psi_{i,j} >= \int_a^b x(s)\psi_{i,j}(s)ds \tag{3.39}$$

Now, if the function $x(s)$ is closely approximated (wherever $\psi_{i,j}$ is non-zero) by some polynomial:

$$x(s) \approx \alpha_0 + \alpha_1 s + \alpha_2 s^2 + \cdots + \alpha_{M-1} s^{M-1}$$

[4]where 'cost' equates to the number of significant coefficients in (3.38)

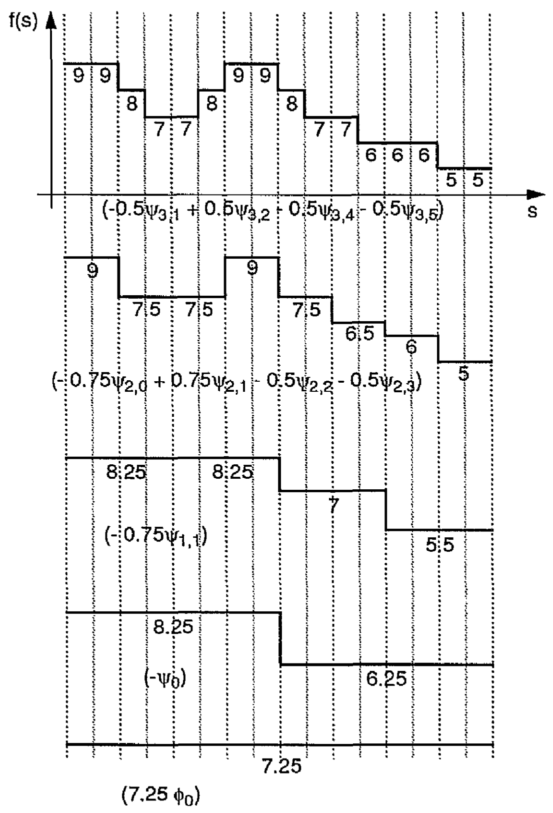


Figure 3.9: Projecting a function into a wavelet basis may lead to only a small number of significant coefficients. From bottom to top, more and more basis functions are included, only 11 coefficients are non-zero in this example.

then (3.39) can be re-written:

$$< x, \psi_{i,j} > \approx \sum_{k=0}^{M-1} \alpha_k \int_a^b \psi_{i,j}(s) s^k ds$$

And it becomes clear that a sufficient condition for the inner product (3.39) to be small is:

$$\int_a^b \psi_{i,j}(s) s^k ds = 0, \quad \forall k = 0, \ldots, M-1 \tag{3.40}$$

Functions $\psi_{i,j}$ which satisfy (3.40) are said to have M *vanishing moments*. Wavelet bases whose detail functions have M vanishing moments will generate near-zero coefficients in regions where the function can be closely approximated by a polynomial of degree $M - 1$. The Haar wavelet basis has one vanishing moment, and (consequently) detail coefficients are small wherever the function is nearly constant, over a suitable region of the domain. In figure 3.9, 11 Haar coefficients completely describe a function which one might have expected to take 16 coefficients.

3.3.4 Flatlets and multiwavelets

Two families of wavelet bases whose detail functions can have an arbitrary number of vanishing moments are the *flatlets* [45, 92] and the *multiwavelets* [1].

Flatlet bases, like the Haar basis, are made up entirely of piecewise constant functions. Unlike the Haar basis, a flatlet basis is not, strictly speaking, a wavelet basis, since its detail functions will not all be scales of a single shape $\psi(s)$. For example, the flatlet basis $\mathcal{F}_2$, whose detail functions each have 2 vanishing moments, is constructed by taking translates of 2 adjacent box functions ϕ^1, ϕ^2, then constructing a two-scale relationship which ensures that the next level up

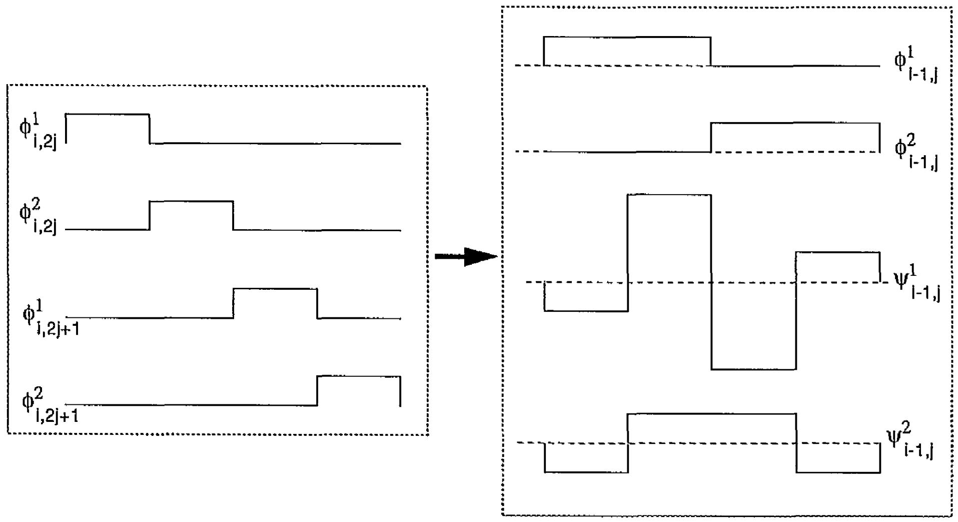


Figure 3.10: Constructing the $\mathcal{F}_2$ flatlet basis.

the hierarchy consists of box functions twice as wide as the original ones, and detail functions which each have 2 vanishing moments (see figure 3.10). This two-scale relationship can be conveniently represented in matrix/vector form:

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ -1 & 3 & -3 & 1 \\ -1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} \phi^1_{i,2j} \\ \phi^2_{i,2j} \\ \phi^1_{i,2j+1} \\ \phi^2_{i,2j+1} \end{pmatrix} = \begin{pmatrix} \phi^1_{i-1,j} \\ \phi^2_{i-1,j} \\ \psi^1_{i-1,j} \\ \psi^2_{i-1,j} \end{pmatrix} \tag{3.41}$$

The first two rows of this matrix have been chosen to give the two wide box functions. The second two rows have been constructed to give functions with 2 vanishing moments (i.e., the rows are orthogonal to constant and linear variation; represented by the vectors $(1,1,1,1)$ and $(0,1,2,3)$, respectively).

Similarly, the flatlet basis $\mathcal{F}_3$, whose 3 detail functions each have 3 vanishing moments, is constructed by taking translates of 3 adjacent box functions, and building a two-scale relationship (a 6×6 matrix, now) whose top 3 rows ensure 3 box functions twice as wide as the originals, and each of whose bottom 3 rows gives a detail function with *at least* 3 vanishing moments. Clearly, in constructing each of these bottom rows, there are 5 degrees of freedom, but only the need to keep them orthogonal to constant, linear and quadratic variation (for 3 vanishing moments). The extra degrees of freedom are accounted for by forcing one detail function to have 5 vanishing moments, another to have 4 vanishing moments and be orthogonal to the first, and the last to have 3 vanishing moments and be orthogonal to the first two. In fact, the generalisation of this construction method can be used to generate a two-scale relationship for flatlets $\mathcal{F}_M$ whose detail functions each have at least M vanishing moments [92].

Notice that (figure 3.10) since the degree of the flatlet basis functions remains constant (0), it is necessary to widen their support as M increases, with a consequent increase in computational cost.

The multiwavelet family of bases which, like the flatlets, are not strictly wavelets, are where hierarchical methods meet higher order methods. A Multiwavelet basis $\mathcal{M}_M$, is constructed by taking translates of the first M Legendre polynomials, and combining them with a suitable two-scale relationship to give smooth functions which are scaled Legendre polynomials, and detail functions which each have M vanishing moments. The $\mathcal{M}_2$ multiwavelet basis functions are shown in figure 3.11. Its (normalised) two-scale relationship is given by:

$$\frac{1}{2\sqrt{2}}\begin{pmatrix} 2 & 0 & 2 & 0 \\ -\sqrt{3} & 1 & \sqrt{3} & 1 \\ 0 & -2 & 0 & 2 \\ 1 & \sqrt{3} & -1 & \sqrt{3} \end{pmatrix}\begin{pmatrix} \phi^1_{i,2j} \\ \phi^2_{i,2j} \\ \phi^1_{i,2j+1} \\ \phi^2_{i,2j+1} \end{pmatrix} = \begin{pmatrix} \phi^1_{i-1,j} \\ \phi^2_{i-1,j} \\ \psi^1_{i-1,j} \\ \psi^2_{i-1,j} \end{pmatrix} \tag{3.42}$$

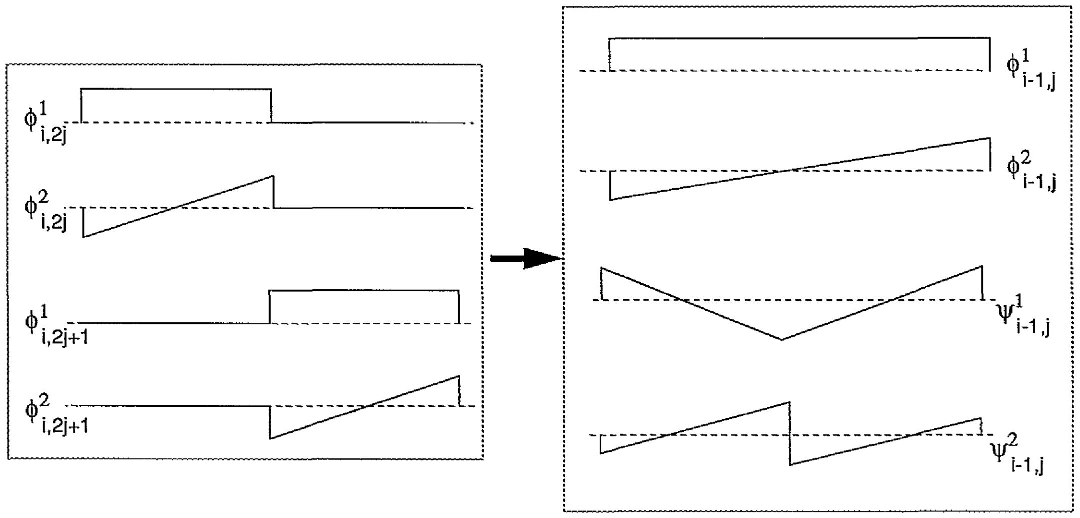


Figure 3.11: Constructing the $\mathcal{M}_2$ multiwavelet basis.

Whereas the multiwavelets necessitate higher order quadrature methods, for function projections, they do offer the advantages of increased convergence rates [28] and smooth basis functions with which to represent the (mainly smooth, in the case of radiosity) results.

The bases $\mathcal{F}_1$ and $\mathcal{M}_1$ are both identical to the Haar wavelet basis.

3.3.5 Flatland radiosity with wavelets

The previous sections have outlined how wavelet (and related) bases can be used to represent a one-dimensional function, to a high degree of accuracy, with less terms than one would normally expect, given the dimension of the function spaces under consideration. In order to model radiative transfer between surfaces, using wavelets, it will not only be necessary to develop two-dimensional wavelet bases with which to model the radiosity across a bi-parametric surface, but also a four-dimensional wavelet basis for the corresponding discretized kernel function. The hope is that (as was the case with one-dimensional functions) representing the kernel in a *wavelet* basis will result in many near-zero terms which, when ignored, will lead to faster algorithms which do not suffer unduly from loss in accuracy.

Before tackling those problems, this section will consider the case of *flatland* radiosity [54], where the radiosity along a *line*, rather than across a surface, is considered, and the discretized kernel can be represented in a two-dimensional function space:

Recall the projection (3.38) of an arbitrary function $x(s)$ into an L-level ($n = 2^L$ dimensional) wavelet basis:

$$P_L x(s) = \hat{x}(s) =< x, \phi_0 > \phi_0(s) + \sum_{i=0}^{L-1} \sum_{j=0}^{2^i-1} < x, \psi_{i,j} > \psi_{i,j}(s)$$

In the related integral equation (3.31) the kernel is first allowed to act on the projected function $\hat{x}$, and then the resulting function is projected back into V_n, the function space we are restricting ourselves to. So, whereas the original integral equation can be written:

$$x = y + \mathcal{K}x$$

The related integral equation can now be written:

$$\hat{x} = \hat{y} + P_L \mathcal{K} P_L \hat{x}$$

The two applications of the operator P_L (one before the application of the kernel, and one after) ensure that

1. the kernel only operates on the basis functions of interest, and
2. rather than obtaining the full function that results from this limited application of the kernel, the result is confined to the function space of interest.

Standard basis kernel

If the operator P_i denotes projection into the basis $\{\phi_{i,j}\}_{j=0}^{j=2^i-1}$, and Q_i denotes projection into the basis $\{\psi_{i,j}\}_{j=0}^{j=2^i-1}$, then the operator $P_L\mathcal{K}P_L$ can be expanded, using the identity $P_L = P_0 + \sum_{i=1}^{L-1} Q_i$:

$$\begin{aligned} P_L\mathcal{K}P_L &= (P_0 + \sum_{i=1}^{L-1} Q_i)\mathcal{K}(P_0 + \sum_{i=1}^{L-1} Q_i) \\ &= P_0\mathcal{K}P_0 + \sum_{i=1}^{L-1} P_0\mathcal{K}Q_i + \sum_{i=1}^{L-1} Q_i\mathcal{K}P_0 + \sum_{i,k=1}^{L-1} Q_i\mathcal{K}Q_k \end{aligned} \tag{3.43}$$

Each of the terms $L\mathcal{K}R$ in the above expansion[5] describes how the kernel projects a function lying in the space corresponding to L into the space corresponding to R. The expansion effectively represents a projection into the space spanned by the basis functions:

$$\begin{array}{cc} \phi_o(s)\phi_0(t) & \phi_0(s)\psi_{i,j}(t) \\ \psi_{k,l}(s)\phi_0(t) & \psi_{k,l}(s)\psi_{i,j}(t) \end{array} \tag{3.44}$$

Where $i, k = 0, \ldots, L-1, j = 0, \ldots, 2^i - 1$, and $l = 0, \ldots, 2^k - 1$.

The discretized kernel coefficients corresponding to this basis, can be found from the coefficients of the matrix (3.32) — which maps the $\{\phi_{L,j}\}_{j=0}^{j=2^L-1}$ onto themselves. This is achieved by taking one column of the matrix at a time and, as was done in the one-dimensional case, recursively applying the two-scale relationship (3.36). The j^{th} such column (transformed) describes the projection of $\mathcal{K}\phi_{L,j}$ into the wavelet basis. The discretized kernel now describes a mapping from $\{\phi_{L,j}\}_{j=0}^{j=2^L-1}$ into $\{\phi_0, \psi_{i,j}\}_{i,j=0}^{i=L-1,j=2^i-1}$. The desired matrix is achieved by repeating the recursive application of the two-scale relationship on the *rows* of the

[5]where L & R are operators which project a function into one of the rows of figure 3.8.

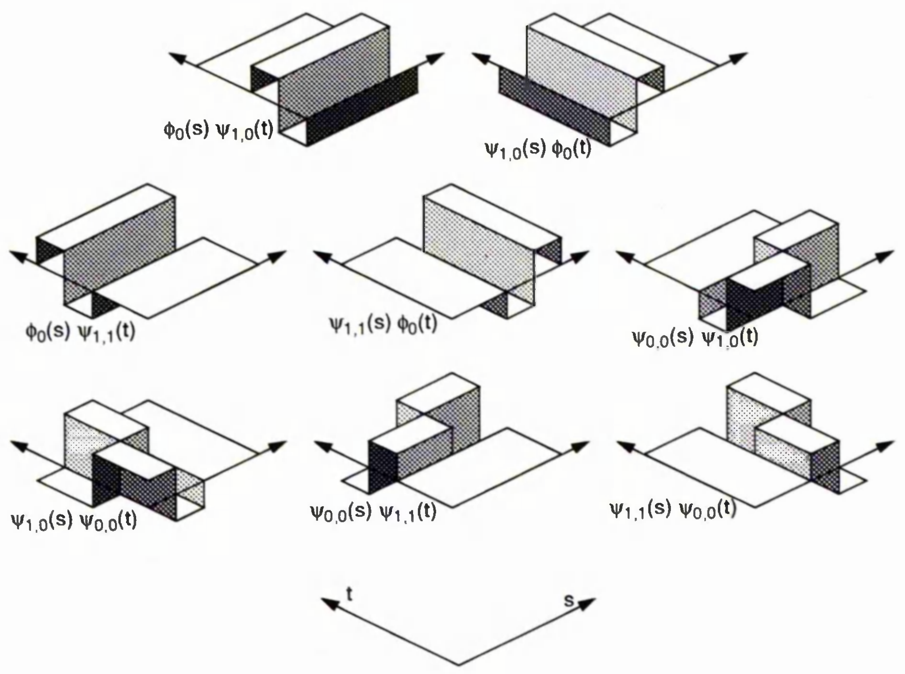


Figure 3.12: The eight unique functions of the standard 2d Haar basis, for the case $L = 2$. The other eight functions, which are shared with the non-standard basis, can be seen in figure 3.13

recently column-transformed matrix. Each of the terms $L\mathcal{K}R$ in expansion (3.43) represents a block of this projected kernel matrix.

It is now possible to store all radiosity functions (along a line) in a one-dimensional wavelet basis, and use this (hopefully sparse) projected kernel to solve directly in terms of the wavelet basis.

The standard two-dimensional wavelet basis (3.43) has not proved popular, however, when compared to the non-standard version [11, 45, 92]. By first column-transforming the original matrix, and then carrying out transformations on rows consisting of already-transformed entries, it was felt [92] that any smoothness present in the rows may have been lost during column-transformation, and so was not being exploited. A more rigourous argument is given in [11], where it is shown that whilst one can ignore (i.e., set to zero) all but the biggest $\mathcal{O}(n \log n)$

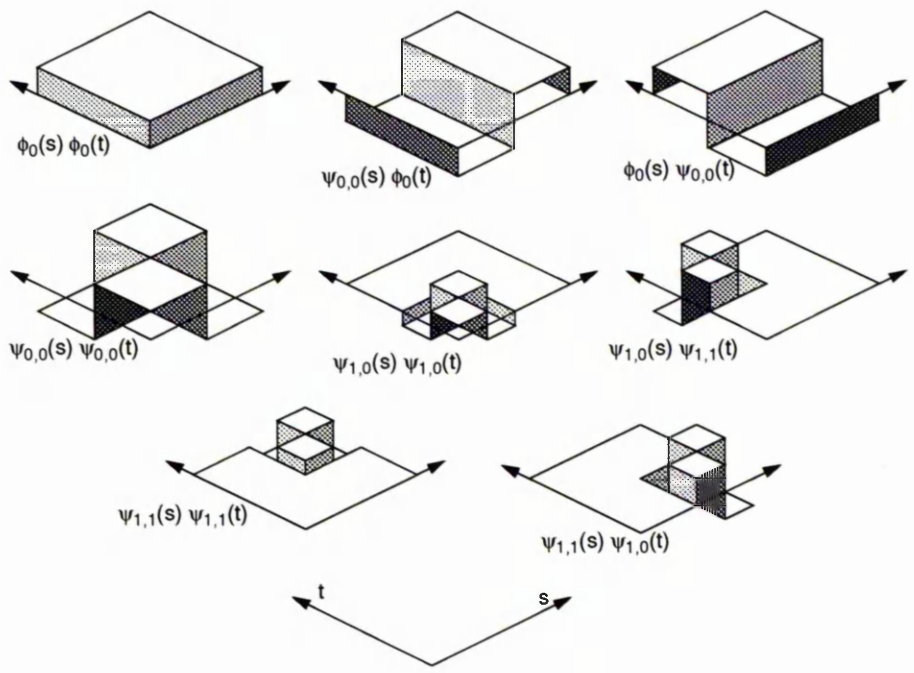


Figure 3.13: The eight functions common to both the standard and non-standard 2d Haar bases, for the case $L = 2$. See also figures 3.12 and 3.14

terms in the standard kernel, this figure is only $\mathcal{O}(n)$ for the non-standard kernel. It is for these reasons that the non-standard option is now examined.

Non-standard basis kernel

As has already been briefly mentioned, the one-dimensional *non-standard* wavelet basis consists of all those functions found in the standard basis, together with all of the smooth functions (generated by application of the two-scale relationship) which were discarded when constructing the standard basis, i.e.,

$$\{\phi_{i,j}, \psi_{i,j}\}_{i,j=0}^{i=L-1, j=2^i-1}$$

This[6] is better viewed as an over-representation of a basis set, rather than as a basis set *per se*, since if either of the $\phi_{i,j}, \psi_{i,j}$ pyramids were discarded, the

[6]The function $\phi_{0,0}$ is simply ϕ_0, and appears in this thesis with a double-zero wherever a single zero would lead to overly-complex indexing.

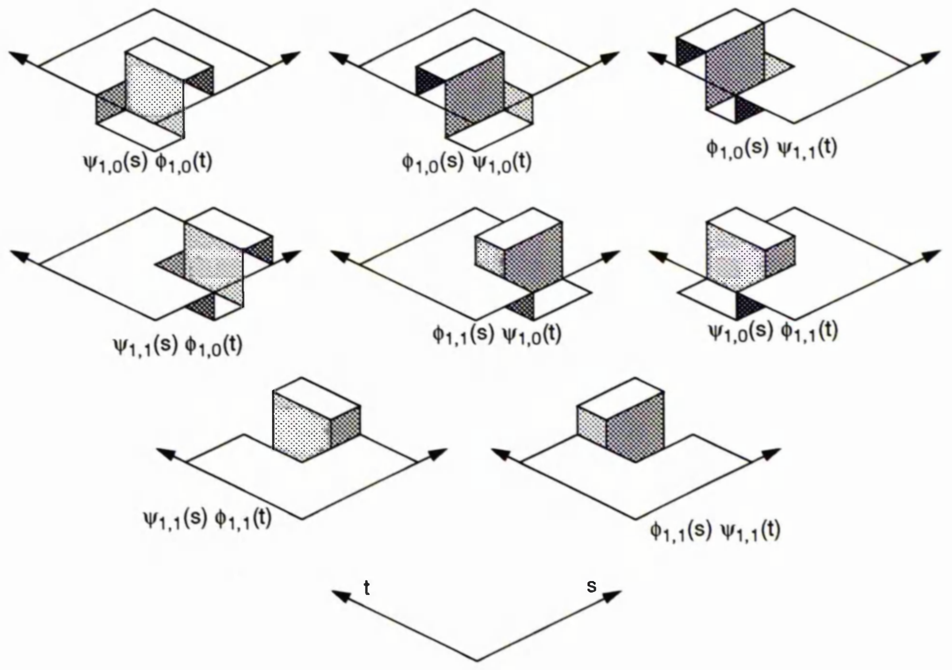


Figure 3.14: The eight unique functions of the non-standard 2d Haar basis, for the case $L = 2$. The other eight functions, which are shared with the standard basis, can be seen in figure 3.13

remaining functions would still span the same space.

In order to overcome the problems which seem to have arisen, by first column-transforming the whole matrix and *then* row-transforming, the transformations in the different directions are interleaved. As before, the operator of interest is $P_L\mathcal{K}P_L$. Consider the following, which is identically true:

$$P_L\mathcal{K}P_L = P_0\mathcal{K}P_0 + \sum_{i=0}^{i=L-1} (P_{i+1}\mathcal{K}P_{i+1} - P_i\mathcal{K}P_i)$$

But, by definition; $P_{i+1} = P_i + Q_i$, so:

$$\begin{aligned} P_{i+1}\mathcal{K}P_{i+1} - P_i\mathcal{K}P_i &= (P_i + Q_i)\mathcal{K}(P_i + Q_i) - P_i\mathcal{K}P_i \\ &= P_i\mathcal{K}Q_i + Q_i\mathcal{K}P_i + Q_i\mathcal{K}Q_i \end{aligned}$$

And so:

$$P_L \mathcal{K} P_L = P_0 \mathcal{K} P_0 + \sum_{i=0}^{i=L-1} P_i \mathcal{K} Q_i + \sum_{i=0}^{i=L-1} Q_i \mathcal{K} P_i + \sum_{i=0}^{i=L-1} Q_i \mathcal{K} Q_i \tag{3.45}$$

Which represents a projection into the space spanned by the basis functions:

$$\begin{array}{ll} \phi_o(s)\phi_0(t) & \phi_{i,l}(s)\psi_{i,j}(t) \\ \psi_{i,l}(s)\phi_{i,j}(t) & \psi_{i,l}(s)\psi_{i,j}(t) \end{array} \tag{3.46}$$

Where $i = 0, \ldots, L-1, j, l = 0, \ldots, 2^i - 1$.

Notice now that only functions on the same level in the basis hierarchy interact with one another. Practically-speaking, such a kernel is obtained by taking the usual kernel matrix (mapping the $\{\phi_{L,j}\}_{j=0}^{j=2^L-1}$ onto themselves) and applying the two-scale relationship once to every row, and then once to each of the resulting columns. The resultant matrix maps the $\{\phi_{L-1,j}, \psi_{L-1,j}\}$ onto themselves — specifically, the matrix will consist of four $n/2 \times n/2$ blocks which describe how the kernel maps:

- the $\{\phi_{L-1,j}\}$ onto the $\{\psi_{L-1,j}\}$,
- the $\{\psi_{L-1,j}\}$ onto the $\{\phi_{L-1,j}\}$,
- the $\{\psi_{L-1,j}\}$ onto themselves, and
- the $\{\phi_{L-1,j}\}$ onto themselves.

The first three of these blocks correspond to terms in expansion (3.45): $P_{L-1}\mathcal{K}Q_{L-1}$, $Q_{L-1}\mathcal{K}P_{L-1}$ and $Q_{L-1}\mathcal{K}Q_{L-1}$ respectively. To obtain the rest of the terms in (3.45), the interleaved transformation procedure recurses with the fourth block (corresponding to $P_{L-1}\mathcal{K}P_{L-1}$), resulting in four $n/4 \times n/4$ blocks, corresponding to the

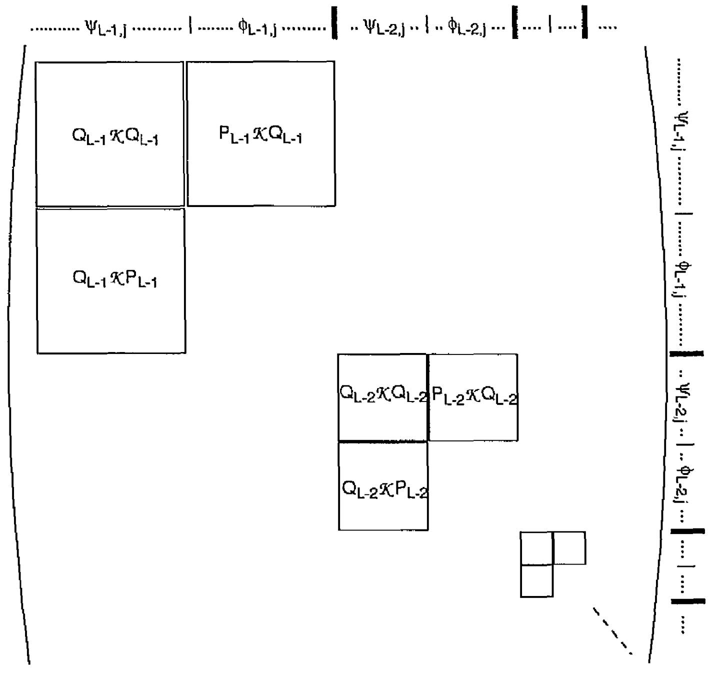

$\psi_{L-1,j}$
$\phi_{L-1,j}$
$\psi_{L-2,j}$
$\phi_{L-2,j}$
$Q_{L-1}\mathcal{K}Q_{L-1}$
$P_{L-1}\mathcal{K}Q_{L-1}$
$Q_{L-1}\mathcal{K}P_{L-1}$
$Q_{L-2}\mathcal{K}Q_{L-2}$
$P_{L-2}\mathcal{K}Q_{L-2}$
$Q_{L-2}\mathcal{K}P_{L-2}$

Figure 3.15: Re-arranging the kernel matrix for a non-standard wavelet basis.

terms $P_{L-2}\mathcal{K}Q_{L-2}$, $Q_{L-2}\mathcal{K}P_{L-2}$, $Q_{L-2}\mathcal{K}Q_{L-2}$ and $P_{L-2}\mathcal{K}P_{L-2}$ in (3.45). Always recursing with the $P_i\mathcal{K}P_i$ block eventually accounts for all of the terms in the expansion (3.45) of the projected kernel.

Arranging these terms in an $n \times n$ matrix does not make matrix/vector multiplication convenient when the vector is expanded in terms of the non-standard basis. Consequently, the blocks are arranged as per figure 3.15, with all entries outside the blocks being equal to zero. Whilst this matrix is twice as wide and twice as high as the original matrix, it has no more non-zero, and many more near-zero, entries.

Each triplet of blocks (figure 3.15) describes how one level of the wavelet hierarchy interacts with itself. Consequently, there are L such groupings, in total, together with a solitary entry corresponding to $P_0\mathcal{K}P_0$. The individual entries in the i^{th} such triplet are given by:

$Q_i\mathcal{K}Q_i$ **block:** $K^{\alpha}_{i,j,k} = K_{\psi_{i,j},\psi_{i,k}} = \int_a^b \int_a^b K(s,t)\psi_{i,j}(s)\psi_{i,k}(t)dsdt$

$P_i\mathcal{K}Q_i$ **block:** $K^{\beta}_{i,j,k} = K_{\phi_{i,j},\psi_{i,k}} = \int_a^b \int_a^b K(s,t)\phi_{i,j}(s)\psi_{i,k}(t)dsdt$

$Q_i\mathcal{K}P_i$ **block:** $K^{\gamma}_{i,j,k} = K_{\psi_{i,j},\phi_{i,k}} = \int_a^b \int_a^b K(s,t)\psi_{i,j}(s)\phi_{i,k}(t)dsdt$

where $j,k = 0,\ldots,2^i-1$.

The solitary entry, corresponding to $P_0\mathcal{K}P_0$ is given by:

$$K^{\phi} = K_{\phi_0,\phi_0} = \int_a^b \int_a^b K(s,t)\phi_0(s)\phi_0(t)dsdt$$

Energy transfer

Now that the kernel matrix can be directly manipulated so that it corresponds to the non-standard wavelet basis, how can this be used to solve for the radiosity distribution along the *lines* that make up the two-dimensional flatland scene?

If the radiosity along some line is given by the function $x(s)$, then the projection of this function into the L-level non-standard wavelet basis is given by:

$$\begin{aligned}\hat{x}(s) =< x,\phi_0 > \phi_0(s) \quad &+ \quad \sum_{i=1}^{L-1}\sum_{j=0}^{2^i-1} < x,\phi_{i,j} > \phi_{i,j}(s) \\ &+ \quad \sum_{i=0}^{L-1}\sum_{j=0}^{2^i-1} < x,\psi_{i,j} > \psi_{i,j}(s) \qquad (3.47)\end{aligned}$$

Allowing the projected kernel to act on this function, we have:

$$P_L\mathcal{K}P_L\hat{x}(s) = K^{\phi} < x,\phi_0 > \phi_0(s) + \sum_{i=0}^{L-1}\sum_{j,k=0}^{2^i-1} K^{\alpha}_{i,j,k} < x,\psi_{i,k} > \psi_{i,j}(s) +$$
$$\sum_{i=0}^{L-1}\sum_{j,k=0}^{2^i-1} K^{\beta}_{i,j,k} < x,\psi_{i,k} > \psi_{i,j}(s) + \sum_{i=0}^{L-1}\sum_{j,k=0}^{2^i-1} K^{\gamma}_{i,j,k} < x,\phi_{i,k} > \phi_{i,j}(s) \qquad (3.48)$$

It is important to remember, however, that whilst (3.48) contains n^2 entries from the projected kernel matrix $P_L\mathcal{K}P_L$, great care has been taken to project the kernel into a basis which will result in many of these entries being near-zero. If the effect of the projected kernel is only needed to within some finite precision, then all but the largest m (say) of these terms can be set to zero, and matrix/vector multiplication becomes an $\mathcal{O}(m)$ problem, rather than $\mathcal{O}(n^2)$.

```
$\hat{K}$ = projectKernel();
$< x, \phi_{L,j} >=< y, \phi_{L,j} >$;
while ( not converged )
        ($< x, \phi_{i,j} >, < x, \psi_{i,j} >$) = Pull( $< x, \phi_{L,j} >$ );
        ($< g, \phi_{i,j} >, < g, \psi_{i,j} >$) = Gather( $\hat{K}, < x, \phi_{i,j} >, < x, \psi_{i,j} >$ );
        $< g, \phi_{L,j} >$ = Push( $< g, \phi_{i,j} >, < g, \psi_{i,j} >$ );
        $< x, \phi_{L,j} >=< g, \phi_{L,j} > + < y, \phi_{L,j} >$;
Display ( $< x, \phi_{L,j} >$ );
```

Figure 3.16: Radiosity pseudo-code for a non-standard wavelet basis

How can this be usefully applied to solve for the unknown $\hat{x}(s)$ in the related integral equation (3.31)? Gortler *et al* [45, 92] describe a three-phase algorithm, using terminology from previous hierarchical algorithms [50]:

Pull: Project $x(s)$ into the non-standard wavelet basis: Given the n inner products $< x, \phi_{L,j} >$, recursively apply the two-scale relationship (3.36) until the $2n$ inner products which appear in (3.47) have been found.

Gather: Allow the projected kernel $P_L \mathcal{K} P_L$ to operate on the projected function $\hat{x}(s)$, using (3.48). This corresponds to one *bounce* of light around the scene, or accounting for a single term in the Neumann expansion (3.7).

Push: Project the resulting function back into the basis $\{\phi_{L,j}\}_{j=0}^{2^L-1}$; an operation needed between each `Gather` and also for the displaying of the results. This involves recursive use of the *inverse* two-scale relationship which, given 2^{i-1} $\phi_{i-1,j}$ functions and 2^{i-1} $\psi_{i-1,j}$ functions will return the 2^i $\phi_{i,j}$ functions from the next-most-fine level in the basis function hierarchy.

