SHADING AND INVERSE SHADING FROM DIRECT ILLUMINATION

By

Pierre Poulin

M.Sc., University of Toronto, 1989

B.Sc., Université Laval, 1986

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming
to the required standard

....

....

....

....

THE UNIVERSITY OF BRITISH COLUMBIA

December 1993

Abstract

An understanding of light and its interaction with matter is essential to produce images. As the modeling of light sources, light transport and light reflection improves, it becomes possible to render images with increasing realism. The central motivation behind this thesis is to improve realism in computer graphics images by more accurate local shading models and to assist the user to obtain the desired lighting effects with these more complex models.

The first part of the thesis addresses the problem of rendering surfaces illuminated by extended (linear and area) light sources. To compute the light reflected by a surface element in a given direction, one needs to determine the unoccluded regions (shadowing) of each light source and then to compute the light reflection (shading) from each of these regions. Traditionally, point light sources are distributed on the lights to approximate both the shadowing and the shading. Instead, an efficient analytical solution is developed for the shading. Shadowing from extended light sources is a fairly expensive process. To give some insights on the complexity of computing shadows, some properties of shadows and algorithms are presented. To reduce the cost of computing shadows from linear light sources, two acceleration schemes, extended from ray tracing, are introduced and evaluated.

The second part of this thesis addresses the problem of achieving a desired shading effect by building up systems of constraints. This approach is called inverse shading. It allows a user to obtain a desired lighting effect by interacting directly with the scene model. An interactive modeling system has been built to study the introduction of rendering issues into the modeling process itself. Two rendering aspects are considered: shading and shadows. By specifying a highlight position and size, the unique direction of a directional light source as well as the surface glossiness are determined. Interactively moving this highlight on the surface redefines the light direction while changing its size redefines the surface glossiness. By manipulating shadow volumes (the volume within which a particular light cannot be completely seen), the light source definition and position can be modified. By assigning colours to various points in a 3D scene, information about the colour of a surface as well as other surface characteristics can be deduced. This relatively new concept of defining the causes by manipulating the effects is expected to have a major impact on the design of future computer graphics modeling systems. It begins a new family of tools for the design of more intuitive user interfaces.

Acknowledgement

As I am writing these lines in the middle of the night, it is hard to believe this part of my life is going to be over soon. It seems just like yesterday, I was this poor Québécois, driving through Canada in my old Lynx (may it rust in peace), so eager to discover...

My supervisor, Alain Fournier, is by far the most predominant figure in my graduate years. His guidance, enthusiasm, friendship, generosity and strength are remarkable. He taught me so much more than what could ever appear in a thesis. It is with pride that I will always remain "one of Alain's students". Kelly Booth and Dave Forsey with their "almost Dr. Poulin" treated me as an "almost junior faculty" and I really appreciated their support.

I thank the members of my supervisory committee: Uri Ascher, David Forsey, Günther Schrack, Jack Snoeyink, and Robert Woodham, for their comments and support. Also Kellogg Booth, Dale Rolfsen and Tomoyuki Nishita for their role on the examination committee.

My friend, Juancho Buchanan, deserves special thanks. Together, "We came, we saw, we conquered!" His friendship means a lot to me. Even so far apart, we will still share. For instance, there are these -40°C's. So many other friends made these years so enjoyable... We started a graphics lab from scratch with the great support of Maria Klawe. This lab very quickly exploded with many great people. Everybody contributed one way or another to this thesis, but to name a few: Atjeng Gunawan, Chris Romanzin, Chris Healey, Vishwa Ranjan, Peter Cahoon, and Rob Scharein. Bob Lewis helped satisfying the library requirements while I was back in Montréal. Without him, the thesis might have never made it. Then there are the cool guys (Chris, Deb and Tien), the kids (Gene, Torre, Larry, Bill, Kevin, et cie), the system guys (and ze solo numerical analist guy), the staff people, the softball, soccer and ballhockey teams, the babies from the next graphics generations, the old generation of DGPers (Andrew, George, Johnny, Eugene, Marc, ...) and so many others with whom I spent long hours chatting. So many...

Mikio Shinya lit up many of my original thoughts on inverse shading during a great visit of his lab at NTT, Japan. Michael Gleicher and Paul Heckbert discussed some of their ideas with me. Pat Hanrahan, Michael Cohen and the graphics students of Princeton helped me on some of my latest work while on leave for my pre-postdoc.

Deborah Wilson, Caroline Houle, Marie-Claire Forgue, Adrienne Drobnies, and Ying Li kept the sun shining under the cloudy days. They made me laugh and they listened to me. Ariel Fournier delighted me with her childhood life. Precious moments of my days...

Finally, the last and most important thanks go to my family, whom I love very much and am so proud of.

Table of Contents

Chapter 1

Introduction

Computer graphics creates images to provide information or to induce emotions. These images are made to be looked at. Because we live in a 3D world, it is often efficient to draw from our experience of our world to convey information. Although realism is not always necessary, or even the best way to convey the information, it can help in many situations. Sometimes, realism *is* the application. For instance, when computer generated objects are merged with real objects to produce a realistic combined image, the rendered objects must be undistinguishable from the real ones to achieve the desired effect.

What we see originates from the interaction of light with our environment. Emitted light is absorbed, reflected and refracted by the objects in the environment. The spectral and spatial distributions of the light can be altered at every step in the paths taken by the light. These changes in the nature of light allow us to *see* in our world.

In order to graphically simulate parts of our world, understanding light and its interaction with matter is essential. The better we can approximate light's behaviour, the closer we can get to capturing the richness of our world in realistic computer graphics images. The factors influencing the way light is reflected include the nature of light itself and the nature and geometry of the surfaces of objects in an environment.

Physics explains light as both a massless particle of energy (photon) and an electromagnetic wave. When a photon strikes an atom on a surface, its energy is absorbed and parts of it may be reemitted as light. Fortunately for us, this *atomic* rendering is not necessary to accurately approximate the behaviour of light in most applications. In fact, in many applications, geometric optics provides a sufficient basis to correctly render most surfaces.

Unfortunately, while geometric optics simplifies the nature of the light, it does not reduce the complexity of our world enough. Take a single tree as an example. Light is reflected from every branch and leaf. But it is also reflected differently on the various parts of a single leaf because of the changing concentrations of materials in the leaf. To properly render the reflection from one leaf, one would need to model these various parts. Our tree is made of thousands of different leaves. A forest is made of thousands of different trees. A forest is also made of grass, flowers, hills and many other complex objects. Very quickly, this hierarchy of objects and details explodes in an overwhelming complexity. Because we are only interested in the light reflection from these objects, it becomes important to find reflection models that can approximate the reflections for the levels of complexity that an application requires.

To reduce the complexity of reflection models, we can also study the medium on which the images will ultimately be reproduced. Computer graphics images are rendered to be viewed on CRT monitors, prints or through projections. Most of these media can reproduce only a subset of the visible colour space (the gamut of the device). So we have to think of realism of rendered 3D scenes as photorealism within the gamut of the device. The limitations of the display system as well as computational costs can force us to reevaluate our simulation of light to *acceptable* approximations, where acceptable is determined by the application for which the image is computed.

1.1 Light Sources

One simplification often used in current reflection models applies to the light emitter. Representing a light as an infinitely distant source (directional light) reduces the problem of illumination to a series of parallel light rays. Representing a light as a 3D point from which the light rays emanate (point light) allows for some simplifications, but not as many. In fact, for many applications where all surfaces form together a small solid angle as seen from a distant light, a directional light is very efficient and quite adequate. When the light emits equally in every direction and when the light forms a small solid angle from each surface element, a point light is very efficient. However, our world is largely made of higher dimensional light emitters, so when the small solid angle assumption is broken, we must compute the reflection originating from such a higher dimensional light.

The first part of this thesis is concerned with improving the shading of surfaces illuminated by extended (linear and area) light sources. In many rendering systems, linear and area light sources are modeled with a series of point light sources. This is done because point light sources are simple to use and can approximate the light emission from a light source of any shape. Unfortunately, the large number of points necessary to approximate a light[1] can require considerable computing resources to produce realistic looking results. In this thesis, a general analytic solution for shading surfaces illuminated by a linear light source is proposed. A linear light approximates a light that is long and narrow. Fluorescent and neon tubular lights are two examples of such light sources in our world. A solution for the diffuse component of reflection is derived for linear lights. By assuming a simple specular reflection component (Phong), inexpensive yet convincing images are produced with the use of a polynomial approximation for the specular reflection curve. An extension of the solution for shading with linear light to shading with area light is also presented. The introduction of these illuminants and their integration in a commonly used reflection model broaden the range of realistic scenes that can be efficiently rendered.

The presence of shadows forms another aspect of realism in computer graphics imagery. Shadows can provide additional information about the shape of an object, its nature (if it exhibits some degree of transparency), its relative position with respect to other objects and

[1]Such an approximation is based on the solid angle the light forms from the surface element to shade.

light sources, and even about the light sources themselves. Shadowing from a directional or a point light source is a binary decision. A point in a scene is or is not illuminated by a directional or a point light source. Shadowing from an extended light source is more complicated because a point in a scene can be illuminated by the entire source, not illuminated at all or be partially illuminated. The information about the shading then depends on the portion of the light that is visible from the point being shaded. Determining this portion can be computationally expensive. We present some properties of shadows from extended lights and show how this information has been used to more efficiently compute shadows. This should help to put in perspective the complexity of correctly computing shadows. In the particular case of linear light sources, it becomes possible to reduce the average cost of computing shadows. Two new acceleration schemes are evaluated. They both take advantage of the 2D nature of the shading problem of a point illuminated by a linear light source. Each technique processes information about the geometry of a scene in order to test only a subset of the objects for shadowing. The increase in cost due to analytically solving the light reflection from extended lights and computing their shadows is substantial compared to a simple directional or point light source. However, this increase is usually a fraction of the cost of using a series of point light sources to simulate the same extended light to the degree to which the shading effects can be considered realistic.

1.2 Lighting Design

Better reflection models and illuminants can greatly improve the realism of computer generated images. However, a model is only as good as the designer using it. It is therefore essential to provide a designer with a set of intuitive parameters that allow one to create the desired lighting effects. Unfortunately, when the number of parameters and their inter-relationships grow, a designer has to understand all of them in order to create the final look of a picture. For instance, once the geometry of a scene has been built, the designer has to position the lights and assign them intensities in order to cast a shadow here or get a highlight there. The designer must also determine values for all of the parameters of every surface so that each highlight will have the proper size and so that the surface colours will approximate well enough what the designer had in mind when building the scene. Changing the intensity of one light can affect the shading of every surface. Moving a light can make a highlight disappear or can cast undesirable shadows or can completely over-saturate the colour of a surface. With current modeling systems, the designer must either compute in her head the *inverse shading* in order to obtain a certain effect or attempt to achieve the desired effect through trial and error. Even for a simple scene, this process is very difficult. Typically, the designer will have to iterate between adjusting the values of the parameters and rendering the entire scene. Depending on the experience of the designer (i.e. how well she can perform mental inverse shading) and the time required to perform the rendering, this iteration will continue until all the desired effects are achieved or until the frustrated designer decides to stop and accept the current image as partially satisfying. The second part of this thesis addresses the problem of achieving a desired shading effect. A system of constraints is employed to bring the user a desired effect by letting her interact directly with specific features in the scene model, rather than indirectly, as is current done in most computer graphics modeling systems.

Instead of observing the results of lighting only after the rendering stage, we introduce some rendering considerations directly into the modeling process. In our system, a user interacts directly with the computer graphics model. Constrained by a system of equations, the user can alter the shading of a surface leaving the remaining unknowns to be solved automatically by the system. We investigate two rendering effects: shading and shadows. The user can control the direction of a directional light by selecting a point on a surface. If this point represents the point of highest diffuse reflected intensity, it identifies a unique direction for the light. If instead it represents the point of highest specular reflected intensity, it gives another direction to the light. The specular intensity corresponds to the highlight region. The user can specify a size for this highlight and by doing so, it becomes possible to identify a unique value for the surface roughness[2] that would create the highlight. The highlight can be moved interactively to specify a new light direction. The size of the highlight can be changed as well, which determines a new value to the roughness coefficient.

Another technique to define and manipulate light consists of manipulating the shadow volume (the semi-infinite volume within which points cannot completely see the light) of a point, linear or area light source. In this way, the shadow is interactively modified and automatically specifies a light that would produce the shadow.

Finally, the user can paint colours on a surface and the modeler will attempt to determine appropriate surface characteristics for which the colour points would retain their colour (within a certain tolerance) when the surface is finally shaded. The under-constrained systems that result from these manipulations are solved via a nonlinear constrained optimisation while the over-constrained systems are solved via a weighted least-squares approximation with penalty functions to constrain the values of some surface parameters. Surface attributes such as surface colour, quantity of ambient, diffuse, and specular reflections and the ratio of dielectric and conductor properties are therefore automatically determined as the user adds colour points and interactively moves them on the surface.

This relatively new concept of defining the causes by manipulating the effects is expected to have a significant impact on the design of future computer graphics modeling systems. It leads to a new family of tools for the design of more intuitive user interfaces by freeing the designer from solving mentally many of the inverse shading problems.

[2]The surface roughness in Phong and Blinn shading models determines the surface glossiness. This term will be defined in the next chapter

1.3 Organisation

The thesis is organised as follows. In the next chapter, light and the radiometric terms used to measure it are explained with an example. Then the light actually captured by the human visual system is described and we briefly discuss the perception of the visual signal and how realism is related to perception. Finally, we present and analyse various reflection models previously introduced in computer graphics.