This `Pull-Gather-Push` process can be wrapped in a loop and repeatedly applied until convergence — i.e., Jacobi iteration: This algorithm (figure 3.16) is specific to the non-standard basis. The `Push` and `Pull` operations are only *inside* the `while` loop because of the presence of the the $\phi_{i,j}$ functions in the non-standard basis: these functions represent the *average* over some region, so the energy collected at one level (as with conventional hierarchical radiosity [50]) cannot be regarded in isolation. At first glance, the presence of the `Push/Pull` routines inside the `while` loop may seem excessive, but they account for the energy interchange between functions at *different levels* of the basis hierarchy — which was specifically *not* accounted for in the non-standard projected kernel matrix (expansion (3.45)). With the standard basis, this is not the case, but the cost for taking the `Push/Pull` routines outside of the `while` loop comes in the form of $\log n$ times more links in the `Gather` routine: intuitively, each level now links with every other level in the hierarchy ($L = \log_2 n$ levels). It was for this reason that the standard basis was shunned in the first place.

3.3.6 3D radiosity with wavelets

For 3D radiosity, where the radiosity across a surface is now regarded as being a function of two variables, the algorithm proceeds exactly as described in the last section, but with the radiosity now being represented by a 2D wavelet basis, and the kernel now a function of 4 variables.

How can the radiosity across some surface be represented in a wavelet basis?

Consider a projection of the radiosity function $x(s,t)$ into some finest-level, tensor product, 2D basis:

$$\hat{x}(s,t) = \sum_{j,k=0}^{2^L-1} < x, \phi_{L,j}\phi_{L,k} > \phi_{L,j}(s)\phi_{L,k}(t) \tag{3.49}$$

This can be represented as an $n \times n$ matrix of inner products which can be transformed (to represent a basis change into a 2D non-standard wavelet basis) in exactly the same way the two-dimensional kernel matrix was transformed in the flatland case. This results in a $2n \times 2n$ matrix arranged as shown in figure 3.15. This is a 2D `Pull` routine, a 2D `Push` routine can be similarly constructed using the inverse two-scale relationship.

The basis change for the four-dimensional kernel matrix (stored in a 4D array with entries K_{ijkl}) is achieved in a manner analogous to the 2D case: the two-scale relationship is first applied to each of n^3 vectors $\{K_{ijkl}\}_{i=0}^{n-1}$ (fixed j, k, l). A similar transformation is then carried out on the $\{K_{ijkl}\}_{j=0}^{n-1}$, then the $\{K_{ijkl}\}_{k=0}^{n-1}$, and finally the $\{K_{ijkl}\}_{l=0}^{n-1}$. This process then recurses with the $(n/2)^4$ entries which represent $P_{L-1}P_{L-1}\mathcal{K}P_{L-1}P_{L-1}$, exactly as per the 2D case. The transformed kernel matrix can now be incorporated into a 4D `Gather` routine, and the solution process proceeds exactly as in the flatland case.

$\mathcal{O}(m)$ kernel construction

One problem is apparent: the 'bottom-up' kernel construction algorithm just described will require $\mathcal{O}(n^4)$ storage and time, since no matter how few entries are significant at the end of the process, we begin with n^4 such terms. If the wavelet kernel matrix contains m significant terms, then figure 3.16 certainly describes an $\mathcal{O}(m)$ *solution* process — ideally, this should be accompanied by an $\mathcal{O}(m)$ algorithm, for the *construction* of the projected kernel matrix.

A 'top-down' approach, which utilises a decision-making *oracle*, is proposed by Gortler *et al* [45], in an attempt to reduce the complexity of constructing the projected kernel matrix.

A quadtree algorithm is implemented, as per conventional hierarchical radiosity [49, 50]. Given two patches (or parts thereof) the oracle must decide whether or not the region of the kernel responsible for the interaction of the two patches[7] is sufficiently smooth, or not. If the wavelet basis being used, has detail functions with M vanishing moments, then the oracle must decide whether the region of the kernel under consideration can be reasonably represented by a polynomial of degree $M - 1$ or less.

In conventional hierarchical radiosity (i.e., $M = 1$), information concerning the size, orientation, separation distance, and inter-visibility of the two patches was used to construct a similar oracle [49, 50]. This was possible because it is well understood how all of the these quantities contribute to the fraction of energy leaving one patch which reaches another. In the case of wavelets, particularly those flatlets and multiwavelets which have such desirable vanishing moment properties, it is *not* well-understood how such geometric snippets can be used to reach conclusions about the function under consideration.

Gortler *et al*'s oracle [45] begins by trying to establish interaction between the two patches at a coarse level of detail and then, if that proves unsatisfactory,

[7]Actually, the basis functions whose domains correspond to the two patches.

recurses to the patches' children and checks if *they* will interact at the next-most-fine level. In order to establish whether the current level of interaction is fine enough, their oracle directly evaluates the part of the kernel corresponding to the current level and the two patches[8], using a Gauss-Legendre quadrature rule [28]. A polynomial of degree $M-1$ is then used to interpolate between the values sampled during quadrature. A second quadrature rule, which samples *in between* those points used by the Gauss-Legendre routine, is then employed to obtain a measure of how much the interpolating function differs from the Gauss-Legendre result — an error measure. If this error is too large, the oracle advises recursion, otherwise it is assumed that the kernel is sufficiently smooth.

Note that the part of the projected kernel matrix, which corresponds to the interaction between two patches, actually covers sixteen different interactions (corresponding to $\{AB\mathcal{K}CD\}_{A,B,C,D=P_i,Q_i}$) between the different 2D wavelet basis functions whose domains *define* the patches. But since the inner products which detail these interactions can all be expressed as a linear combination of the inner products:

$$K^{\phi}_{i+1,j,k,l,m} = \int\int\int\int K(s,t,p,q)\phi_{i+1,j}(s)\phi_{i+1,k}(t)\phi_{i+1,l}(p)\phi_{i+1,m}(q)dsdtdpdq \tag{3.50}$$

— which describe the projection $P_{i+1}P_{i+1}\mathcal{K}P_{i+1}P_{i+1}$. It is only the these inner products which ever need be directly evaluated in any top-down `ProjectKernel()` algorithm. It is for this reason that Gortler *et al*'s algorithm establishes whether the patches' *parent* level is smooth before storing the inner product (3.50) with a

[8]The *level* determines which triplet is being dealt with (figure 3.15), the *patches* determine which rows/columns within the three blocks are of interest — in *flatland* terms, at any rate.

link joining the two patches. In a quadtree algorithm, the radiosity across a patch is (similarly) found by applying the two-scale relationship to the inner products $\int\int x(s,t)\phi_{i+1,j}(s)\phi_{i+1,k}(t)$ stored with their immediate children[9]. A `Gather` consists of moving these values across the kernel links, scaling them by the $K^{\phi}_{i+1,j,k,l,m}$ as they go. This is very similar to the conventional hierarchical algorithm [50].

Visibility is accounted for exactly as it was in the conventional algorithm; by spraying a constant number of rays between patches to give $V \in [0,1]$, which is used to scale energy transfer across the appropriate link.

3.4 Closing remarks

As proved to be the case with non-hierarchical higher order radiosity methods [129], values for this visibility term V which are other than 0 or 1 can cause problems. If a discontinuity[10] exists, in the radiosity function across a surface, then the difference between smooth (averaging) functions at adjacent levels will often be significant, and this prevents negligible coefficients for the detail functions. Visibility discontinuities are thus a major source of non-negligible terms in the projected kernel matrix — an issue worsened by the use of higher order flatlets which, having such wide support, are more likely to encounter such discontinuities.

The algorithms of this chapter mark a fundamental change in the approach the computer graphics community is taking, to solving the rendering equation: the

[9]The only inner products which get stored.

[10]Due, say, to a major difference in the view nearby points on the surface have of an important emitter.

trend is very much towards setting the problem in a sound mathematical context and, by considering the problem in these terms, exploring any solution methods which apply to problems of this type. It is the author's view that it is exactly the algorithms described in this, and the next, chapter which illustrate this trend in the radiosity community. It is for this reason that care has been taken to describe the algorithms in such detail.

The first part of this chapter looked at the non-hierarchical higher order radiosity methods. Whilst no shattering conclusions were reached about the superiority of these methods over conventional approaches, some clear improvements have resulted:

Accuracy: By using piecewise polynomial, rather than piecewise constant functions, it is possible to represent the radiosity across much of the scene, to within a very high degree of accuracy. Incorporation of *interpolation* into the solution process, rather than the rendering phase, removes an anomaly that has long been present in constant radiosity algorithms. The introduction of the Galerkin method, rather than the more conventional collocation approach, results in a more accurate kernel matrix than is typically found in classical methods.

Framework: By couching the problem in terms of integral equations, and restricting the solution of these equations to manageable function spaces, the higher order radiosity methods have set the problem in a solid mathematical framework. In turn, this allows researchers to make concrete assertions about the validity of their results, as well as clearly categorising the problem at hand, so that research efforts can be usefully directed into other fields; which may already have solved some of the problems faced by the computer graphics community.

Convergence: Whilst each iteration, in the solution process, may be more costly than a compatible iteration in a constant radiosity algorithm, the higher order solution will converge in less iterations than its piecewise constant counterpart [28].

Storage costs: Higher order polynomials remove much of the need to mesh a surface: Zatz [129] reported solutions which differed little from high-quality, conventionally-produced, solutions for the same scene, but which required an order of magnitude less memory to store.

These improvements have not been without their costs:

Discontinuities: Gibbs ringing behaviour results when finite order polynomials attempt to model significant discontinuities in the radiosity function across a surface.

Expense: Energy interchange between higher order polynomials may require a very large number of samples before a solution is possible. If the two polynomials are of order N, then $(N+1)^4$ samples are required before energy exchange can be evaluated — it is easy to see how this figure could soar to unacceptable levels.

The second part of this chapter has been devoted to wavelet radiosity which, by using a hierarchy of basis functions, of arbitrary order, has inherited all of the advantages of conventional higher order methods, whilst keeping storage and speed costs down. Non-standard wavelet methods result in linear algorithms, both in storage and time, and whilst energy interchange between high order polynomials remains expensive, the method ensures that all such transfers are significant to the final result.

The common problem shared by all of the radiosity algorithms discussed thus far concerns the difficulties that arise when discontinuities exist in the radiosity function:

- In constant radiosity algorithms, shadow leaks and light leaks occur;
- In the conventional higher order algorithms, Gibbs ringing results;
- In wavelet radiosity, large number of significant kernel entries result;
- In all methods, convergence rates are adversely affected.

It is apparent that correct handling of these discontinuities could help most of the algorithms discussed thus far. A careful look at such discontinuities seems in order. The next chapter describes algorithms which locate these discontinuities *a priori*, and which utilise higher order basis functions constructed so that their support does *not* cross over any discontinuity deemed to be significant.

Chapter 4

Discontinuity meshing radiosity

Thus far, this thesis has described a number of radiosity algorithms which, whilst each having their own good and bad points, have all shared a common stumbling block: their inability to handle shadows properly — particularly sharp shadows. It is not, however, shadows *per se* which are the problem; it is the inability of a piecewise constant[1] function to accurately model a significant discontinuity lying within the support of one of its basis functions. Figure 4.1 shows an example where linear basis functions have been used to model a function twice; once with uniformly-positioned nodes, and again with carefully-positioned nodes. It is clear that simply by taking care over node positioning (ensuring that discontinuities lie *between* elements rather than *within* them) a considerable increase in accuracy can be achieved.

This chapter describes algorithms which recognise the need for careful positioning of nodes/elements across the support of the function being modelled, and which

[1]or low order polynomial

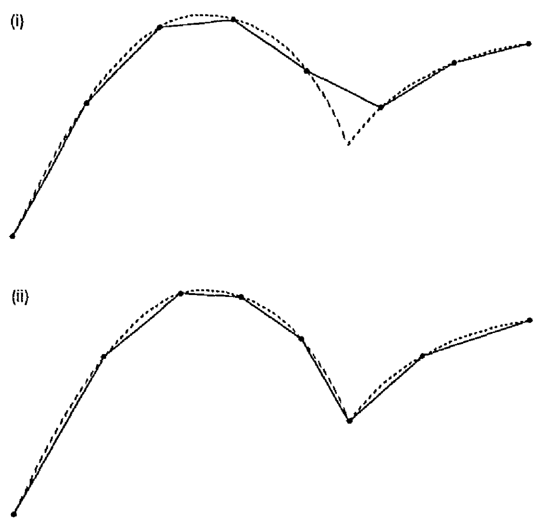


Figure 4.1: Accurate modelling of the underlying function is not so much a problem of choosing the order of the basis functions, and their number, but of carefully choosing their supports.

use geometric information to calculate these positions *before* light transfer takes place.

What constitutes a *significant* discontinuity in the radiosity function across a surface? How can discontinuities be located and/or quantified? A number of terms and definitions are now introduced which should help answer these questions.

4.1 Discontinuities and their significance

A function $x(s)$ is said to be C^0 across the domain Ω if:

$$\forall\, s \in \Omega, \quad \lim_{\delta \to 0}(x(s) - x(s \pm \delta)) = 0 \tag{4.1}$$

Functions which satisfy (4.1) are *continuous in value*. Functions whose k^{th} derivative satisfy (4.1) are known as C^k functions — continuous in the k^{th} degree. A function which is continuous in every degree is said to be C^∞.

A function $x(s)$ which fails to satisfy (4.1) exhibits a *discontinuity in value*, and is called a D^0 function. A function whose $(k-1)^{th}$ derivative satisfies (4.1), but whose k^{th} derivative does not, is known as a D^k function, and is said to exhibit a *discontinuity in the* k^{th} *degree*.

In the case of the radiosity function across a surface, where are these discontinuities, if anywhere, likely to occur? Consider the radiosity across a receiver surface `rcv` lit by an emitter `src`. The final radiosity across `rcv` is the sum of any emitted radiosity together with whatever arrives from `src` (scaled by `rcv`'s diffuse reflectance). Consequently, a discontinuity in

- the emitted radiosity, or
- the diffuse reflectance, or
- a surface normal, or
- the visibility term between `rcv` and `src`

are all likely to cause discontinuities in the final radiosity function.

In this thesis, it is the last of these possibilities that is of interest. Such discontinuities, in the visibility term between two surfaces, correspond to *visual events* [39]. Whilst the other listed reasons may well cause discontinuities in scenes we wish to render, they do not commonly cause problems with image quality, or even accuracy. This is not true of *shadow discontinuities*, caused by discontinuities in the visibility term between `rcv` and `src`, and due to the presence of one or more occluders lying between the two: all of the radiosity methods examined thus far have encountered difficulties with shadow discontinuities. The human eye is particularly attuned to *contrast*, rather than absolute intensity [115] and, as such, finds the absence of the perceptual cues usually provided by shadows (one of the chief areas of contrast in many scenes) most noticeable. Indeed, the importance of realistic shadows, in computer-generated images, has long been apparent to those creating the images: shadow algorithms are almost as old as rendering algorithms [108].

The importance of discontinuities in gradient, to a human observer's perception of an image, is well known to those familiar with Mach banding — a feature not uncommon to computer-generated images. With Mach banding, it is the *presence* of a discontinuity in gradient, where it *was not* expected, that attracts the human eye. With a D^1 shadow discontinuity, it is the *absence* of the discontinuity where it *was* expected, that the observer finds so noticeable.

Early radiosity algorithms were praised [13] for their realistic, soft, area light source, shadows. Not all shadows are soft, however, but if the radiosity method were able to model *all* significant shadows in a scene, then the realism of the resulting images would surely be impressive. It is the desire to accurately account for *all* significant shadow discontinuities within the radiosity method, that drives this study.

Heckbert [54] has shown that in an arbitrary environment, the radiosity function is capable of exhibiting discontinuities of *every* degree. It is easy to visualise a situation where a scene has a number of light sources, none of which exhibit discontinuities of any degree, which then directly illuminate all surfaces visible to them, and these surfaces then illuminate all the surfaces they can see, and so on (iteratively accounting for more and more terms in the Neumann series (3.7)).

However, shadow discontinuities *will* typically appear across the surfaces lit directly by the sources, because of discontinuities in visibility between source and receiver. These discontinuities will then propagate to other surfaces (on the next application of the kernel function) and this transfer too will encounter obstacles; causing more discontinuities in the radiosity function. As the process continues, and more and more terms in the Neumann series (3.7) are accounted for, more and more discontinuities of higher and higher order will appear in the radiosity function. A more formal proof, by counter-example, can be found in [54].

It should be clear that specifically accounting for an infinite number of discontinuities, in the function being modelled, will not be practical. Instead, only those discontinuities which are deemed *significant* are accounted for. The following sections examine some low order discontinuities in the radiosity function, with a view to establishing which, if any, of these discontinuities should be accounted for when meshing an illuminated surface into elements.

4.1.1 D^0 Discontinuities

Discontinuities in value are a common feature of images rendered using the radiosity method[2]. Most often visible in the form of *light leaks* or *shadow leaks*, they occur where different parts of the same element[3] have a radically different view of the scene (figure 4.2). For example, a floor in our scene may have been uniformly meshed into a number of elements, and then a box placed on the floor. Unless the

[2]They are even visible on the cover photograph of [24]; a highly-respected book on the radiosity method.

[3]Using finite element nomenclature, rather than classical radiosity nomenclature (patch).

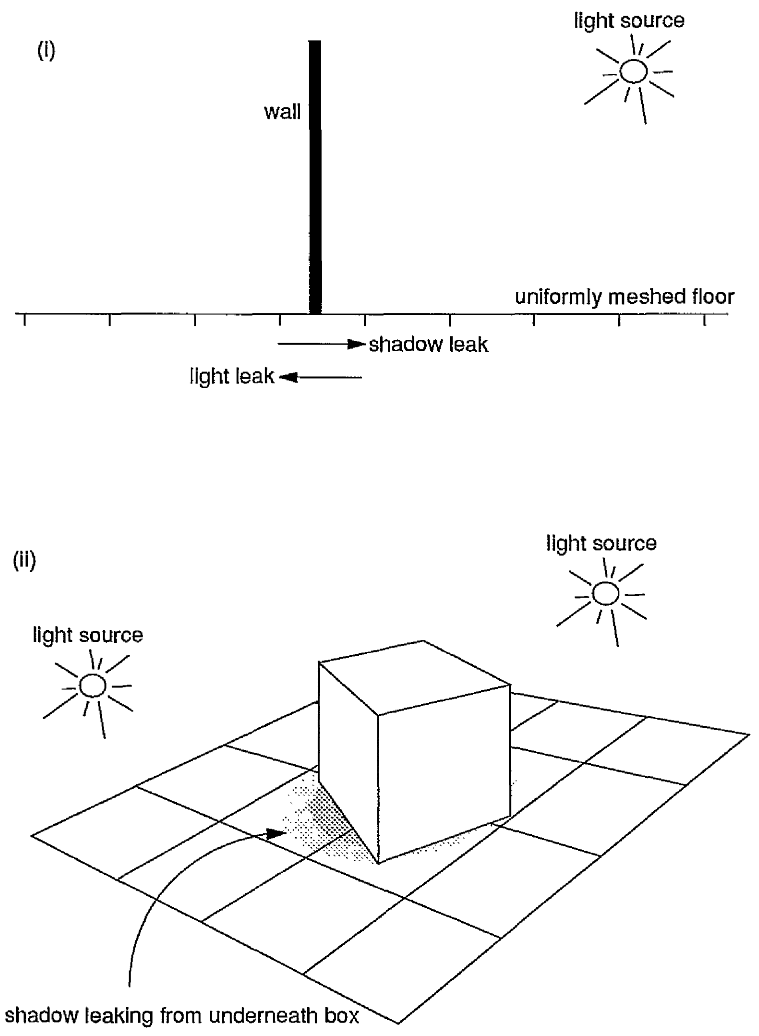


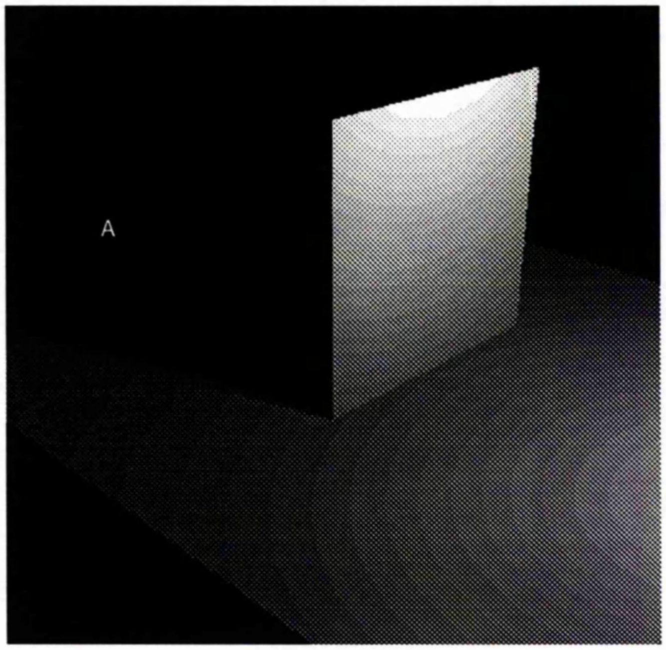
Figure 4.2: Light leaks and shadow leaks occur when different parts of the same element have radically different views of the scene.

vertical sides of the box meet the floor along element/element boundaries, then different parts of any elements lying *partially* under the box will certainly have radically different views of the scene (figure 4.2.ii). Consider one such element, with some of its nodes hidden under the box, and some of its nodes well lit. It is easy to see how, when the nodal values are interpolated across the element, light from the lit nodes will *leak* under the box, and 'shadow' (for want of a better word) from the dark nodes will leak out and appear on the floor, beside the box.

In the correct radiosity function across the floor, there is a discontinuity in value along the box/floor boundary. It is the failure of the uniform element mesh to coincide with this boundary that causes the problems. Figure 4.3 shows an example of a D^0 discontinuity in the radiosity function across a surface. Also shown is a graph depicting the radiosity along a line AB in the surface — the discontinuity is clearly visible as a 'step' in the graph. A scene containing e edges, and hence $\mathcal{O}(e)$ faces, can have $\mathcal{O}(e^2)$ such critical boundaries, since $\mathcal{O}(e)$ edges could intersect with each face in this way. In reality, the figure is unlikely to ever be this large.

The problem remained largely unaddressed, in the radiosity community, until Baum *et al* took a careful look at D^0 discontinuities in [7]. Baum *et al* pre-processed their scenes with some intelligent mesh-generation code; ensuring that implicit surface boundaries (such as the box/floor case, just discussed) *did* coincide with boundaries in the mesh. They also accounted for the D^0 discontinuities caused by situations such as a carpet lying on a floor, as well a number of other awkward cases that can crop up if databases are rendered naïvely.

Once the surfaces to be rendered have been successfully pre-processed, generating



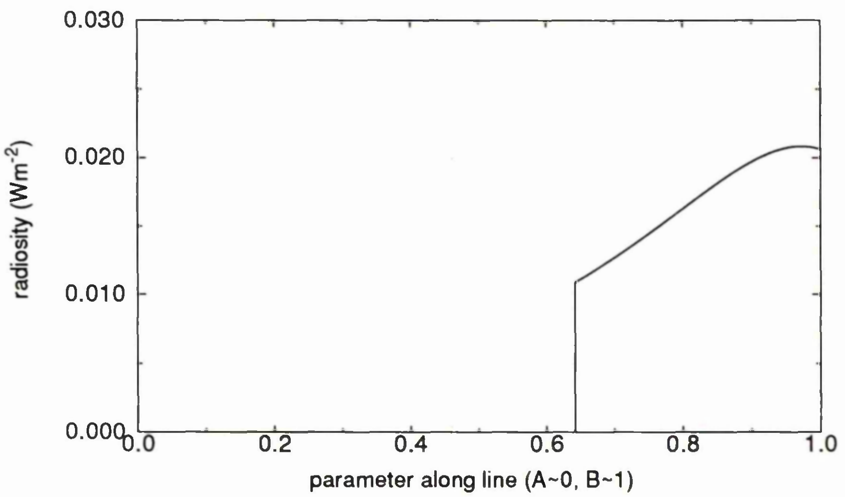


Figure 4.3: An example of a discontinuity in value. The discontinuity (corresponding to the polygon-meets-floor boundary) is clearly visible in the lower graph.

a mesh whose boundaries *do* coincide with any possible D^0 discontinuities, the application of the usual solution method should now lead to an image free from any light or shadow leaks. Care must still be taken, however, when evaluating the radiosity at those nodes positioned *on* the critical boundaries: floating-point precision can cause problems here, and the node may feel its view of important sources is being blocked by the 'box'/'carpet' causing the discontinuity. As long as nodes lying on critical boundaries are labelled as such, and care is then taken when evaluating how they are lit, such problems can be avoided [70].

4.1.2 D^1 and D^2 Discontinuities

Having established that D^0 discontinuities are of significance, and should be incorporated into the meshing process, what conclusions can be reached about higher order discontinuities? Figures 4.4 and 4.5 show, respectively, examples of first and second order discontinuities in the radiosity function. Such discontinuities occur when, as one is moving across the shadowed surface, the view of the light source changes dramatically — a visual event occurs. In order to expand on this point, let us introduce some simple terminology:

- A *lit* region is one in which every point in the region has a wholly unoccluded view of the entire light source.
- A *penumbra* region is one in which every point in the region can see some, but not all, of the light source.
- An *umbra* region is one in which every point in the region can see no part of the light source.

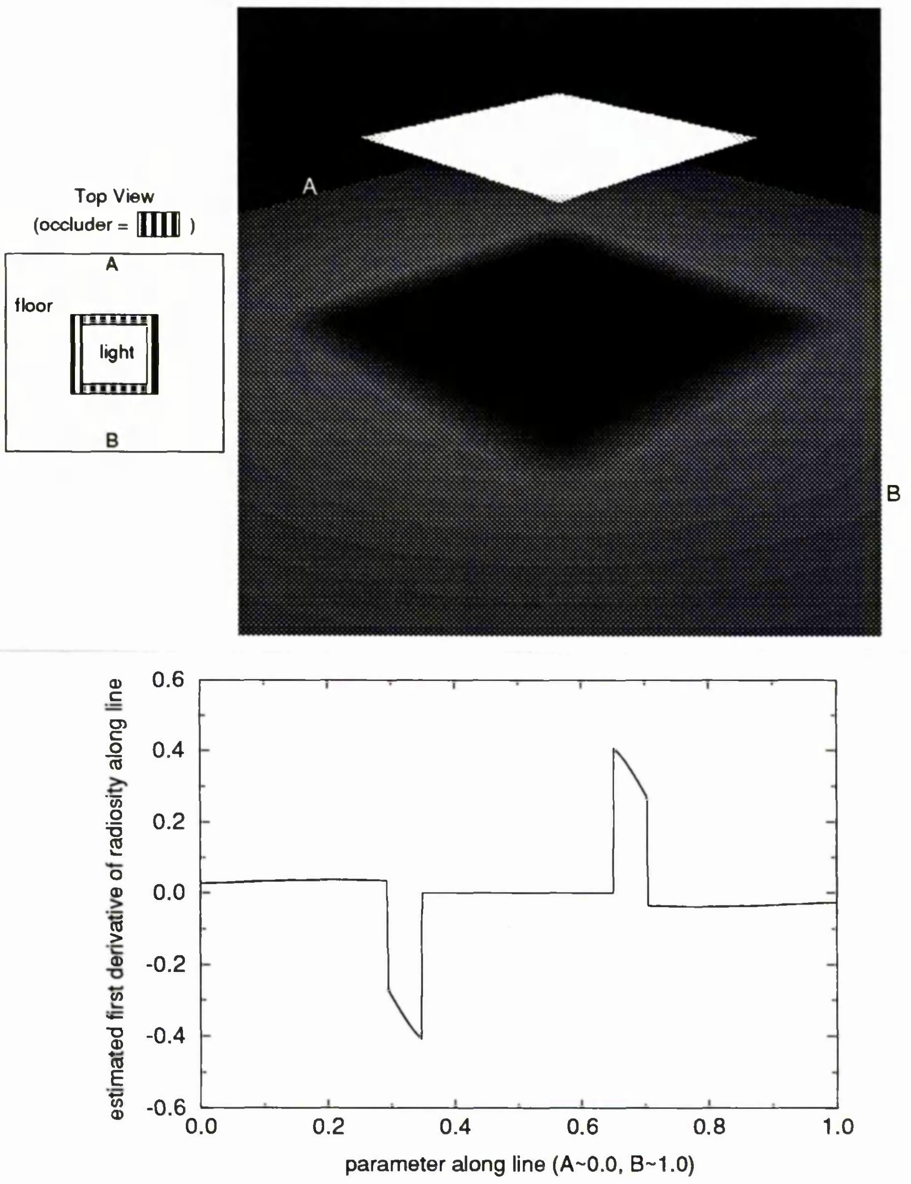


Figure 4.4: An example of a first order discontinuity in the radiosity function across a surface. The accompanying graph — which plots the derivative of radiosity along the line AB, shows the discontinuities as sudden jumps in value.

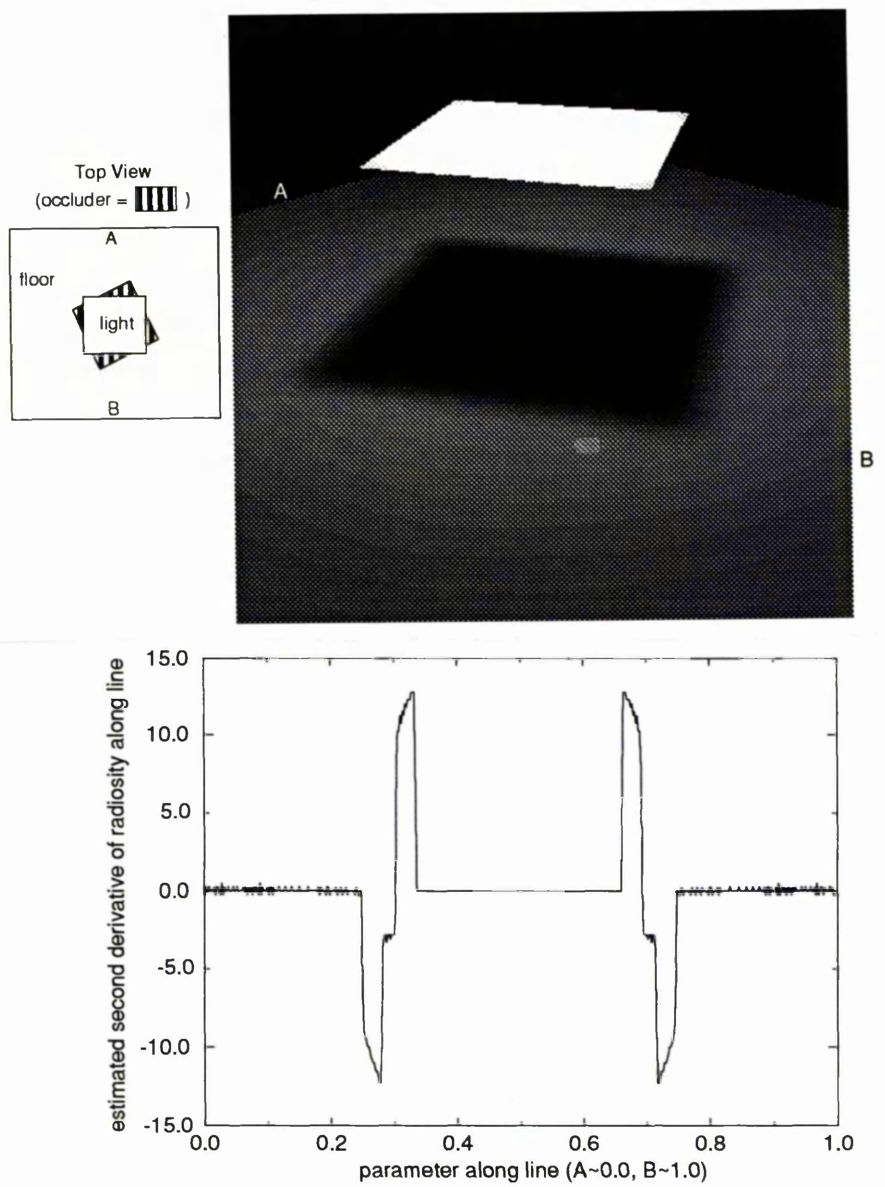


Figure 4.5: An example of a second order discontinuity in the radiosity function across a surface. The accompanying graph — which plots the second derivative of radiosity along the line AB, shows the discontinuities as sudden jumps in value.

Clearly, when moving across a lit region, no dramatic changes will occur in the view one has of the light source. Consequently, the radiosity function across a lit region is completely free of shadow discontinuities. However, when crossing from a lit region into a shadowed region, or *vice versa*, it is easy to see how whatever occluder is casting the shadow must affect one's view of the source, and thus discontinuities are to be expected here. Similarly for the boundary between umbra and penumbra regions.

A comprehensive study of visual events; covering where and why they are likely to occur, was made by Gigus and Malik [39], who constructed the *aspect graphs* for polyhedral objects, for an application in machine vision. For polyhedral scenes in three dimensions, Gigus and Malik identified two distinct types of visual event: *edge-vertex* (EV) events, and *edge-edge-edge* (EEE) events, both of which define three-dimensional *critical surfaces*. Because of the difficulties associated with locating critical surfaces for anything other polyhedral scenes, this thesis (as others have done [54, 70, 31]) limits itself to such scenes.

EV events correspond an inter-visible edge e and vertex v. The critical surface corresponding to such an event is a subset of the plane which passes through e and v (figure 4.6(i)). Specifically, given any point P on e, all points which lie on the straight line Pv, but which do not lie *between* P and v, lie in the EV critical surface. This surface is referred to as an EV *wedge*, for obvious reasons. In fact, because of occlusion, the wedge may not extend as far as figure 4.6(i) implies — as illustrated by figure 4.6(ii).

EEE events correspond to three inter-visible skew edges. The critical surface corresponding to such an event is the locus of a view-point from which all three

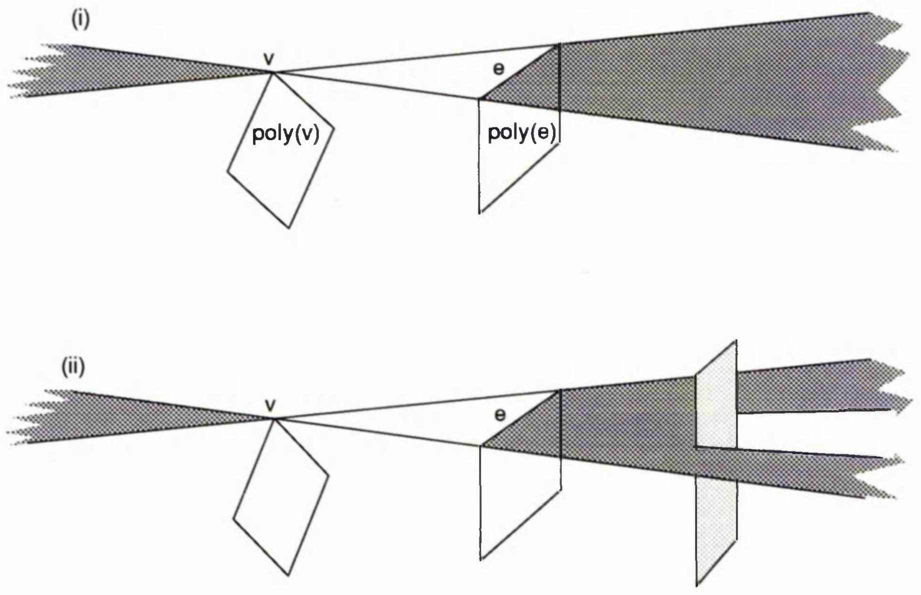


Figure 4.6: (i) An edge-vertex (EV) visual event, with its corresponding wedge-shaped critical surface. (ii) Note how the rest of the scene may clip the critical surface.

edges are seen to cross at a point (figure 4.7). This defines a quadric ruled surface [39].

Figure 4.8 illustrates three visual events involving a light source, where shadow discontinuities are expected. The figure shows the view from the shadowed surface, looking towards the light source:

D^2 **EV:** Figure 4.8(i) shows an EV event where, as one moves from an umbra region, into a penumbra region, a vertex of the source rises over the horizon (which is an edge of the occluder). The further one moves into the penumbra region, the more one can see of the source — its (triangular) visible area increasing quadratically with distance moved along the shadowed surface. This quadratic increase implies a *second* order discontinuity along this umbra/penumbra boundary. Figure 4.5 shows an example of this phenomenon; the graph showing eight sudden jumps in value where the line

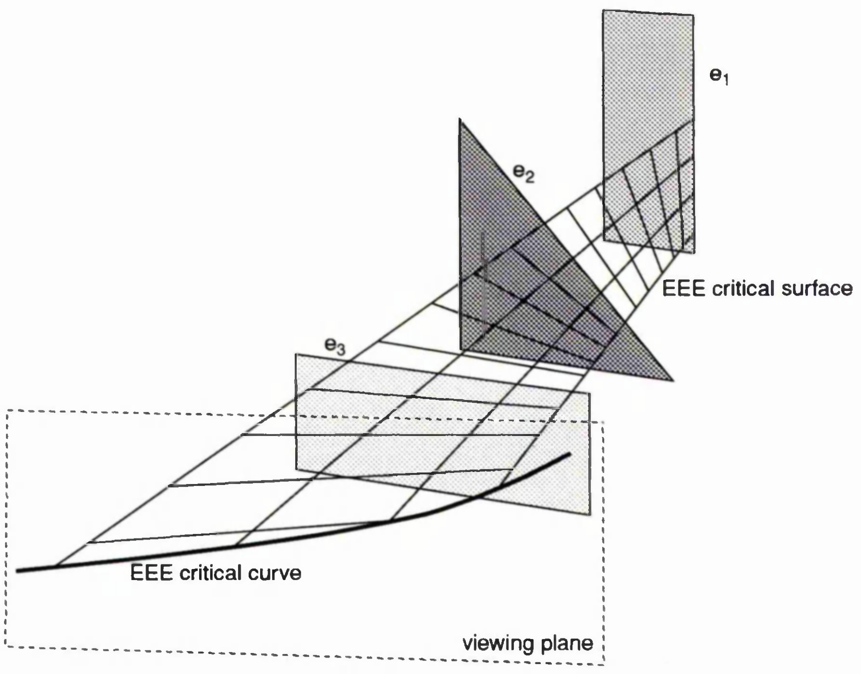


Figure 4.7: An edge-edge-edge (EEE) visual event, with its corresponding critical surface.

AB has crossed a D^2 discontinuity.

D^1 **EV:** Figure 4.8(ii) shows an EV event where, as one moves from an umbra region, into a penumbra region, an edge of the source rises over the horizon (which is an edge of the occluder). This is a special case of the EV event just described: everything is the same, except that now, one of the source edges that subtends the 'rising' vertex, is parallel to the 'horizon-edge' of the occluder. In general, in such cases, the visible portion of the source is a trapezium, and so increases linearly in size, with distance moved along the shadowed surface. This linear increase implies a *first* order discontinuity along the umbra/penumbra boundary. One can regard this (more severe) discontinuity as being due to the *superposition* of two second order discontinuities of the type just described: each due to the 'horizon-edge' of the occluder, and one end of the 'rising' source edge (figure 4.8(ii)). Figure 4.4

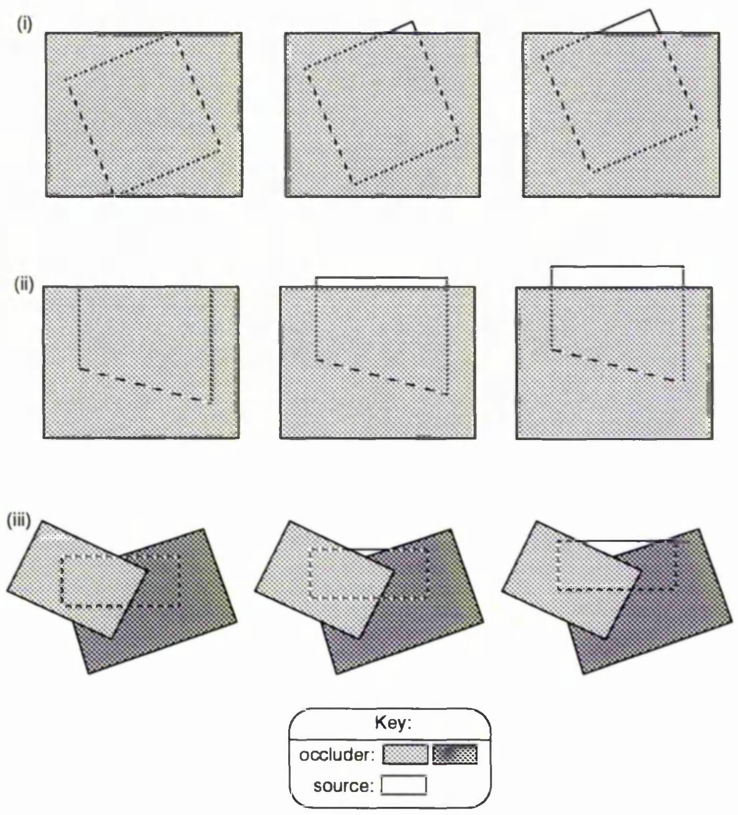


Figure 4.8: The figure shows three views moving out of an umbra region into a penumbra region, looking from the shadowed surface, towards the source. The source can be seen rising over the horizon formed by the edge(s) of the occluder(s). Each case corresponds to a different *visual event*.

shows an example of this phenomenon; the four sudden jumps in the graph marking where the line AB has crossed a D^1 discontinuity.

D^2 **EEE:** Figure 4.8(iii) shows an EEE event where, as one moves from the umbra region into the penumbra region, an edge of the source rises over the horizon (consisting of edges from two occluders). The further one moves into the penumbra region, the more one can see of the source — its (triangular) visible area increasing quadratically with distance moved along the lit surface. This quadratic increase implies a *second* order discontinuity along this umbra/penumbra boundary.

Critical surfaces, corresponding to visual events such as those just described, define the boundaries between regions across which the visible portion of the source remains *topologically* constant [39] (figure 4.9 illustrates this point, and figure 4.5 shows a graph which demonstrates such discontinuities). Across such regions, no dramatic change ever occurs in the view one has of the source. Consequently, critical surfaces define the boundaries between regions where one can expect the radiosity function to be well-behaved.

In a scene containing e edges, there can be as many as $\mathcal{O}(e^2)$ EV critical surfaces, each of which may intersect with a face, making a total of $\mathcal{O}(e^3)$ EV *critical curves*. Similarly, there may be up to $\mathcal{O}(e^3)$ EEE critical surfaces, with up to $\mathcal{O}(e^4)$ EEE critical curves. These figures seem prohibitively large. By limiting our interest to critical surfaces which involve a single light source only, there may be up to $\mathcal{O}(e^2)$ EV critical curves, and as many as $\mathcal{O}(e^3)$ EEE critical curves. In most scenes, occlusion (which invalidates *inter-visible*) and parallel edges (which invalidates *skew*) will result in a much smaller number of critical curves.

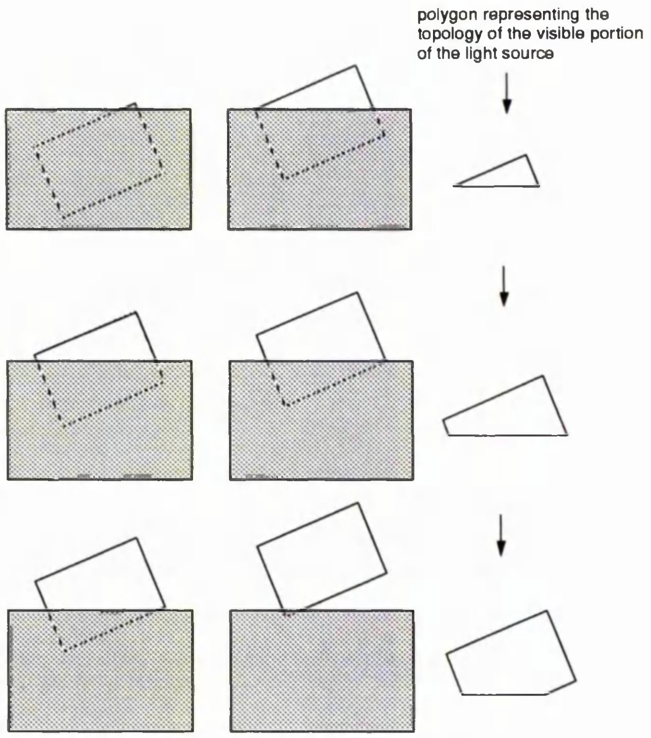

polygon representing the topology of the visible portion of the light source

Figure 4.9: Visual events correspond to a change in the *topology* of the visible portion of the light source.

4.2 Discontinuity meshing

Consider a mesh, across any shadowed surface, constructed by evaluating the intersection of the surface with all relevant critical surfaces. One can reasonably expect the radiosity function, across the elements in such a mesh, to be well-enough behaved to validate the approximation of the function, across each element, by some low order polynomial. If this transpires not to be the case, then one *can* be confident that this is not because of a shadow discontinuity lying within the support of the element. If this *discontinuity mesh* is now used as the basis for a radiosity solution of the scene, then all shadow discontinuities of first and second order, in the radiosity function, will have been captured by the mesh, and a highly accurate solution should result.

Such an approach, first proposed by Heckbert (a colleague of Gigus) in [54], and implemented for the *flatland* case only, constitutes *discontinuity meshing radiosity* (DMR).

A number of algorithms have come close to implementing DMR, without actually doing so. Nishita and Nakamae [79][4] used umbra and penumbra volumes to classify vertices in their mesh as being either in umbra or penumbra, but did not actually incorporate the shadow boundaries into their mesh. Campbell [15], following on from his earlier work with Fussell [14], used the same shadow volumes used by Nishita and Nakamae, but *did* incorporate the shadow boundaries into his meshing process. The only discontinuities *within* the penumbra, modelled by Campbell, were umbra EV events. Unfortunately, both of these approaches were unable to extract any useful object-space ordering from their data structures, and so every polygon being classified as shadowed/not-shadowed had to be tested against the shadow volumes of *all* other polygons in the scene.

This situation was improved upon somewhat by Chin and Feiner [20], who split the light source by the planes of those polygons which were visible to it and whose planes passed through it. This resulted in a number of source *fragments*, from which one could tour the scene in a unique front-to-back order, by utilising the BSP tree [35] in which the scene was stored. This avoided redundant shadow-volume/polygon comparisons, but had to be repeated for each source fragment. Also, Chin and Feiner's algorithm only dealt with *direct* illumination.

A more detailed account of BSP trees and SVBSP trees [19], is given in appendix A.

[4]following on from their previous work with point and linear sources [80].

Heckbert [53] and Lischinski *et al* [70] were the first to implement DMR for a three-dimensional scene; their work having been carried out separately, but simultaneously. Both papers describe algorithms which do *not* handle EEE events, the reasoning being that:

1. Such events are difficult to locate; much simpler algorithms can be used for EV events: an EV critical surface lies in a plane, an EEE critical surface lies in a quadric (figure 4.7).
2. Such events constitute second order discontinuities, and these are the least significant of the three orders being modelled.
3. Such events always occur within the penumbra, which is bounded by EV critical surfaces, so EEE critical lines are not so noticeable when excluded.

More recent algorithms [30, 105] have included EEE events, arguing that they are important because, when they are present, they define part of the boundary of the umbra region. Ignoring EEE events leads to small parts of the umbra region being incorrectly classified as penumbra — an error soon rectified when one attempts to find out how the source illuminates this region. Ignoring EEE events amounts to a trade-off: wasteful sampling of the source *versus* expensive location and manipulation of quadric surfaces. Heckbert [53], Lischinski *et al* [70], and the work described here, all opt for the former option; EEE events are ignored.

4.2.1 Meshing considerations

Having decided to limit the critical surfaces under consideration to those which correspond to an EV event, one is left with the problem of locating these (semi-infinite) *wedges*, and finding the line segments where the (possibly clipped) wedges meet the scene polygons.

Heckbert [53] describes an algorithm where one evaluates 'all significant' EV wedges *a priori*, and then proceeds with a hemi-cube-based, linear element, collocation radiosity solution, *as per* [73]. Heckbert gives no indication of how one determines which wedges are 'significant' and, when presenting his paper, freely admitted that the work was not complete and only considers a single light source. For this reason, it is Lischinski *et al*'s algorithm [70] which is of more interest here.

Lischinski *et al* describe an algorithm where one does not attempt to locate all, or all significant, discontinuities *a priori*. Instead, they adopt a progressive refinement [21] approach whereby:

- A discontinuity mesh is built corresponding to an important light source;
- The light then illuminates the mesh;
- A new source is chosen and a new mesh created and illuminated;
- The new mesh and the original mesh are merged, and the process repeats.

In this way, the final mesh will certainly contain all significant EV discontinuity lines, and the user will have been able to view the scene as extra meshes were added, rather than waiting for a lengthy meshing and solution process before seeing an image. Also, the progressive refinement approach has the added advantage that the user can stop the solution process as soon as the image seems 'good enough'.

By considering the visual events associated with one polygon only (some important emitter), the number of EV critical surfaces to be evaluated drops from $\mathcal{O}(e^2)$ to $\mathcal{O}(e)$, for a scene with e edges. There may still be as many as $\mathcal{O}(e^2)$ critical lines due to this single source.

Considering only one polygon at a time puts a somewhat different complexion on EV events in general: recall figure 4.6(i). The part of the EV wedge which lies to the left of v, represents a discontinuity in the radiosity which is leaving poly(e), heading towards v. The part of the EV wedge to right of e, represents a discontinuity in the radiosity which is leaving poly(v), heading towards e. Once it is known that poly(v) (say) is the light source and that poly(e) is unlit, the left hand part of the wedge no longer represents any such discontinuity, only the right hand part represents a real discontinuity. The converse is true if poly(e) is the emitter, and poly(v) is unlit. Dealing with the emission from each polygon in separate steps, splits the EV wedge of figure 4.6 into two distinct pieces.

A change of notation is called for, in order to represent this situation throughout the remainder of this thesis. A visual event which involves an edge of an emitter and a vertex of a non-emitter, is referred to as an *edge-vertex* (EV) event — with its associated critical surface, an EV wedge. A visual event which involves a vertex of an emitter and an edge of a non-emitter, is referred to as a *vertex-edge* (VE) event — with its associated critical surface, a VE wedge (figure 4.10).

When referring to visual events involving one edge and one vertex, regardless of where the emitter is, the term EVE will be used.

4.2.2 Mesh generation using BSP trees

Having decided to find all significant EVE critical lines on a per-polygon basis, there remains the problem of efficiently locating these lines, given a single polygonal source.

Heckbert's algorithm [53] compared all polygons in the scene with each critical wedge, resulting in a number of 2D *spans* lying in the plane of the wedge — each corresponding to the intersection of the wedge with a scene polygon. In order to account for parts of the wedge being clipped away (figure 4.6(ii)) a 2D sweepline algorithm was used, to determine which parts of these spans were hidden from the wedge vertex.

A more efficient algorithm was outlined by Lischinski *et al* [70], who utilise the fact that only polyhedral scenes are being considered, by storing their scene in a BSP tree [35]. This means that one can traverse the scene front-to-back, from the vertex of each VE wedge, visiting each BSP polygon[5] in order. The wedge is tested against each polygon encountered in the front-to-back traversal — if an intersection is found, a discontinuity line is added to the mesh of the polygon, the wedge is clipped, and the traversal continues. Traversal can stop as soon as the wedge is completely clipped away; saving unnecessary wedge/polygon comparisons. Notice that a wedge/polygon intersection, in such a routine, should

[5]Notation: a *BSP polygon* is a polygon stored at a node of the BSP tree; this may be an original scene (OS) polygon, or may be some fraction of an OS polygon.

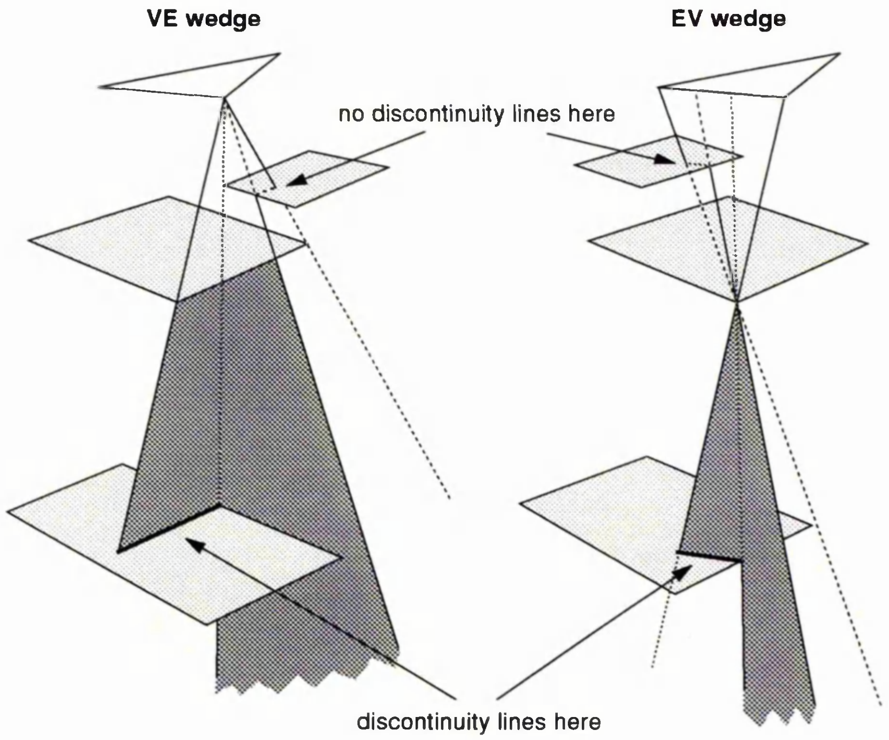


Figure 4.10: Occluders lying between edge and vertex clip the critical surface, but no discontinuity line results. Occluders which intersect the critical surface result in a discontinuity line.

only result in a discontinuity line being added to the polygon's mesh if the polygon lies *behind* the occluding polygon causing the wedge. Otherwise, the wedge should be clipped, but no discontinuity line added (figure 4.10).

Lischinski *et al* [70] are not clear about how they actually go about deciding *which* VE wedge to process. In the implementation described here, BSP trees are used to make this selection, and to decide whether or not a wedge/polygon intersection should result in a discontinuity line, or not. By giving the VE wedge front-to-back routine a list of *current wedges* to carry (initially NULL), the author's implementation (figure 4.11) makes a front-to-back traversal of the scene, from each source vertex, which at each BSP node:

1. Decides which polygons stored at the node are facing the source vertex,

2. Compares each polygon with each wedge on the current wedge list, adding a discontinuity line and clipping a wedge, for each intersection. Clipping may remove wedges from the current wedge list.

3. Generates a list of new wedges, corresponding to the edges of the polygons currently being considered[6] — each wedge is tagged with a pointer to the original scene polygon that is causing it.

4. Clips each of the new wedges by the polygons responsible for the wedges on the current wedge list. Clipping may remove wedges from the new wedge list.

5. Adds the new wedges to the current wedge list, before continuing with recursion.

Lischinski *et al* are also unclear about how they process EV wedges, saying this 'requires only minor changes to the algorithm used to process VE events' [70]. This is somewhat misleading, since it implies that EV wedges are different to, but no more of a problem than, VE wedges. This is not the case. Problems with EV wedges include:

- Finding which occluders might provide the vertex for such a wedge, and
- Finding the polygons which lie between the source edge and the occluder vertex: these polygons must clip the wedge before it is used to generate any discontinuity lines.

[6]Some of these edges are not original scene edges, but have resulted from the construction of the BSP tree: these edges are ignored.

```
void VEfront-to-back ( VElist, pt, tree )
{
   if ( tree == NULL )
      return;

   if ( in-front( pt, tree.plane ) {
      VEfront-to-back ( VElist, pt, tree.pos );
      disc-edges ( VElist, tree.posPolys );
      new-wedges ( VElist, tree.posPolys );
      VEfront-to-back ( VElist, pt, tree.neg );
   } else if ( behind ( pt, tree.plane ) ) {
      VEfront-to-back ( VElist, pt, tree.neg );
      disc-edges ( VElist, tree.negPolys );
      new-wedges ( VElist, tree.negPolys );
      VEfront-to-back ( VElist, pt, tree.pos );
   } else {
      VEfront-to-back ( VElist, pt, tree.neg );
      VEfront-to-back ( VElist, pt, tree.pos );
   }
}
```

Figure 4.11: Pseudocode for the author's method of handling all VE wedges due to a single source vertex. The routine `disc-edges` compares the current wedges with the pertinent polygons; the routine `new-wedges` generates new wedges, clips them, and updates the current wedge list.

With VE wedges, both of these problems were neatly handled by the object-space ordering inherent in the BSP front-to-back routine. Whilst the first of these problems can be handled fairly cheaply (clip the edge with the occluder's plane), this is not true of the second problem.

Before an EV wedge can be taken through the scene, front-to-back with respect the wedge (occluder) vertex, it must be clipped by all those polygons lying *between* the source edge and the occluder vertex (figure 4.10). Regardless of how this is done, it is a considerable computational task, and makes EV wedges markedly more expensive than VE wedges to process. One approach which suggests itself, is to traverse the scene front-to-back (from the vertex, as far as the edge) to *clip* the wedge, and then pass those wedges which survive this clipping routine to the VE front-to-back routine, to go through the scene creating EV discontinuity lines. Another approach is suggested in section 5.1.1. Suffice to say here that EV wedges are not as cheap to process as VE wedges.

4.2.3 Meshing an original scene polygon

Now that routines which can generate EVE critical lines are available, the problem of incorporating these lines into a useful mesh, across each polygon in the scene, remains. Following Lischinski *et al*'s [70] example, the author's implementation utilises a *discontinuity meshing tree* (DM-tree) for this purpose.

A DM-tree consists of the union of a 2D BSP tree, with a winged-edge data structure [9, 41] (WEDS). Each internal node of the 2D BSP tree (hereinafter *uvBSP tree*) contains the 2D line equation of an edge in the mesh, and pointers to the regions which lie on either side of this line. Each leaf node of the uvBSP tree points to a face of the WEDS (figure 4.12). A DM-tree is stored with each original scene (OS) polygon.

During the construction of the scene BSP tree, the face representing the OS polygon may be split, in which case the single leaf node in the tree is replaced by an internal node storing the 2D line equation of this split, and the tree now has two leaf nodes, with pointers to the faces which resulted from the split.

When adding a discontinuity line to the tree, the line segment representing the discontinuity is filtered down the tree, possibly being split into a number of smaller segments in the process, until finally every such segment arrives at a leaf node of the tree. The WEDS face representing such a leaf will now usually[7] be split in two, and an edge representing the discontinuity inserted into the WEDS. If the edge does not completely span the face being split, then extra *construction* edges are inserted into the WEDS to keep it consistent (this is shown in figure 4.12). Each discontinuity edge is labelled with the order of the discontinuity it represents.

Having processed all EVE wedges for the current light source, each scene polygon has a mesh consisting of a number of construction edges, and probably a number of discontinuity edges. It now remains only to fit appropriate low-order elements across each WEDS face in the mesh, and evaluate the contribution of the light source at each node.

[7] A new edge may lie along an existing edge.

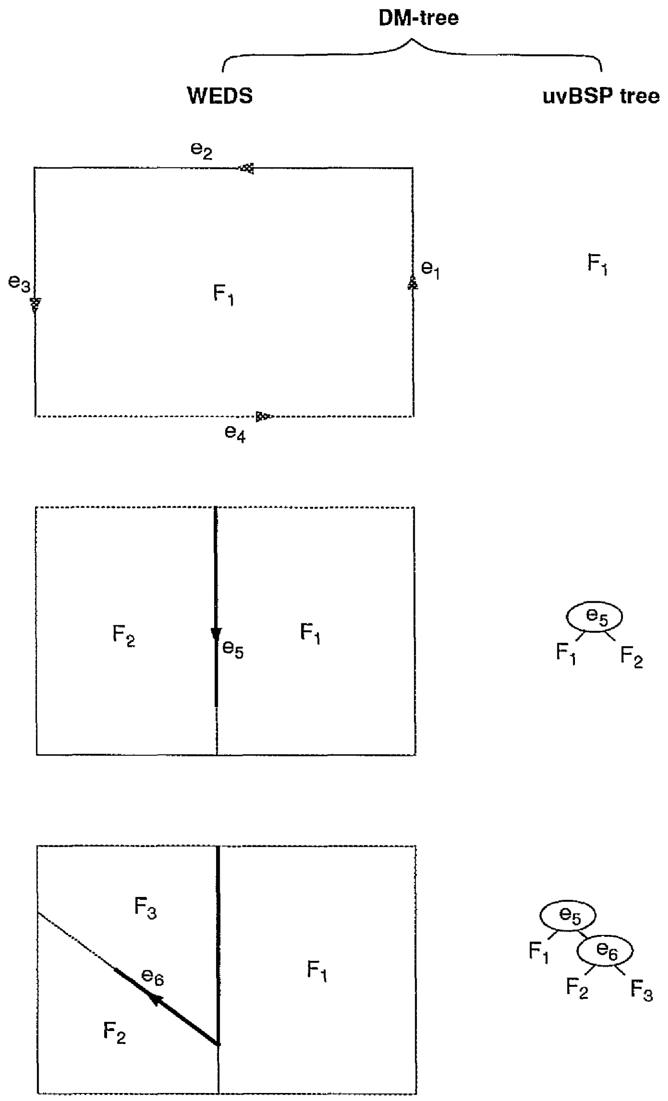


Figure 4.12: The figure shows a DM-tree for an OS polygon as two discontinuity edges are added to the tree. Bold lines represent discontinuity edges; thinner lines are construction edges.

Mesh illumination

Because of the way the mesh across a receiver polygon was constructed, one can be certain that the resulting mesh faces, whilst all being convex, will vary massively in size, shape and number of edges. For these reasons, the faces are triangulated, to make element-fitting practicable. Heckbert [53], Lischinski *et al* [70], and the work described here, have all opted for triangulation of the mesh.

In [70], mesh faces are triangulated by finding the mid-point of the longest diagonal, and joining this new vertex to the remaining face vertices. This simple approach certainly yields a valid triangulation of the mesh, but is far from optimal. A number of improvements are suggested in the next chapter.

Having triangulated the scene, elements of any order can now be chosen to fit across each triangular face. Because of the inability of constant and linear elements to resolve second order discontinuities, which we have gone to great lengths to accurately model, quadratic elements are used by the author and by [70]. Quadratic triangular elements require 6 radiosity values to be stored for each wavelength of interest — one for each vertex, and one corresponding to the mid-point of each edge [131]. The WEDS provides an ideal storage vessel for such values, since radiosity values can be stored with a vertex (edge) in the WEDS, and these can then be easily accessed by faces which share the same vertex (edge).

For a constant emission polygonal light source, Lischinski *et al* evaluate the contribution of the source, to a single point on the receiver, using the analytic form factor formula introduced to the computer graphics community by [8], originally from [58]:

$$F_{dA_r \rightarrow A_s} = \frac{1}{2\pi} \sum_{k=0}^{n-1} \frac{\cos^{-1}(\mathbf{R}_{i\oplus 1} \cdot \mathbf{R}_i)}{\|\mathbf{R}_{i\oplus 1} \times \mathbf{R}_i\|} (\mathbf{R}_{i\oplus 1} \times \mathbf{R}_i) \cdot \mathbf{N}_r \qquad (4.2)$$

This gives the fraction of energy leaving the n-sided source A_s which arrives at the elemental region dA_r on the receiver. The vector $\mathbf{R}_i$ joins dA_r to the source's i^{th} vertex. The symbol '$\oplus$' denotes addition modulo n. The interested reader is referred to [36] for a derivation of this formula from first principles.

If the view of the source, from the point being lit, is unoccluded, then (4.2) can be applied directly. If the view is partially occluded, then the occluders are projected onto the light source plane, where they are used to clip away the hidden parts of the source — (4.2) can then be applied to the source fragments that remain. Using an analytic formula, in this way, guarantees the accuracy of the radiosity values stored with the mesh nodes.

Lischinski *et al* [70] are unclear about how they go about establishing whether the node being illuminated can actually see the source, or not. It seems that they compare the entire scene with the frustum defined by the source and the node — presumably after having clipped away all polygons lying behind the source and behind the receiver, and possibly after culling those polygons which lie outside of a *shaft* [15, 47] joining the two. This is still an $\mathcal{O}(e)$ problem, for a scene with $\mathcal{O}(e)$ polygons, and seems an expensive solution, particularly when one considers how many times the operation has to be carried out. An alternative approach, implemented by the author, is described in the next chapter.

In later iterations of the progressive refinement routine, after light has been shot from all the sources, the shooting polygon's radiosity will be represented by a

DM-tree. As such, the radiosity across its surface is likely to be far from constant, thereby invalidating any simple application of (4.2). Lischinski *et al* [70] are sketchy about how they account for this case, saying they use an algorithm 'similar to' Tampieri and Lischinski's algorithm [110] which adaptively splits the source until the radiosity across each fragment is roughly constant, then sums the contribution from each fragment. Presumably, 'similar to' means they utilise the uvBSP structure of the source's DM-tree to fragment the source, rather than a quadtree (which was used in [110]), but one cannot be sure.

Adaptive subdivision

Given an initial triangulation of the discontinuity mesh across a receiver surface, one cannot guarantee the user's desired level of accuracy, simply by fitting a quadratic element across each triangle in the mesh. Large triangles, poorly-shaped triangles, and triangles across which the radiosity is varying rapidly, can all lead to situations where the true radiosity function is ill-represented by the quadratic element.

In order to account for such situations, a routine is needed whereby one establishes which triangles are causing problems, and either:

- replaces the quadratic element with an element better-able to represent the true radiosity function across the triangle, or
- subdivides the triangle into a number of smaller triangles, so that the single quadratic element is replaced by a number of smaller quadratic elements.

Lischinski *et al* [70] opted for the latter option. Salesin *et al* [90], when faced with an identical problem, opted for the former option; replacing the quadratic element with a *Clough-Tocher* element, consisting of three triangular cubic elements manipulated so as to ensure C^1 continuity across those element/element boundaries where it is expected. The implementation described here currently employs the *subdivide* option.

Having decided to subdivide problem triangles, there remains the problem of efficiently locating these triangles. Lischinski *et al* [70] approximate the infinity-norm error metric, which gives the maximum absolute difference between the true radiosity and the interpolated radiosity, across the element. This approximation is achieved by evaluating the true radiosity at the centroid of the triangle, and comparing this value with the interpolated value there (subdivide if this is larger than some user-defined threshold). Clearly, this is an expensive option, requiring the location of any occluders which may lie between the centroid and the source, before any contribution can be evaluated. Whilst this method has been successfully implemented by the author, an alternative (cheaper) strategy is suggested in the next chapter.

Whatever method is used to locate problem triangles, a suitable algorithm for actually subdividing the triangle is still needed. As per [70], the author has implemented Rivara's [88] *2-triangle* subdivision method. Rivara's algorithm was designed for multi-gridding in finite element analysis (described in section 2.2.4), whereby an initial mesh is selectively refined until the desired level of accuracy is reached — making it ideally suited to the problem at hand. Rivara's algorithm is particularly appropriate since it leads to well-shaped triangles, and guarantees

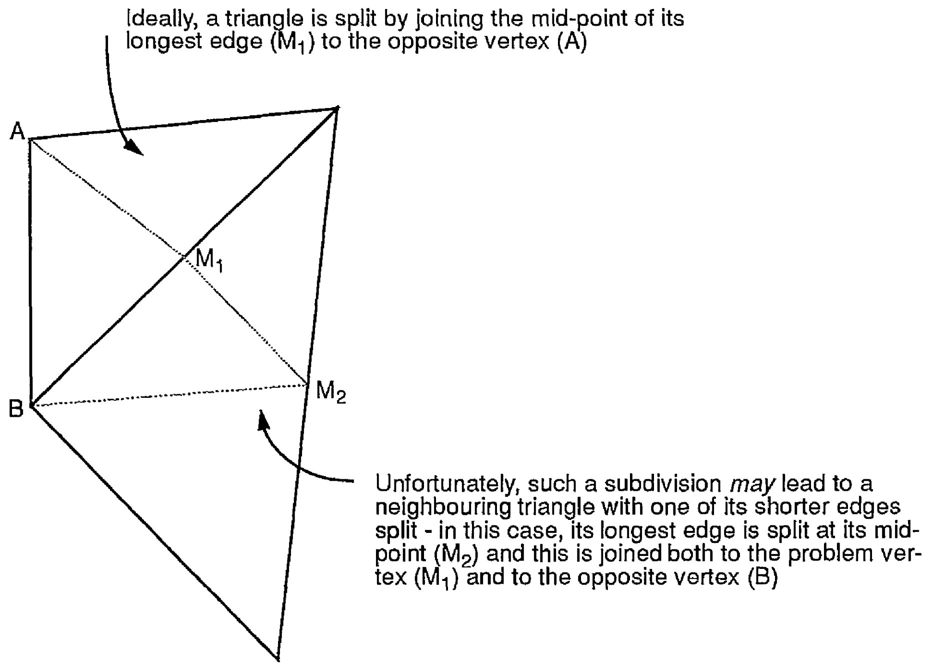

Ideally, a triangle is split by joining the mid-point of its longest edge (M_1) to the opposite vertex (A)
A
M_1
M_2
B
Unfortunately, such a subdivision *may* lead to a neighbouring triangle with one of its shorter edges split - in this case, its longest edge is split at its mid-point (M_2) and this is joined both to the problem vertex (M_1) and to the opposite vertex (B)

Figure 4.13: Rivara's 2-triangle subdivision method.

a smooth transition from large triangles to small triangles — both desirable properties for our mesh [8]. Rivara's algorithm is outlined in figure 4.13.