In Chapter 3, we review the formulations for computing the reflection from directional, point, linear and area lights. We then present our analytical solution for shading with a linear light and how our solution has been derived to shade with a polygonal light.

Because extended lights can produce shadows with both umbra and penumbra regions, we present in Chapter 4 some properties of these regions for both linear and polygonal lights. Two acceleration schemes for shadowing with linear lights are then presented and evaluated. Algorithms for shadowing with polygonal lights are also presented.

In the last chapter, we investigate how to help a lighting designer to efficiently obtain a desired shading effect by indirectly controlling the lights via the position of points of maximum diffuse reflection, highlights and shadows. We also discuss a system in which the designer interactively specifies surface characteristics by simply applying colour points on a surface.

Chapter 2

Local Illumination

Light illuminates objects and makes them visible to humans. This role of light in our daily life is the predominant factor in our appraisal of realism in computer graphics imagery. It is therefore imperative to understand how light interacts with surfaces and how our visual system perceives the results of this interaction. This chapter starts by giving a brief introduction about light and its interaction with matter. The next section will briefly describe how this interaction is perceived by the human visual system. Once this base is established, we enter into the world of computer graphics. In reality, light interaction with matter is very complex. Simplifying this reality is our only practical approach to simulating light interaction in computer generated pictures. We review the various illumination models used in computer graphics and show how far the underlying assumptions depart from realism. Unfortunately, the trade-off with realism is not only a question of computational cost, but also of ease of use. We conclude this chapter with a discussion of ease of use, computational load and realism to justify our choice of illumination models in subsequent chapters.

Before delving into the nature of light, we must clarify a few points. In computer graphics, we always consider the light reflection off surfaces. In reality light can penetrate the surface and interact with the atoms within the surface. However, for simplicity's sake and because a very important factor governing the reflection depends on the orientation of the surface, we model light reflection only as reflection from the surface.

A *surface element* is defined as an infinitesimal portion of a surface. Sometimes, we also refer to a surface element as a point on the surface that has exactly the same properties as the surface element. There is a surface normal at the point and the radiance at the point represents the radiance at the surface element. This analogy will often simplify our explanations.

Finally, there is the distinction between global illumination and local illumination. Consider a single illuminated surface element. Light comes into contact with the surface element. Some portion of this incoming light is absorbed, another portion is directly reflected, while yet another portion interacts with the surface (due to interreflections between surface bumps or interactions with colour pigments within the surface) before being reflected back into the environment. This whole phenomenon is called *local reflection*. The radiance of this surface element once projected onto a pixel produces the *local shading*. If light comes only from light sources, we are dealing with *local illumination* or *direct illumination*. However, this phenomenon does not describe the full behaviour of light. Once light reflects off a surface, it can illuminate other surfaces and reflect from one surface to another an infinity of times. This phenomenon is called *global* *illumination* because the reflected light is potentially a function of every surfaces in a scene. This distinction is a matter of scale. In fact, the interreflections between micro-facets of a surface can be considered as a *global illumination* phenomenon at the scale of the micro-facets. To not confuse these concepts any further, we restrict our definitions of local and global illuminations to the geometric level at which the objects are defined.

In this thesis, we concentrate our efforts on local illumination issues. The results, however, can have a direct impact on global illumination because local light reflection is an underlying part of global illumination.

2.1 Quantitative Measures of Light

In this section, we will look at light as a physical phenomenon measured independently of the observer. In order to simulate the behaviour of light, it is important to understand how light propagates and how it is measured.

Light has the dual nature of a massless particle of energy (photon) traveling as an electromagnetic wave. As photons hit a surface, some energy is absorbed by the atoms at the surface and is re-emitted later in the form of radiation. If the re-emitted radiation is within the narrow band between $380nm$ and $770nm$, this radiation is within the visual range of the human visual system. It is then called *light*.

Radiometry is concerned with measuring radiant energy as a function of the wavelength λ. The following definitions of the most relevant radiometric terms are from Nicodemus *et al.* [nico77]. The characters in parenthesis ( ) represent the symbols used to refer to these concepts while characters in square brackets [ ] represent their units in the system international (SI). Figure 2.1 defines the polar coordinates used throughout the thesis.

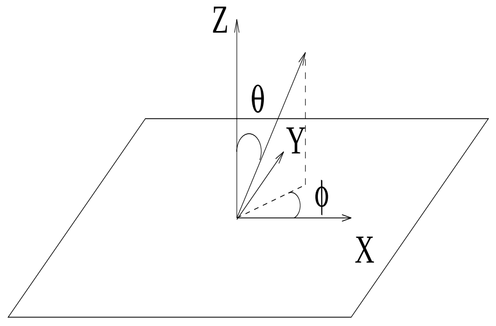


Figure 2.1: Polar coordinate systems

Radiant Energy (Q) [Joules — J]

The radiant energy is the energy propagated in the form of electromagnetic waves or streams of particles (quanta or photons).

Radiant Power or Flux (Φ) [Watts — W]

The radiant power or flux is the radiant energy emitted, transferred, or received through a surface, in a unit time interval.

$$\Phi = \frac{dQ}{dt}$$

Radiant Exitance (M) [Watts per square meter — W/m^2]

The radiant exitance at a point of a surface is the quotient of the radiant power emitted by an infinitesimal surface element containing the point, over the area of that surface element.

$$M = \frac{d\Phi}{dA}$$

Irradiance (E) [Watts per square meter — W/m^2]

The irradiance at a point of a surface is the quotient of the radiant power incident on an infinitesimal surface element containing the point, over the area of that surface element.

$$E = \frac{d\Phi}{dA}$$

Radiant Intensity (I) [Watts per steradian — W/sr]

The radiant intensity of a source in a given direction is the quotient of the radiant power emitted by the source in an infinitesimal element of solid angle containing the given direction, over the element of solid angle.

$$I = \frac{d\Phi}{d\omega}$$

Radiance (L) [Watts per steradian per square meter — $W/(m^2 \cdot sr)$]

The radiance in a given direction at a point on the surface of a source or a receiver, or at a point on the path of a beam, is the quotient of the radiant power leaving, arriving at, or passing through an element of surface at this point and propagated in directions defined by an elementary cone containing the given direction, over the product of the solid angle of the cone and the area of the orthogonal projection of the surface element on a plane perpendicular to a given direction.

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

BRDF (f_r) [inverse steradian — sr^{-1}]

The bidirectional reflectance-distribution function (BRDF) is defined to be the fraction of the incident radiation coming from the direction (θ_i, ϕ_i) that is re-emitted in the direction (θ_r, ϕ_r). The BRDF is expressed as $f_r(\theta_i, \phi_i; \theta_r, \phi_r)$.

$$f_r = \frac{dL_r}{dE_i}$$

2.2 Human Visual System

Illumination concerns us only so long as we can see its effects in our environment. Although this is not primordial in the framework of this thesis, it is important to never forget that the end result of our light simulation is the transmission of information to a viewer. Understanding the human visual system can help in simplifying the illumination computations while retaining the properties of most significance in the human visual system.

Light goes through the human eye and is detected by two types of receptor cells: the rods and the cones. The rods are the most sensitive to light and are responsible for our vision under dim illumination (night vision). The cones are responsible for most of our daylight vision and for our vision of colour. Each receptor cell has a different sensitivity to light of a particular wavelength. The spectral luminous efficiency curves of each receptor cell resemble a Gaussian distribution with its peak at 555 nm and 507 nm for the cones and the rods, respectively. These measure the response of the human visual system to light as a function of the wavelength.

Photometry studies light measurements as experienced by the human visual system. Human eye characteristics vary from one human to another, and even from one eye to the other in the same person. However many photometric measurements have been performed on many persons and a set of accepted standard values exists.

We conclude this brief incursion into the properties of the human visual system by stating that for each radiometric quantity, there exists a photometric one [wysz82]. Therefore it is always possible to convert a radiometric value into its equivalent photometric one by convolution with the appropriate spectral luminous efficiency curve. To avoid any confusion in the remainder of this thesis, we always use radiometric terms.

2.3 Perception and Realism

Of all the light being reflected in our environment, only the very small portion impinged upon our visual system is of potential visual significance. Somehow, the human brain interprets the spectral composition and spatial distribution of the signals it receives from the visual system. It is unclear how the information is extracted from the visual signals. Moreover, this information can be interpreted in highly different ways from one person to another.

To produce realistic images (photorealistic experience), an artist must face several dilemmas. An image conveys conflicting information because of its duality as a scene with 3D visual cues and as a flat 2D representation of this scene. Pictures generally occupy only a small portion of our field of view. They are computed from a single point of view and do not allow for different depths of field. They are also subject to illumination from where they are observed. Moreover, the display media currently used exhibit a much smaller dynamic range than what our visual system transmits to us of our world.[1] So it is likely that the visual experience of looking at reality will be different than looking at a 2D representation of this same reality. The knowledge accumulated by the viewer will lead to different perceptions of this reality.

Nevertheless, constructing an image closer to reality provides us with the possibility of reaching more people who have experienced similar structures that will lead to similar interpretations. For instance, hidden surface removal in line drawings removes some information about the occluded structures, but the disambiguation of the spatial relationships often makes an image much easier to interpret. Similarly, the gradient of the shading on a surface can reveal important information about the shape of the surface. Additional cues like shadows, textures, reflections, transparencies, light interreflections, material properties, and motion blur are all examples of contributions that make the experience of looking at a computer graphics image more *real* because they simulate what we experience in our world. Therefore, even though we cannot quantify the *realism* within an image, we can judge if the presence or absence of certain features in an image contributes to making the image closer to our experience of our world. It is this *unscientific* feeling we will be relying on when arguing that a technique leads to more convincing photorealistic images.

2.4 Local Shading in Computer Graphics

For many years now, simple local illumination models such as those introduced by Phong [phon75], Blinn [blin77] and Cook [cook82] have been successfully used to render 3D scenes with a certain degree of realism. Improvements for these basic models have been proposed, but somehow they have had a limited impact within the 3D graphics community.

In this section, we analyse some of the proposed improvements and try to determine why they are not widely used today. With this background, we establish some criteria that should be considered when developing a new reflection model. These criteria help us justify our choices for local illumination models in subsequent chapters.

As we already know, shading in computer graphics is an essential part of realism. Shading carries information about the geometry of objects, the orientation of lights and the nature of surfaces. In the late 1970's and early 1980's, most of the currently used local illumination models were introduced into computer graphics. Work by Phong [phon75], Blinn [blin77] and Cook [cook82] paved the way to well accepted and commonly used illumination models. We

[1]Barbour and Meyer [barb92] describe some of these dilemmas in more details and explain some techniques that can be used to remedy some of these limitations. They base their work on a large body of literature in psychology, photography, art, optics and computer graphics. The interested reader is referred to their paper and its references for a more exhaustive enumeration and treatment of pictures as representations of reality.

can summarise the various formulations into a single generalised equation:

$$\begin{aligned} L_{pixel} &= f_{ra} L_{ia} \pi + \\ & \sum_{i=1}^{m} \int_{\omega_i} f_{rd}(\theta_i, \phi_i; \theta_r, \phi_r) L_i \cos\theta_i \, d\omega_i + \\ & \sum_{i=1}^{m} \int_{\omega_i} k_s F_s(\theta_i, \phi_i; \theta_r, \phi_r) L_i \, d\omega_i \end{aligned} \tag{2.1}$$

L_{pixel} is the radiance reflected by the surface element in the direction of the pixel of interest. Therefore, if more than one surface element reflects light in the direction of a given pixel, the total radiance in this pixel direction will be a weighted average of the radiance of all these surface elements. The weight of a given surface element corresponds to the visible area this surface element projects onto the pixel.

The reflected radiance from m light sources is subdivided into three terms: ambient, diffuse and specular. Figure 2.2 shows how a combination of each of these three reflections can provide a more complete surface shading.

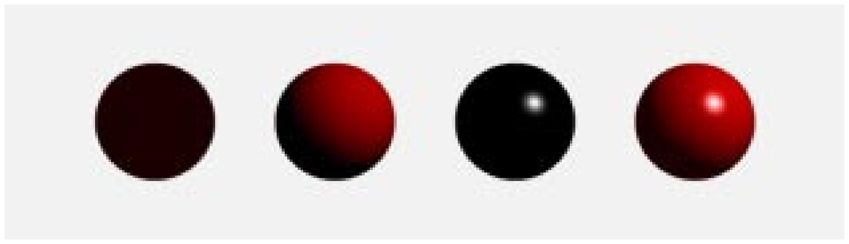


Figure 2.2: Ambient, Diffuse and Specular Terms

The first term corresponds to the ambient reflection. In early reflection models, interreflection between surfaces was very crudely approximated by this term. It assumes that the indirect light comes to the surface element equally from every directions in the hemisphere above it. This means that a surface element shaded with only ambient reflection will always reflect the same radiance whatever its orientation and its position (as long as it is visible through a pixel). L_{ia} is the ambient radiance of a scene. It is assumed constant at any point on a surface. If we integrate this ambient radiance over the entire hemisphere above the surface element, we get

$$\int_{-\pi}^{\pi} \int_{0}^{\frac{\pi}{2}} L_{ia} \cos\theta \sin\theta d\theta d\phi = L_{ia}\pi$$

f_{ra} is the ambient BRDF associated with the surface element. Normally, there should be only one BRDF f_r associated with a surface. Dividing it into a summation of three functions f_{ra}, f_{rd}

and f_{rs} for the three types of reflection provides the possibility to simplify each one individually. Usually in computer graphics, k_a replaces $f_{ra} \cdot \pi$.