Once a mesh has been refined to the satisfaction of the user, an image can be generated by ray tracing (eye rays only) the BSP tree containing the scene. Because of the object-space ordering one is able to extract from a BSP tree, ray tracing BSP trees is a particularly fast form of ray tracing [112]. Whilst Lischinski *et al* [70] generated their images using ray tracing, an even faster image could probably be achieved by implementing the front-to-back BSP display method described in [44].

Mesh merging

As has already briefly been mentioned, Lischinski *et al* [70] implemented a system whereby at each iteration:

- the contribution from the shooting polygon, on a receiver, is stored in a DM-tree;
- this is then merged with the mesh (DM-tree) corresponding to the contributions of previous iterations.

In order to achieve this *mesh merging*, Lischinski *et al* [70] take the two discontinuity meshes, extract the discontinuity edges from each, and build a new mesh using these edges as its starting point. This new mesh is then triangulated, and quadratic elements are fitted across each triangle, as before. In order to calculate the radiosity value stored at each node in the new mesh, no source sampling need be carried out. Instead, *interpolated* values, corresponding to the node's location, are extracted from each of the first two meshes, and then simply added together. The first two meshes can now be deleted, an image (if called for) can be generated, and the code can move on to the next iteration.

Whilst such a mesh-merging approach seems fine in theory, in practise this transpired not to be the case. Lischinski *et al* [70] report being able to achieve converged solutions for simple scenes, but for larger scenes their code was unable to handle the excessive memory requirements. They report results for a complicated[8] scene after only two iterations — one of which produced a particularly simple mesh, being due to a small, distant, polygon representing the sun. In order to merge meshes, it is necessary, at the end of each iteration, to simultaneously store *three* DM-trees on each scene polygon: the old mesh, the latest iteration mesh, and the merged mesh. Once merging has been completed, two of these meshes can be deleted, but it is easy to see how three winged-edge

[8]1382 polygons

data structures (even without their accompanying uvBSP trees) could take up an *awful lot* of space (see appendix B). An alternative approach is suggested by the author, in the next chapter.

A closing remark

In chapter 3, the Galerkin method was shown to be an improvement over the collocation method, because the Galerkin method makes a more stringent evaluation of the kernel function. Whilst the method described here amounts to a return to the collocation method (in that radiosity is evaluated on a vertex-by-vertex basis) it is important to note that great care has accompanied the *positioning* of these vertices and, because of this, the earlier statements about the relative accuracy of the collocation method are not pertinent here.

4.3 Combining discontinuity meshing and hierarchical radiosity

Recognising some of the deficiencies in their own method, and the advantages of hierarchical radiosity [50], Lischinski *et al* produced a second paper [71]; describing a combination of the two methods. The illumination part of their new algorithm is split into two distinct parts: a *global* pass and a *local* pass.

Their global pass proceeds exactly *as per* Hanrahan's hierarchical algorithm [50] (described in section 2.2.4 of this thesis) with the exception that rather than using a quadtree subdivision method, Lischinski *et al* [71] use a BSP-based subdivision method which favours division along known lines of discontinuity in the radiosity function. This is achieved by storing, with each node in the hierarchy, a list of *primary discontinuities* which lie in the region represented by the node. Primary discontinuities are D^0 discontinuities and D^1, D^2 shadow discontinuities due to *primary* light sources. Discontinuities due to surfaces which are *not* primary light sources are not modelled by their algorithm. In this way, when a node needs subdividing, one can:

- examine the list of discontinuities stored with the node, and choose one which is of low order[9] and which most-nearly bisects the region represented by the node;
- create two new lists of discontinuities by splitting those on the current list with the line just chosen;
- subdivide the node along the chosen line, giving each child one of the new lists.

Subdividing a node which has no discontinuities is even simpler — one need only consider the geometry of the problem. Having refined the hierarchy in this way, a data structure results which is identical to Hanrahan's [50], except for its binary nature and its lists, and so the system can now be solved by passing radiosity values down the data structure's links. This results in a piecewise constant approximation to the global illumination in the scene. This approximation is markedly more accurate [71] than a comparable conventional hierarchical solution, because the new algorithm is so quick to separate regions which have an

[9]i.e., the order of preference being D^0, D^1, D^2.

occluded view of the primary sources from those which do not — a major source of error in most radiosity systems.

The local pass, which follows the global, is designed to accurately reconstruct the radiosity function across each polygon in the scene, using the global solution as its starting point. The local pass begins by triangulating each polygon. This is achieved using a constrained Delauney triangulation (CDT) [18] routine, which takes a set of points, and a set of edges, and makes an optimal triangulation of the points which includes all of the given edges. Given a tree representing the radiosity across a polygon, the CDT routine is passed the vertices of the leaf-node regions in the tree, plus any discontinuity edges stored in the tree. Once this triangulation has been carried out, one can fit quadratic elements across each resulting triangle, and it only remains to evaluate accurate radiosity values for the *e-nodes*[10] of every such element; using the information stored during the global pass.

Lischinski *et al* [71] describe a number of methods for implementing this last step; the different methods varying considerably in terms of both accuracy and cost. They begin with an analysis of the *causes* of error in such a situation, making the observation that: given a constant representation for the radiosity across a receiver, due to some source, the amount by which the *true* value, at any point on the receiver, varies from the constant value, depends on:

- how much the radiosity varies across the source,
- how much the (unoccluded) form factor between receiver and source varies

[10]The term *e-node* is introduced so that one can distinguish between nodes of a quadratic element and nodes in the hierarchical representation of the radiosity across a polygon.

over the receiver, and

- how the visible fraction of the source varies over the receiver.

Lischinski *et al* [71] point out that, when developing an algorithm to accurately evaluate the radiosity at some point on a receiver, improving the way the algorithm evaluates *any* of these three variables, will improve the overall accuracy of the method.

Based on this analysis, four different methods for evaluating the contribution of the global pass solution to the e-nodes created in the local pass, are described. These are:

method A: If an e-node lies entirely within the region corresponding to a leaf node, take the node's radiosity value. If an e-node lies on the boundary of two or more leaf node regions, average the nodes' radiosity values.

method B: Unlike conventional hierarchical radiosity, links do not store a single form factor value, but an *unoccluded* form factor (from the centre of one node's region to the other node's region) and a corresponding visibility term $V \in [0, 1]$. In Lischinski *et al*'s second local pass method, they recalculate the unoccluded form factors on all links of nodes containing the e-node (from the e-node itself), but still use the visibility term stored with the links.

method C: Taking method B one step further, the third method not only re-calculates unoccluded form factors, ignoring the values stored with the links, but also re-calculates the visibility terms from the point of view of the e-node, not the centre of the some node in the hierarchy.

method D: The last method describes tries to reduce costs by using method C for links to primary light sources, whilst using method B for all other links.

To test their different local pass strategies — designed to give the user an opportunity to trade accuracy for speed — Lischinski *et al* took three different hierarchical (global pass) solutions of a test scene (low, medium and high accuracy) and applied the different local pass methods to each one.

Their results indicate that, even given a coarse hierarchical solution, by including all discontinuity edges in the local pass, and using method C, a surprisingly accurate solution results. Method B and method A are quicker, but less accurate, and method D (which is all but indistinguishable from method C) is hardly more costly than method B, but noticeably more accurate. Their results also confirm that the local pass is well worth the time spent carrying it out: a medium accuracy global pass solution, when passed to the most expensive local pass routine, still produced an image far faster than and more accurate than, a high accuracy global pass solution followed by the simplest of the local pass routines.

Chapter 5

Optimising discontinuity meshing radiosity

The previous chapter gave a comprehensive account of discontinuity meshing radiosity, as it has been implemented by others [53, 70, 71]. In this chapter, the author's work is described in greater detail; with the emphasis on those areas where the algorithm differs from previous approaches.

Essentially, the algorithm described here is similar to that described in [70], with a number of notable differences. How the new algorithm differs, and how the overheads of discontinuity meshing radiosity have been consequently reduced, is described in detail in this chapter. Briefly, the enhancements include:

Shadow classification: Unlike many previous DMR algorithms, each element in every DM-tree is classified as being either *lit*, *in penumbra* or *in umbra*, before any effort is made to illuminate the mesh. Those elements which lie in penumbra, store a list of the polygons in whose shadow they lie.

Minimum candidate lists: A novel ordering is extracted from the BSP tree in which the scene is stored. It has long been possible to make an in-order traversal of the polygons stored in a BSP tree, with the respect to a given *point* [35]. The same has not been true of a traversal with respect to a given *polygon* [20]. The new algorithm goes some way towards addressing this problem.

Mesh triangulation: Two new triangulation schemes are introduced — both of which seem to exhibit a number of advantages over the method implemented by Lischinski *et al* in [70].

Mesh merging: In the last chapter, the memory requirements associated with mesh merging were shown to be considerable. Because of the difficulties of merging DM-trees, the implementation described here stores DM-trees in *layers*. The storage savings of such an approach are significant.

5.1 Optimising mesh illumination

There is a surprising feature of discontinuity meshing radiosity (DMR) algorithms [53, 70, 71], which separate them from their predecessors [79, 15]. Whilst DMR algorithms expend great effort in accounting for lines of probable discontinuity in shadow regions, only in very recent algorithms [37, 30, 105] are any actual shadow regions located.

The advantages of knowing exactly which regions are lit, which are shadowed, and what is casting the shadow, become apparent as soon as one tries to illuminate a DM-tree. For every e-node of a triangular mesh element, it is necessary to establish which polygons, if any, lie between the e-node and the source. Without this information, it is impossible to tell:

- whether one should be applying (4.2, pg 159) directly, or
- whether regions of the source are hidden from the e-node, and so one should only apply (4.2) to the source's visible parts, or
- whether the source is completely hidden from the e-node, and *any* further computation will be wasteful.

In order to find any occluding polygons, and whether or not they completely, or only partially, hide the e-node's view of the source, one must build, for each (source, receiver) pair, a *candidate list* of polygons which lie between the two. Then, for every e-node in the receiver's DM-tree, one must build the frustum[1] defined by the source edges and the e-node itself. This frustum can now be compared with every polygon on the candidate list to establish whether the e-node is lit, in penumbra, or in umbra (figure 5.1).

Given the regularity with which the code will need to illuminate an e-node by a given source, it is clear that short, cheaply-obtained, candidate lists would be desirable. Traditionally [15, 47], candidate lists are obtained by comparing every polygon in the scene with the plane containing the source and the plane containing the receiver — any polygon lying totally behind either of these planes is excluded from further consideration. An optional further step is to *shaft cull* [47] those polygons which remain.

[1]possibly clipped by the plane containing the receiver

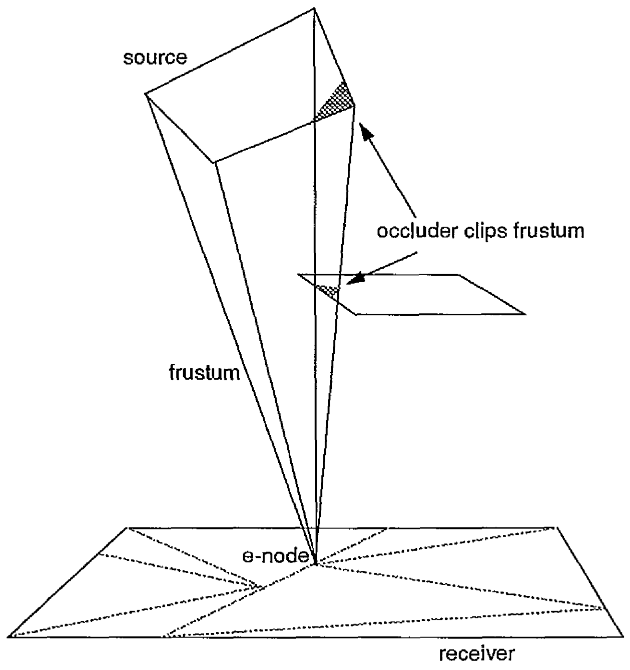


Figure 5.1: The frustum corresponding to each e-node on a receiver must be compared with all polygons which may block the node's view of the source.

5.1.1 Minimum candidate lists

In the algorithm implemented by the author, a similar[2] *clip/clip/cull* procedure is carried out. However, rather than beginning the process by considering every polygon in the scene, an ordering is extracted from the scene BSP tree which allows the algorithm to ignore many scene polygons which would otherwise have to regarded as potential occluders. These *minimum candidate lists* can then be compared with source and receiver planes, and shaft-culled, as before. The following paragraphs outline the process for obtaining these lists.

When generating VE wedges, for a particular polygonal light source, the scene BSP tree is traversed once for each source vertex. For each such traversal a list (a *vertex ordering*), storing the order in which the routine visited the BSP tree's nodes, is generated. A bit-mask is stored with each BSP tree node, and this is

[2]to shaft-culling [47]

```
void VEfront-to-back ( VElist, pt, tree, ordering )
{
   if ( tree == NULL )
      return;

   if ( in-front( pt, tree.plane ) ) {
      VEfront-to-back ( VElist, pt, tree.pos, ordering );
      ordering = ordering + tree;
      update ( tree.bits, POS );
      disc-edges ( VElis, tree.posPolys );
      new-wedges ( VElist, tree.posPolys );
      VEfront-to-back ( VElist, pt, tree.neg, ordering );
   else if ( behind ( pt, tree.plane ) ) {
      VEfront-to-back ( VElist, pt, tree.neg, ordering );
      ordering = ordering + tree;
      update ( tree.bits, NEG );
      disc-edges ( VElis, tree.negPolys );
      new-wedges ( VElist, tree.negPolys );
      VEfront-to-back ( VElist, pt, tree.pos, ordering );
   else {
      VEfront-to-back ( VElist, pt, tree.neg, ordering );
      VEfront-to-back ( VElist, pt, tree.pos, ordering );
   }
}
```

Figure 5.2: The VE wedge front-to-back routine is altered to return a vertex ordering, and to record at each BSP node how the node was seen by each source vertex.

updated during each traversal so that one can tell:

- which orderings the node appears on, and
- whether the various source vertices lie in front of, or behind[2], the plane stored with the node.

Notice that both these pieces of information come for free, if one is making a front-to-back traversal of a BSP tree (figure 5.2). In the current implementation,

[2]If a source vertex lies *in* the plane of a BSP tree node the node will not appear on the list corresponding to the vertex.

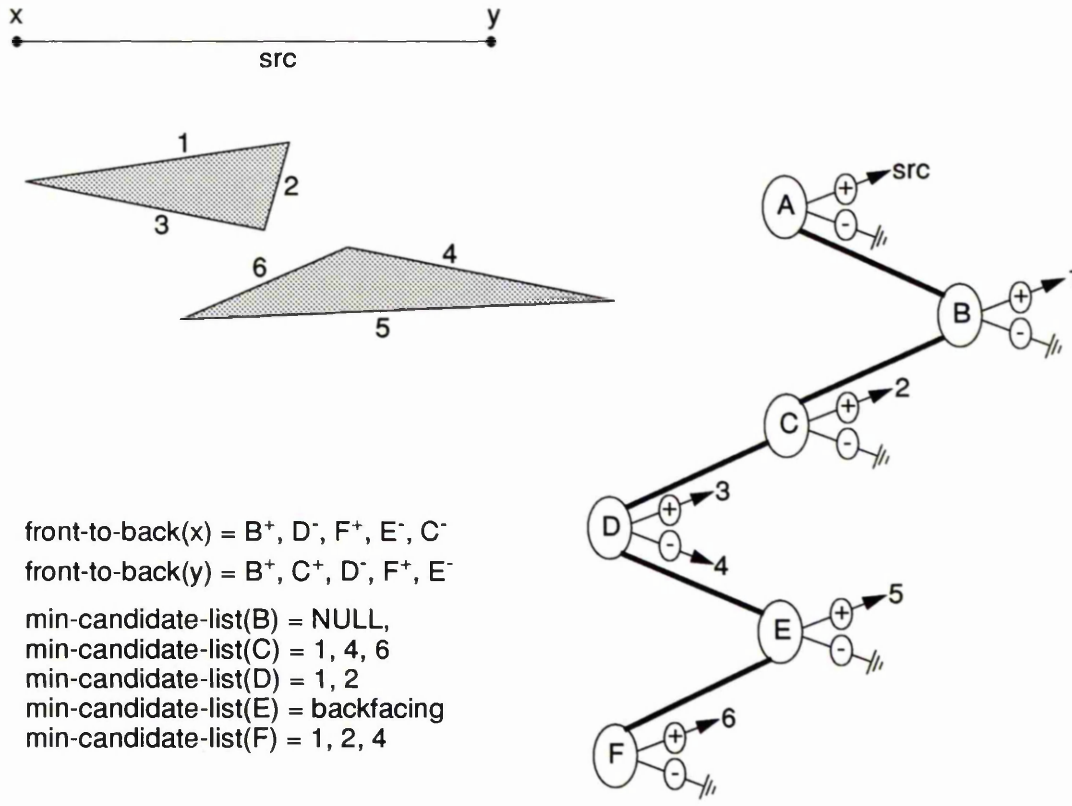

x
y
src
1
2
3
4
5
6
A
B
C
D
E
F
src
front-to-back(x) = B⁺, D⁻, F⁺, E⁻, C⁻
front-to-back(y) = B⁺, C⁺, D⁻, F⁺, E⁻
min-candidate-list(B) = NULL,
min-candidate-list(C) = 1, 4, 6
min-candidate-list(D) = 1, 2
min-candidate-list(E) = backfacing
min-candidate-list(F) = 1, 2, 4

Figure 5.3: The minimum candidate list algorithm applied to a simple scene. Nodes which face 0 source vertices and which have no back-facing polygons, can be ignored (node E).

sources with up to four vertices are considered, so each node has four bits indicating which orderings the node appears on, and four bits indicating whether the vertex lay in front of, or behind, the node's plane. Once these orderings — one for each source vertex — have been generated, it is possible to *combine* the ordering information, encapsulated in each list, to obtain a minimum candidate list for the polygons stored with each node.

Most nodes will appear on an ordering. The only nodes which appear on no orderings correspond to polygons which lie in the plane of the light source.

Given any node which appears on at least one ordering, the minimum candidate list algorithm proceeds as follows: For every ordering containing the node, step

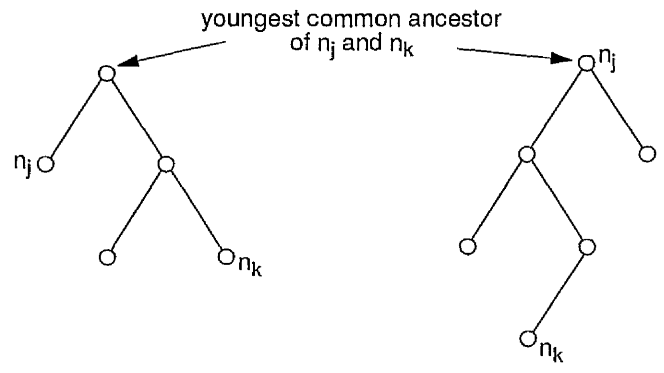


Figure 5.4: The youngest common ancestor of two BSP tree nodes.

through the ordering until the target node is reached. Whilst stepping through the orderings, if a node is encountered whose bit-mask tells us is front-facing with respect to the ordering's vertex, then the node's front-facing polygon(s) are added to the candidate list. If a node is encountered which is back-facing with respect to the ordering's vertex, the node's back-facing polygon(s) (if any) are added to the candidate list. Notice that this approach automatically excludes any polygons which are facing away from the source (polygons 3 & 5, in figure 5.3). A simple alteration would result in a similar algorithm which excluded all polygons facing the source. This algorithm may well try to add the same polygon to the candidate list more than once, but this is trivially guarded against.

How can one be sure that the polygons, generated by the algorithm just described, include all polygons lying 'between' the source and the receiver? The algorithm assumes that the vertex orderings capture *all* possible orderings of the scene, from points on the source: this is affirmed by the following theorem.

Theorem: Consider a polygonal light source, from each of whose vertices a front-to-back traversal of a BSP tree is made. Each traversal is recorded in a separate *vertex ordering*. Now, if a front-to-back traversal of the BSP tree, from any point A on the source, encounters a node n_j *before* some other node n_k, then there is at least one vertex ordering which encounters those nodes in this same order.

Proof: Let $n_{\mathbf{yca}}$ be the BSP tree node which is the youngest common ancestor of n_j and n_k. This common ancestor may be n_j or n_k (figure 5.4). Let P be the plane stored with $n_{\mathbf{yca}}$.

All front-to-back traversals, from points which lie on the same side of P as A, will visit n_j before n_k. All traversals from points which lie on the opposite side will visit n_k before n_j.

If the source lies wholly on one side of P, then *every* vertex lies on the same side of P as A, so every vertex ordering will encounter n_j before n_k.

If P passes *through* the source then, because the source is a polygon, there is at least one vertex which lies on the same side of P as A, or lies in P with A, and the ordering corresponding to this vertex will visit n_j before n_k.

Q.E.D.

The algorithm applies to convex and concave polygons alike. In fact, a short-cut is possible for concave polygons whereby only orderings corresponding to a vertex of the source's *convex hull*, are considered by the algorithm.

Polygons close to the vertex end up with very short candidate lists (empty lists are not uncommon, for such polygons). Polygons which are further from the source end up with longer lists. The mean length of a candidate list, for the

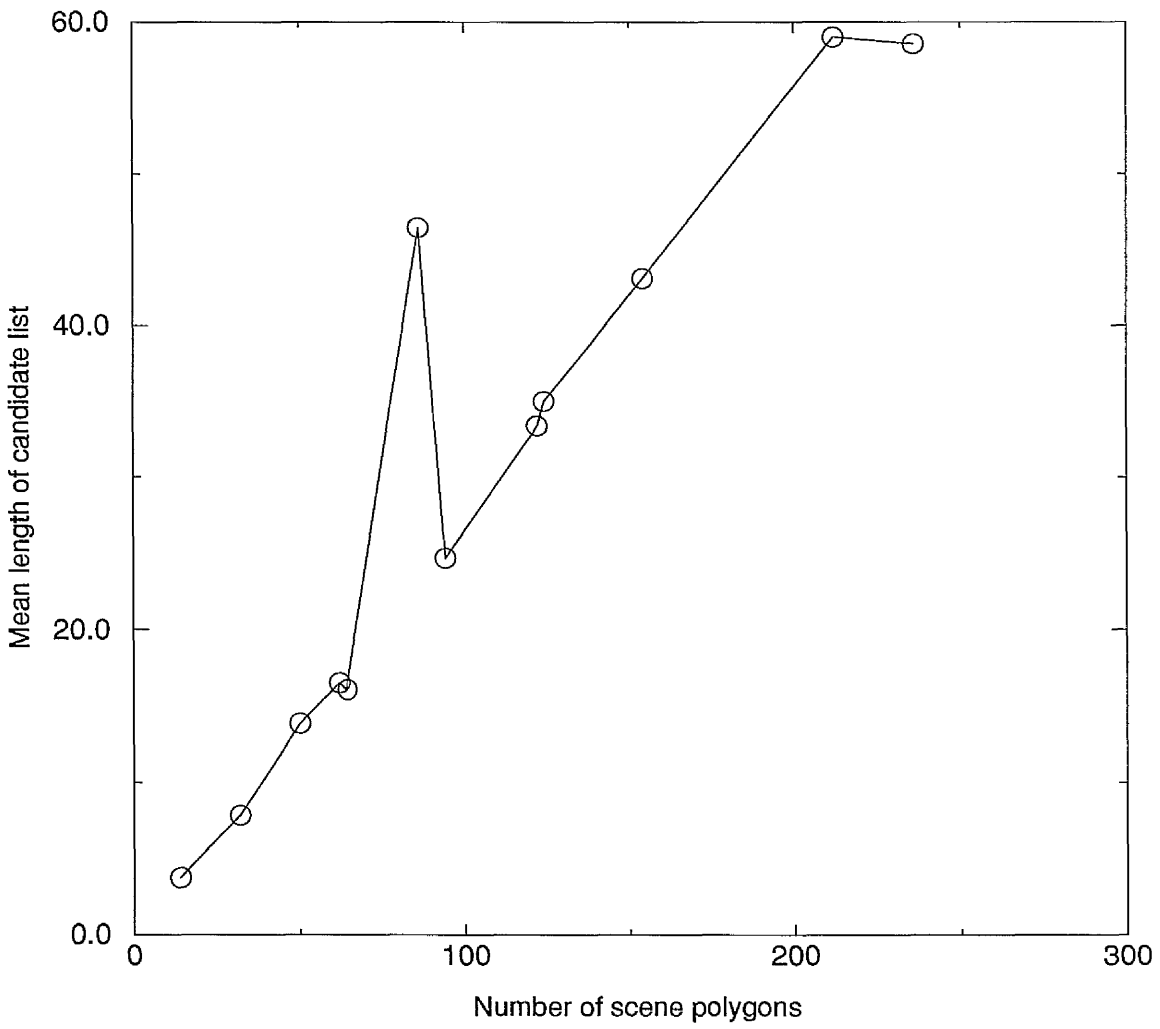


Figure 5.5: Extracting object-space ordering information from the scene BSP tree can prevent testing hopeless polygons for occlusion.

scenes tested by the author, is usually somewhere between 1/4 and 1/2 of the total number of polygons in the scene (figure 5.5). Long candidate lists result when the source is split by the plane of a node close to the root of the scene BSP tree — in which case radically different vertex orderings arise.

The algorithm can also be used to establish which polygons may lie between the edge and the vertex of an EV wedge — a problem discussed in section 4.2.2. Each source edge corresponds to two vertex orderings: these orderings can be processed exactly as before to give a list of polygons which may lie between the source edge and the occluder vertex. The EV wedge can be clipped by these polygons, before going front-to-back through the scene (from the occluder vertex) to generate discontinuity lines.

Having generated a minimum candidate list for a (source, receiver) pair, the list could be used naïvely, and every e-node/source frustum could be compared with the list. This is not the approach taken here: the list is used to efficiently establish which polygons are shadowing which parts of the receiver, so that each e-node can consult its own (tailored) candidate list.

5.1.2 Shadow classification

If each element in the DM-tree stores the specifics on how it is shadowed, then one can establish very quickly whether an e-node is lit, in penumbra or in umbra. Only in the second of these cases need one proceed any further, and look-up the list of occluders in whose penumbra this e-node lies. This list of occluders can then be used to clip the e-node/source frustum, so that only visible parts of the source contribute to the illumination of the e-node.

Initially, it had been hoped that it would be possible to shadow classify mesh elements simply by touring their edges and examining the information stored there: which wedge caused the edge; which (if any) shadow volume did the wedge bound; which occluder was involved. Early results were promising, but it becomes apparent that whilst such approaches are suitable for finding the *boundary* of shadow regions [31], they are not practical for establishing accurate occluder candidate lists for the region. An occluder *can* shadow a region without actually having any EVE wedges intersect with it. An alternative approach was sought.

In the implementation described here, original scene (OS) polygons store two shadow visibility BSP (SVBSP) trees [19]; one for the penumbra (due to the current source) and one for the umbra. The shadow volumes are constructed incrementally, as the EVE shadow wedges are processed:

- when a wedge is created, the implementation first decides whether its plane bounds the penumbra volume, the umbra volume, or neither.
- If the wedge plane *does* bound a shadow volume, then the plane is added to the appropriate SVBSP tree, for use later.

SVBSP tree-building is particularly cheap; each tree is built solely from planes which are a by-product of EVE wedge processing. When discontinuity meshing is complete, all OS polygons have two shadow volumes stored.

Not all polygons have a non-empty umbra volume. These polygons are quickly found by examining the bit-masks[3] stored with each BSP tree node (figure 5.6): some nodes will face towards some source vertices, and away from others; some nodes will not appear on all orderings. The relevant polygons, stored with such nodes, have their umbra volumes deleted.

In order to generate a list of polygons which actually cast a shadow on a given receiver, the BSP node at which the receiver is stored is passed to the minimum candidate list routine. The resulting candidate list is then clipped by the source

[3]set when handling the VE wedges

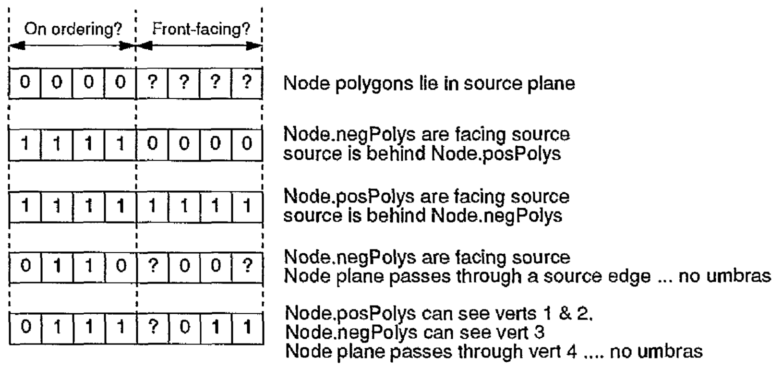

On ordering?
Front-facing?
0 0 0 0 ? ? ? ?
Node polygons lie in source plane
1 1 1 1 0 0 0 0
Node.negPolys are facing source
source is behind Node.posPolys
1 1 1 1 1 1 1 1
Node.posPolys are facing source
source is behind Node.negPolys
0 1 1 0 ? 0 0 ?
Node.negPolys are facing source
Node plane passes through a source edge ... no umbras
0 1 1 1 ? 0 1 1
Node.posPolys can see verts 1 & 2.
Node.negPolys can see vert 3
Node plane passes through vert 4 .... no umbras

Figure 5.6: Some conclusions which can be reached by examining BSP bitmasks and receiver planes, and shaft culled [47].

The polygons left on the pruned candidate list are now used to classify each element in the receiver's DM-tree as being either lit, in penumbra, or in umbra: lists of occluders are stored with penumbra regions. One could utilise the hierarchical nature of the DM-tree to achieve this, as Campbell has done [15]. The current implementation, however, does not.

Shaft culling

Shaft-culling takes numerous forms [47, 15]. Essentially, given a (source, receiver) pair, shaft-culling involves the creation of a set of planes which enclose, as *snugly* as possible, all the line segments obtained by joining a point on the receiver to a point on the source. Given this collection of planes, one can shorten a given candidate list by excluding all those polygons on the list which lie outside of the *shaft* defined by the planes.

A number of different methods have been proposed for creating the shaft planes between a polygonal source and receiver. Haines and Wallace [47] examine the axis-aligned bounding boxes of the two polygons, and build a shaft from planes which pass through an edge of each box. Whilst this is suitable for polygons and non-polygons alike, and probably optimal for polygons with large numbers of vertices, a polygon-specific approach is described in [15]: Campbell's shaft culling produces a tighter-fitting set of planes, by choosing the planes which bound the convex hull of the source and receiver, taken together.

Neither of these methods are ideal for the problem at hand, because they necessitate the *computation* of the planes which make up the shaft. Ideally, information which has already been stored should be used to prune a candidate list. This is achieved by comparing the receiver with the penumbra volume (SVBSP tree) of every polygon on the candidate list. Note that, for four-sided polygons, the penumbra volumes will consist of between between four and eight planes, so pruning a candidate list containing c polygons will involve between $4c$ and $8c$ polygon/plane comparisons. This is exactly the cost of both of the other shaft culling methods [15, 47].

As well as avoiding any unnecessary computation, a second advantage of this culling approach is that when the receiver lies wholly inside a shadow volume, the whole polygon can be classified as lying in penumbra[4]. Such conclusions cannot be drawn when using either of the other two culling methods [15, 47] — a polygon either shadows, or not — extra work has to be done to establish *how* an occluder shadows the receiver.

[4]These polygons can then be tested against the occluder's umbra volume, if present, and possibly classified as being wholly in umbra

5.2 Optimising mesh triangulation

After all EVE discontinuity lines have been located, stored in DM-trees, and the resulting elements shadow classified, the elements are meshed into triangles, so that quadratic elements can be easily fitted across them. The elements which need triangulating will vary considerably in size, shape and number of vertices; but all will be convex.

Lischinski *et al* [70] triangulate elements by finding the mid-point of the longest diagonal, and joining this to all remaining vertices. A number of problems are associated with this approach, and the situation was improved upon in [71], where constrained Delauney triangulation is used to triangulate the mesh.

In the implementation described here, two alternative triangulation methods are examined. The original triangulation scheme, involving the mid-point of the longest diagonal, was deemed unsuitable for a number of reasons:

- An extra vertex is created for each element being triangulated.
- The number of triangles produced often seems excessive.
- The method does not produce particularly well-shaped triangles.

Both of the new algorithms involve choosing a diagonal, splitting the element by this diagonal, and recursing with the two resulting halves until triangles result. The new methods only differ in their choice of diagonal: in the first method, the shortest diagonal is chosen; in the second method a heuristic is used to make this choice (figure 5.7). The heuristic is designed to favour diagonals which:

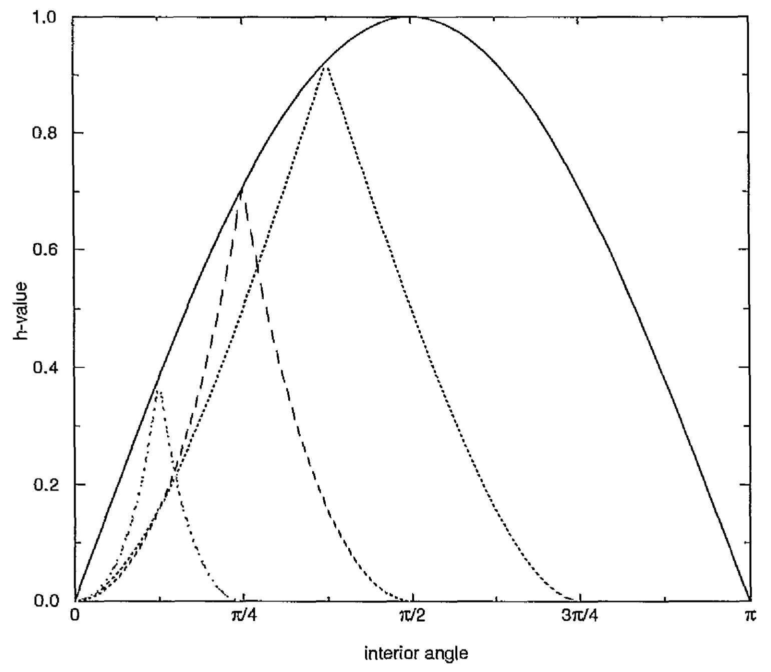


Figure 5.7: The plot shows h-values corresponding to attempts to split the angles $\pi/4$, $\pi/2$, $3\pi/4$ and π by a diagonal. Large angles are favoured over small ones. Unequal splits are punished.