The second term captures the diffuse reflection. Diffuse reflection occurs when light hits an ideal diffuse surface (also known as a Lambertian surface). A diffuse surface element will appear equally bright from whatever angle it is looked at. This phenomena is a function of $\cos\theta_i = (\vec{N} \cdot \vec{L})$, i.e., it is proportional to the inner product between the surface element normal ($\vec{N}$) and the light direction ($\vec{L}$) from the surface element.[2] L_i (not vectorised) is the light source radiance in the direction of the surface element and $d\omega_i$ is the solid angle formed by the light from the surface element. These values must be computed and summed for each of the m lights in the scene. For a perfectly diffuse surface, it turns out that f_{rd} is constant and therefore

$$\begin{aligned} L_{pixel} &= f_{rd} \int_{\omega_i} L_i \cos\theta_i \, d\omega_i \\ &= f_{rd} E_i \end{aligned}$$

which means that the diffuse reflected radiance is proportional to the irradiance on the surface. Usually in computer graphics, f_{rd} is simply called k_d. Surfaces covered by matte paint or chalk dust are two examples of real surfaces that radiate light mostly in a diffuse way. Two models can explain the diffuse reflection. In one, light reflects from a surface made of a multitude of mirror-like micro-facets before leaving the surface. In the other model, light penetrates the surface and is scattered internally onto colour pigments before emerging out of the surface. Figure 2.3 illustrates these two models.

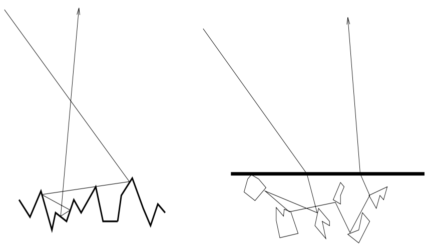


Figure 2.3: Two models for diffuse reflection

The third and last term simulates the specular reflection. Some surfaces reflect light more around one direction. This reflection is associated with what is usually perceived as a highlight. For a perfect mirror surface, light coming from direction (θ_i, ϕ_i) will reflect in direction $(\theta_i, \phi_i + \pi)$, where ϕ is the azimuthal angle of the light direction. To capture this behaviour, we need to use the Dirac delta function $\delta(\theta_i - \theta)\delta(\phi_i - (\phi + \pi))$ which is non-zero (i.e., 1) only when

[2] Please note that all through this document, vectors are identified by a capital letter crowned by an arrow ($\vec{A}$) and are all assumed normalised unless otherwise specified.

$\theta = \theta_i$ and $\phi + \pi = \phi_i$. The reflected radiance $L_r(\theta_r, \phi_r)$ in the perfectly reflected direction must therefore satisfy

$$L_r(\theta_r, \phi_r) = \int_{-\pi}^{\pi} \int_0^{\frac{\pi}{2}} F_s L_i(\theta_i, \phi_i) \sin\theta_i d\theta_i d\phi_i = L_i(\theta_r, (\phi_r + \pi)).$$

This is achieved when f_{rs} is expressed as

$$\begin{aligned} F_s &= \frac{\delta(\theta_i - \theta_r)\delta(\phi_i - (\phi_r + \pi))}{\sin\theta_i} \\ &= \delta(\cos\theta_i - \cos\theta_r)\delta(\phi_i - (\phi + \pi)). \end{aligned}$$

Phong [phon75] observed that for a mirror, the specular direction corresponds to the perfectly reflected light direction ($\vec{R_L}$) around the normal at the surface. See Figure 2.4. Only when a

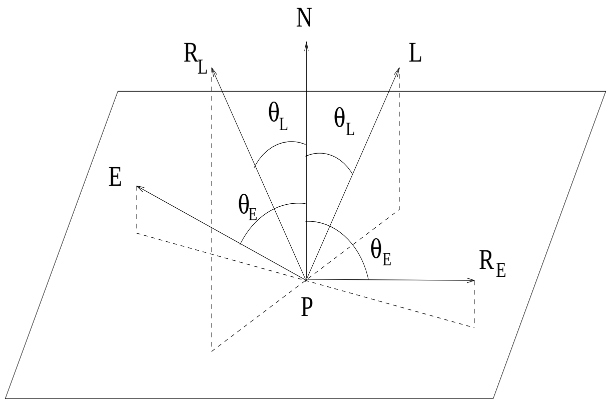


Figure 2.4: General Reflection

viewer ($\vec{E}$) is positioned along ($\vec{R_L}$), should she see the specular reflection. However, most real surfaces are non-ideal specular reflectors and some light is also reflected slightly off axis from the ideal reflected light direction $\vec{R_L}$. A possible explanation for this phenomena is that a real surface is never perfectly flat but contains microscopic deformations. To simulate the decay of radiance reflected off axis, Phong controlled the specular term by raising the inner product between the eye direction ($\vec{E}$) and the perfectly reflected light direction ($\vec{R_L}$) to a certain power n. In this model, n is a positive real number which is a function of the surface roughness. Later, Blinn [blin77] incorporated various distributions, D, of these microscopic deformations. Blinn also included in his model other surface parameters like geometric attenuation, G, due to self-blocking, and the Fresnel coefficient, F, for grazing angle effects. To incorporate these factors, F_s is expressed as

$$F_s = \frac{DGF}{\cos\theta_r}.$$

The factor $(\vec{N} \cdot \vec{H})^n$ provides a distribution approximating the specular reflection model proposed by Phong, although with more sound physical bases. Here, $\vec{H}$ is the bisector direction between the eye $\vec{E}$ and the light $\vec{L}$ directions.

The reader familiar with the traditional nomenclature of the reflection models in computer graphics will notice that Equation 2.1 departs from it in many symbols, although the equivalent graphics terms can easily be substituted. The connexion between f_r and the standard graphics formulation [lewi93] corresponds to

$$f_r(\theta_i, \phi_i; \theta_r, \phi_r) = k_d + k_s \frac{F_s(\theta_i, \phi_i; \theta_r, \phi_r)}{\cos \theta_i}$$

We do so because of the need to use proper radiometric values and units. This can help avoid inconsistencies when going from one reflection model to another, whether that model is developed in computer graphics, computer vision or optics. In this thesis, the radiometric terms are used.

Several improvements to the basic reflection model described above have been proposed. The parameters controlling the specular reflection have also been applied to refraction [hall83] [hall89]. Even energy conservation formulations have been presented for many of the popular models [lewi93]. Each of these improvements was justified as a method to render scenes with a higher degree of realism. We will examine in more detail some other improvements in the next sections.

2.4.1 Spectral curves

Many of the terms in Equation 2.1, reflection and emission, are actually functions of the wavelength λ. Much effort has been devoted to representing and efficiently computing this wavelength dependency [cook82] [meye88] [yuan88] [glas89] [musg89] [gart90] [raso91] [meye91] [borg91] [peer93]. Although physical accuracy requires that computations be carried out over the entire visible spectrum, most of the renderers in use today simply sample this spectrum in the red, green and blue regions. This is done for several reasons. There is no generally accepted representation for the spectral curves. While surface reflectance spectral curves are rather smooth [raso91], light emittance spectral curves can be noisy and exhibit strong, narrow peaks (e.g., fluorescent light) [wysz82]. One can always use a very accurate representation, but at the cost of an increase of computation and storage. The unavailability of spectral data makes it unappealing to any neophyte in colour science. Meyer and Hale [meye91] measured the spectral properties (reflectance, emittance, absorption and transmittance) of some materials and proposed to build a spectral database for surface rendering. Such a database would certainly help in spreading the use of spectral curves.

2.4.2 Wave Propagation

One can also treat light propagation as a wave. Moravec [mora81] proposes to simulate this form of light transport by using wave fronts rather than light rays. However, computing these wave fronts as they progress through the environment and reflect off surfaces is a very expensive process. Some of the effects produced can fortunately be simulated using the simpler ray model for light propagation. This simple model has even been used to simulate the interference of reflected light [firb85] [smit89] [dias91]. Wolff and Kurlander [wolf90] present a model for ray tracing surfaces lit by polarised light.

2.4.3 Average Approximations of BRDF

The experimental approach to reflectance accuracy involves measuring the light reflected in all directions for all possible directions of the incoming light. A BRDF is defined over infinitesimal intervals and therefore cannot be measured directly. However, average values over finite intervals can be measured and used as an approximation to a BRDF. Such approximations have been measured [krin47] into tables for various earth surfaces (wheat fields, ocean and forest) as viewed from an airplane. Cabral *et al.* [cabr87] compute reflectance tables for computer generated height fields.[3] They consider shadowing and masking effects over a triangular mesh representation of the height field, but, they neglect interreflections between surface elements. They discretise the reflectance hemisphere (hemisphere above the height field) into bins in order to accelerate the computations.

These measurements should lead to very accurate reflectance functions if the sampling is done at a sufficiently fine discretisation. Unfortunately, the obvious drawback of this approach lies on the huge amount of data needed. At a 5^o precision in both θ and ϕ, we have a four dimensional table with more than 1.5 million entries. And all of this for only a single surface. To reduce the amount of data, Westin *et al.* [west92] use samples to define spherical harmonics as do Sillion *et al.* [sill91b] to represent BRDFs for global illumination. Both techniques conserves energy (important in global illumination) and their representation is suitable for filtering between geometric levels of details. Unfortunately for irregular BRDFs with long and narrow spikes, even spherical harmonics require a very large number of coefficients to accurately approximate the BRDFs. Westin *et al.* give a few tricks to reduce the number of coefficients but never provide any kind of description of the order of magnitude of storage their representation requires.

Fournier [four92a] [four92b] introduces the concept of a distribution of surface normals to capture reflections from the microscopic level (Phong, Blinn, Cook) to the macroscopic (geometric) level. A distribution of surface normals is represented by a few directions (main normals) around which the micro-facets of a surface are mostly oriented. The distribution of the normals around each direction is controlled by a Gaussian function. Fournier establishes

[3]A height field is defined as a function defined over a surface

a parallel between the standard BRDFs and these distributions. He uses them to produce simple definitions for complex surfaces and explains how to render the surfaces efficiently. His approach leads to important savings because only a two-dimensional distribution of surface normals needs to be stored instead of a four-dimensional table.

Becker and Max [beck93] also allow surfaces to be viewed at different geometric levels (BRDF, bump map and displacement map). A smooth transition is provided between the BRDF, the redistributed bump map (corrected to produce the same average reflected radiance as the other levels) and the displacement map. Their technique still requires the use of several large tables.

Ward [ward92a] [ward92b] introduces a different shading model to address the issue of data size. He designed a new physical device to simplify the measurement of BRDFs. An external light illuminates the surface being measured. The light reflects onto an hemispherical reflective dome covering the surface. A CCD camera, positioned in the dome, measures the reflected radiance on the dome. Ward thereby reduces the measurements to only one array of intensities for each different light direction (a reduction from 4D to 2D). One drawback is that his device lacks accuracy when the light or its reflection are at an angle larger than 60° to the surface normal of the sample. The radiances measured are currently restricted to values between 0 and 255 (CCD resolution). Ward observed in his measurements that the reconstructed BRDFs could be approximated by two elliptical Gaussians, perpendicular to each other. He therefore reduces to only a few parameters the huge tables of Cabral *et al.*, the large number of spherical harmonics of Westin *et al.*, or the distribution of normals of Fournier. One can, however, wonder how many surfaces his model can represent with sufficient visual accuracy.

2.4.4 More Sophisticated Physical Models

He *et al.* [he91] introduce a more sophisticated reflection model based on a more complete description of surface characteristics. To derive their model, they introduce several variables that capture certain previously neglected aspects of actual reflection. Their model includes polarisation of light, height, slope and statistical connection of the surface bumps, and a notion of effective roughness. Although their model has been demonstrated to be close to actual surface reflectance for certain materials, the number of parameters is disconcerting. The situation is further complicated because each variable may be dependent on the others and the effect of any one parameter on the final shading is not obvious.

Hanrahan and Krueger [hanr93] propose a model to simulate reflection off layered surfaces. Their model replaces the diffuse term approximated by Lambert's cosine law ($\vec{N} \cdot \vec{L}$) by a much more complete model where light penetrates the surface and is scattered between the various layers before being reflected off the surface. They present applications of their model to rendering human skin and tree leaves.

All the values for the variables in the models of He *et al.* and Hanrahan and Krueger could be provided in a special library of surfaces for photorealistic rendering. However, measuring all these values is a very difficult task requiring much more equipment, time and expertise than most computer graphics laboratories can afford.

2.5 Failures of Current Reflection Models

In the previous sections, we reviewed some proposed improvements to the basic reflection models. None of these improvements really succeeded in making any of the commonly used reflection models obsolete. They each contribute to the creation of more realistic rendering of surfaces, but somehow the improvements were not worth the costs involved.

We identify a few important qualities necessary for a reflection model to be accepted. First, the model must produce *realistic* looking surface shadings and and it must be *consistent.* Varying the direction of the light sources should create continuous changes in the shading (no unexpected discontinuities). The model must be *easy to use* and *intuitive.* The model must also be *flexible*, so that it produces a wide variety of shading effects. Finally, it should be *reasonably fast* to compute, it should be *memory efficient* and, it should be *relatively simple to implement.*

Naturally, choosing a shading model is also a function of the application domain. In global illumination, conservation of energy is very important, while in the entertainment business, achieving a desired effect is the only issue that matters.