1. split large internal angles of the element;

2. bisect the angles being split.

By favouring those vertices of the element where the internal angle is large, the aim is to avoid triangles with internal angles which are very small. By favouring the *equal* splitting of an internal angle, the heuristic aims to avoid the same problem arising from a particularly uneven split.

If a diagonal splits an internal angle into two angles θ_1 and θ_2, the *h-value* given to this end of the diagonal is:

$$\left(\frac{\min(\sin\theta_1, \sin\theta_2)}{\max(\sin\theta_1, \sin\theta_2)}\right)\min(\sin\theta_1, \sin\theta_2) \tag{5.1}$$

The value given to a choice of diagonal is the product of the *h-values* given to its ends. In this way, a value in the range $[0, 1]$ is obtained, which is large when both ends of the diagonal nearly-bisect large angles. As with the other two methods (shortest and longest diagonals), every possible diagonal is tested, and the diagonal with the largest heuristic (5.1) value is used to split the element.

5.2.1 Comparison of triangulation schemes

In order to measure the comparative effectiveness of the three methods, all three algorithms were implemented, and the resulting meshes (after adaptive subdivision) were compared. The comparison concerned itself with:

- the number of resulting triangles, and
- how well-shaped these triangles were.

During adaptive subdivision, a triangle is subdivided when the computed radiosity value at its centroid differs from the interpolated value there by more than some user-specified threshold value. Figure 5.8 shows a graph comparing this threshold value to the total number of mesh triangles in a test scene, for each triangulation scheme.

In order to decide how well-shaped a particular triangle was, its *radius-ratio* was calculated: the ratio of its inscribed circle radius to its circumscribed circle radius [7] (figure 5.10). In order to decide how well-shaped a particular scene triangulation was, *radius-ratios* were obtained for every triangle in every mesh in the scene, and their mean was evaluated. Figure 5.9 shows a graph depicting how these mean values vary with triangulation scheme and user-specified error threshold, for a test scene.

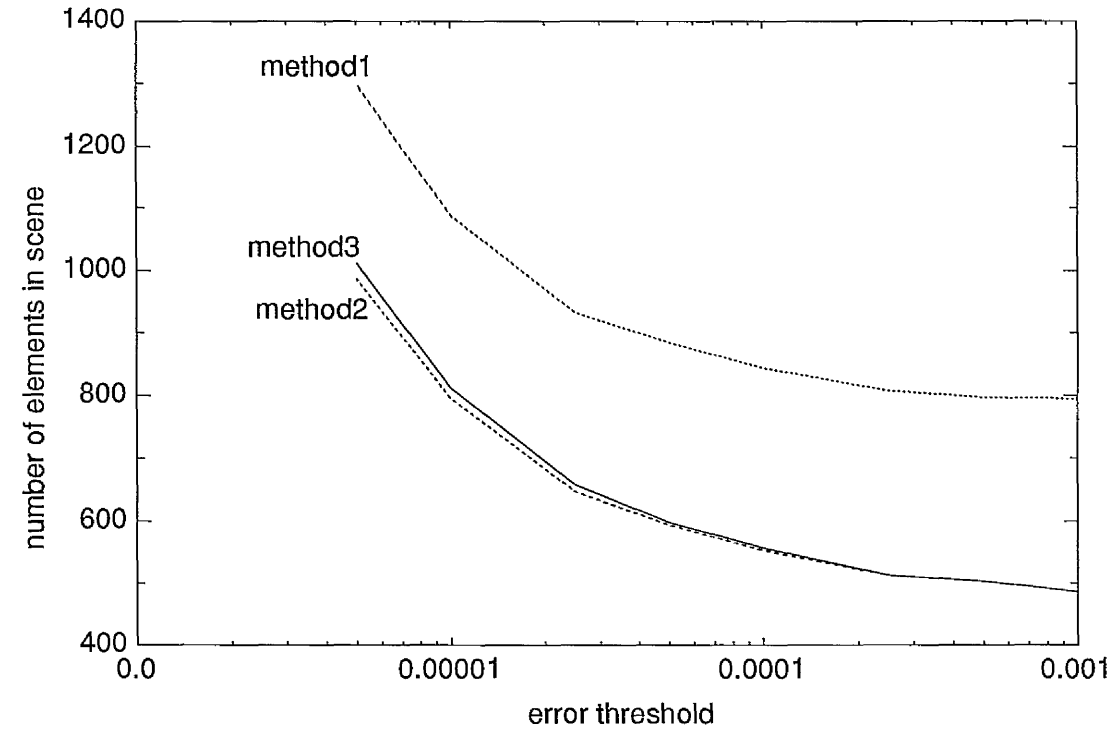


Figure 5.8: Method 1 is longest diagonal, Method 2 is shortest diagonal, Method 3 is heuristic diagonal. Either of the new methods produce markedly less triangles for the same error.

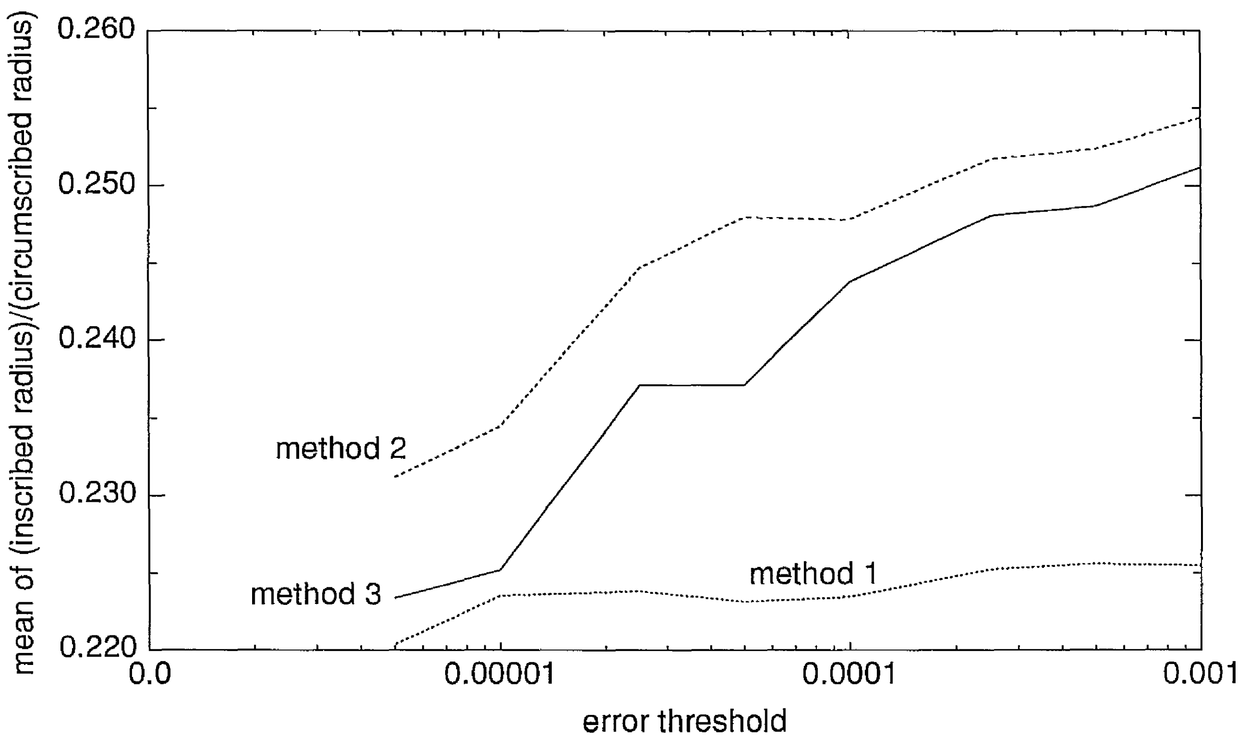


Figure 5.9: Method 1 is longest diagonal, Method 2 is shortest diagonal, Method 3 is heuristic diagonal. Either of the new methods produce better-shaped triangles for the same error.

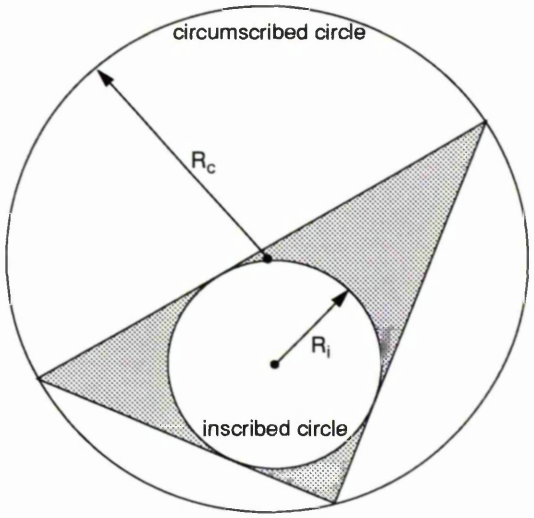


Figure 5.10: One can judge how 'well-shaped' a triangle is by evaluating the ratio $R_i/R_c \in (0, 0.5]$.

Each triangulation scheme was applied to five test scenes, with eight different threshold values — a total of forty runs per triangulation scheme. Statistics relating the number of mesh triangles, and mean radius-ratios, to the user-specified error threshold were recorded. A full account of these findings is presented in appendix C. Suffice to say here that figures 5.8 and 5.9 give a fair indication of the situation *vis-à-vis* the three triangulation schemes: either of the new methods will not only mesh to the required tolerance with less triangles than the previous method, but will do so with better-shaped triangles. In particular, the shortest-diagonal algorithm seems to have the best characteristics for the task in hand.

5.2.2 Other triangulation thoughts

Two other points which relate to the optimisation of mesh triangulation, are now considered. Neither of these algorithms have been implemented by the author, but are included here in an effort to give a fuller picture of how this crucial part of DMR can be optimised.

Deciding when to subdivide

When building a DM-tree, it is usually not sufficient to simply triangulate the mesh which results from the location of discontinuities. Often, the quadratic elements [131] across the triangles in a mesh will not match the true radiosity across the triangle closely enough, and the triangle is subdivided. In section 4.2.3, the implemented method for establishing *which* triangles are subdivided, was described.

The implemented method is expensive, because finding the exact radiosity value anywhere on a receiver is expensive. Shadow classification makes the process cheaper, because many centroids are known to have an unoccluded view of the source, and those which do not, can quickly find which polygons are blocking the view. There is, however, some more information available, which may help in the search for a cheap algorithm to decide whether a triangle should be subdivided. By locating all D^0, D^1 and D^2 discontinuities in the radiosity function across the receiver, one can assert that the radiosity across edges which do *not* represent such discontinuities, should be at least C^1 [90]. For triangles with such edges, a subdivision algorithm which proceeds as follows, seems practicable:

- For each edge not corresponding to a discontinuity, find the normal to the quadratic interpolant surface at the mid-point of the edge.
- Using the winged-edge data structure, find the element lying on the other side of this edge.

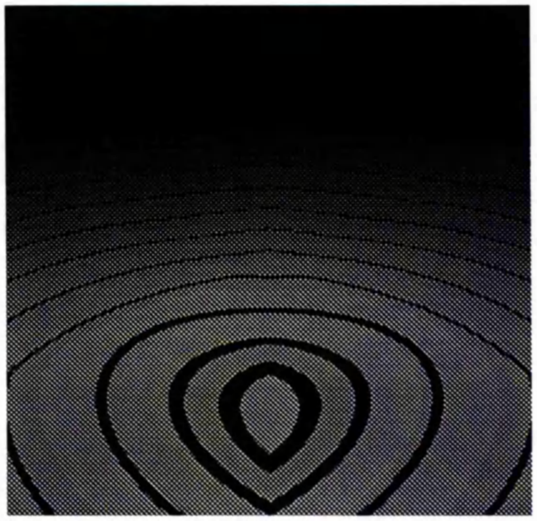


Figure 5.11: The figure shows an image whose colour map has been altered so as to highlight isolux contours, in the interpolant surfaces across a polygon which has been poorly subdivided: the discontinuities in gradient, in the interpolated surface, are clearly visible.

- Find the normal to the interpolant surface of the adjacent element, at the mid-point of the edge.
- Compare the two normals:
  - if they are nearly parallel, all is well;
  - if not, subdivide the triangles which share this edge.

For triangles without such edges, Lischinski *et al*'s approximation [70] to the L_∞ error norm could be applied. A further degree of flexibility still remains: in how finds the normal to an interpolant surface. One first needs to find two tangent vectors to the surface, at the point of interest, and then find their cross product. These tangent vectors could either be found analytically, by taking partial derivatives, or numerically, by examining points close to the point of interest. Since the interpolant surface is only a quadratic, it may well be that the numerical approach would be optimal. These issues require further research.

Figures 5.11 and 5.12 each show an image which has had its colour map altered, so as to highlight isolux contours in the interpolant surface representing the true radiosity. Each image shows a different situation where poor mesh quality appears to be linked to D^1 discontinuities in the interpolant surface.

Problems with DM-trees

Whilst the DM-tree allows one to construct a mesh containing discontinuity lines particularly efficiently [15, 70], it is not without its drawbacks [15, 53]. When building the tree, discontinuity edges arise which do not fully span the face to which they are being added. Such edges are augmented by *construction edges* which lie along the same line, but which reach out to the edges of the face (figure 4.12). Construction edges can cause problems when a number of small, parallel, discontinuity edges (due, say, to a number of VE wedges involving the same edge) are added to a large face in the DM-tree. Faces result which are very long and thin and which lead to particularly poorly-shaped triangles (figure 5.12).

One possible solution might be to split the location of discontinuity lines into two passes: a first which dealt only with wedges forming part of a *penumbra* volume, and a second which handled all other wedges. In this way, wedges which do not form part of a shadow boundary will not force their presence into those parts of the mesh which are lit, and the resulting mesh should consist of markedly better-shaped elements (figure 5.13). The use of front-to-back vertex orderings, to generate minimum candidate lists, facilitates such an approach: one would not have to repeat front-to-back traversals of the BSP tree in the second pass, one could simply step along the vertex orderings.

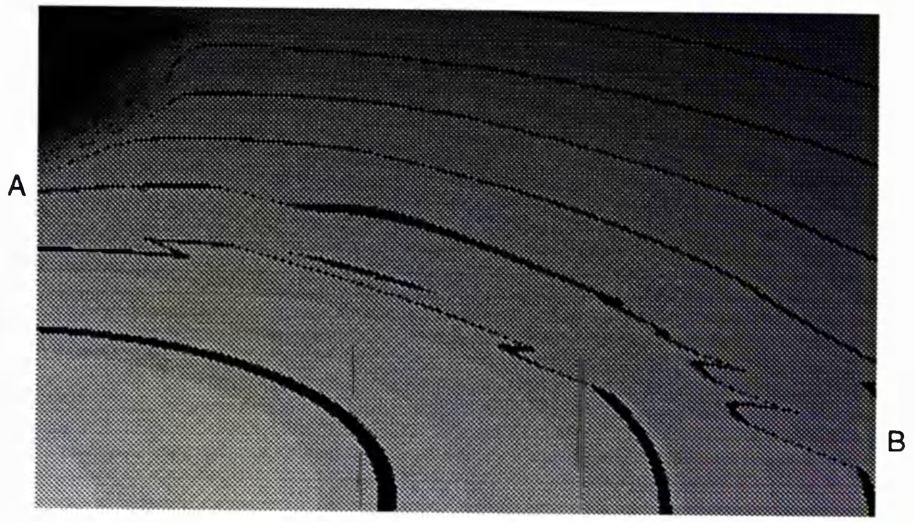


Figure 5.12: Long thin triangles affect image quality. The mesh used to generate this image has long thin triangles stretching from A, towards B. The colour map has been altered to emphasise the anomalies.

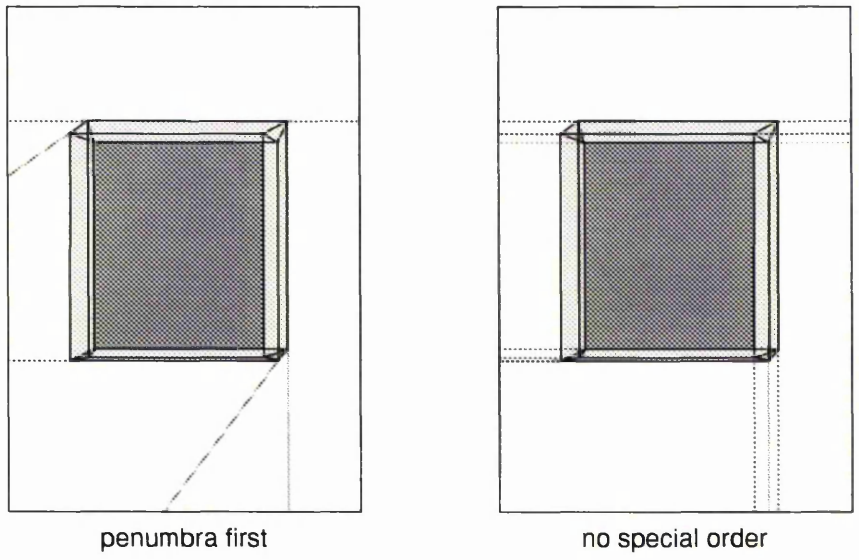


Figure 5.13: Mesh quality could be improved by handling penumbra wedges first.

A second possible solution to poorly shaped meshes may be to precede discontinuity meshing by a coarse premeshing of the surfaces [31, 106]. This would avoid undermeshing (figure 5.11) and localise any mesh anomalies like long thin triangles (figure 5.12). A combination of the coarse pre-meshing with a penumbra-first approach, may be doubly beneficial.

5.3 Merging *versus* layering

Probably the most fundamental difference between the algorithm described here, and radiosity as it has been implemented elsewhere, is the way in which the contribution from each light source is stored, with a receiver polygon. As with Lischinski *et al*'s algorithm [70], each light source results in a mesh on all those receiver polygons which can see it. Each mesh stores the contribution of this particular light source to the receiver. In [70], the mesh from one light source is merged with the mesh from the next source — a process which proved prohibitively expensive. In the implementation described here, the meshes from different light sources are not merged, but are stored separately, in *layers*.

The reasons for the adoption of a layering, rather than a merging, approach are multifarious. The driving force behind the development of the algorithm was the fact that the merging approach was reported, by Lischinski *et al* [70], to have failed for sensibly large scenes. In the merging approach, the need to merge meshes, due to different light sources, is made tractable by the winged edge data structure (WEDS, appendix B) in which much of the mesh is stored. If one forgoes merging, then whilst one still needs a WEDS (or similar) to *build* the mesh, no such constraint applies to its *storage*. After a mesh has been created, the only task ever requested of it is to return a radiosity value, given a point on the receiver.

5.3.1 Layering specifics

With this reasoning in mind, the implementation builds, for each light source, a triangulated, adaptively subdivided DM-tree, on every polygon visible to the source. Every such DM-tree consists of a uvBSP tree whose leaf nodes point to a WEDS face. Before moving on to the next source, the new algorithm:

- replaces the WEDS vertex ring with an array of *compressed vertices* which store only a (u, v) position, and a radiosity value;
- replaces the WEDS edge ring with an array of *compressed edges* which store only a pointer to the edge's (u, v) line equation, and a radiosity value;
- replaces each WEDS face by a *compressed face*, which stores a boolean indicating whether the face is lit at all, and an array of 6 integers which index into the vertex and edge arrays. This array is NULL if the face is dark — making umbra nodes particularly cheap to store. Each leaf node of the uvBSP tree now points to a compressed face;
- deletes the WEDS;
- adds the uvBSP tree and accompanying data to an array of *layers* stored with each OS polygon;
- re-initialises the DM-tree ready for the next shoot.

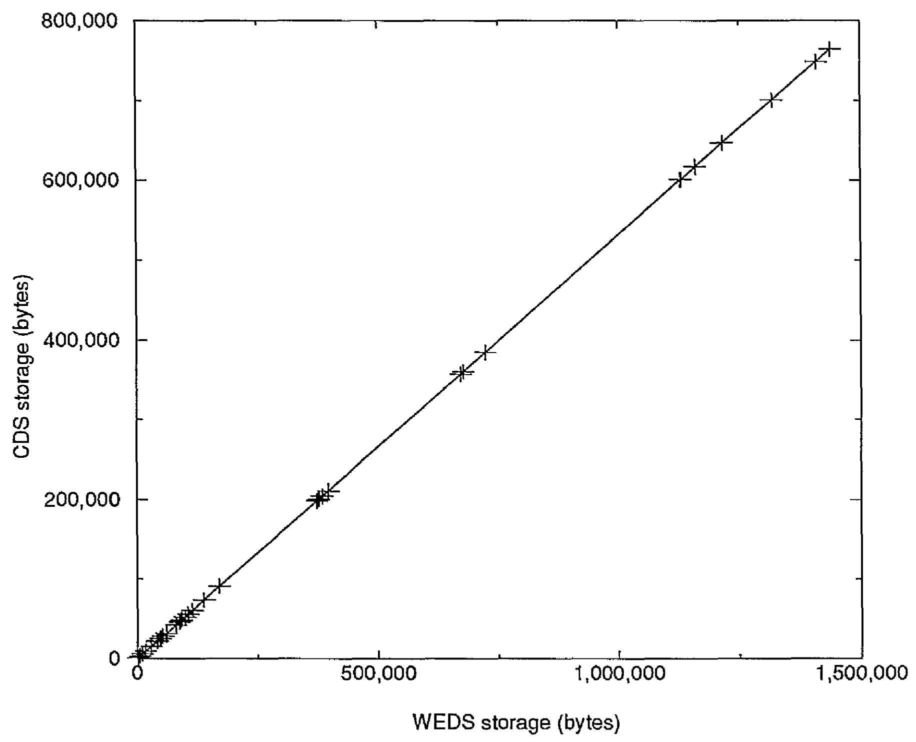


Figure 5.14: The compressed data structure takes up markedly less storage than its WEDS counterpart.

The new compressed data structure (CDS) takes up markedly less space than a WEDS (figure 5.14). When seeking the radiosity value, at some point on a receiver across which the radiosity is stored in layers, one simply looks up a value in each layer and adds the values together.

Leaves of the uvBSP tree representing umbra regions are known as *umbra nodes*. A further storage optimisation should be possible, whereby one traverses the uvBSP tree associated with a particular CDS, and recursively deletes (bottom up) all *sister*[5] umbra nodes — changing their parent node into an (umbra) leaf node. This may well prune the tree considerably, but has not yet implemented by the author.

[5]Leaf nodes which share the same parent node.

5.3.2 Advantages of layering

A number of advantages are associated with layering.

Simplicity: The compressed data structure is very simple, and one avoids the (non-trivial) task of merging meshes.

Accuracy: Mesh merging constitutes resampling of interpolated data. As such, accuracy is lost. This does not occur with layering.

Storage: A CDS takes up about half as much space as its causal WEDS. Also, merging two meshes requires the simultaneous storage of three DM-trees; with layering this cost is avoided.

Efficiency: Rivara's two-triangle method [88], which is used during adaptive subdivision, results in a triangulation where the transition between large and small triangles is smooth. A merged mesh, which contains shadow information from many light sources, will have few regions which are far from a shadowed (densely meshed) area. Consequently, a merged mesh will be hard-pushed to have large triangles anywhere. If the shadows from different sources are kept separate (layered), the same problem is less likely to arise (figure 5.15).

Parallelism : The layering approach lends itself particularly well to being parallelised.

Umbras: When layering is combined with the shadow classification of mesh elements, as in the implementation described here, one need never triangulate

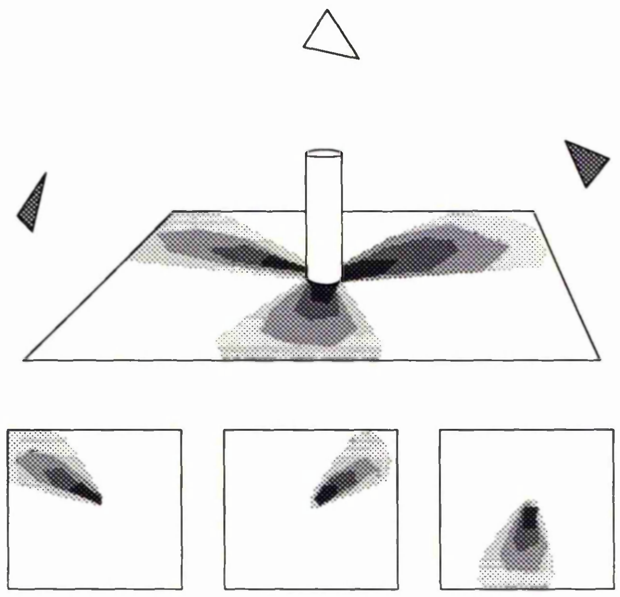


Figure 5.15: Separate layers are more likely to have efficiently meshed regions. Merged meshes have shadows almost everywhere.

umbra regions. Being kept separate from other sources, and therefore always representing a contribution of 0, one need never interpolate in these regions. This can greatly reduce the number of triangles in the mesh. Such regions take up next to no storage in a CDS.

Complexity: In theory, a scene with l light sources and e polygon edges can result in $\mathcal{O}(le)$ discontinuity lines on each polygon. When merged into a single mesh, these lines could result in a mesh containing $\mathcal{O}(l^2e^2)$ elements. When stored in seperate meshes, the total number of elements can only be $\mathcal{O}(le^2)$. These are, however, worse-case figures [71]; further research is needed to establish which approach is actually optimal.

5.3.3 Converged solution

One feature of the layering method, as it has been described thus far, is apparent: no mention of *convergence* has yet been made — the method simply locally illuminates the scene, by each of the primary light sources. Whilst such images are visually impressive (appendix C), this is a long way short of global illumination, which is the author's aim. One could continue, as Lischinski *et al* [70] did, and keep on building layers, and adding[6] them into the whole, but this does not seem a promising approach: the storage costs will simply keep on building up, eventually becoming prohibitive.

Whilst the code implemented thus far, only deals with the primary light sources, and does not reach a converged solution, a strategy for doing so is now outlined.

Two things are immediately apparent.

- If the shadow discontinuity lines, found by treating *every* polygon as a source, are accounted for, then the memory costs will almost certainly become crippling [70].
- One cannot ascertain *a priori* which polygons should be treated as important sources [63].

The solution outlined here proposes that one *first* accounts for the direct illumination, due to the primary light sources, and *then* solves for the indirect diffuse illumination, in the scene. The primary light sources are handled as has already been described, in an iterative, layering algorithm, accounting for all D^0, D^1 and

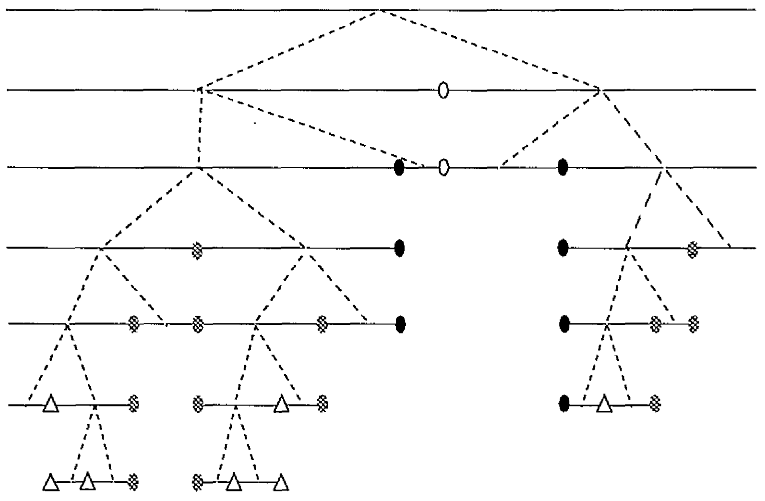
[6]Merging, in the case of [70], layering here.

D^2 shadow discontinuities. Note, however, that once the primary light source layers are *in situ*, the only illumination left unaccounted for, in a diffuse scene, is indirect diffuse illumination (IDI). Also, recall the discussion of section 2.3.3 and figure 2.9; here it was noted that IDI is the most slowly-varying of all the various illumination components and, as such, is particularly cheap to model [97, 63].

Consider how a layer is built, from scratch: the highest nodes in the uvBSP tree correspond to splits which occurred during the building of the scene BSP tree, and to D^0 discontinuity edges. The next group of nodes down the uvBSP tree, correspond to splits which occurred when discontinuity meshing. The nodes nearest the leaves of the tree, correspond to splits made during triangulation, and adaptive subdivision (figure 5.16).

Now consider how a uvBSP tree was constructed in Lischinski *et al*'s hierarchical algorithm [71]. The first discontinuity lines added to their tree are the D^0 lines. The next lines added are the D^1 lines, then the D^2 lines, and then the CDT lines. The point being, that the two trees (the layers described here, and the DM-trees in [71]) share a common feature: both are quick to separate umbra, penumbra and lit regions.

As noted in [71] (section 4.3), the fact that shadow regions and lit regions are separated near the top of the tree, makes the hierarchy ideally suited to a hierarchical radiosity solution. By *pulling* the leaf radiosity information up a uvBSP tree (using area-weighted averaging) one can store radiosity values with nodes throughout the tree. A standard [50] refining and linking process can now take place; the initial linking only involving those nodes with a radiosity value above some user-defined B_ε. This means only nodes near the top of tree will take part



Splits near the top of the tree occur during BSP tree creation:	0
The next splits down the tree occur when handling D^0 edges:	●
The next splits down the tree occur when handling D^1/D^2 edges:	▨
The lowest splits in the tree correspond to mesh triangulation:	△

Figure 5.16: The 2D line represents a 3D polygon. The DM-tree which results from the method described in this work, quickly separates lit and shadow regions.

in initial linking, and links will have to be made between all layers on all (inter-visible) scene polygons. A hierarchical radiosity solution can now proceed exactly as described in [50, 71], the final radiosity values stored at the nodes being *pushed* down to the leaves, for display.

This approach has a number of advantages:

- Because the only transport being accounted for in the hierarchical solution, is IDI, the solution can proceed with a fairly large error threshold, confident that this will be sufficient to model the relevant transport. If p scene polygons each have l layers, then $\mathcal{O}(lp)$ links may be required: in fact, the large error threshold should keep this figure manageable.

- The user does not have to wait for initial linking before an image is available.

Images from each iteration of the progressive refinement (direct illumination) pass, are available quickly: only when an impressive image has already been achieved, does one have to wait for a converged solution.

- One need never concern oneself with reconstructing the radiosity function across a polygon: re-calculating form factors, or visibility terms (section 4.3); the *nature* of the radiosity across the surface has already been established in the direct illumination pass.

- During the hierarchical solution, one need never split a node when an attempt to link fails to satisfy the error criterion; one simply recurses down the uvBSP tree. Links can be forced at leaf/leaf interactions.

Chapter 6

Conclusions

6.1 Previous work

This thesis has presented a review of recent innovations in the radiosity method in computer graphics, together with a number of optimisations which have helped to improve the performance of one of the more complex algorithms.

Beginning with the development of a physically-based transport model, the rendering equation [61], the aim from the start has been to reach conclusions which would be applicable to problems found in real-world applications, such as lighting and architectural design.

An investigation, of a number of classical solutions to the rendering equation, was made in chapter 2. Whilst many impressive images have resulted from the constant radiosity and ray tracing algorithms of chapter 2, the latter are somewhat deficient in that they struggle to reach a truly global solution, and the former are somewhat deficient in that they suffer from a number of accuracy problems; due to their insistence on representing the radiosity across a surface as a piecewise constant function.

In chapter 3, a thorough investigation of higher order radiosity methods, based on finite element theory, was presented. A number of advantages, to such an approach, were indentified:

- When representing the radiosity as a piecewise polynomial, rather than piecewise constant, function, the true radiosity may be *very* closely represented by the solution function, across much of the scene. The same can rarely be said of classical radiosity.
- Convergence rates are impressive when using higher order polynomials.
- Far less patches are required than in classical radiosity; storage costs are much reduced.

The weighted residual radiosity methods of chapter 3 also represent a slightly more subtle improvement in the radiosity method: by setting the problem in a *sound mathematical framework*, the problem becomes well-defined, and its limitations clear. Avenues for improvement can be sought within the context of the new framework.

Wavelet theory, and a discussion of how it might be applied to radiosity, was also covered in chapter 3. Wavelets were found to be a particularly powerful tool for the discrete representation of functions and operators, in a number of dimensions. By *projecting* the radiosity function, across a bi-parametric surface, into a wavelet basis, one obtains a cheap, accurate representation of the function — made possible by the wavelet basis' ability to represent *smooth* regions very cheaply. By projecting the kernel function into a compatible basis, which also results in a highly sparse representation, an $\mathcal{O}(n)$ solution process is possible.

Whilst the methods of chapter 3 were shown to be a marked improvement over the constant radiosity algorithms of chapter 2, they are not without their problems:

- Evaluating energy interchange between two order N basis functions is an $\mathcal{O}(N^4)$ problem. Whilst one might expect a typical scene to include less higher order basis functions, than if constant basis functions were used, it is not yet clear whether this advantage will be dragged down by the high costs of calculating energy interchange.
- As yet, wavelet methods cannot be applied to triangular mesh elements, which may make their useful application problematic.

All of the radiosity methods examined, in chapters 2 and 3, share a common stumbling block. This concerns the inability of a piecewise polynomial function to accurately represent any sort of significant discontinuity, within the span of one its basis functions. Unless care is taken to avoid such situations, they will invariably occur, and the costs can be severe:

- in constant radiosity algorithms, light and shadow leaks result;
- in conventional higher order methods, Gibbs ringing results;
- in wavelet radiosity, the sparseness of the kernel is adversely affected;

- In all of the approaches examined, convergence rates are hampered.

A closer look at such discontinuities, seemed in order.

Chapter 4 presented an examination of discontinuities in the radiosity function, the conclusion being reached that shadow discontinuities, up to order 2, should be explicitly accounted for during the radiosity solution. A review of previous discontinuity meshing radiosity (DMR) algorithms was presented, with clarification being given by the author on a number of issues left unclear, by others. Whilst the DMR algorithms of chapter 4 are impressive in and of themselves, they were found to be lacking in a number of respects:

- Whilst great effort is expended finding shadow discontinuity edges, many algorithms reached no conclusions about which regions lie in shadow.
- The costs of the non-hierarchical approach were seen to be prohibitive.
- The mesh building and illumination was often far from optimal.

6.2 Author's work

All of these issues were addressed in chapter 5, where the author's optimisations for DMR were presented. The major optimisations being:

Minimum candidate lists: A new algorithm for extracting object-space ordering from a BSP tree was presented. The algorithm is simple to implement, and allows one to exclude between 50% and 75% of the polygons in a scene, when building an occluder candidate list for a (source, receiver) pair. The information used to build these lists was shown to be a by-product of discontinuity meshing, and so costs are low.

Shadow classification: Each mesh element is classified as being either lit, in penumbra, or in umbra. Elements which lie in penumbra store a list of polygons in whose shadow they lie. Shadow classification involved the use of SVBSP trees, which were also used to *shaft cull* occluder candidate lists. The planes used to build these trees were shown to be a by-product of discontinuity meshing, and so costs are minimal.

Mesh layering: A new method for handling contributions from multiple light sources, in DMR, was presented. By *layering* the contributions from each source, a host of advantages were shown to have emerged. The layering method was shown to be simpler, have lower storage costs, be more accurate, and have a lower complexity, then the merging approach.

Mesh triangulation: Two new mesh triangulation schemes were presented, both of which were shown to result in meshes containing less triangles, for a given error tolerance, than the previous method. Both methods were also shown to produce better-shaped triangles, for a given error tolerance, than the previous method.

Chapter 5 also presented a suggestion for a hierarchical indirect diffuse illumination pass, to follow the direct lighting, layering, pass. This seems a particularly promising approach, due to the slowly-varying nature of the illumination component being handled by the pass.

6.3 The future

A number of issues remained unaddressed, the foremost of which is the modelling of indirect diffuse illumination. The author would very much like to persue the hierarchical algorithm described in section 5.3.3, which he is confident would lead to a converged, global solution.

The one overriding thought in the author's mind which remains, at the conclusion of this work, concerns the sheer complexity of discontinuity meshing radiosity. The vast number of problem cases that can arise, once the scene geometry becomes intricate, far exceeded the number envisioned by the author at the outset of this research. This might suggest a limited application the DMR, where discontinuity meshing takes place only in those parts of the scene where it is deemed necessary. Whether a human or a machine would decide where such meshing was necessary, or a combination of the two, is an open question: both options seem plausible.

Discontinuity meshing radiosity is still limited in that it can only handle polygonal scenes. Extensions to curved surfaces seem unlikely, given the complexities of the polygonal case, but if such meshing is only being applied in areas where it was deemed important, then perhaps such extensions should be investigated.

Another area for improvement involves the reflection properties of the surfaces being considered. In the same way that previous radiosity algorithms have been extended to handle specular reflection, DMR could be extended.

Parallelism is another area which deserves close attention. Whilst DMR algorithms are complex, they are certainly manageable, perhaps a parallel approach would make the costs of DMR more acceptable. The layering method, introduced in this thesis, seems are particularly likely candidate for parallelism.

Whatever the future holds for discontinuity meshing radiosity, the author is confident that it has earned its place there: radiosity methods which do not account for discontinuities will *always* encounter problems. The radiosity community, and the computer graphics community in general, have recognised, in recent years, the importance of setting their work in a sound theoretical framework if work is to progress; the results so far, have been impressive. The future holds no bounds.

Appendix A

Binary Space Partitioning trees

Many of the algorithms, described in this thesis, are based around the notion of Binary Space Partitioning (BSP) trees. This appendix presents a précis of BSP trees; how to build them, their advantages, and their uses as they relate to this thesis.

BSP trees have been used extensively in the computer graphics community for many years; their success has probably been helped by their simplicity. Also, the ease with which they allow one to extract *order* from the seeming chaos which is the data being processed. BSP trees can be used in an arbitrary dimensional space, but this discussion will concentrate on the three-dimensional case.

In three dimensions, a BSP tree represents a hierarchical partitioning of 3-space by a set of planes. Clearly, a BSP tree is binary, each node storing: a plane, a list of polygons which lie in this plane, and pointers to two branches. Each node represents a region of space; the root node represents the whole of whatever space is being partitioned. The plane stored with each node splits the space, represented by the node, into two half-spaces — one wholly in front of the plane, and one wholly behind it.

Given a list of polygons, a BSP tree is constructed as follows (figure A.1):

- select a polygon from the list;
- note the plane in which this polygon lies; store this plane with a new node;
- create two new lists, corresponding to
    1. polygons which lie in front of the chosen plane, and
    2. polygons which lie behind it.

  Any polygons which lie *across* the plane, are split by it, and each half is added to the appropriate list. Any polygons which lie *in* the plane are stored with the new node.
- Create the new node's positive branch by recursing with the first list;
- create the new node's negative branch by recursing with the second list.

Such an approach was first formulated by Fuchs *et al* in [35, 34]; their research having been motivated by Schumacker *et al*'s earlier work [95][1]. Notice that, because polygons are split during the construction of a BSP tree, the polygons stored in the tree may not be exactly those polygons which were originally provided. Throughout this thesis, polygons stored at the node of a BSP tree are referred to as *BSP polygons*, to distinguish them from *original scene (OS) polygons*.

[1]described in [108]

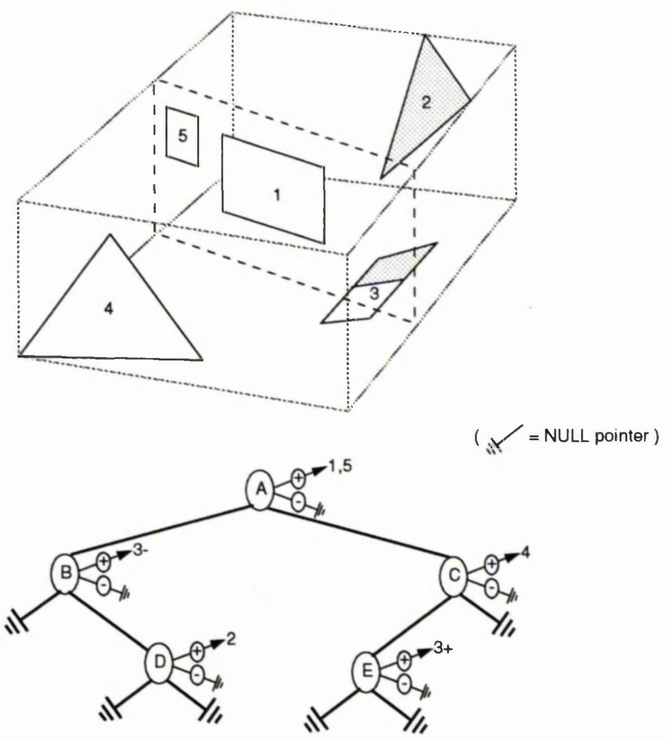


Figure A.1: Building a binary space partitioning tree.

If a polygon is regarded as having distinct front and back sides, then one can store two lists of coplanar polygons with each BSP tree node: one for polygons which share the same normal as the plane stored with the node, and one for polygons which share the opposite normal. This second list is often empty. Such an approach is implemented in the work described here.

The *raison d'être* of BSP trees, is that they allow an observer to make conclusions about the *relative position* of the scene polygons. Given any point, in the space represented by a BSP tree, it is possible to create an *ordering* of the polygons stored in the tree, whereby: the point has a wholly unoccluded view of the i^{th} polygon in the ordering, as long as the only candidates for occlusion are the $(i+1)^{th}, (i+2)^{th}, \ldots$ polygons in the ordering. This a *front-to-back* [35] ordering. A reverse (*back-to-front*) ordering is also possible, where the point has an unoccluded view of the i^{th} polygon as long as the only candidates for occlusion

```
void front-to-back ( pt, tree )
{
   if ( tree == NULL )
      return;

   if ( in-front( pt, tree.plane ) ) {
      front-to-back ( pt, tree.pos );
      output ( tree.posPolys );
      front-to-back ( pt, tree.neg );
   } else {
      front-to-back ( pt, tree.neg );
      output ( tree.negPolys );
      front-to-back ( pt, tree.pos );
   }
}
```

Figure A.2: Pseudocode for the front-to-back traversal of a BSP tree.

are the $(i-1)^{th}, (i-2)^{th}, \ldots, 1st$ polygons.

Fuchs *et al* [35] employed a back-to-front ordering to implement a *painter's* rendering algorithm. In such an approach, the scene is traversed back-to-front with respect to the eye/camera position, and polygons are output directly to the image buffer — without any regard for which pixels have already been written to. The nature of the ordering ensures that the resulting image hides (and shows) the correct regions of the scene. A problem with such an approach, is that any one region of the buffer may be written to many times. Polygons may be rendered, only to be completely obscured later; this can become expensive when large scenes are involved [44]. Gordon and Chen [44] improved on the situation by employing a front-to-back approach, where care was taken to write to buffer locations once only.

A.1 BSP ray tracing

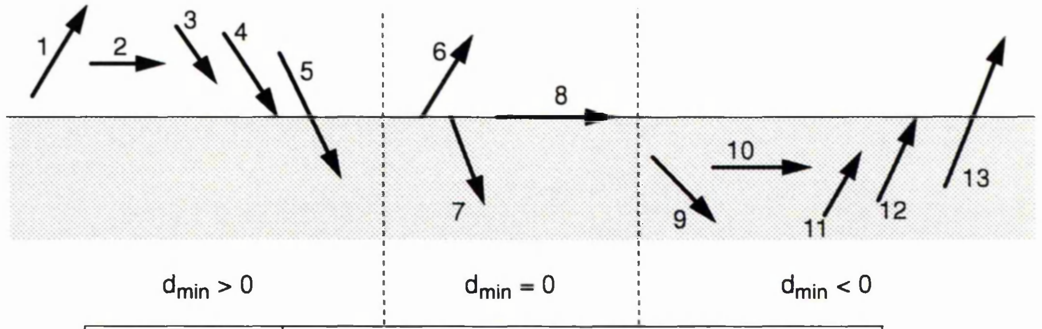
One area where the front-to-back ordering of scene polygons has proved highly beneficial, is in ray tracing. Thibault [112] described an algorithm whereby a ray is treated like a line segment as it is filtered down a BSP tree; the plane stored with each node being used to clip the ray[2], as it recurses down the tree.

Filtering a ray (a directed line segment) down a BSP tree is not as simple as the previous front-to-back algorithm considered (figure A.2), but it is not too far removed. Essentially, whenever a node plane splits a line segment (representing some part of the ray) the algorithm recurses, with the half which lies nearest the ray origin, into the half-space containing this part of the ray. If this recursion fails to find an intersection, the ray is compared with the polygons stored at the node. If this fails to find an intersection, the algorithm recurses with the other ray half into the other half-space (figure A.4).

If the axis-aligned bounding box of all the polygons stored in a BSP branch, is stored with the branch's root node, a further optimisation is possible; by carrying out a ray/box comparison before recursing. This has also been implemented by the author.

In such an approach the *first* ray/polygon hit found, is the nearest hit the ray has to its origin. This is a great plus compared to many ray tracing algorithms, which must compare the ray with other parts of the scene before they are able to ascertain which hit is closest.

[2]A ray is defined by an origin, a direction, and two reals which specify the particular segment of the ray being considered: these give the distance from the ray origin of the segment's two ends (initially $t_{\min} = 0$ and $t_{\max} = \infty$).



case:	action:
1,2,3,6,...,11:	$[t_{min}, t_{max}]$ *vs.* near
4	$[t_{min}, t_{max}]$ *vs.* near; posPolys test
5	$[t_{min}, t_{int}]$ *vs.* near; posPolys test; $[t_{int}, t_{max}]$ *vs.* far
12	$[t_{min}, t_{max}]$ *vs.* near; negPolys test
13	$[t_{min}, t_{int}]$ *vs.* near; negPolys test; $[t_{int}, t_{max}]$ *vs.* far

Figure A.3: By carefully handling each of the thirteen cases that can arise when ray tracing BSP trees, an optimal algorithm results.

As can be seen from figure A.3, the algorithm is not quite as simple as recently described — there are thirteen possible cases that can arise when comparing a plane with a directed line segment. The implementation described here (figure A.4) is different from Thibault's original algorithm, which worked on an augmented BSP tree [111, 107], where the ray was tested against the objects stored at the leaves, not the polygons stored at each node.

Sung and Shirley [107] who, like Thibault [111] implemented a ray tracer for an augmented BSP tree, have described ray tracing with BSP trees as outperforming 'all of the spacial subdivision approaches we have experienced'.

```
Tboolean ray_int_bsp ( ray, tmin, tmax, tree )
{
   if ( tree == NULL )
      return ( FALSE );

   dmin = signed_dist ( ray, tmin, tree.plane );
   dmax = signed_dist ( ray, tmax, tree.plane );

   if ( dmin > 0 OR (dmin == 0 AND dmax > 0) ) {
      near = tree.pos;
      far = tree.neg;
      candidates = tree.posPolys;
   } else {
      near = tree.neg;
      far = tree.pos;
      candidates = tree.negPolys;
   }

   if ( ray_in_plane ( ray, tree.plane ) )  /* ray lies in node.plane */
      return ( ray_int_bsp ( ray, tmin, tmax, near ) );

   tint = ray_intersect_plane ( ray, tree.plane );

   if ( tmin < tint < tmax ) {   /* cases 5 and 13 */
      if ( ray_int_bsp ( ray, tmin, tint, near ) )
         return ( TRUE );

      if ( ray_vs_polys ( ray, candidates, &hit_pt, &hit_pol) )
         return ( TRUE );

      return ( ray_int_bsp ( ray, tint, tmax, far ) );
   } else if ( tint != tmax )    /* cases 1,2,3,6,7,9,10 and 11 */
      return ( ray_int_bsp ( ray, tmin, tmax, near ) );
   else                          /* cases 4 and 12 */
      if ( ray_int_bsp ( ray, tmin, tmax, near ) )
         return ( TRUE );
      else
         return ( ray_vs_polys( ray, candidates, &hit_pt, &hit_pol) );
}
```

Figure A.4: By comparing each end of the ray segment with the node plane, one only tests the segment against polygons which it can possibly intersect.

A.2 Shadow visibility BSP trees

The uses of BSP trees are not simply restricted to ordering algorithms. Thus far, algorithms have been described which filter points down BSP trees, and another which filters a line segment. This section concerns algorithms which filter polygons down a BSP tree.

In [111], Thibault & Naylor extended BSP trees so that a tree could be used to represent a polyhedron. In their approach, each leaf node is labelled as such; and similarly for each internal node. Each node represents a region of space; if the polygons used to build the tree are the faces of a polyhedron, then some leaf nodes will correspond to regions within the polyhedron, and others will correspond to regions without. By labelling leaf nodes as being either `in` or `out`, a BSP polyhedron representation results.

Thibault & Naylor used their representation to evaluate set operations on polyhedra. Originally, their work combined a B-rep with a BSP tree, but later they extended their work to deal solely with the BSP representation [76]. This extension entailed the development of a general algorithm for merging BSP trees [76] — an algorithm which has since proved most useful to researchers dealing with *shadow volumes* [15, 20].

Shadow volumes, originally introduced by Nishita and Nakamae [78], define regions of space where the view of a particular source is affected by a particular occluder. A *penumbra volume* defines a region where this occlusion may only be partial; an *umbra volume* defines a region where this occlusion is total (figure A.5). Nishita and Nakamae recognised that the surface bounding such a

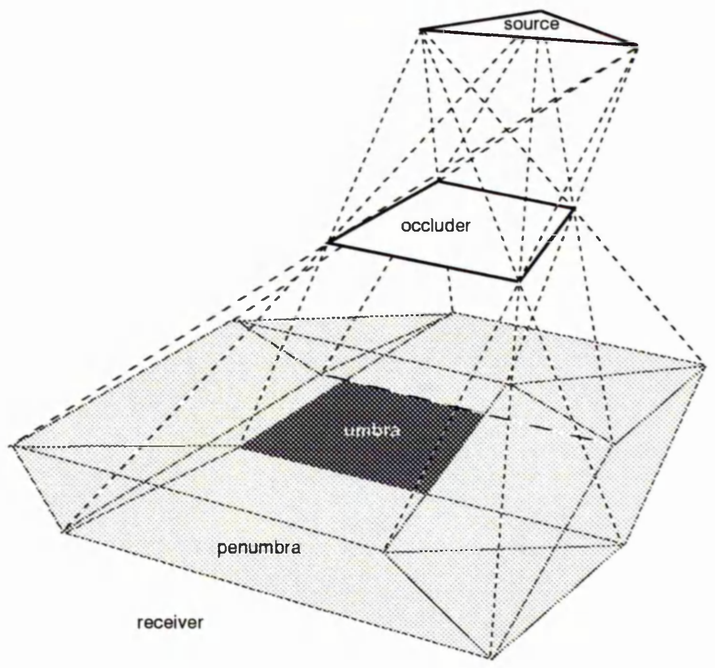


Figure A.5: The penumbra and umbra shadow volumes for a polygonal source and polygonal occluder.

penumbra region, has the source wholly in front of it, and the occluder wholly behind it. The surface bounding an umbra region has both the source and the occluder lying in front of it[3].

When the source and occluder are both polygons, or polyhedra, the penumbra volume is bounded by a set of *planes*, satisfying the in-front/behind condition, and chosen from those which either:

- pass through a source vertex and an occluder edge, or
- pass through a source edge and an occluder vertex;

The umbra volume, in this case, consists of the set of planes (satisfying the in-front/in-front condition) which pass through a source vertex and an occluder

[3]Clearly, the notions of 'in front' and 'behind' are somewhat ambiguous here, but if the side containing the source is *defined* as being the front side, this ambiguity vanishes.

edge. A more rigid analysis of the planes which bound such shadow volumes, is presented in [15].

Whilst shadow volumes were used by Nishita and Nakamae to aid with their shading calculations, it was Chin and Feiner [19] who first exploited their full potential, with the introduction of shadow visibility BSP (SVBSP) trees: Chin and Feiner [19] built an *extended* BSP tree [111, 112], using those planes which bound the umbra volume of a polygonal occluder, due to a point light source. In this way, any point filtered down the tree will end up at a leaf node labelled either in (the point is in shadow) or out (the point is not in shadow).

By filtering a polygon down such a tree — splitting the polygon by each node plane as its recurses, and taking one half down each branch — Chin and Feiner could very quickly establish which parts of the polygon were lit, and which were in shadow. They stored their (polygonal) scene in a BSP tree and incorporated their shadow classification approach into a front-to-back traversal of the tree, from the point source. In this traversal, a *merged* SVBSP tree is carried; representing the combined shadow of all polygons encountered thus far on the traversal (initially a single out node). As each polygon is encountered, it is filtered down the SVBSP tree; its shadowed parts are classified as such, and its lit parts are noted. An SVBSP tree is now built for each lit part, and this is merged into the tree representing the combined shadow, using an algorithm from [111].

SVBSP trees are more efficient than linked lists for comparing a number of planes with a polygon [15]. Also, in Chin and Feiner's approach, the front-to-back traversal ensures that no polygon is tested against the shadow volume of a polygon which lies behind it. Such an approach is generally not possible with area sources, as there is not usually a single ordering of the scene polygons available [20, 37].

SVBSP trees were extended to model the shadow volumes due to *polygonal* light sources by Campbell [15], who combined the shadow volume planes used by Nishita and Nakamae [78, 79] with the data structure introduced by Chin and Feiner [19]. With polygonal sources, each occluder has two SVBSP trees; one representing the penumbra, and one representing the umbra. By testing for penumbra regions *before* umbra regions, one can avoid comparing lit regions with the umbra volume.

Merging SVBSP trees, due to polygonal rather than point sources, is not as simple as the problem solved by Chin and Feiner [19]. Nevertheless, Campbell [15] obtained such a merged shadow volume by utilising another algorithm due to Thibault *et al* [76]. Campbell used the merged SVBSP tree to shadow classify the polygons in the scene, but without the aid of the object space ordering that Chin and Feiner had been able to exploit, for a point source [19].

The most recent development in shadow volume usage, apart from the work described in this thesis, is again due to Chin and Feiner. In their improved algorithm [20], polygonal sources are used, and object space ordering is included. This last point is made possible by storing the scene in a BSP tree, and then splitting the source by all those polygons which lie in front of it, and whose planes pass through it. This results in a number of source *fragments*, from which one can traverse the scene in a unique front-to-back order; finding shadowed regions and building a merged shadow volume as one proceeds. The obvious drawback with such an approach is that one may have to make very many traversals before the scene is shadow classified with respect to the whole source, and not just one of its fragments. The SVBSP trees used are exactly those used by Campbell [15], but the merging process is again the simpler algorithm due to Thibault *et al* [111]; made possible by the object space ordering employed.

Appendix B

Winged Edge Data Structure

The winged-edge data structure (WEDS) is not new to computer graphics. Originally developed by Baumgart [9] in 1975, for an application in computer vision, it has since become popular not only for the representation of polyhedra (for which it was originally intended) but also for a number of other structures where a *consistent* representation of faces, edges, and vertices, is required. One such structure, is the element mesh from the radiosity algorithm described in this thesis.

The central concept of the WEDS, is an edge. An edge stores most of the information which allows one to tour the data structure. Given a face, one can use the information stored in the WEDS to:

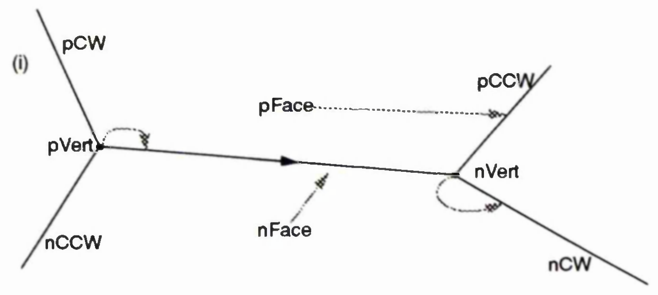
- visit all of edges which bound the face, or
- visit all of the vertices which bound the face, or
- visit all of the faces which share a common edge with the face, or

- visit all the faces which share a common vertex with the face.

Any number of other traversals are also possible.

The WEDS (as implemented by Baumgart [9]) consists of three doubly-linked lists: one storing all the faces, one storing all the edges, and one storing all the vertices. As well as storing all the application-dependent data that they need, each face, edge and vertex in a WEDS stores topological information, which enables the traversal of the data: each vertex, stores a pointer to an edge (of which it is one end); each face stores a pointer to an edge (which lies on its boundary); each edge stores:

- Pointers to the vertices, at its ends. These are ordered; one is the *previous vertex*, the other is the *next vertex*. In this way, each edge is *directed.*
- Pointers to the faces which lie on either side of it. Because the edge is directed, these can be labelled as lying on either the left (previous) or right (next) side of the edge.
- A pointer to the adjacent edge, on the previous face, which is reached by going clockwise around the face, from the edge.
- A pointer to the adjacent edge, on the previous face, which is reached by going counter-clockwise around the face, from the edge.
- A pointer to the adjacent edge, on the next face, which is reached by going clockwise around the face, from the edge.
- A pointer to the adjacent edge, on the next face, which is reached by going counter-clockwise around the face, from the edge.



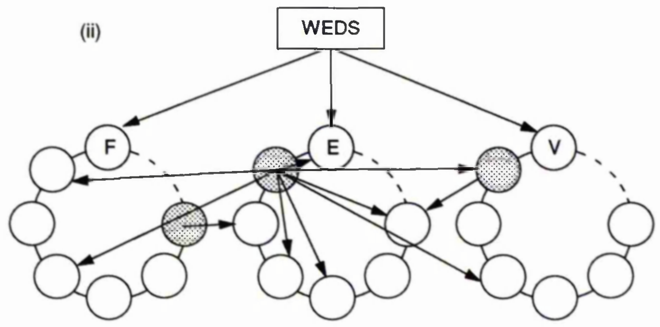


Figure B.1: The figure illustrates (i) An edge, and the information it stores; also shown are the edge pointers stored by vertices and faces. (ii) A symbolic diagram of the whole data structure; consisting of a face, edge and vertex ring. One edge, face and vertex (shaded) are each shown with their 'topology' pointers.

These last four are known as the edge's *wings* (figure B.1).

It is plain to see, that whilst such a structure gives one tremendous scope, when traversing the data stored therein, the storage costs of such a scheme, are considerable.

By using a number of generic routines (for tasks such a splitting an edge by a given vertex, splitting a face by a given edge), the otherwise laborious task of building and maintaining a WEDS, as data is and deleted, becomes tractable [41].

Appendix C

Results

This appendix presents a number of images and graphs which support the original work described in chapter 5. A few of these are repeated elsewhere in the thesis, but this is not true in the main. Any timings given refer to C code running on a clustered HP9000/735crx48z.

C.1 Mesh optimisation statistics

This section presents a series of graphs which relate to the three triangulation schemes described in section 5.2. Two series of five graphs are presented, one series showing how the number of mesh elements varies for the different triangulation schemes, and one showing how the shape of the mesh triangles vary for the different schemes. Each of the five graphs, in a series, corresponds to a different test scene. The test scenes varied considerably in complexity, the five scenes being:

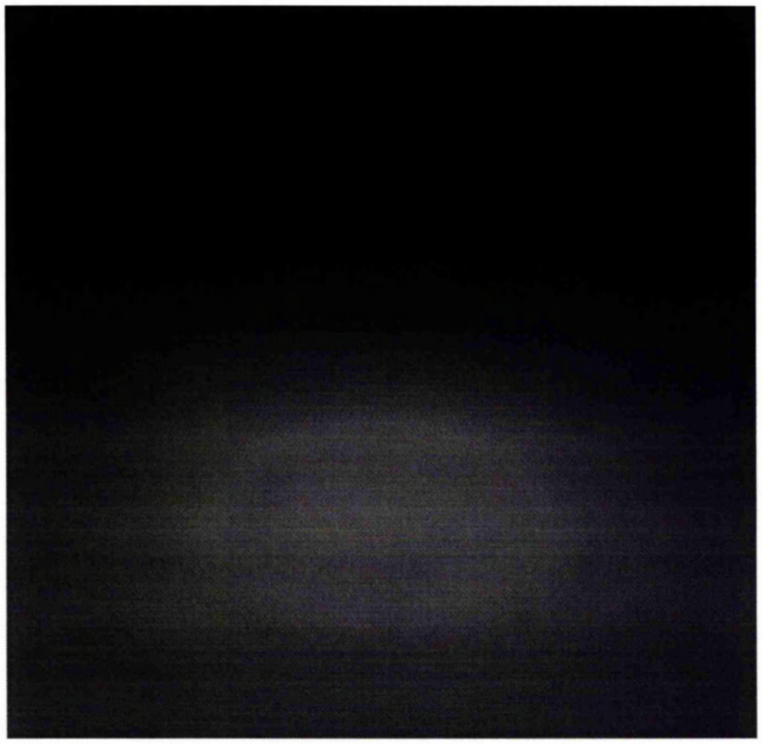


Figure C.1: Test scene `flat`. 2 polygons.

flat: A simple (source, receiver) pair; no occlusion (2 polygons).

boxes: Two boxes, lit by a single light source, both boxes casting a shadow onto the floor, the high box also casting a shadow onto the lower box (14 polygons).

table: A table, lit by a single source, casting a shadow onto the floor (34 polygons).

chair: A chair, lit by a single source, casting a shadow onto the floor (65 polygons).

float: A table and five chairs, lit a single source. Four chairs are at the table, one chair is floating above the table (214 polygons).

Figures C.1 to C.5 show the five test scenes.

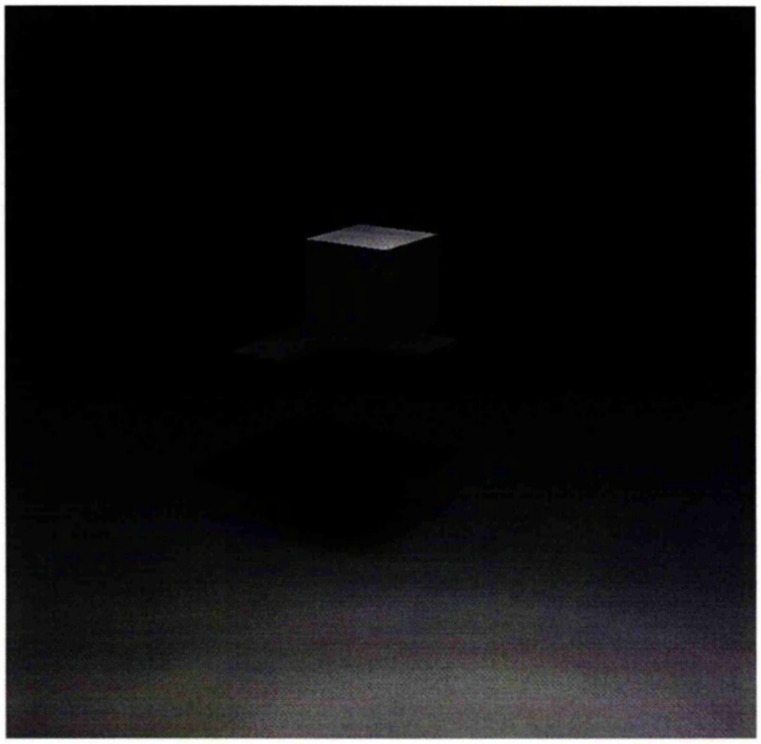


Figure C.2: Test scene `boxes`. 14 polygons.

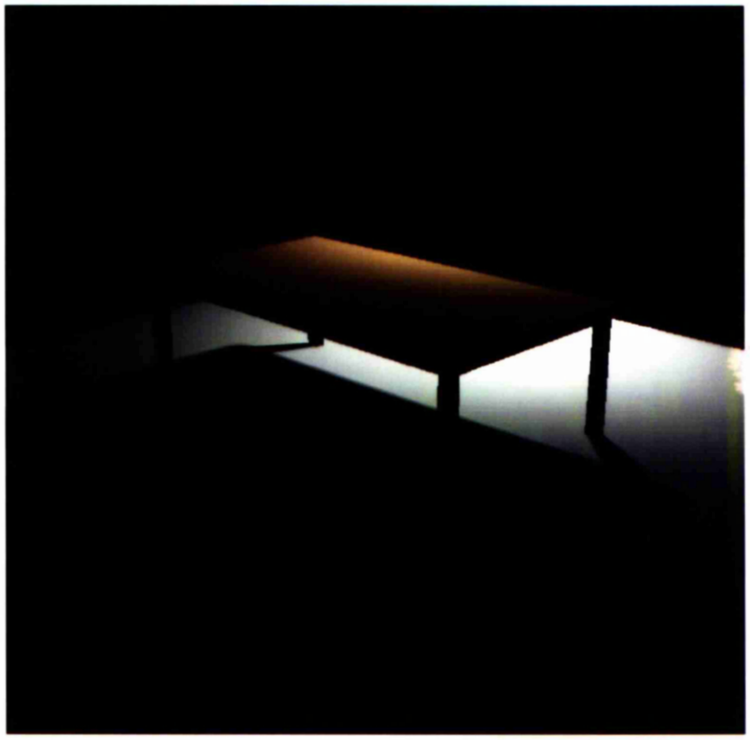


Figure C.3: Test scene `table`. 34 polygons.

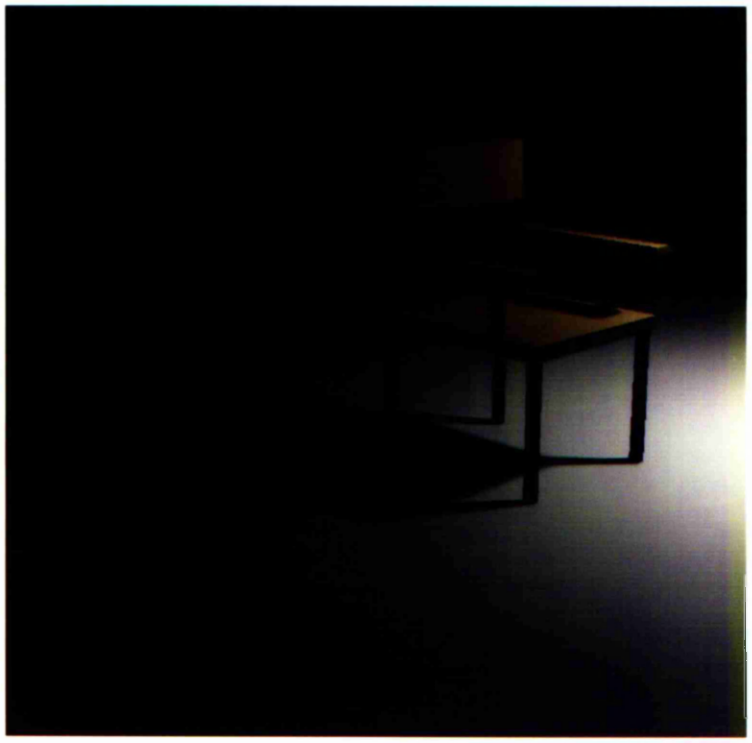


Figure C.4: Test scene **chair**. 65 polygons.

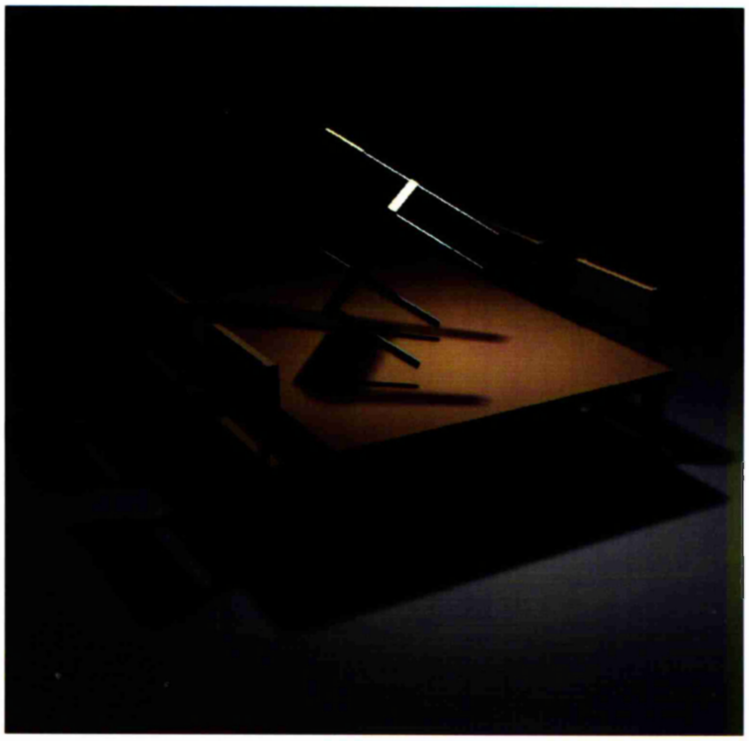


Figure C.5: Test scene **float**. 214 polygons.