A new reflection model recently proposed by Schlick [schl93] seems to fulfill well many of the criteria listed above. His model is an interesting compromise between the theoretical accuracy and simplicity of use. It is controlled by only a few intuitive parameters. Moreover, his parameters can be associated to measurable physical surface characteristics. While it is still too early to evaluate the impact this approach will have on the models currently used, Schlick's new model is a step forward in a very interesting direction.

Because we would like the techniques developed in this thesis to be accepted and widely used by the graphics community, we concentrate our efforts around Phong's and Blinn's shading models. As we have seen, this is not a physically accurate but it has proven to satisfy a wide variety of applications. Equation 2.1 can be rewritten with the specular reflection from Blinn as

$$\begin{aligned} L_{pixel} &= f_{ra} L_{ia} \pi + \\ & \sum_{i=1}^{m} \int_{\omega_i} f_{rd} (\vec{N} \cdot \vec{L_i}) L_i \, d\omega_i + \\ & \sum_{i=1}^{m} \int_{\omega_i} k_s F'_s(\theta_i, \phi_i; \theta_r, \phi_r) (\vec{N} \cdot \vec{H_i})^n L_i \, d\omega_i \end{aligned} \tag{2.2}$$

In the case of Blinn's full formulation,

$$F'_s = \frac{GF}{\cos\theta_r}.$$

In the general formulation of Phong, the Fresnel term, F, and the geometric attenuation function, G, are not considered and F_s' is taken to be constant. We will write it as F_s. Equation 2.2 is the shading equation on which we base our techniques. This formulation is quite popular. However, it is our hope that the techniques developed in this thesis will somehow transcend the shading model used.

Chapter 3

Shading with Extended Light Sources

In the production of computer graphics images, after designing the 3D geometry of a scene the artist must choose the number, position, and emitted radiance of the illuminants. Of equal importance are the values assigned to the various reflection properties of each surface. Together, the illuminants and the characteristics of the surfaces determine the final appearance of the scene once it is rendered.

In the previous chapter, we reviewed some of the reflection models proposed to simulate the optical properties of a wide variety of surfaces. In this chapter, we look at the various types of light sources used in computer graphics and integrate the properties of lights (in radiometric terms) into the basic reflection models of Phong and Blinn.

One of the requirements for generating realistic images is the capability to simulate a wide variety of light sources. Small changes in the emitted radiance of illuminants can significantly affect the appearance of a scene. To get the desired shading effect, an artist may add dozens of lights of different types. In computer graphics, the two most commonly used light sources are the directional light and the point light. They are simple to use and the irradiance they produce on a surface is relatively easy to compute. However, they are of limited use in the simulation of the irradiance patterns of real lights such as fluorescent light or neon tube light. Neither do they capture the shading gradients of area lights. Both types of lights are part of our everyday existence and so, to be complete, computer rendered scenes should be able to handle these types of illuminants.

After reviewing the models developed to simulate these extended lights, we present our analytical solution to shading with a linear light. Our model allows us to simulate the emission of a neon tubular light where each point in the light radiates light equally in every direction. This model is extended to simulate fluorescent tubular lights where each point on the surface of the tube radiates light mostly in the direction perpendicular to the main axis of the tube. We also show how our solution was used by Tanaka and Takahashi [tana91b] to simulate light emitted from polygonal lights.

3.1 Concepts and Related Work

In the next sections, we will describe the related work in function of each type of light source.

3.1.1 Directional Light

The simplest light source in use in computer graphics is the directional light source. It models parallel light rays coming from an infinitely distant light source of infinite power.

Assume that a directional light shining from (θ_0, ϕ_0) produces an irradiance E_0 on a surface oriented perpendicularly to the light direction. The irradiance in this case is expressed as

$$E_0 = \int_{-\pi}^{\pi} \int_0^{\frac{\pi}{2}} L_i \cos\theta_i \sin\theta_i d\theta_i d\phi_i.$$

Because the surface is perpendicular to the light direction, $\cos\theta_i = 1$. Obviously, the radiance L_i of this light is zero except in the light direction. We use the Dirac delta function $\delta(\theta_i - \theta_0)\delta(\phi_i - \phi_0)$ to ensure that the light radiance L_i is non-zero only in the direction (θ_0, ϕ_0). The normalisation factor to express E_0 as a function of L_i is simply

$$\begin{aligned} E_0 &= \int_{-\pi}^{\pi} \int_0^{\frac{\pi}{2}} \frac{\delta(\theta_i - \theta_0)\delta(\phi_i - \phi_0)}{\sin\theta_0} L_i \sin\theta_i d\theta_i d\phi_i \\ &= \int_{-\pi}^{\pi} \int_0^{\frac{\pi}{2}} \delta(\cos\theta_i - \cos\theta_0)\delta(\phi_i - \phi_0) L_i d\theta_i d\phi_i \end{aligned}$$

By integrating, we can determine the radiance of the directional light that would produce an irradiance E_0 on a surface oriented perpendicularly to the light direction as

$$L_i = \frac{E_0 \delta(\theta_i - \theta_0)\delta(\phi_i - \phi_0)}{\sin\theta_0} = E_0 \delta(\cos\theta_i - \cos\theta_0)\delta(\phi_i - \phi_0).$$

We can use this radiance in the reflection model of Equation 2.2 described in the previous chapter. To simplify the equations, we do not consider in this section the ambient reflection, which can be added directly to any formula because it does not depend on the light definition or position.

The radiance of a surface element in the direction of a given pixel while illuminated by a directional light is

$$\begin{aligned} L_{pixel} &= k_d \int_{\omega_i} (\vec{N} \cdot \vec{L}_i) L_i \, d\omega_i + \\ & \quad k_s \int_{\omega_i} F_s (\vec{N} \cdot \vec{H}_i)^n L_i \, d\omega_i \\ &= k_d \int_{\omega_i} (\vec{N} \cdot \vec{L}_i) E_0 \tfrac{\delta(\theta_i - \theta_0)\delta(\phi_i - \phi_0)}{\sin\theta_0} d\theta_i d\phi_i + \\ & \quad k_s \int_{\omega_i} F_s (\vec{N} \cdot \vec{H}_i)^n E_0 \tfrac{\delta(\theta_i - \theta_0)\delta(\phi_i - \phi_0)}{\sin\theta_0} d\theta_i d\phi_i. \end{aligned}$$

The radiance of the surface element becomes simply

$$\begin{aligned} L_{pixel} &= k_d (\vec{N} \cdot \vec{L}_i) E_0 + && (3.3) \\ & \quad k_s F_s (\vec{N} \cdot \vec{H}_i)^n E_0. && (3.4) \end{aligned}$$

There are several reasons for the popularity of the directional light model. First, the vector $\vec{L}$ that points to the light source is constant and does not have to be computed at each point to shade. This vector, in fact, defines the directional light source. Because the vector $\vec{L}$ is constant, the shading computation for polygons is simplified. Thus many computations can be removed from the inner loops of the shading computations [phon75]. The radiance of the light source is constant and independent of the position of an object in space. Therefore, to make a surface radiance twice its value, one can simply double the light radiance.

3.1.2 Point Light

The second most popular light source is the point light source, which is defined geometrically by a point in space radiating light equally in every direction. Such a point light source emits light over the entire sphere. Its radiant intensity is

$$I = \frac{\Phi_p}{4\pi}.$$

The solid angle subtended by a surface element dA as seen from the point light source corresponds to

$$d\omega = \frac{dA \cos\theta}{r^2},$$

where θ is the angle between the surface element normal and the direction to the point light source and r is the distance between this surface element and the point light source.

The irradiance of the point light source onto a surface element is computed as

$$E = \frac{I\, d\omega}{dA} = \frac{\Phi_p \cos\theta_i}{4\pi r^2}.$$

The radiance of a surface element in the direction of a given pixel illuminated by a point light is

$$\begin{aligned} L_{pixel} &= k_d \frac{\Phi_p(\vec{N} \cdot \vec{L}_i)}{4\pi r^2} + && (3.5) \\ & \quad k_s F_s \frac{\Phi_p(\vec{N} \cdot \vec{H}_i)^n}{4\pi r^2}. && (3.6) \end{aligned}$$

The directional and point light sources are generally included in most rendering systems due to their simplicity. In fact, many graphics hardwares implement only these two types of light sources. Modifications to the radiant intensity distribution of point light sources can give the designer more creative freedom. Warn [warn83] attempts to create more realistic light sources by simulating the lights used by photographers. His extensions include making the radiant intensity of the point light sources a function of direction to produce spotlights and providing flaps that can cut off the light in certain directions. Verbeck and Greenberg [verb84] and Nishita *et al.* [nish85] extend Warn's work of by providing more sophisticated lighting design tools. They allow the user to specify the shape of the radiant intensity distribution by drawing goniometric diagrams.[1]

3.1.3 Linear Light

Linear light sources simulating tubular lights (neon and fluorescent) open a new dimension of shading effects.

We express the radiant flux Φ_l of a linear light in terms of watt per unit length (W/m). Assume that every point on a linear light radiates light equally in every direction. The irradiance of a linear light source onto a surface element is extended from the formula developed for a point light source. It corresponds to

$$E = \int_{\mathcal{L}} \frac{\Phi_l}{4\pi} \frac{\cos\theta}{r^2} dl = \frac{\Phi_l}{4\pi} \int_{\mathcal{L}} \frac{\cos\theta}{r^2} dl.$$

where $\mathcal{L}$ is the length of the linear light and Φ_l is assumed constant over every part of the light.

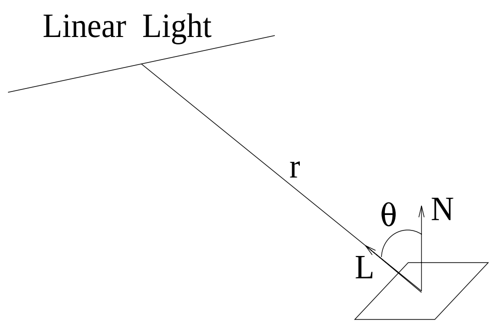


Figure 3.5: Irradiance of a linear light

The radiance of a surface element in the direction of a given pixel illuminated by a linear light is

$$L_{pixel} = k_d \frac{\Phi_l}{4\pi} \int_{\mathcal{L}} \frac{(\vec{N} \cdot \vec{L_i})}{r^2} dl + \quad (3.7)$$

$$k_s F_s \frac{\Phi_l}{4\pi} \int_{\mathcal{L}} \frac{(\vec{N} \cdot \vec{H_i})^n}{r^2} dl. \quad (3.8)$$

Linear light sources are modeled by Verbeck and Greenberg [verb84] using a series of collinear point light sources. By only sampling the radiant intensity of the linear light at discrete positions, artifacts in the form of discrete regions may appear in the diffuse reflection, in the

[1]Goniometric diagrams specify relative intensity as a function of direction. See the IES Lighting Handbook [kauf81] for a good source of lighting definitions.

specular reflection, and in the shadows. These artifacts are mostly apparent on surfaces with sharp specular highlights and in the shadows cast from this linear light. To alleviate these sampling problems, it is always possible to add more point light sources. However, this is done at an important increase in cost because each new point light contributes to a very significant portion of the entire rendering process. Moreover, it is very difficult to estimate the best number of point light sources that would be cost effective while still producing images exempt of major shading artifacts. This number depends on the distribution of objects around the linear light in the scene, the type of surfaces (diffuse vs. highly specular), the viewer position, the positions of the shadows and the contrasts in the final image. It is conceivable that for each frame of an animation sequence, the number of point light sources needed to approximate a simple linear light could vary. Since many of these factors are geometric or are a function of some mathematical model, one could develop some heuristics to estimate the number of point light sources by which a linear light could be approximated with limited shading artifacts, but there is always the concern that changing the number of point light sources might produce temporal aliasing. However, as it will be argued later in this chapter, an analytical solution proposed will provide perfect shadings at a relatively low cost.

Nishita *et al.* [nish85] simulate a fluorescent light where each point on the light emits light as a cosine function around the direction perpendicular to the linear light. They derive an analytic solution for the diffuse reflection when the linear light source is parallel or perpendicular to the surface. For the general case, where the surface element is neither parallel or perpendicular to the linear light, they transform the light and the surface element to a new coordinate system (2D) where they perform a numerical integration via increments. Multiple point light sources are used to approximate the specular integral with possibly the exact same problems that can appear in Verbeek and Greenberg [verb84] (i.e. discrete regions in the highlight). In subsequent work, Nishita *et al.* [nish92] simulate linear lights in a similar way to their earlier work [nish85], although the diffuse integral is performed over a long thin rectangle instead of a line. The specular integral is still approximated by point sampling the line as before [nish85]. However, by assuming that the linear lights all reside on a plane (the ceiling of classroom for instance), they present techniques to speed up shadow computation using a scanline approach.

Picott [pico92] simulates a linear light where each point radiates light equally in every direction. He solves the diffuse reflection analytically but replaces the entire specular integral by a single sample point. This point on the linear light represents the point of highest specular reflected radiance from the point being shaded for the eye position under consideration. This crude approximation of the specular integral can produce satisfactory results as long as the roughness coefficient of the surface is very high (the intensity curve then exhibits a single narrow peak) and as long as the angle formed by the end points of the linear light at the point being shaded is relatively small. Breaking any of these assumptions leads to aliasing artifacts in the highlight regions. To reduce this aliasing, Picott proposes to sample the linear light around the point of highest specular reflected intensity. This sampling is done according to the roughness coefficient of the surface. However, as will be demonstrated with our formulation, computing analytically this entire integral costs the equivalent of just a few samples but avoids sampling artifact altogether.