Figures C.6 to C.10 show graphs illustrating how the number of mesh triangles varies with user-defined error tolerance, for the different triangulation schemes (section 5.2) running on the five test scenes. Each graph shows three plots: one for each triangulation scheme. Each plot is labelled as being due to either `method 1`, `method 2` or `method 3`; these refer to longest-diagonal, shortest-diagonal and heuristic-diagonal, respectively. All of these graphs, with the exception of figure C.6, show that the two new triangulation schemes result in meshes which satisfy a given error tolerance, using markedly less triangles than the original [70] scheme. The author is content to regard figure C.6 as being anomolous, since it corresponds to a scene which is completely free from occlusion. Whilst the difference between the two new schemes (shortest-diagonal and heuristic-diagonal) is tiny, it seems that shortest-diagonal manages to mesh to a given error tolerance using slightly less triangles than heuristic-diagonal.

Figures C.11 to C.15 show graphs which consider how well-shaped the mesh triangles were, for the different triangulation schemes running on the five test scenes, at a selection of error tolerances (section 5.2.1). Whilst shortest-diagonal and heuristic-diagonal both consistently produce better shaped meshes than longest-diagonal (apart the anomolous test scene `flat`), shortest-diagonal only comes out as a close-run winner.

The tail-off in both graphs involving test scene `float` are due to the large number of subdivisions which were prevented because the element size was too small.

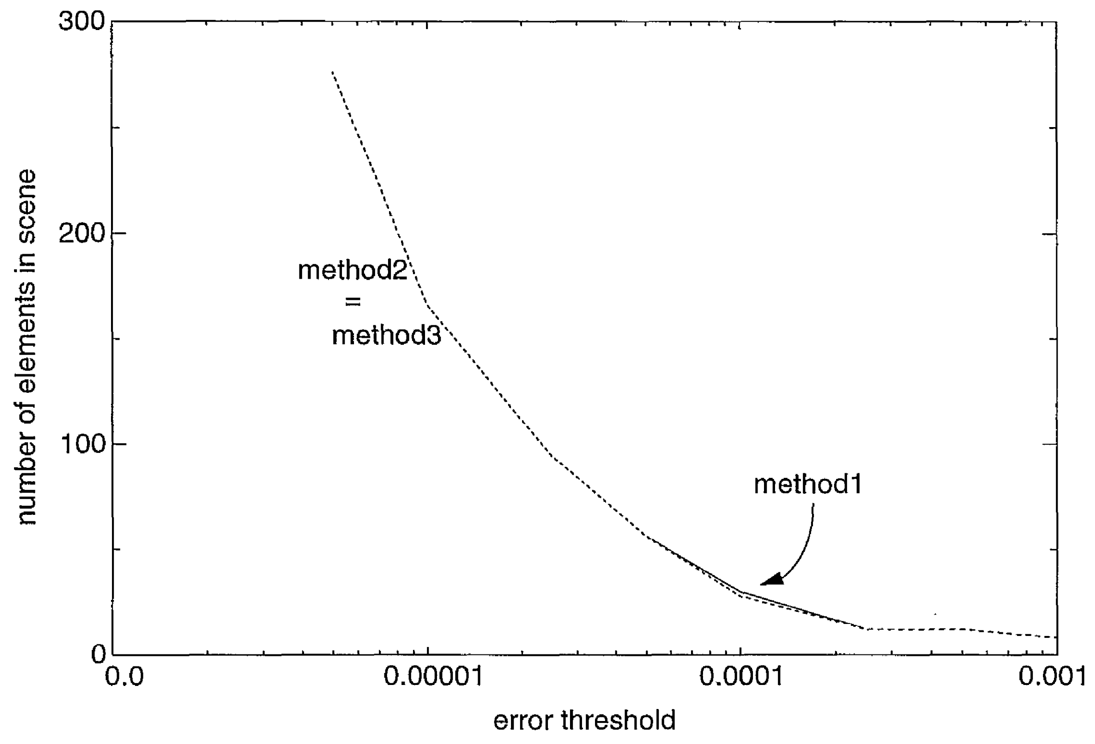


Figure C.6: Comparing the number of mesh elements in each of 3 triangulation schemes (test scene `flat`).

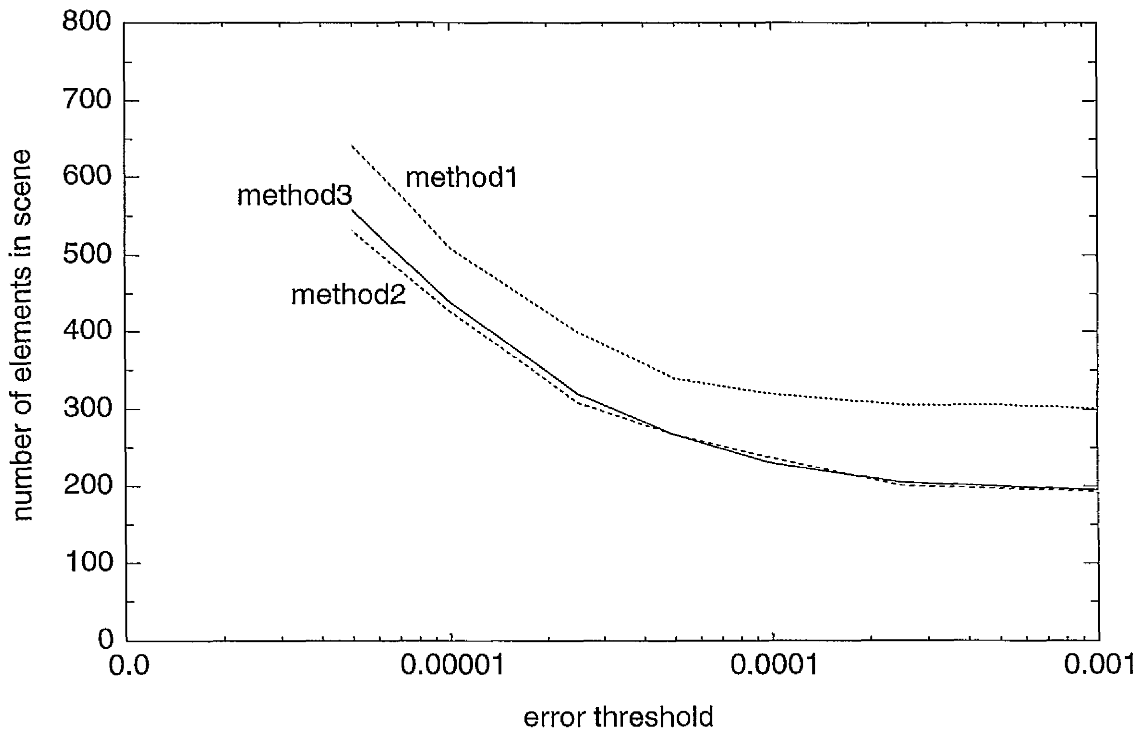


Figure C.7: Comparing the number of mesh elements in each of 3 triangulation schemes (test scene `table`).

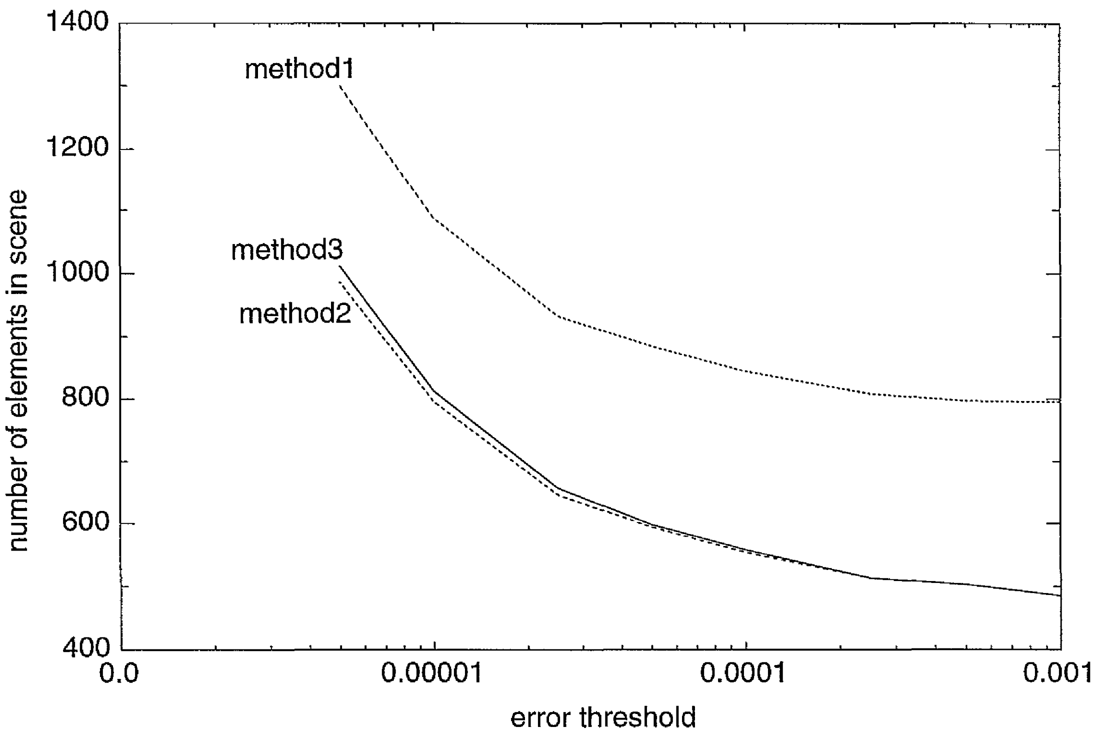


Figure C.8: Comparing the number of mesh elements in each of 3 triangulation schemes (test scene `boxes`).

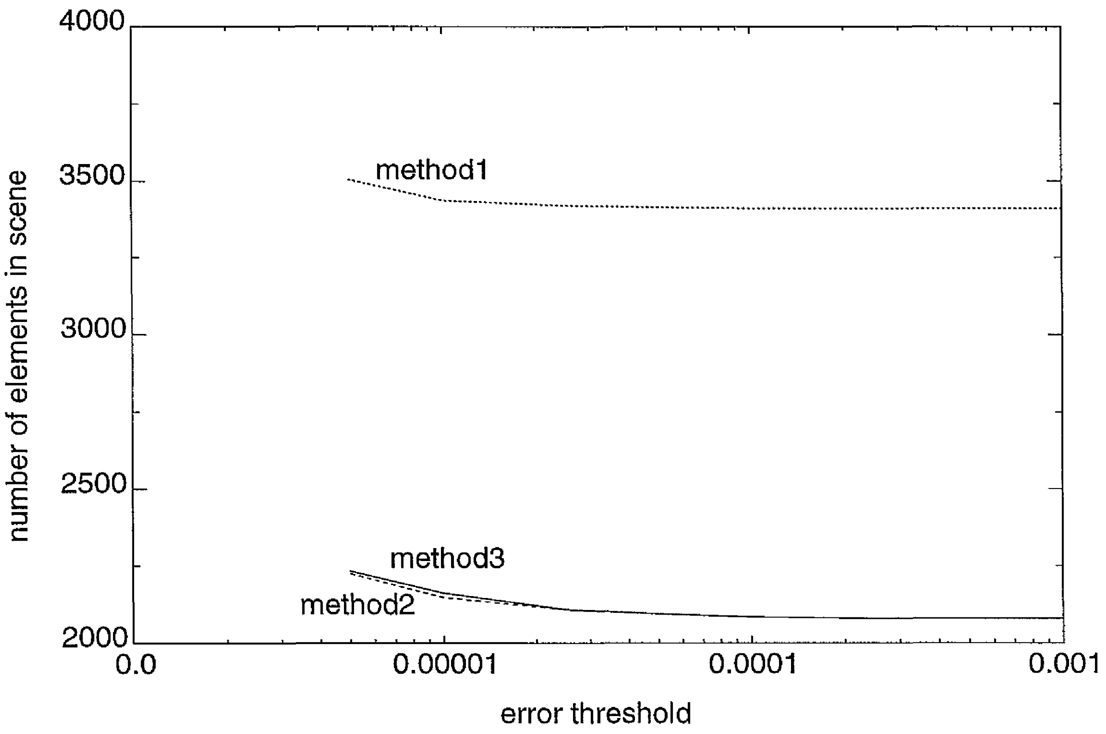


Figure C.9: Comparing the number of mesh elements in each of 3 triangulation schemes (test scene `chair`).

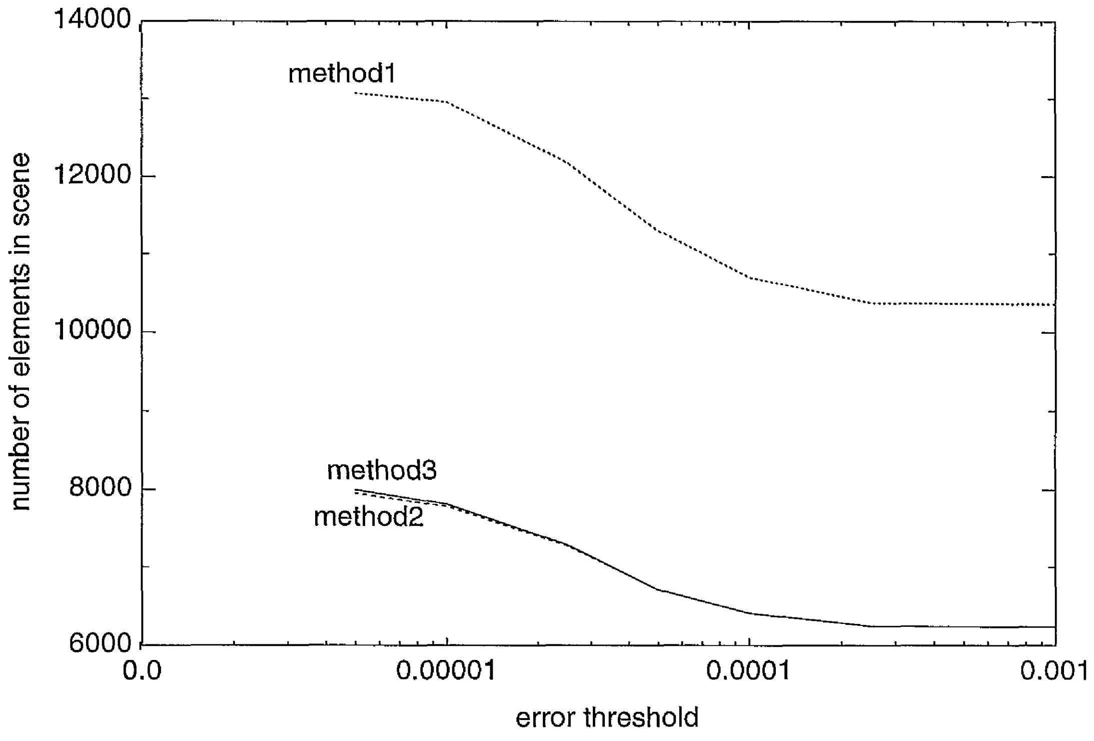


Figure C.10: Comparing the number of mesh elements in each of 3 triangulation schemes (test scene `float`).

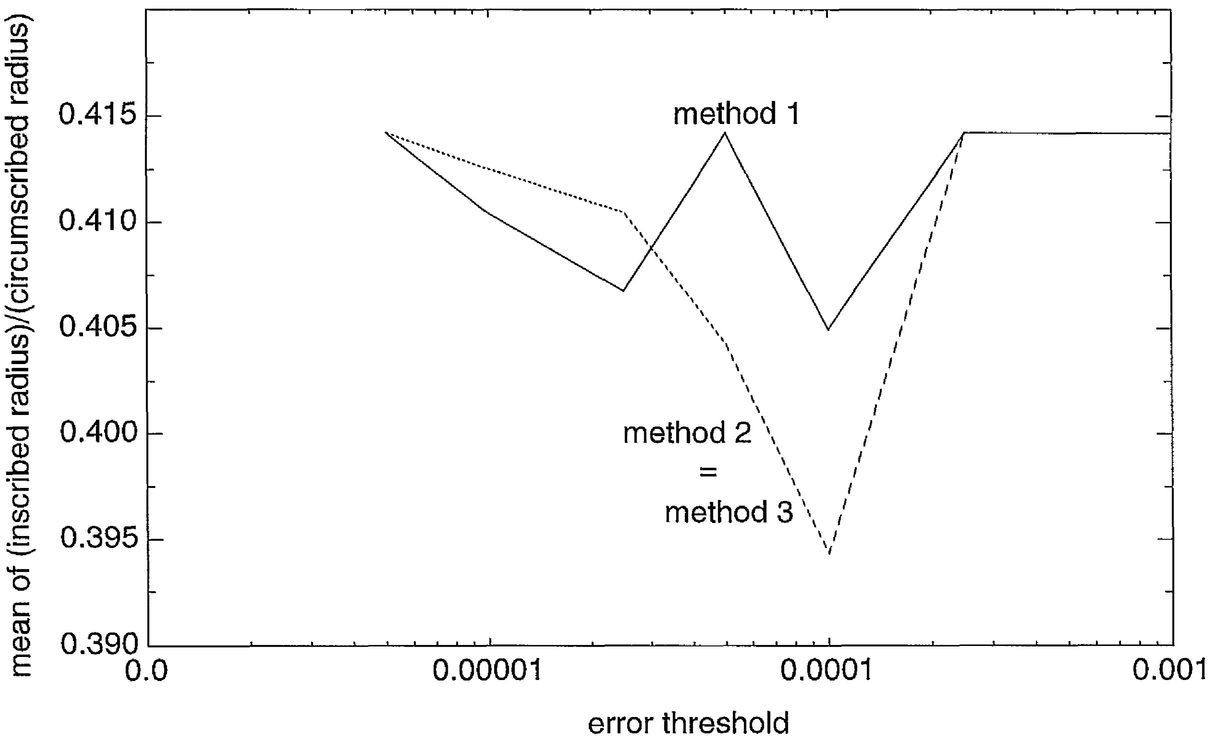


Figure C.11: Comparing the mean inscribed/circumscribed ratio for the different triangulation schemes (test scene `flat`).

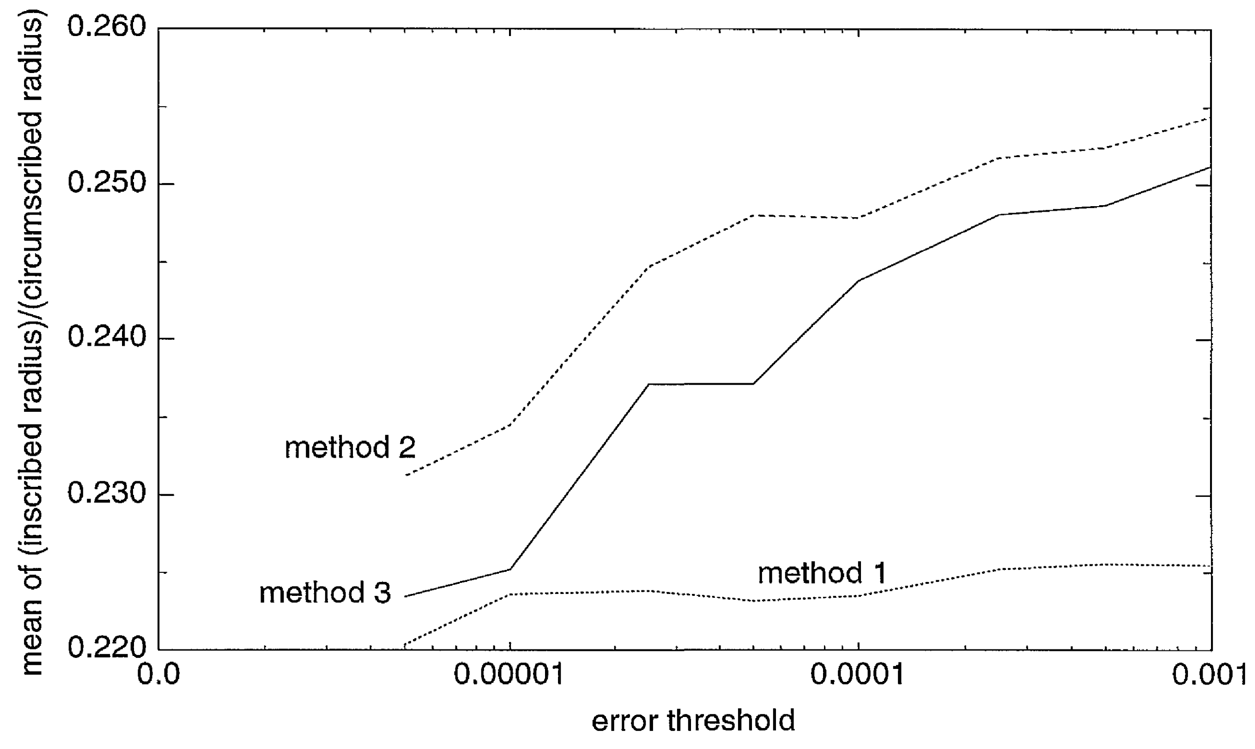


Figure C.12: Comparing the mean inscribed/circumscribed ratio for the different triangulation schemes (test scene `boxes`).

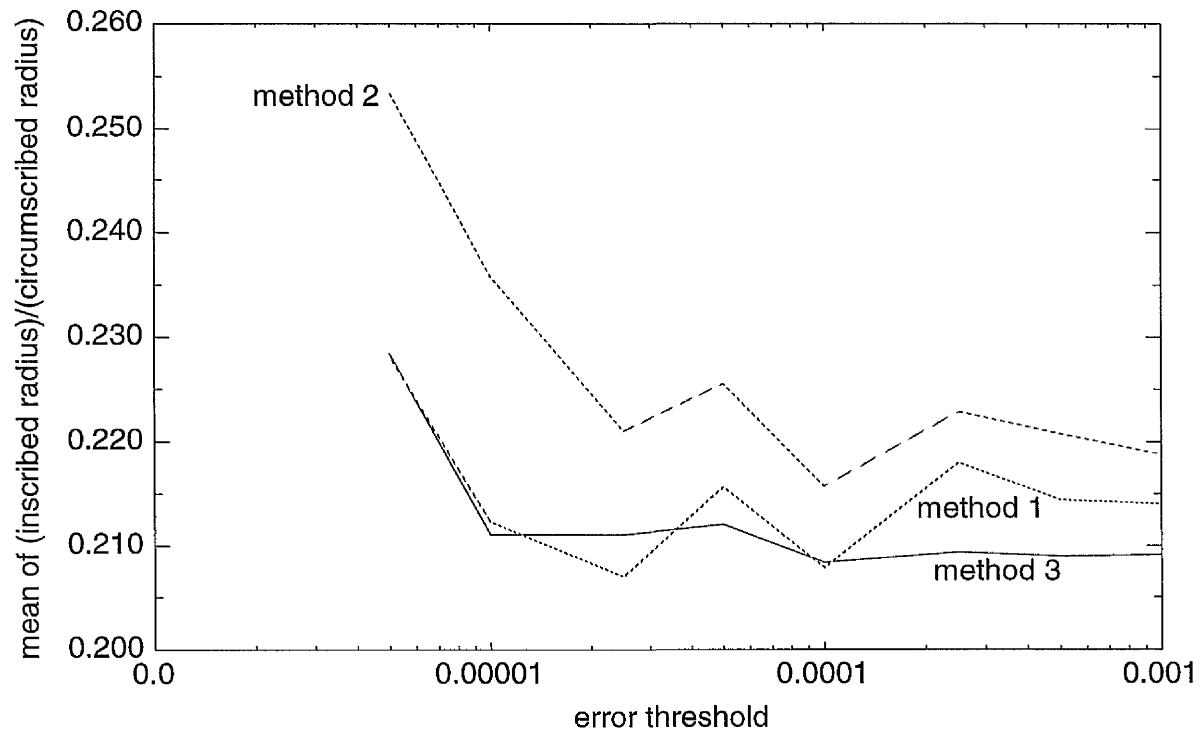


Figure C.13: Comparing the mean inscribed/circumscribed ratio for the different triangulation schemes (test scene `table`).

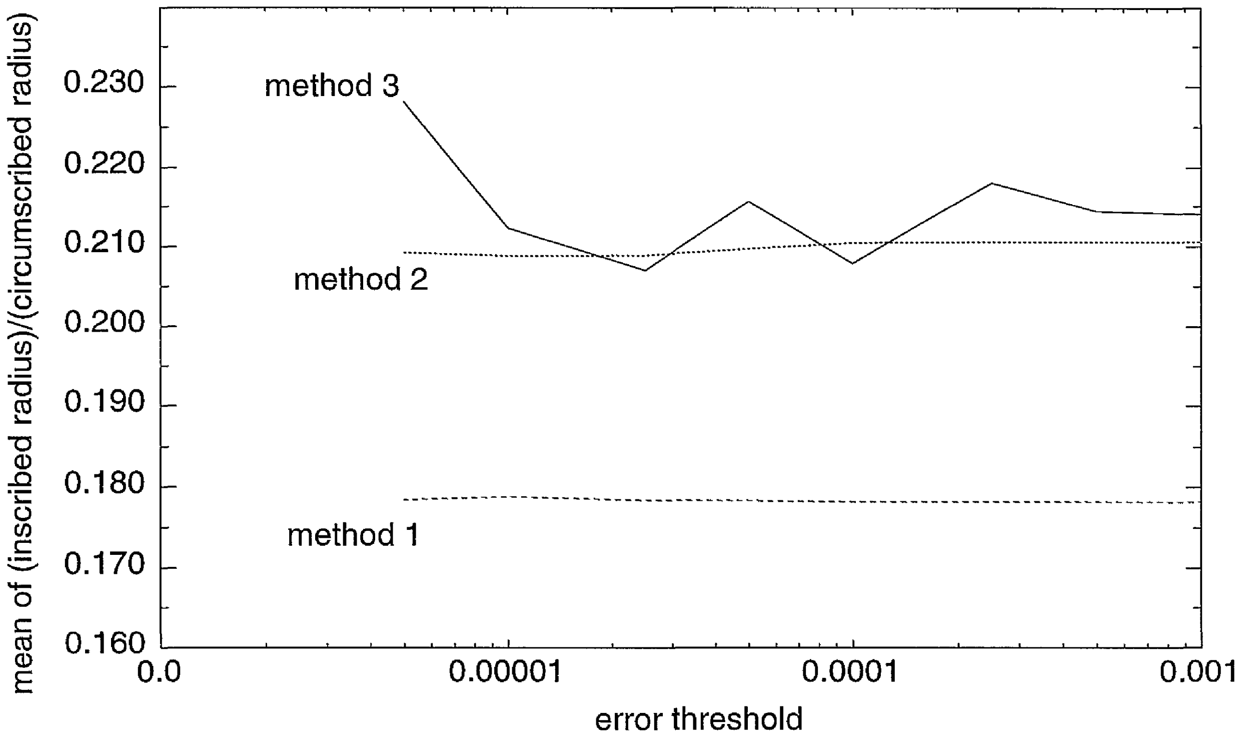


Figure C.14: Comparing the mean inscribed/circumscribed ratio for the different triangulation schemes (test scene `chair`).

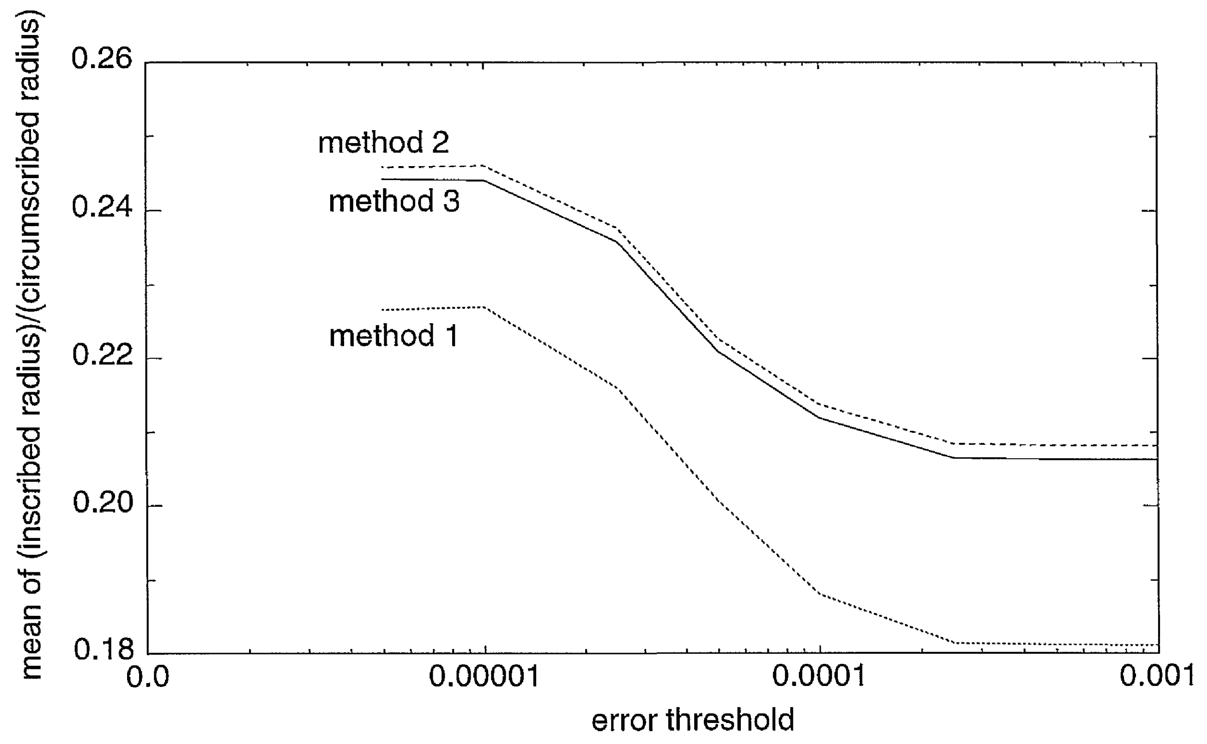


Figure C.15: Comparing the mean inscribed/circumscribed ratio for the different triangulation schemes (test scene `float`).

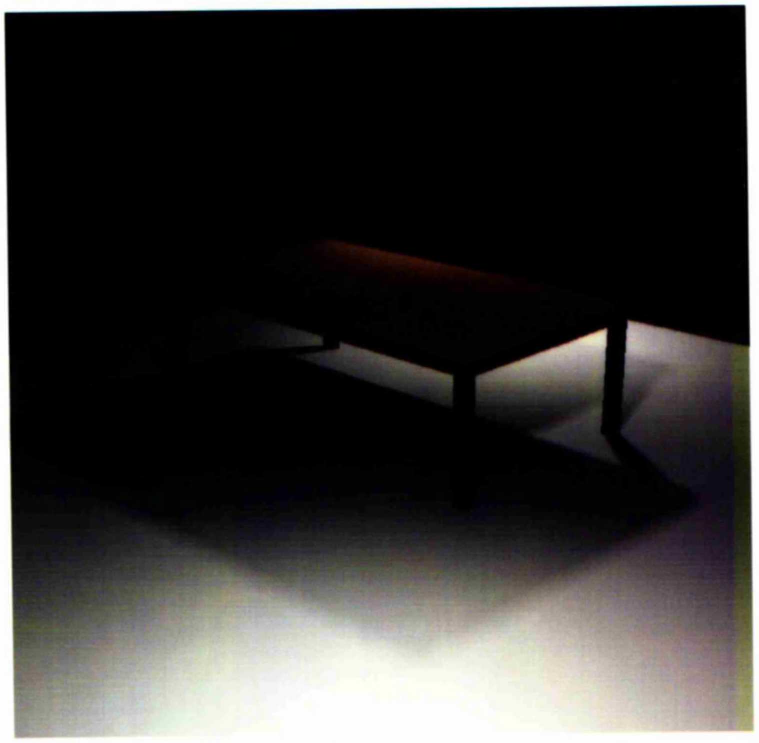


Figure C.16: Two light sources, test scene `table`; radiosity solution: 1 second; ray-traced rendering: 175 seconds.

C.2 Mesh layering

This section presents a number of images corresponding to test scenes lit by multiple light sources.

Figures C.16 and C.17 show the `table` test scene lit by two and three polygonal sources, respectively.

Figures C.18 and C.19 show the `chair` test scene lit by two and four polygonal sources, respectively.

Figures C.20 and C.20 show the `float` test scene lit by two and three polygonal sources, respectively.

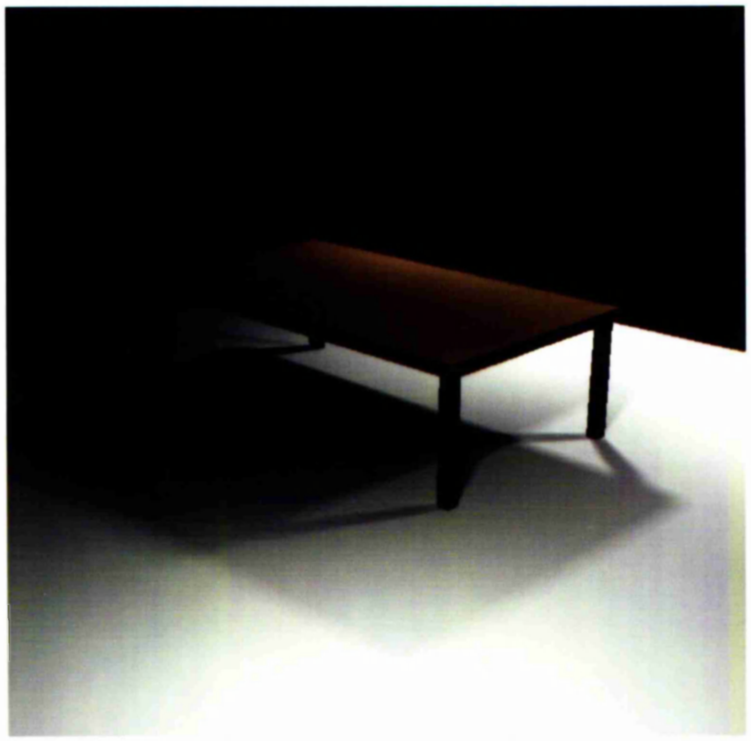


Figure C.17: Three light sources, test scene `table`; radiosity solution: 2 seconds; ray-traced rendering: 162 seconds.