Bao and Peng [bao93] present analytic solutions for both diffuse and specular reflections of linear lights. They derive their solutions in Cartesian 3D space. The analytic solution for the specular reflection is reduced to evaluating a series of $\frac{m(m+1)}{2}$ simple polynomials of degree up to m, where $m = \frac{n+3}{2}$ is a function of the surface roughness n. This solution is valid only if n is an odd integer. This limitation on n mainly affects the rough surfaces for which n is small. As the surface becomes smoother (a large value for n), the function $\cos^n \phi$ behaves more and more like a delta function and the requirement that n be an odd integer is not as restrictive. However, for large n, the computational cost of evaluating $\frac{m(m+1)}{2}$ polynomials of degree up to m can quickly become prohibitively expensive.

3.1.4 Area Light

Nishita and Nakamae [nish83] were the first in computer graphics to present a solution to the diffuse integral for polygonal and polyhedral light sources. By assuming a Lambertian light (each light element emits light equally in every direction), the radiance falling on a surface element is proportional to the solid angle subtended by the light. For a polygonal light, this corresponds to:

$$L_{pixel} = k_d \frac{L}{2\pi} \sum_{i=1}^{m} \beta_i \cos \delta_i \tag{3.9}$$

where	P	is a point on the surface element being shaded
	k_d	is the proportion of diffuse reflection of the surface
	L	is the light radiance
	m	is the number of vertices of the polygonal light
	β_i	is the angle formed by two adjacent light vertices
	δ_i	is the dihedral angle between the plane at P and the plane of the face formed by the segment on the light and P.

These variables are illustrated in Figure 3.6.

A parallel can be drawn between the irradiance and the radiosity, which is also called radiant exitance. If we neglect the presence of the radiance, the surface BRDF and the diffuse coefficient, the formula of Equation 3.9 is what is called in global illumination the form-factor between a surface element and a polygon. The concept of form-factor is the basis of most global illumination algorithms in use today.

Some techniques that simulate linear light sources have also been adapted to deal with the specular integral of area light sources. It is fairly simple to see how the approaches of Verbeck and Greenberg [verb84] and Picott [pico92] can be extended for area light sources. In fact, several other techniques have addressed various issues in sampling the radiance of area

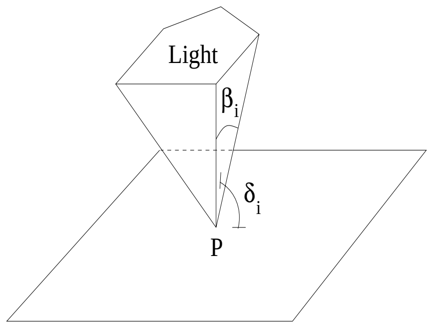


Figure 3.6: The geometry describing diffuse shading from an area light

light sources [cook84b] [shir91]. Houle [houl91] represents her area lights as radiance direction distributions and uses a pyramidal approach to filter the light radiance.

Bao and Peng [bao93] also derived a solution for shading with an area light source. An alternate solution to Equation 3.9 is developed to compute the diffuse reflection. To approximate the specular reflection, they use a Taylor expansion of their formula with only a few terms. A better solution is obtained by adding more terms from the Taylor's series.

Some algorithms have been proposed to treat specular surfaces within the radiosity algorithm. These techniques can be applied to compute the reflected radiance off a surface illuminated by a polygonal light. They have the advantage of conserving energy but unfortunately rely on discretisations of every surface. Immel *et al.* [imme86] add discrete directions to each patch and compute the radiosity solution with these directions. Sillion *et al.* [sill91a] represent the reflected radiance with spherical harmonics at discrete points on each patch, assuming that the radiance should remain uniform within neighbouring points. Rushmeier and Torrance [rush90] and Hall and Rushmeier [hall93] assume fewer surfaces are specular in a scene and thus reduce the number of form-factors required for treating specular surfaces. Aupperle and Hanrahan [aupp93] automatically subdivide all patches according to the three point transport. They use a variation of the Torrance-Sparrow [torr67] specular term.

Other rendering techniques have also been reported to simulate different types of area light sources. In cone tracing [aman84], Amanatides renders images illuminated by spherical lights. In pencil tracing [shin87], Shinya *et al.* introduce the paraxial approximation theory to ray tracing, substituting a single ray by a *pencil* of rays, thus keeping information about the pencil's concentration or expansion. With this concept, it is possible to render scenes illuminated by area light sources. Unfortunately, the inclusion of these specialised rendering techniques into more conventional techniques requires extensive revisions of the algorithms that are not always possible or desirable.

3.2 Linear Light

3.2.1 Our Solution

As discussed in the previous sections, some attempts to simulate linear light sources have been proposed but no general solution other than sampling (apart from later development by Bao and Peng [bao93]) has ever been proposed for approximating the specular integral. In the next two sections, we develop a formulation of the problem that lends itself to a simple solution for both diffuse and specular terms.

3.2.1.1 Diffuse Integral

The problem of shading a surface illuminated by a linear light source can be considered in 2D. We transform the coordinate system such that P, the point representing the surface element to be shaded, is at the origin and the light source lies on the plane $Z = 0$. The diffuse integral of Equation 3.7 can be expressed in this system as

$$\int_{\mathcal{L}} \frac{(\vec{N} \cdot \vec{L_l})}{r_l^2}\, dl = (\vec{N} \cdot \vec{N_p}) \int_{\mathcal{L}} \frac{(\vec{N_p} \cdot \vec{L_l})}{r_l^2}\, dl$$

where $\mathcal{L}$ is the length of the linear light and $\vec{N_p}$ is the projection of $\vec{N}$ onto the plane $Z = 0$.

Consider an infinitely small segment dl on the light source. The projected length of dl in the direction of P is $dq = dl \cos\beta$, where β is the angle between the direction perpendicular to dl and the direction from dl to P. On the other hand, consider a circle of radius r_l centered at the origin. The length of the arc of $d\phi$ is $dq = r_l\, d\phi$. By combining these two equations, we can express dl as a function of $d\phi$ as

$$dl = \frac{r_l\, d\phi}{\cos\beta}.$$

We can replace the integral along the length of the light source by an integral along the angle subtended by the light source at P. If γ is the angle made by joining the two end points of the light source to P (two light vectors) and θ is the angle between $\vec{N_p}$ and the closest light vector[2], the integral becomes

$$(\vec{N} \cdot \vec{N_p}) \int_{\theta}^{\theta+\gamma} \frac{\cos\phi}{r_l^2} \frac{r_l}{\cos\beta}\, d\phi$$

Figure 3.7 illustrates this situation. Expressing r_l as a function of ϕ gives $r_l = \frac{d}{\cos\beta}$, where d is the distance between the supporting line of the linear light source and P.

This formulation is valid for modeling a linear light source as an infinity of point light sources where each point on the light source radiates light equally in all directions. The final diffuse integral is simply

[2] If $\vec{N_p}$ is within γ, we can take any end points light vector. In such a case, θ is negative.

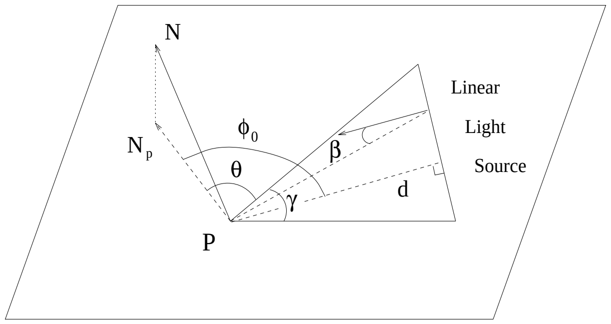


Figure 3.7: Integrating over the angle γ

$$k_d \frac{\Phi_l}{4\pi} \frac{(\vec{N} \cdot \vec{N}_p)}{d} \int_{\theta}^{\theta+\gamma} \cos\phi \, d\phi. \tag{3.10}$$

3.2.1.2 Specular Integral

By substituting the popular Phong specular term $(\vec{R_E} \cdot \vec{L})^n$ instead of the Lambertian formulation $(\vec{N} \cdot \vec{L})$ in the diffuse term,[3] we obtain the following expression for the specular integral

$$k_s F_s \frac{\Phi_l}{4\pi} \frac{(\vec{R_E} \cdot \vec{R_{Ep}})^n}{d} \int_{\theta}^{\theta+\gamma} \cos^n \phi \, d\phi. \tag{3.11}$$

It is important to note that if P is along the supporting line of the light source, the formulation of our integral is indeterminate as γ and d are both zero. In this situation, the integral can be easily handled as a special case. Take the original formulation to be

$$\int_{\mathcal{L}} \frac{(\vec{N} \cdot \vec{L})}{r_l^2} dl.$$

Since P is on the supporting line of the linear light, $(\vec{N} \cdot \vec{L})$ is constant over the entire integral unless P is actually right on the linear light. An analytical solution

$$(\vec{N} \cdot \vec{L}) \int_0^{\mathcal{L}} \frac{1}{(l+b)^2} dl = (\vec{N} \cdot \vec{L}) \frac{\mathcal{L}}{b(\mathcal{L}+b)},$$

exists, where b is the distance to the closest end point of the linear light. If P is on the linear light, the solution to this integral is infinity.

The integral for the diffuse term in Equation 3.10 has an exact solution that is easily solved. However, the integral for the specular term is harder to integrate exactly in an efficient manner.

An alternative is to approximate $\cos^n \phi$ between 0 and $\frac{\pi}{2}$. Figure 3.8 illustrates a few examples of $\cos^n \phi$ for different values of n. This family of curves is relatively smooth, monotonically

[3] $\vec{R_E}$ replaces $\vec{N}$ and $\vec{R_E p}$ replaces $\vec{N_p}$

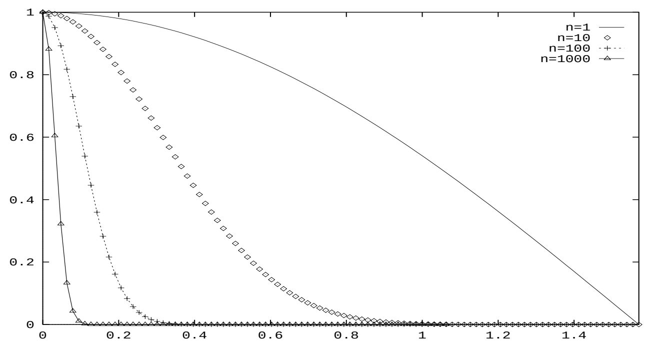


Figure 3.8: $\cos^n \phi$ for (right to left) $n = 1$, 10, 100, and 1000 and $\phi \in [0, \frac{\pi}{2}]$

decreasing, and can efficiently be approximated by polynomials which in turn are simple to integrate analytically. Several approximation techniques can be used. For a similar problem, Poulin and Fournier [poul90b] studied the use of Chebyshev polynomial approximation, piece-wise splines and the Parzen window. Because the formulation of Phong specular reflection is just an approximation based on no physical surface properties other than observation, we can approximate this function by various other functions with a similar behaviour. Blinn [blin77] reported various such functions. Bishop and Weimer [bish86] use a truncated Taylor expension of the Phong function.

The pseudo-code of Figure 3.9 computes the Chebyshev polynomial approximating the function $\cos^n \phi$, where $\phi \in [0, x_M]$. The array $a[0..D(a)]$ will contain the coefficients of the polynomial approximating the curve. Note that this pseudo-code has been provided here to explain how to compute the approximation. Tables can easily be used to speed up many of these computations.

In order to reduce the degree of the polynomial approximating the function, we cut off the function $\cos^n \phi$ at the angle where the function is less than a chosen ϵ. Looking at the curves of Figure 3.8, one can observe that as n becomes larger the area under the curve tends to be very small for the angles larger than a certain angle x_M. We can compute for given n and ϵ the value for x_M such that

$$x_M = (\cos^{-1} \epsilon)^{\frac{1}{n}}.$$

When the curve is evaluated at an angle ϕ higher than this value of x_M, it is considered zero. Choosing a value for ϵ depends on the light radiance, the surfaces characteristics, the geometry of the scene, the contrasts in the final rendering and other variables. In practice, we have found that a polynomial of degree 6 approximates within $\epsilon = \frac{1}{250}$ the $\cos^n \phi$ curve and we never observed shading artifacts in any of the images we rendered with this value for ϵ. The degree of the polynomial can be reduced further if ϵ is larger.

```
**Approximate** $(n, x_M, D(a), a[])$

/* $n$ is the degree of the function $\cos^n\theta$ */
/* $x_M$ is the maximum angle of the approximation */
/* $D(a)$ is the degree of the polynomial approximation */
/* $a[]$ contains the coefficients of the polynomial approximation */

for $i = 0$ to $D(a)$
    $sum = 0;$
    for $k = 0$ to $D(a)$
        $x_{cheby} = \cos(k\pi/D(a))$
        $x = x_M \times (x_{cheby} + 1)/2$
        if $(k = 0$ or $k = D(a))$
            $sum = sum + 0.5\cos^n(x)\times$ Cheby$(i, x_{cheby})$
        else
            $sum = sum + \cos^n(x)\times$ Cheby$(i, x_{cheby})$
    endfor $k$
    b$[i] = 2/D(a) \times sum$
    endfor $i$
for $i = 0$ to $D(a)$
    if $(i = 0$ or $i = D(a))$
        $a[i] = 0.5b[i]\times$ Cheby$(i, x_{cheby})$
    else
        $a[i] = b[i]\times$ Cheby$(i, x_{cheby})$
    endfor

**Cheby** $(degree, x)$
    if $(degree = 0)$ return $(1)$
    if $(degree = 1)$ return $(x)$
    return $(2x$ Cheby$(degree - 1, x) -$ Cheby$(degree - 2, x))$
```

Figure 3.9: Pseudo-code for the Chebyshev approximation

Once as preprocessing, and for each of the coefficients n in a scene, the Chebyshev polynomials are computed and stored. They are used in the rendering stage to efficiently approximate the specular integral.

3.2.2 Extending the Shading Model

If each point on a linear light source does not radiate light equally in every direction, a new model can be developed from the previous one. One such model assumes that a point radiates light with a cosine distribution from the directions perpendicular to the linear light source [nish85]. The modified diffuse integral is simply rewritten as

$$(\vec{N} \cdot \vec{N_p}) \int_{\mathcal{L}} \frac{(\vec{N_p} \cdot \vec{L_l})}{r_l^2} \cos\beta \, dl.$$

β is expressed as $(\phi_0 - \phi)$ where ϕ_0 is the angle between $\vec{N_p}$ and the vector formed by P and the closest point on the supporting line of the linear light. These variables are shown in Figure 3.7. The diffuse integral becomes

$$\frac{(\vec{N} \cdot \vec{N_p})}{d} \int_{\theta}^{\theta+\gamma} \cos\phi \cos(\phi_0 - \phi) \, d\phi$$

which can be integrated analytically. The specular integral is solved as before a the Chebyshev polynomial approximation.

If a point on the linear light emits even less in the directions further away from the direction perpendicular to the light, we can replace the previous distribution by $\cos^m \beta$ instead. The same polynomial approximation technique can be used for this distribution. In the case of the specular intensity, we evaluate the product of two polynomials.

$$\frac{(\vec{R_E} \cdot \vec{R_{Ep}})^n}{d} \int_{\theta}^{\theta+\gamma} \cos^n \phi \cos^m(\phi_0 - \phi) \, d\phi$$

Just like the extensions of Warn [warn83] for point light sources, the same types of extensions can be applied to the current shading model for linear light sources. For instance, one can restrict the emission of every point on the linear light source with flaps at a given angle η. This is done by making certain that the limits of the integrals, θ and $\theta + \gamma$, are always confined within the *flaps'* range, $[\phi_0 - \eta, \phi_0 + \eta]$.

Other extensions can be applied to the linear light source as a whole instead of applying functions to each point individually. It is easy to apply flaps to the entire light. One only needs to determine if the point being shaded is within the illuminated section and, if so, the shading is taken care by the functions established previously. Similarly, a function more complicated than flaps could be applied radially to the linear light source. Figure 3.10 illustrates these extensions of light emissions.

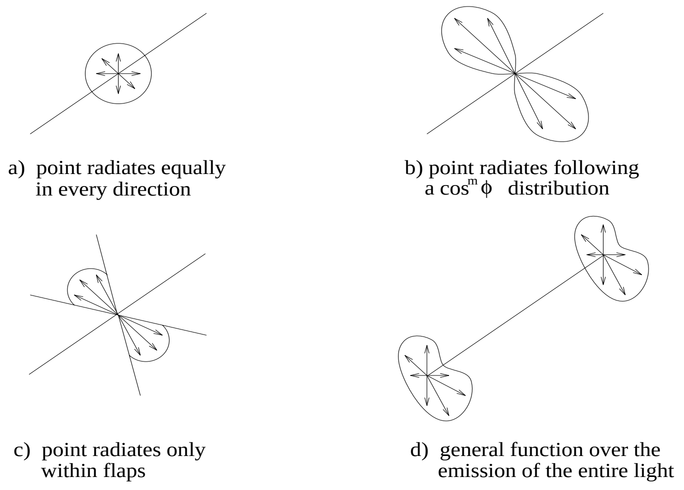

a) point radiates equally in every direction
b) point radiates following a $\cos^m \phi$ distribution
c) point radiates only within flaps
d) general function over the emission of the entire light

Figure 3.10: The extended shading model

Table I: Linear Light versus Multiple Point Lights

Primitive	Shading Only		Shading and Shadowing	
	Function	Equivalent	Function	Equivalent
Square	$0.39 + 0.14\, l$	4.36	$0.16 + 0.10\, l$	8.35
Cube	$0.34 + 0.11\, l$	6.00	$0.15 + 0.10\, l$	8.50
Sphere	$0.31 + 0.11\, l$	6.27	$0.16 + 0.10\, l$	8.40
Disk	$0.40 + 0.14\, l$	4.29	$0.09 + 0.06\, l$	16.28
Patch	$0.50 + 0.08\, l$	6.25	$0.06 + 0.04\, l$	23.50

3.2.3 Results of Shading with Linear Light

Replacing a linear light source by a series of point light sources is, as we said earlier, prone to sampling problems. The problems can appear in both the shading and the shadowing. Table I investigates the relative cost of substituting a linear light source as described in this paper by a series of point light sources. Figures 3.11 and 3.12 illustrate a typical scene we tested.

A linear light source of length ten is positioned five units above a primitive (square, cube, sphere, disk, patch) located one unit above a plane. Each primitive has been scaled so its shadow covers a similar area. The specular coefficient n for $(\vec{R_E} \cdot \vec{L})^n$ is 64 for the primitive and 32 for the plane. The relative timings are given from the implementation on our local ray tracer [aman92]. The timings are normalised by the time required to render the same scene

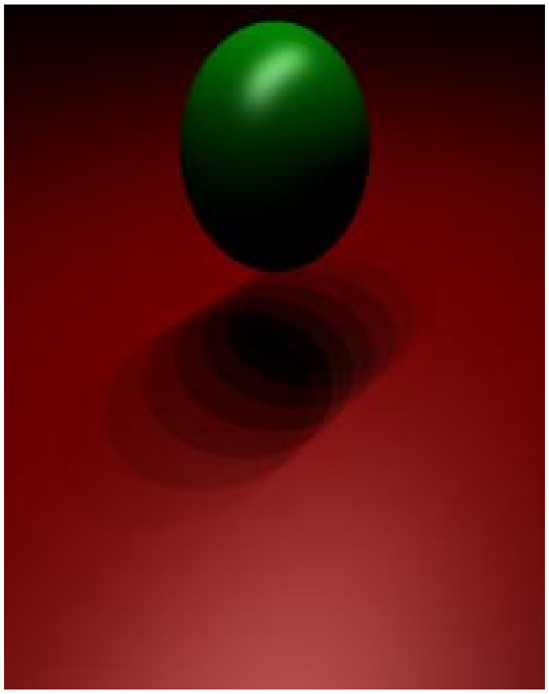


Figure 3.11: Sphere under 7 point lights

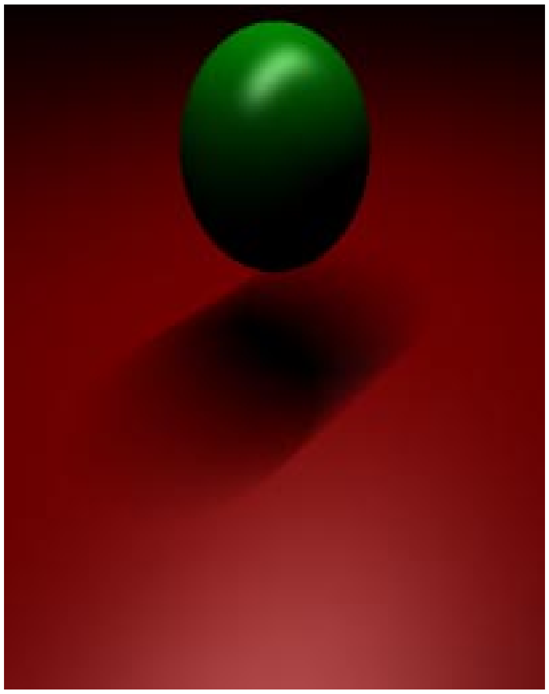


Figure 3.12: Sphere under a linear light

with a linear light source. For instance, rendering a sphere (ray casting) in our scene without any light source takes only 31% of the time required to shade it with a linear light source. If point light sources are used, 11% must be added for each light. Therefore in our benchmark, approximating a linear light source by more than seven point light sources would be more expensive. If shadows are considered, this number increases to nine. Figure 3.11 was computed with seven point light sources and Figure 3.12 with our approximate linear light source. On a 24-bit frame buffer display, the same scene was rendered with as many as 41 point light sources and Mach bands within the penumbra region are still visible. Unfortunately, due to dithering, the two images incorporated here do not give full credit to our new shading algorithm.

For a patch primitive, considering both shading and shadowing, the number of point light sources equivalent in rendering time to a linear light source is as high as 24 because we rely on a distributed ray tracing approach (described in the next chapter) to determine the projection of the patch onto the light source.

From our observations in Table I and our experience with rendering scenes with linear light sources, shading a surface illuminated by a linear light is equivalent in cost to shading the same surface illuminated by less than seven point light sources. However, our analytical solution is valid for any length of linear light source, in arbitrary position relative to the point being shaded and for any diffuse, specular and roughness coefficients. Our solution avoids the discretisation problems prevalent in all previous sampling approaches.

Although we have considered only timings to compare a number of point light sources, we argue that our solution has an even stronger advantage over sampling by the peace of mind it gives to a designer. Choosing the right number of samples depends on the lights' positions, the surfaces' positions, the surfaces' characteristics, the shadows cast and the viewing position. Not all of these can be fully automated so it is possible that during an animation sequence several frames will exhibit some shading artifacts. Rerendering these frames with more samples is a cost that should be factored in, but that is difficult to evaluate. For these reasons we believe our analytical solution to shading with linear light sources is an important contribution to shading with extended light sources.

Shadowing increases the complexity of rendering scenes illuminated by linear light sources. We will show in the next chapter that our technique, especially after the introduction of some algorithmic speedups, is still quite competitive.

Figure 3.13 shows an image of a desk illuminated by a single linear light source. The soft shadows add a more realistic look to the scene. Figure 3.14 shows a typical cafeteria illuminated by 12 linear light sources (2 per white rectangle lying on the ceiling). Notice how by simply increasing the dimension of the light sources to linear, the scene gives an impression closer to what is usually achieved at a much higher computing cost by radiosity algorithms.

In the next section, we will present how our analytical solution to shading with linear light sources has been extended to shading with polygonal light sources.

3.3 Area Light

As described in the previous sections, the full shading of a surface illuminated by an area light source had never been completely computed before. The diffuse reflection could be computed analytically, but the specular reflection was only approximated by point sampling.

3.3.1 The Diffuse Integral

Tanaka and Takahashi [tana91b] extend our formulation of the specular term from linear light sources to polygonal light sources by incorporating some of their results on highlighting rounded edges [tana90][tana91a]. By assuming a Lambertian distributed polygonal light, they use the solution proposed by Nishita and Nakamae [nish83] to compute the diffuse reflection:

$$L_{pixel} = k_d \frac{L}{2\pi} \sum_{i=1}^{m} \beta_i \cos \delta_i .$$

Please refer to Section 3.1.4 and Figure 3.9 for more information about the diffuse solution.

3.3.2 The Specular Integral

The development for the specular integral is more complex. The use of the specular reflection from Phong is important due to its property that the specular reflectivity is rotationally