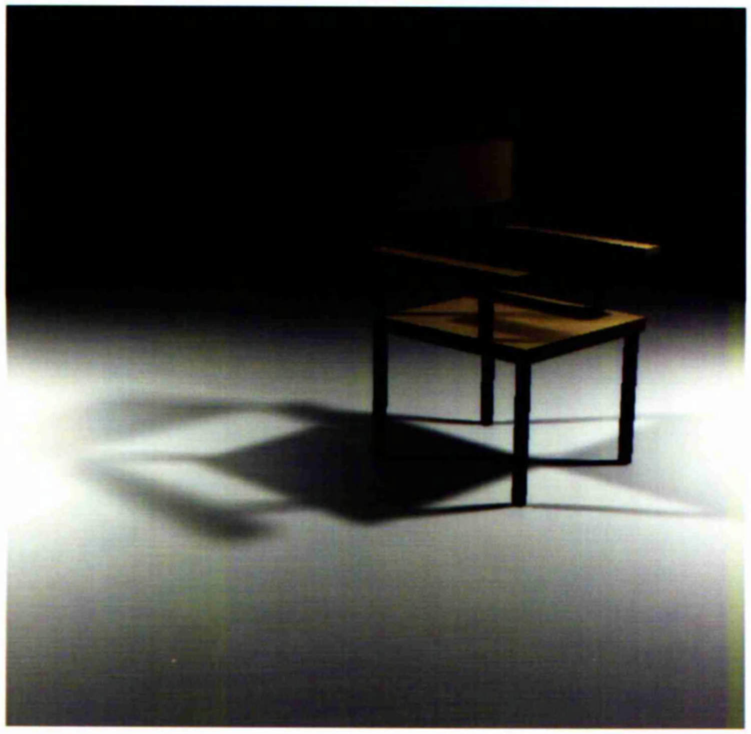


Figure C.18: Two light sources, test scene `chair`; radiosity solution: 14 seconds; ray-traced rendering: 140 seconds.

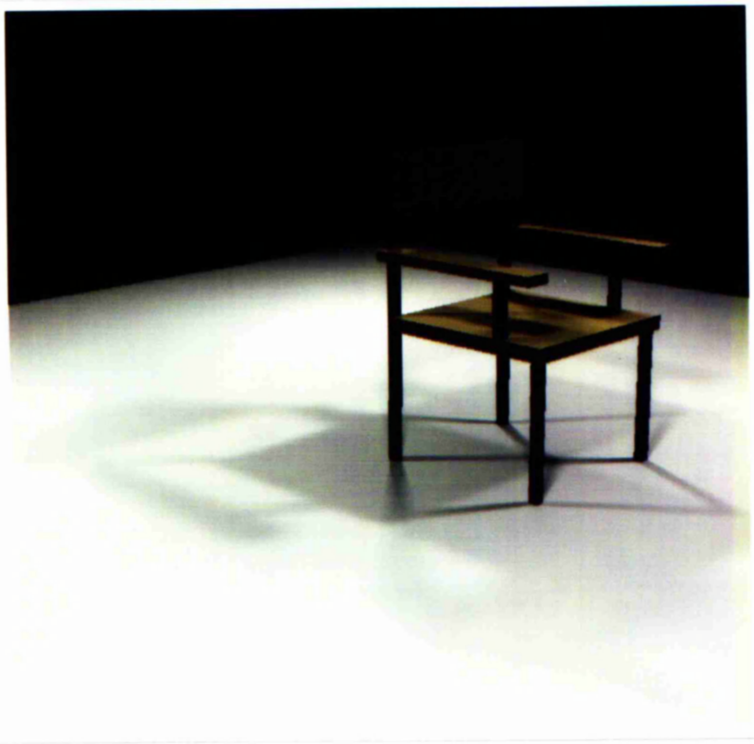


Figure C.19: Four light sources, test scene `chair`; radiosity solution: 29 seconds; ray-traced rendering: 113 seconds.

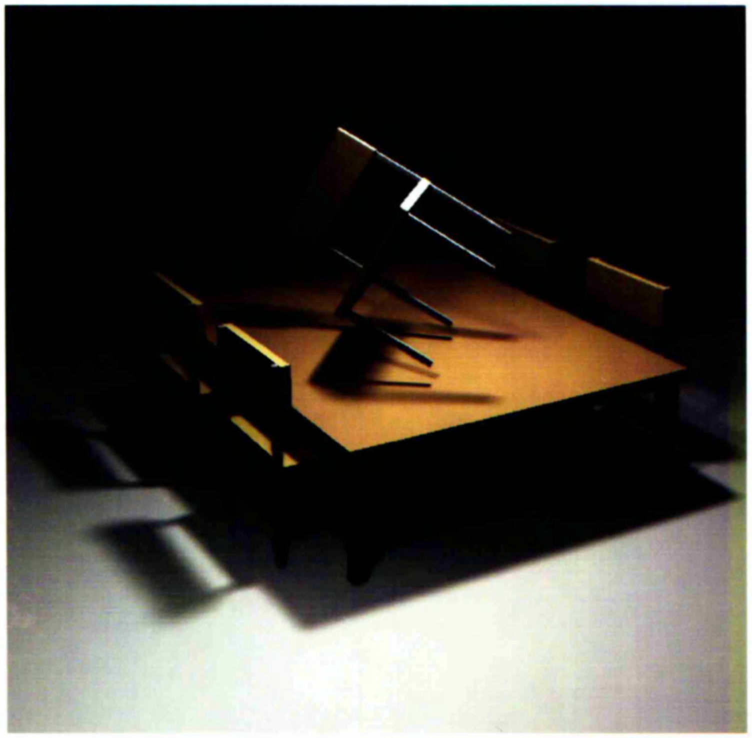


Figure C.20: Two light sources, test scene `float`; radiosity solution: 116 seconds; ray-traced rendering: 185 seconds.

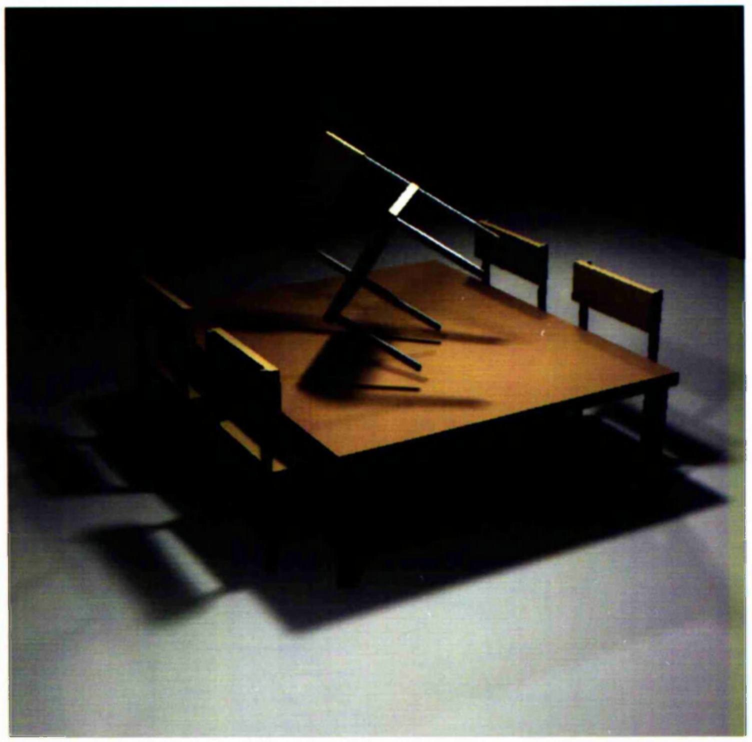


Figure C.21: Three light sources, test scene `float`; radiosity solution: 138 seconds; ray-traced rendering: 174 seconds.

C.3 Shadow classification

Three scenes are presented which illustrate various stages of the solution process. A test scene (spiral staircase) is shown (figure C.22) flat-shaded, with dark discontinuity lines and light construction lines. The same figure is shown after shadow classification (figure C.23), with the shadow regions shown darker. Finally, a ray-traced version is shown (figure C.24).

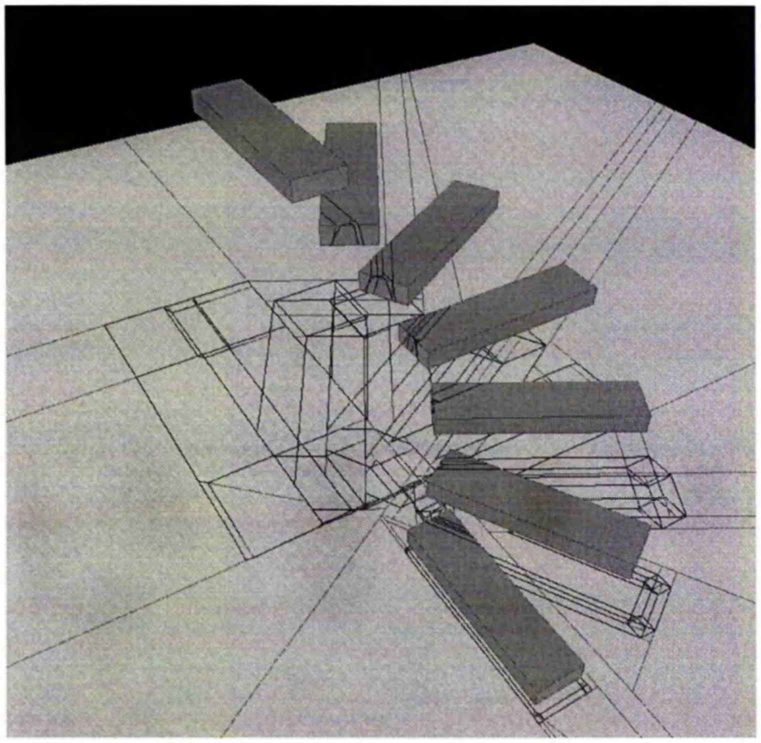


Figure C.22: A test is shown with discontinuity and construction lines only.

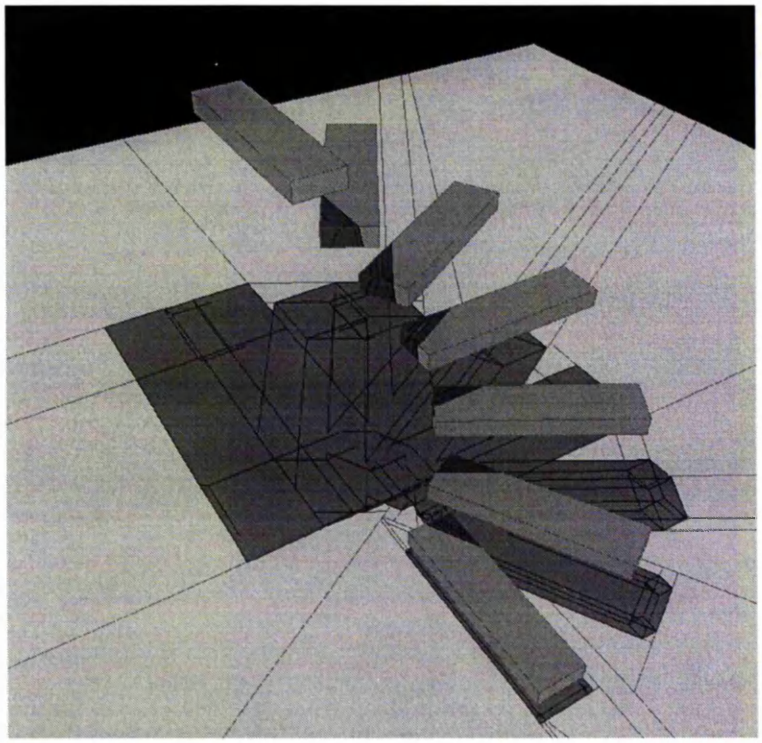


Figure C.23: A test is shown with discontinuity and construction lines, and shadow-classified mesh elements.

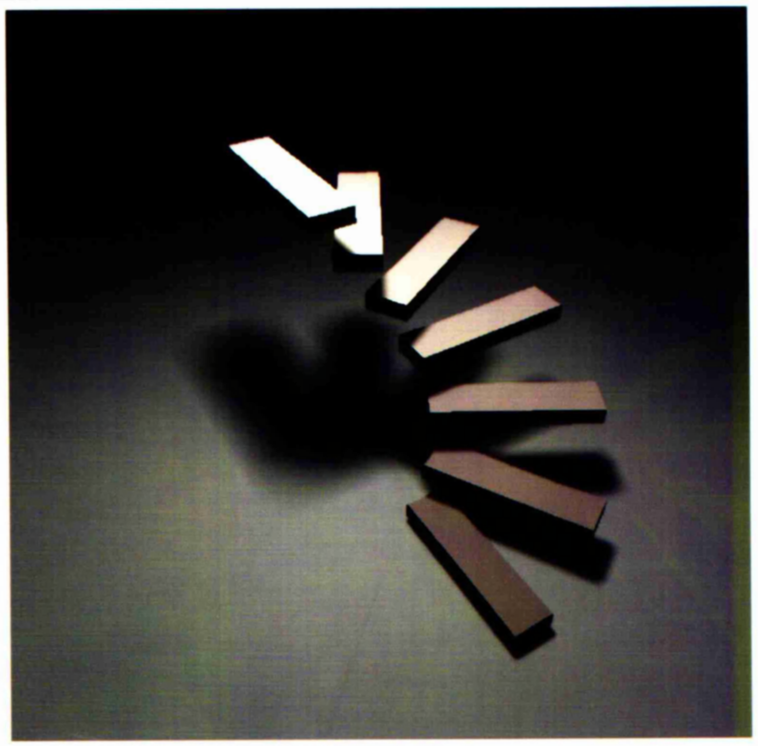


Figure C.24: The final version of the spiral staircase, ray traced.

Bibliography

[1] B. Alpert. A Class of Bases in $\mathcal{L}^2$ for the Sparse Representation of Integral Operators. *SIAM Journal on Mathematical Analysis*, 24(1), January 1993.

[2] B. Alpert, G. Beylkin, R. Coifman, and V. Rokhlin. Wavelet-like Bases for the fast Solution of Second-kind Integral Equations. *SIAM Journal on Scientific Computing*, 14(1):159–184, January 1993.

[3] A. Appel. Some techniques for machine rendering of solids. *AFIPS Joint Computer Conference Proceedings*, 32:37–45, Spring 1968.

[4] James Arvo. Backward Ray Tracing. *Developments in Ray Tracing*, 12:259–263, August 1986. ACM SIGGRAPH course notes.

[5] James Arvo, Kenneth Torrance, and Brian Smits. A Framework for the Analysis of Error in Global Illumination Algorithms. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 75–84. ACM SIGGRAPH, New York, July 1994.

[6] Kendall E. Atkinson. *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind.* Society for Industrial and Applied Mathematics, 33 South 17th Street, Philadelphia, Penn. 19103, 1976.

[7] Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James M. Winget. Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accuarate Radiosity Solutions. *ACM Computer Graphics*, 25(4):51–59, July 1991.

[8] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving Radiosity Solutions through the Use of Analytically Determined Form Factors. *ACM Computer Graphics*, 23(3):325–334, July 1989.

[9] Bruce G. Baumgart. A Polyhedron Representation for Computer Vision. *AFIPS Proceedings*, 44:589–596, May 1975.

[10] H. E. Bennett. Reflection. In Robert M. Besançon, editor, *The Encyclopedia of Physics*, page 1052. Van Nostrand Reinhold Company Inc., 3rd edition, 1985.

[11] G. Beylkin, R. Coifman, and V. Rokhlin. Fast Wavelet Transforms and Numerical Algorithms I. *Communications on Pure and Applied Mathematics*, 44(2):141–183, March 1991.

[12] James F. Blinn. Models of Light Reflection for Computer Sythesized Pictures. *ACM Computer Graphics*, 11(2):192–198, Summer 1977.

[13] David A. Burgoon. Global Illumination Modelling using Radiosity. *Hewlett Packard Journal*, 40(6):78–88, December 1989.

[14] A. T. Campbell, III and Donald S Fussell. Adaptive Mesh Generation for Global Diffuse Illumination. *ACM Computer Graphics*, 24(3):155–164, August 1990.

[15] A. T. Campbell, III and Donald S. Fussell. Analytic Illumination with Polygonal Light Sources. Technical Report TR-91-15, Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712-1188, April 1991.

[16] Sudeb Chattopadhyay and Akira Fujimoto. Bi-Directional Ray Tracing. In T. Kunii, editor, *Computer Graphics 1987 – Proceedings of CG International.* Springer-Verlag, 1987.

[17] Shenchang Eric Chen, Holly E. Rushmeier, Gavin Miller, and Douglas Turner. A Progressive Multi-Pass Method for Global Illumination. *ACM Computer Graphics*, 25(4):165–174, July 1991.

[18] L. Paul Chew. Constrained Delaunay Triangulations. *Algorithmica*, 4:97–108, 1989.

[19] Norman Chin and Steven Feiner. Near Real-Time Shadow Generation Using BSP Trees. *ACM Computer Graphics*, 23(3):99–106, July 1989.

[20] Norman Chin and Steven Feiner. Fast Object-Precision Shadow Generation for Area Light Sources Using BSP Trees. *ACM Computer Graphics*, 26:21–30, May 1992. Special Issue on *1992 Symposium on Interactive 3D Graphics*.

[21] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A Progressive Refinement Approach to Fast Radiosity Image Generation. *ACM Computer Graphics*, 22(4):75–84, August 1988.

[22] Michael F. Cohen and Donald P. Greenberg. The Hemi-cube: A Radiosity Solution for Complex Environments. *ACM Computer Graphics*, 19(3):31–40, July 1985.

[23] Michael F. Cohen, Donald P. Greenberg, David S. Immel, and Philip J. Brock. An Efficient Radiosity Approach for Realistic Image Synthesis. *IEEE Computer Graphics & Applications*, 6(2):26–35, March 1986.

[24] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press Professional, 1993.

[25] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. *ACM Computer Graphics*, 18(3):137–145, July 1984.

[26] Robert L. Cook and Kenneth E. Torrance. A Reflectance Model for Computer Graphics. *ACM Computer Graphics*, 15(3):307–316, August 1981.

[27] John Cunningham. *Vectors.* Heinemann Educational Books Ltd, 1969.

[28] L. M. Delves and J. L. Mohamed. *Computational Methods for Integral Equations.* Cambridge University Press, 1985.

[29] Mark A. Z. Dippé and Erling Henry Wold. Antialiasing Through Stochastic Sampling. *ACM Computer Graphics*, 19(3):69–78, July 1985.

[30] George Drettakis and Eugene Fiume. A Fast Shadow Algorithm for Area Light Sources Using Backprojection. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 223–230. ACM SIGGRAPH, New York, July 1994.

[31] Georgios Drettakis. *Structured Sampling and Reconstruction of Illumination for Image Synthesis.* PhD thesis, University of Tornoto, 1994.

[32] Bruce A. Finlayson. *The Method of Weighted Residuals and Variational Principles.* Academic Press, 1972.

[33] Carl-Erik Froberg. *Introduction to NUMERICAL ANALYSIS.* Addison-Wesley Publishing Company, 2nd edition, 1972.

[34] Henry Fuchs, Gregory D. Abram, and Eric D. Grant. Near Real-Time Shaded Display of Rigid Objects. *ACM Computer Graphics*, 17(3):65–72, 1983.

[35] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On Visible Surface Generation by *a priori* Tree Structures. *ACM Computer Graphics*, 14(3):124–133, 1980.

[36] Neil Gatenby. Radiosity in Computer Graphics: A Proposed Alternative to the Hemi-cube Algorithm. Master's thesis, University of Manchester, 1991.

[37] Neil Gatenby and W. T. Hewitt. Optimizing Discontinuity Meshing Radiosity. In *Proceedings of the Fifth Eurographics Workshop on Rendering, Darmsadt, Germany*, June 1994.

[38] Neil Gatenby and W. Terry Hewitt. Radiosity in Computer Graphics: A Proposed Alternative to the Hemi-cube Algorithm. In *Proceedings of the Second Eurographics Workshop on Rendering*, May 1991.

[39] Ziv Gigus and Jitendra Malik. Computing the Aspect Graph for Line Drawings of Polyhedral Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–122, February 1990.

[40] Andrew S. Glassner. Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics & Applications*, 4(10):15–22, October 1984.

[41] Andrew S. Glassner. Maintaining Winged-Edge Models. In *Graphics Gems II*, pages 191–201. Academic Press, 1991.

[42] Jeffrey Goldsmith and John Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics & Applications*, 7(5):14–20, May 1987.

[43] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennet Battaile. Modelling the Interaction of Light Between Diffuse Surfaces. *ACM Computer Graphics*, 18(3):213–222, July 1984.

[44] Dan Gordon and Shuhong Chen. Front-to-Back Display of BSP Trees. *IEEE Computer Graphics & Applications*, 11(5):79–85, September 1991.

[45] Steven J. Gortler, Peter Schröder, Michael F. Cohen, and Pat Hanrahan. Wavelet Radiosity. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 221–230. ACM SIGGRAPH, New York, August 1993.

[46] Henri Gouraud. Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers*, 20(6):623–629, June 1971.

[47] Eric A. Haines and John R. Wallace. Shaft Culling for Efficient Ray-Traced Radiosity. In *Proceedings of the Second Eurographics Workshop on Rendering*, May 1991.

[48] Roy Hall. *Illumination and Color in Computer Generated Images*. Springer-Verlag, 1989.

[49] Pat Hanrahan and David Salzman. A Rapid Hierarchical Radiosity Algorithm for Unoccluded Environments. In *Proceedings of the First Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, May 1990.

[50] Pat Hanrahan, David Salzman, and Larry Aupperle. A Rapid Hierarchical Radiosity Algorithm. *ACM Computer Graphics*, 25(4):197–206, August 1991.

[51] Xiao D. He, Kenneth E. Torrance, François Sillion, and Donald P Greenberg. A Comprehensive Physical Model for Light Reflection. Technical report, Program of Computer Graphics, Cornell University, Ithica, NY 14853, 1991.

[52] Paul Heckbert. Adaptive Radiosity Textures for Bidirectional Ray Tracing. *ACM Computer Graphics*, 24(4):145–154, August 1990.

[53] Paul Heckbert. Discontinuity Meshing for Radiosity. In *Proceedings of the 3rd Eurographics Workshop on Rendering*, May 1992.

[54] Paul S. Heckbert. *Simulating Global Illumination Using Adaptive Meshing*. PhD thesis, University of California, Berkeley, June 1991. Dept. of Electrical Engineering and Computer Sciences.

[55] Paul S. Heckbert and James M. Winget. Finite Element Methods for Global Illumination. Technical Report UCB/CSD 91/643, Computer Science Division (EECS), University of California, Berkeley, California 94720, July 1991.

[56] F. B. Hildebrand. *Methods of Applied Mathematics.* Prentice-Hall, Inc., 1952.

[57] E. Hinton and D. R. J. Owen. *An Introduction to Finite Element Methods.* Pineridge Press Limited, 1979.

[58] Hoyt C. Hottel and Adel F. Sarofim. *Radiative Transfer.* McGraw-Hill Book Company, 1967.

[59] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A Radiosity Method for Non-Diffuse Environments. *ACM Computer Graphics*, 20(4):133–142, August 1986.

[60] James T. Kajiya. Anisotropic Reflection Models. *ACM Computer Graphics*, 19(3):15–21, July 1985.

[61] James T. Kajiya. The Rendering Equation. *ACM Computer Graphics*, 20(4):143–150, August 1986.

[62] Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*, volume I: Basics. John Wiley & Sons, 1986.

[63] Arjan J. F. Kok and Frederik W. Jansen. Source Selection for the Direct Lighting Computation in Global Illumination. In *Proceedings of the Second Eurographics Workshop on Rendering*, May 1991.

[64] G. E. V. Lambert. Cosine Law of Diffusing Surface. In J. Thewlis, editor, *Encyclopaedic Dictionary of Physics*, pages 123–124. Pergamon Press Ltd., 1961.

[65] Walter Ledermann. *Multiple Integrals.* Routledge & Kegan Paul, 1975.

[66] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically Optimized Sampling for Distributed Ray Tracing. *ACM Computer Graphics*, 19(3):61–67, July 1985.

[67] John R. Levine, Tony Mason, and Doug Brown. *Lex & Yacc*. O'Reilly & Associates, Inc., 2nd edition, 1992.

[68] Robert R. Lewis. Making Shaders More Physically Plausible. In *Proceedings of the Fourth Eurographics Workshop on Rendering*, June 1993.

[69] Dani Lischinski, Brian Smits, and Donald P. Greenberg. Bounds and Error Estimates for Radiosity. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 67–74. ACM SIGGRAPH, New York, July 1994.

[70] Dani Lischinski, Filipo Tampieri, and Donald P. Greenberg. Discontinuity Meshing for Accurate Radiosity. *IEEE Computer Graphics & Applications*, 12(6):25–39, November 1992.

[71] Dani Lischinski, Filipo Tampieri, and Donald P. Greenberg. Combining Hierarchical Radiosity and Discontinuity Meshing. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 199–208. ACM SIGGRAPH, New York, August 1993.

[72] Thomas J. V. Malley. A Shading Method for Computer Generated Images. Master's thesis, University of Utah, 1988.

[73] Nelson L. Max and Michael J. Allison. Linear Radiosity Approximation Using Vertex-to-Vertex Form Factors. In *Graphics Gems III*, pages 318–323. Academic Press, 1992.

[74] Gary W. Meyer. Wavelength Selection for Synthetic Image Generation. *Computer Vision, Graphics, and Image Processing*, 41:57–79, 1988.

[75] Don P. Mitchell. Generating Antialiased Images at Low Sampling Densities. *ACM Computer Graphics*, 21(4):65–72, July 1987.

[76] Bruce Naylor, John Amanatides, and William Thibault. Merging BSP Trees Yields Polyhedral Set Operations. *ACM Computer Graphics*, 24(4):115–124, August 1990.

[77] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Geometrical Considerations and Nomenclature for Reflectance., October 1977. US National Bureau of Standards Monograph 160.

[78] Tomoyuki Nishita and Eihachiro Nakamae. Half-tone Representation of 3-d Objects Illuminated by Area Sources or Polyhedron Sources. In *IEEE COMPSAC*, pages 237–242, November 1983.

[79] Tomoyuki Nishita and Eihachiro Nakamae. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. *ACM Computer Graphics*, 19(3):23–30, July 1985.

[80] Tomoyuki Nishita, Isao Okamura, and Eihachiro Nakamae. Shading Models for Point and Linear Sources. *ACM Transactions on Graphics*, 4(2):124–146, April 1985.

[81] M. Y. Numan and G. R. Moore. Form Factors: The Problem of Partial Obstruction. Technical report, The Martin Centre for Architectural and

Urban Studies, University of Cambridge Department of Architecture, 6 Chaucer Rd, Cambridge, CB2 2EB, England, November 1982.

[82] J. T. Oden and J. N. Reddy. *An Introduction to the Mathematical Theory of Finite Elements.* John Wiley and Sons, 1976.

[83] James Painter and Kenneth Sloan. Antialiased Ray Tracing by Adaptive Progressive Refinement. *ACM Computer Graphics*, 23(3):281–288, July 1989.

[84] Sumant N. Pattanaik. *Computational Methods for Global Illumination and Visualisation of Complex 3D Environments.* PhD thesis, Birlani Institute of Technology and Science, Pilani, Juhu, Bombay, India, 1993.

[85] Sumant N. Pattanaik and Kadi Bouatouch. Haar Wavelet: A Solution to Global Illumination with General Surface Properties. In *Proceedings of the 5th Eurographics Workshop on Rendering*, June 1994. Darmstadt, Germany.

[86] Bui Tuong Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[87] Werner Purgathofer. A Statistical Method for Adaptive Stochastic Sampling. In *Proceedings of the European Computer Graphics Conference and Exhibition.* Eurographics Association, August 1986.

[88] María-Cecilia Rivara. Numerical Generation of Nested Series of General Triangular Grids. In C. K. Chui, L. L. Schumaker, and F. I. Utreras, editors, *Topics in Multivariate Approximation*, pages 193–206. Academisc Press, Orlando, Florida, 1987.

[89] Suggested Units, Symbols and Defining Equations for Computer Graphics Global Illumination Based on ANSI/IES RP-16-1986, September 1992. Compiled by Holly Rushmeier, distributed through globillum@cs.cmu.edu (mailing list).

[90] David Salesin, Dani Lischinski, and Tony DeRose. Reconstructing Illumination Functions with Selected Discontinuities. In *Proceedings of the 3rd Eurographics Workshop on Rendering*, pages 99–112, May 1992. Bristol, England.

[91] Christophe Schlick. A Customizable Reflectance Model for Everyday Rendering. In *Proceedings of the Fourth Eurographics Workshop on Rendering*, June 1993.

[92] Peter Schröder, Steven J. Gortler, Michael F. Cohen, and Pat Hanrahan. Wavelet Projections for Radiosity. In *Proceedings of Fourth EUROGRAPHICS Workshop on Rendering*, pages 105–114, June 1993.

[93] Peter Schröder and Pat Hanrahan. On the Form Factor between Two Polygons. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 163–164. ACM SIGGRAPH, New York, August 1993.

[94] Peter Schröder and Pat Hanrahan. Wavelet Methods for Radiance Computations. In *Proceedings of the Fifth Eurographics Workshop on Rendering, Darmsadt, Germany*, June 1994.

[95] R. A. Schumacker, B. Brand, M. Gilliland, and W. Sharp. Study for Applying Computer-Generated Images to Visual Simulation. Technical Report

AFHRL-TR-69-14, U.S. Air Force human Resources Laboratory, September 1969.

[96] Min-Zhi Shao, Qun-Sheng Peng, and You-Dong Liang. A New Radiosity Approach by Procedural Refinements for Realistic Image Synthesis. *ACM Computer Graphics*, 22(4):93–101, August 1988.

[97] Peter Shirley. A Ray Tracing Method for Illumination Calculation in Diffuse-Specular Scenes. In *Proceedings of Graphics Interface '90*. Canadian Information Processing Society, 1990.

[98] Peter S. Shirley. *Physically-based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois, Urbana, Illinois, 1991.

[99] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfer*. McGraw-Hill Book Company, 1972.

[100] François Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A Global Illumination Solution for General Reflectance Distributions. *ACM Computer Graphics*, 25(4):187–196, July 1991.

[101] François Sillion and Claude Puech. A General Two-Pass Method Integrating Specular and Diffuse Reflection. *ACM Computer Graphics*, 23(3):335–344, July 1989.

[102] Brian E. Smits, James R. Arvo, and David H. Salesin. An Importance-Driven Radiosity Algorithm. *ACM Computer Graphics*, 26(2):273–282, July 1992.

[103] E. M. Sparrow and R. D. Cess. *Radiation Heat Transfer*. Hemisphere Publishing Corporation, 1978. Augmented Edition.

[104] Stephen N. Spencer. The Hemisphere Radiosity Method: A tale of Two Algorithms. In *Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 127–135, May 1990.

[105] A. James Stewart and Sherif Ghali. Fast Computation of Shadow Boundaries Using Spacial Coherence and Backprojections. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 231–238. ACM SIGGRAPH, New York, July 1994.

[106] W. Stürzlinger. Adaptive Mesh Refinement with Discontinuities for the Radiosity Method. In *Proceedings of the Fifth Eurographics Workshop on Rendering, Darmsadt, Germany*, June 1994.

[107] Kelvin Sung and Peter Shirley. Ray Tracing with the BSP tree. In *Graphics Gems III*, pages 271–274. Academic Press, 1992.

[108] Ivan. E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A Characterization of Ten Hidden-Surface Algorithms. *ACM Computing Surveys*, 6(1):1–55, March 1974.

[109] Earl W. Swokowski. *Calculus and analytic geometry.* Prindle, Weber and Schmidt, 3rd edition, 1983.

[110] Filippo Tampieri and Dani Lischinski. The Constant Radiosity Assumption Syndrome. In *Proceedings of the Second Eurographics Workshop on Rendering*, May 1991.

[111] W. Thibault and B. Naylor. Set Operations on Polyhedra Using Binary Space Partioning Trees. *ACM Computer Graphics*, 21(4):153–162, July 1987.

[112] William C. Thibault. *Application of Binary Space Partitioning Trees to Geometric Modelling and Ray-Tracing.* PhD thesis, Georgia Institute of Technology, 1987.

[113] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of the Optical Society of America*, 57(9):1105–1114, September 1967.

[114] K. E. Torrance, E. M. Sparrow, and R. C. Birkebak. Polarization, Directional Distribution, and Off-Specular Peak Phenomena in Light Reflected from Roughened Surfaces. *Journal of the Optical Society of America*, 56(7):916–925, July 1966.

[115] David Travis. *Effective Color Displays.* Academic Press, 1991.

[116] Roy Troutman and Nelson L. Max. Radiosity Algorithms Using Higher Order Finite Element Methods. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 209–212. ACM SIGGRAPH, New York, August 1993.

[117] T. S. Trowbridge and K. P. Reitz. Average Irregularity of a Rough Surface for Ray Reflection. *Journal of the Optical Society of America*, 65(5):531–536, May 1975.

[118] Christophe Vedel and Claude Puech. A testbed for adaptive subdivision in progressive radiosity. In *Proceedings of the Second Eurographics Workshop on Rendering*, May 1991.

[119] 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. *ACM Computer Graphics*, 21(4):311–320, July 1987.

[120] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A Ray Tracing Algorithm for Progressive Refinement Radiosity. *ACM Computer Graphics*, 23(3):315–324, July 1989.

[121] Gregory J. Ward. Irradiance Gradients. In *Proceedings of the Third Eurographics Workshop on Rendering, Bristol, UK*, pages 85–98, May 1992.

[122] Gregory J. Ward. Measuring and Modelling Anisotropic Reflection. *ACM Computer Graphics*, 26(2):265–272, July 1992.

[123] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A Ray Tracing Solution for Diffuse Interreflection. *ACM Computer Graphics*, 22(4):85–92, August 1988.

[124] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice.* Addison Wesley, 1992.

[125] Alan H. Watt. *Fundamentals of Three-Dimensional Computer Graphics.* Addison Wesley, 1989.

[126] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. Predicting Reflectance Functions from Complex Surfaces. *ACM Computer Graphics*, 26(2):255–264, July 1992.

[127] Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–349, June 1980.

[128] S. K. Zaremba. The Mathematical Basis of Monte Carlo and Quasi-Monte Carlo Methods. *SIAM Review*, 10(3):303–314, July 1968.

[129] Harold R. Zatz. *Galerkin Radiosity: A Higher Order Solution Method for Global Illumination.* PhD thesis, Cornell University, August 1992.

[130] Harold R. Zatz. Galerkin Radiosity: A Higher Order Solution Method for Global Illumination. In *COMPUTER GRAPHICS Proceedings, Annual Conference Series*, pages 213–220. ACM SIGGRAPH, New York, August 1993.

[131] O. C. Zienkiewicz. *The Finite Element Method.* McGraw-Hill Book Company (UK) Ltd., 3rd edition, 1977.