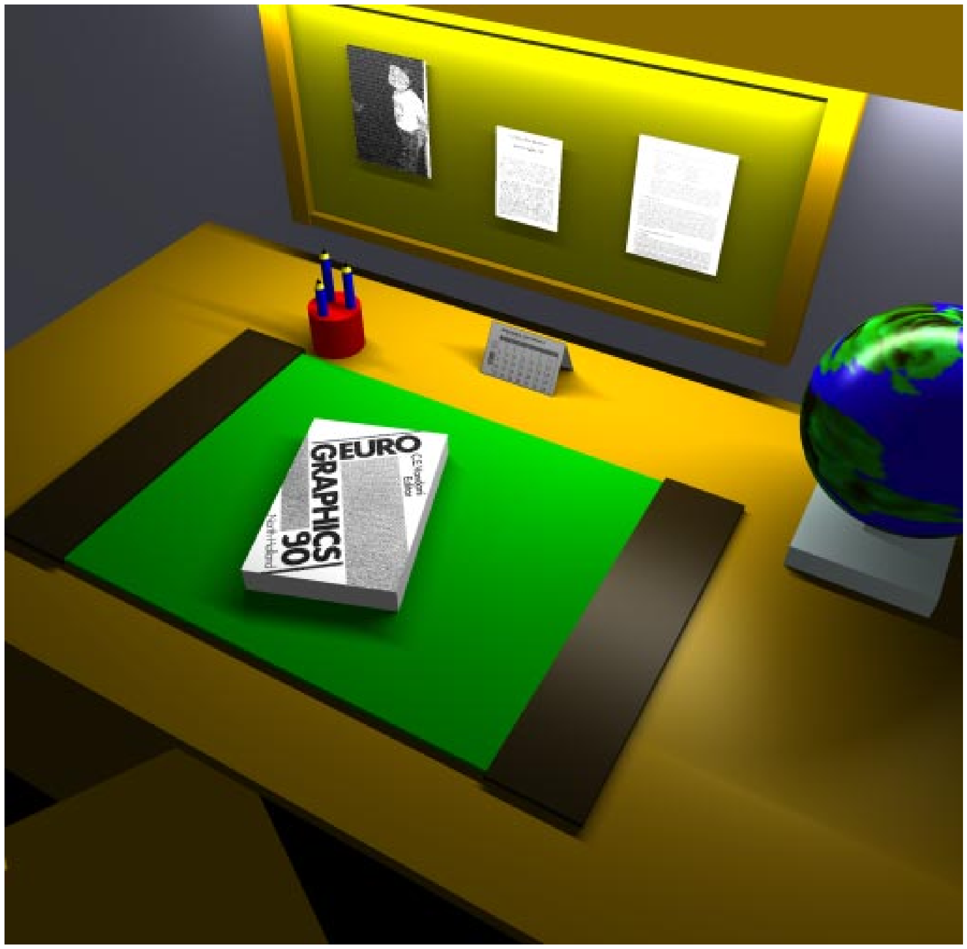


Figure 3.13: A desk under a linear light

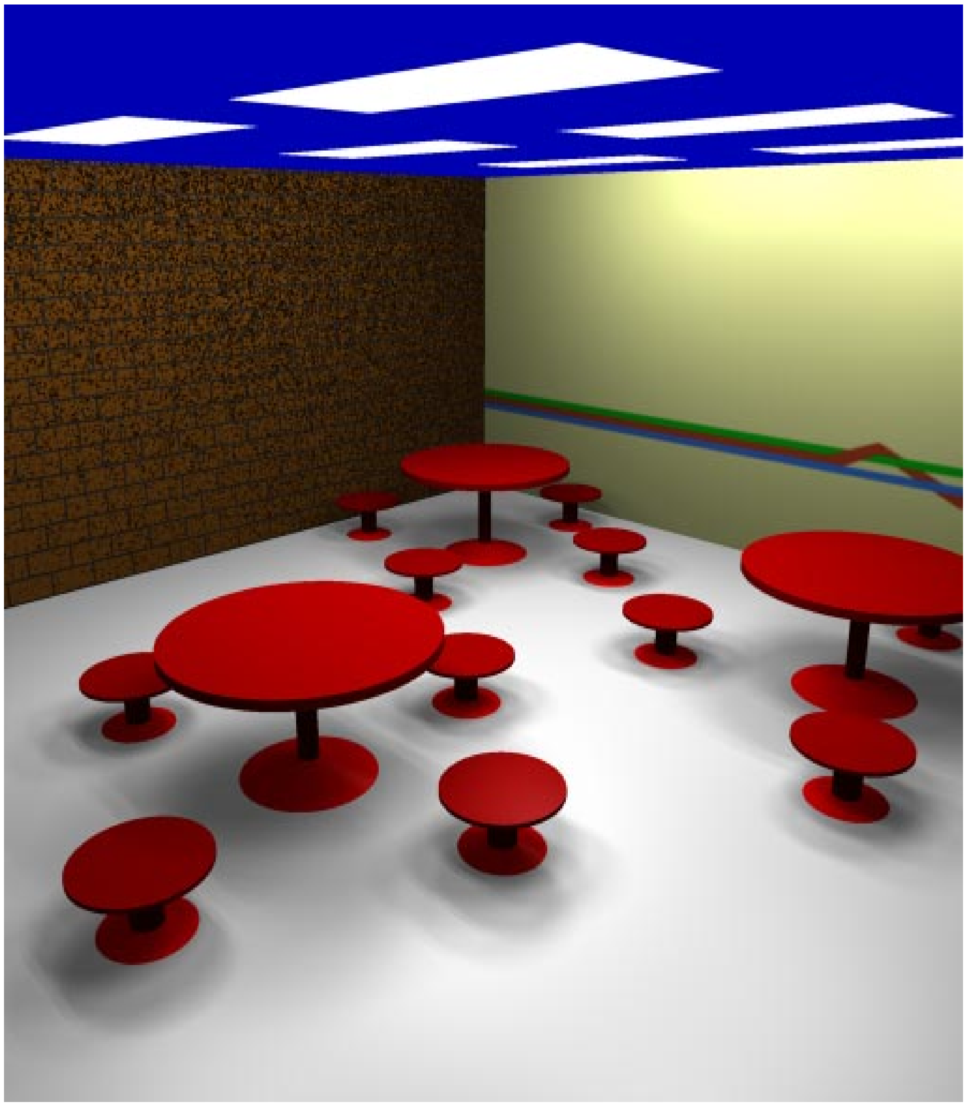


Figure 3.14: A cafeteria under 12 linear lights

symmetric around the perfectly reflected eye direction $\vec{R_E}$. As we did, Tanaka and Takahashi transformed the standard Cartesian formulation into polar coordinates around this vector. In this form, the specular integral corresponds to

$$L_{pixel} = k_s F_s L \frac{(n+1)}{2\pi} \iint\limits_{\Omega(w_1,\ldots,w_m)} \cos^n \theta \sin \theta d\theta d\phi \tag{3.12}$$

where	n	is the Phong roughness coefficient
	θ	is the angle between $\vec{R_E}$ and $\vec{v_i}$, a vector pointing towards a vertex i of the polygonal light
	w_i	is the projection of vertex i onto the unit sphere
	ϕ	is the rotational angle around $\vec{R_E}$
	$\Omega(w_1, \ldots, w_m)$	is the projected light polygon onto the unit sphere.

Look at the area light from point P on the surface element being shaded. Assume that the up vector ($Z = 1$) is $\vec{R_E}$ for simplicity. Look first at a light triangle defined by a point along $\vec{R_E}$ and two vertices of the polygonal light source. Because we deal with a Lambertian light, the radiance of this light is the same as the radiance of a light projected onto the unit sphere centered at P. Figure 3.15 illustrates several variables used in the rest of this section.

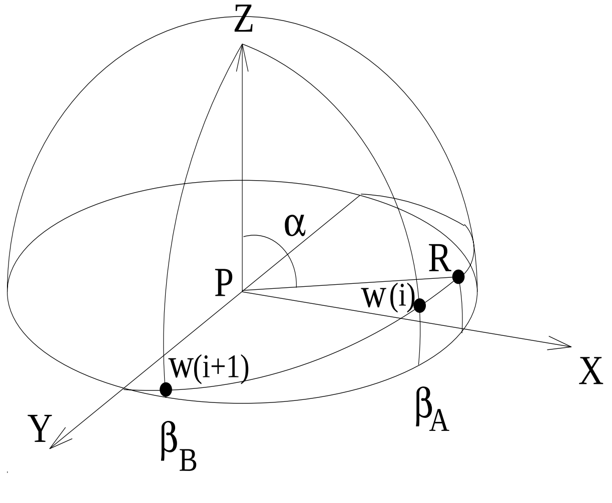


Figure 3.15: Integrating the specular term

The solid angle is defined by three great circles, two passing by $Z = 1$ and one passing by w_i and w_{i+1}. This latter great circle has a single point R closest to $Z = 1$ at an angle we call α. We use the projection of R onto the plane $Z = 0$ to define the origin ($\theta = \frac{\pi}{2}$ or $(X = 1, Y = 0, Z = 0)$).

Assume the integral is evaluated from w_i (β_A on $Z = 0$) to w_{i+1} (β_B on $Z = 0$). Solving the double integral leads to

$$L_{pixel} = k_s F_s \frac{L}{2\pi} \left[(\beta_B - \beta_A) - (H(\alpha, \beta_B) - H(\alpha, \beta_A)) \right]$$

where

$$H(\alpha, \beta) = \int_0^\beta \left(\frac{\cos^2 \phi}{\tan^2 \alpha + \cos^2 \phi} \right)^{\frac{n+1}{2}} d\phi.$$

For our function $\cos^n \phi$, $H(\alpha, \frac{\pi}{2})$ decreases monotonously between 0 and $\frac{\pi}{2}$ and Tanaka and Takahashi used a Chebyshev polynomial approximation of $H(\alpha, \frac{\pi}{2})$ to integrate $H(\alpha, \beta)$.

To integrate the entire polygonal light, each edge is integrated in counter-clockwise order as seen from P and the result of the integral is multiplied by a function

$$F(w_i, w_{i+1}) \begin{cases} 1 & \text{if } (w_i, w_{i+1}) \text{ is counter-clockwise} \\ -1 & \text{otherwise} \end{cases}$$

Each integral is then summed to provide the final specularly reflected radiance.

3.4 Conclusion

In this chapter, we presented an analytical solution for shading surfaces illuminated by linear light sources and explained how our solution was extended to shade surfaces illuminated by area light sources. Rendering with these illuminants is more expensive than rendering with just the same number of directional and point light sources. However the increase in realism justifies the added complexity. Moreover, the increase in rendering cost is not as significant as increasing the number of point light sources to produce a similar lighting effect.

The next chapter addresses in more detail the shadowing computation as the light source dimensions and the scene complexity increase.

Chapter 4

Shadows

In previous chapters, we talked about shading without paying much attention to another phenomenon created by the presence of light sources: shadows. A shadow is produced when a surface element acts as a blocker of some of the light emitted by another surface element. The blocker stops some of the radiant energy along its path. The energy is then absorbed or redirected. The volume of points for which at least a portion of the light is occluded by this blocker is called a *shadow volume*. This shadow is of interest to us when the radiance on a third surface element has to be computed since we must then characterise the illumination on this surface element. This is achieved by intersecting this surface element with the scene shadow volumes. A shadow is *cast* on a surface when this surface intersects a shadow volume. This intersection defines a *cast shadow*. Shadows can provide important information about the shape of objects, their relative positions and the nature of their surfaces (transparency). They also can reveal some information about light locations and shapes. While shadows can also be produced by secondary lights (i.e. a surface reflecting light can act as a indirect light), we will restrain our discussions in this chapter to shadows produced from an emitting light source.

The presence of shadows can reduce the cost of computing shading information. After all, when a point is considered completely in shadow of a light, the direct contribution from this light is zero. Unfortunately, determining if a point is completely, partially or not in shadow is a fairly expensive process that often more than offsets the savings in shading.

We will consider in this chapter only shadows produced by opaque blockers. Semi-transparent material blocking light for certain wavelengths can be considered as opaque blockers if we deal with the shadows on a wavelength basis. However, shadows produced by refractive surfaces are more complicated since they can redirect light depending on their geometry, their index of refraction and the wavelengths of the light. Many of the properties of shadows studied in this chapter do not apply to refractive blockers.

In the following sections, we will first give a few definitions of concepts linked to shadows. Then we will go through a series of properties to establish a foundation for our understanding of what is involved in correctly computing shadows. This will give us some insights on the effectiveness of some shadowing algorithms. With this in mind, we will present our shadowing algorithm for scenes illuminated by linear light sources and two culling algorithms to reduce the number of occluding candidates. Finally, we will review and discuss various algorithms developed to determine the shadows from area light sources.

4.1 Definitions

The *umbra volume* of a light source formed by an occluding object is the set of points such that every point of the light source is blocked by at least a point of the occluding object.

Note that the umbra volume so defined is generally a volume which can be finite, infinite or null. The umbra cast on another object is the set of points of the object (restricted to its surface for an opaque object) that also belong to the umbra volume as previously defined.

The *penumbra volume* of a light source formed by an occluding object is the set of points such that at least one point of the light source is blocked by at least one point of the occluding object while at least one point of the light source is still visible.

Note that the penumbra volume is generally a volume or a set of disconnected volumes. If a penumbra volume is formed by a light and a single blocker, this penumbra is always infinite.

If the light source is a direction (point at infinity) or a point, there exists no penumbra volume as a point is either blocked or not blocked by a point of the occluding object. The points blocked belong therefore to the umbra volume.

The *shadow* is the volume formed by the union of both volumes, the umbra and the penumbra volumes. Each point not in shadow is *illuminated* by the entire light source.

In the next section, we will look at some theoretical characteristics of shadows. We will start by studying the shadows in the plane (2D) because fundamentally, shadows produced by a linear light source need to consider only the visible portion of a line segment viewed from a surface element to shade. We represent this surface element by a single point on its surface for simplicity. Also, some concepts and properties are simpler to illustrate and understand in the plane and can be generalised to the 3D space.

4.2 Properties of Shadows

4.2.1 Directional and Point Light

As already mentioned in the previous section, a directional light and a point light source cannot generate a penumbra volume from a blocker. A point is or is not in shadow. The umbra generated by one or more blockers is the union of every umbra volume, where the volumes are not necessarily connected. This is illustrated in 2D by Figure 4.16.

For a survey of shadowing algorithms for directional and point light sources, we refer the reader to the survey of Woo *et al.* [woo90].

4.2.2 Linear Light

In the plane, a linear light is a line segment and a blocker is an opaque line segment (or a set of line segments) resulting from the intersection of a 3D primitive with the plane defined by the linear light source and the point to shade. The following properties apply to a single line segment acting as a linear light and a single line segment acting as a blocker.

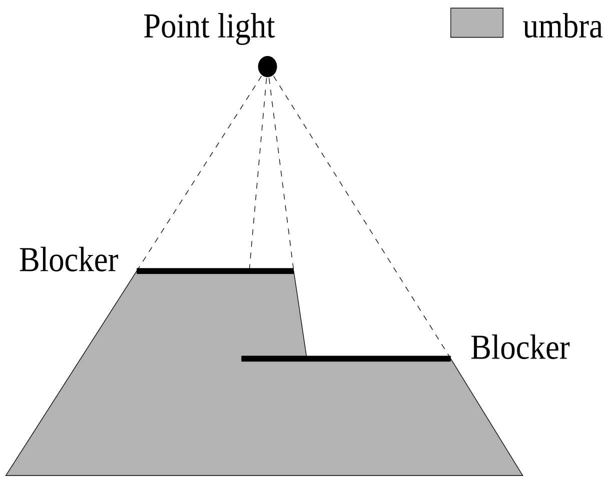


Figure 4.16: Umbra region in 2D

4.2.2.1 One Light and One Blocker

Consider as input to the 2D shadow determination problem a blocker line segment and a line segment defined as a linear light source. Figure 4.17 shows a light segment L and a blocker segment B used in the next definitions. All the definitions are valid as long as the supporting lines of the blocker and of the light do not intersect in the light segment or in the blocker segment. Otherwise, the results are applicable on both sides individually if the light or the blocker is split at their intersection point into two separate segments.

Let b_i be a vertex of the blocker segment B. Consider the two lines defined by this vertex and the two light vertices. The line with the smallest angle α with the segment B is called *minimum blocker extremal line* (Minb_i) and its associated vertex is called *minimum blocker extremal vertex*. The *maximum blocker extremal line* (Maxb_i) and its associated vertex, the *maximum blocker extremal vertex* are linked to the other vertex of the light L. Each of these lines subdivides the plane into two regions. The minimum blocker extremal line divides the plane such that the light segment is on one side and the blocker segment is on the other. We define the *positive* side $(\text{Min}b_i)^+$ as the one containing the light segment. The maximum blocker extremal line divides the plane such that both the light and blocker segments are on one side of the line. We define this side $(\text{Max}b_i)^-$ as the *negative* side. We also define sides for the regions divided by the supporting lines of the light segment and the blocker segment. The *positive* side of the light supporting line (L^+) contains the blocker segment and the *positive* side of the blocker supporting line (B^+) contains the light segment.

Similar definitions are used [camp91] for extremal planes of convex polygonal lights and blockers. Campbell and Fussell prove many properties of shadows in 3D with these definitions.

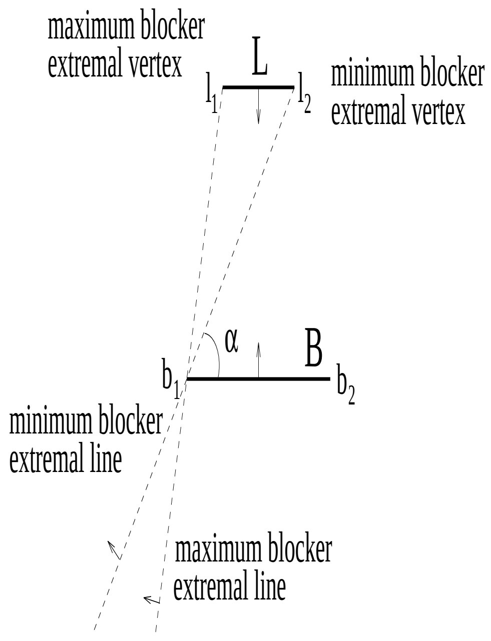


Figure 4.17: Extremal lines

Since our work is mainly based on theirs, we will simply enumerate some properties of shadows in the plane. When the proofs can easily be extended from the ones of Campbell and Fussell, we will omit them and refer the reader to the original report [camp91].

Shadow Properties in the Plane:

1. No point on the positive side of either minimum blocker extremal lines or on the positive side of the blocker B may be occluded by B. This represents the illuminated region and is expressed in set operations as $B^+ \cup (\mathrm{Min}b_1)^+ \cup (\mathrm{Min}b_2)^+$.

2. Any point in the area closed by the negative sides of both minimum blocker extremal lines and the negative side of blocker B is at least partially occluded by B. This represents the shadow region and is expressed in set operations as $B^- \cap (\mathrm{Min}b_1)^- \cap (\mathrm{Min}b_2)^-$.

3. Any point on the positive side of either maximum blocker extremal lines receives some light from L while still in shadow. This represents the penumbra region and is expressed in set operations as $[B^- \cap (\mathrm{Min}b_1)^- \cap (\mathrm{Min}b_2)^-] \cap [(\mathrm{Max}b_1)^+ \cup (\mathrm{Max}b_2)^+]$.

4. Any point in the area formed by the intersection of the negative sides of both maximum blocker extremal lines and the negative side of blocker B do not receive any light from L. This represents the umbra region and is expressed in set operations as $B^- \cap (\mathrm{Max}b_1)^- \cap (\mathrm{Max}b_2)^-$.

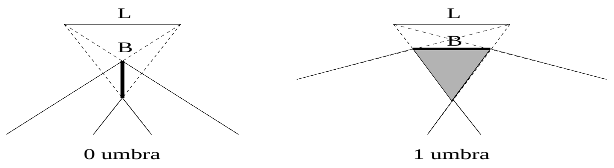


Figure 4.18: Linear light umbra region in 2D

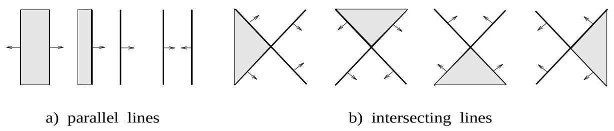


Figure 4.19: One umbra

5. A single linear light and a single blocker can generate 0 or 1 umbra region but not more than 1 umbra region.
   Proof:
   Figure 4.18 illustrates cases with 0 and 1 umbra region. Recall the set definition of an umbra: $B^- \cap (\text{Max}b_1)^- \cap (\text{Max}b_2)^-$ where each region is bounded by an oriented line. Look first at $(\text{Max}b_1)^- \cap (\text{Max}b_2)^-$. If the two lines are parallel, they define three regions and only at most one region can be on the negative side of both lines. Otherwise the two lines intersect and once again only one region can be on both negative sides. Figure 4.19 shows all the possible cases. Note that for intersecting lines, all the cases are identical by rotation. The supporting line of B can only divide this region into two, leading only to a single umbra region.

6. A single linear light and a single blocker can generate 0,1 or 2 penumbra region(s) but not more than 2 penumbra regions.
   Proof:
   Figure 4.20 illustrates cases with 0, 1 and 2 penumbra regions. Recall the set definition of a penumbra: $[B^- \cap (\text{Min}b_1)^- \cap (\text{Min}b_2)^-] \cap [(\text{Max}b_1)^+ \cup (\text{Max}b_2)^+]$. From the previous proof, we can deduct that $[B^- \cap (\text{Min}b_1)^- \cap (\text{Min}b_2)^-]$ determines a maximum of one region. Call this region A. Then we have $A \cap [(\text{Max}b_1)^+ \cup (\text{Max}b_2)^+]$ which is equivalent to $[A \cap (\text{Max}b_1)^+] \cup [A \cap (\text{Max}b_2)^+]$. Since each intersection leads to a maximum of a single region, there may not be more than 2 penumbra regions.

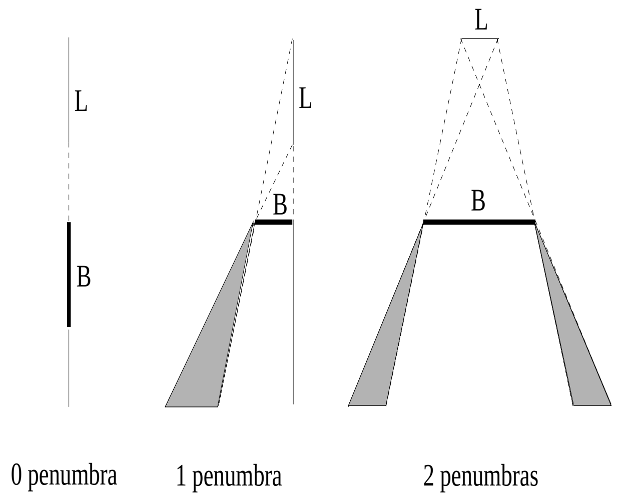


Figure 4.20: Linear light penumbra region in 2D

4.2.2.2 One Light and Many Blockers

The *union* of shadow regions defines a set of points such that each point belongs at least to one shadow region.

1. The union of the shadow regions from two blockers forms an umbra region that is equal or larger than the union of the individual umbra regions. This union forms a penumbra region that is equal or smaller than the union of the individual penumbra regions.
   Proof:
   A point in the umbra of one blocker and in the penumbra of another blocker is in the umbra of the union since the first blocker completely occludes the light. A point in two penumbra regions can be in the umbra or penumbra of the union since the portion of the light visible from this point can be completely occluded by another blocker. This explains the possibly smaller penumbra and larger umbra of the union of shadow regions.

To illustrate this situation, consider Figure 4.21. We label Ui and Pi the umbra and penumbra regions, respectively, formed by blocker $Bi, i \in \{1, 2\}$. The region hashed and labelled $U12$ is in penumbrae of both blockers $B1$ and $B2$ but together, $B1$ and $B2$ completely occlude the light L and therefore this region is in the umbra of the union of the shadows $((P1) \cup (P2) \cup (U1) \cup (U2))$.

If two blockers are connected, the two intersecting penumbra regions formed at the connecting vertex are identical except that the orientations of minimum and maximum blocker extremal lines are in opposited directions. Since there is no gap between these two blockers (i.e. no plane separating the two blockers can intersect the light), this entire region is simply converted into umbra.

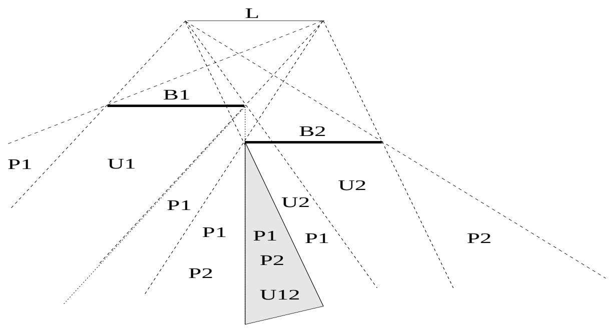


Figure 4.21: Smaller penumbra and larger umbra

When the number of light sources is increased, the shadows must be computed for each light source individually.

The properties enumerated in this section usually extend to 3D for convex lights and convex polygonal blockers. The next section looks at shadows in 3D. It will show that in general, the addition of an extra dimension introduces a substantial increase in shadow complexity.

4.2.3 Area Light

One important difference when going from line segments in the plane to polygons in 3D is the notion of concavity and convexity. A line segment can only be convex while in 3D, a polygon can be convex or concave. Most of the properties in this section assume convex lights and convex blockers. These results are applicable to concave lights and blockers when generalisable to a union of convex polygons.

4.2.3.1 One Light and One Blocker

Most of the following results have been extracted from work by Campbell and Fussell [camp91].

Consider as input to the 3D shadow determination problem a convex blocker polygon and an area light source defined as a convex polygon. Figure 4.22 shows a triangular light L and a rectangular blocker B used in the next four definitions.

Let (b_i, b_{i+1}) be any edge of the blocking polygon B. Consider all the planes defined by this edge and all the vertices of light L. The plane with the smallest angle with the supporting plane of B is called *minimum blocker extremal plane* and its associated vertex is called *minimum blocker extremal vertex*. Similarly for the light, let (l_i, l_{i+1}) be any edge of the light L. Consider all planes defined by this edge and all the vertices of blocker B. The plane with the smallest angle with the supporting plane of L is called *minimum source extremal plane* and its associated vertex is called *minimum source extremal vertex*. The *positive* side of the minimum blocker and source extremal planes contains the light polygon. The *maximum blocker and source extremal*

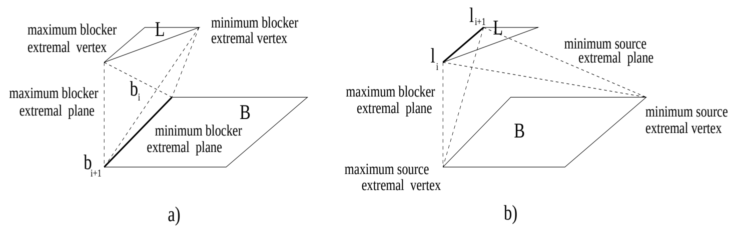

maximum blocker extremal vertex
L
minimum blocker extremal vertex
maximum blocker extremal plane
b_i
B
minimum blocker extremal plane
b_{i+1}
a)
l_{i+1}
L
l_i
minimum source extremal plane
maximum blocker extremal plane
B
minimum source extremal vertex
maximum source extremal vertex
b)

Figure 4.22: Extremal planes

planes and *vertices* are the corresponding concepts associated with the largest angle with the supporting plane of B instead. The *negative* side of the maximum blocker and source extremal planes contains both the blocker and light polygons.

Campbell and Fussell start from these definitions to prove the following:

1. No point on the positive side of a minimum blocker or source extremal planes can occlude B.

2. The occlusion volume (umbra and penumbra) is the volume resulting from the intersection of the negative side of the blocker B with the negative sides of the minimum blocker extremal planes from every edge of B and the negative sides of the minimum source extremal planes from every edge of L. This is equivalent to say that for a convex light source defined by vertices and a convex blocker, the shadow volume is the convex hull of the umbra volumes created by the blocker from each vertex of the light source.

3. The umbra volume corresponds to the intersection of the negative side of the blocker B with the negative sides of the maximum blocker extremal planes from every edge of B. In other terms, for a convex light source defined by vertices and a convex blocker, the umbra volume is the intersection of the umbra volumes created by the blocker from each vertex of the light source.

4.2.3.2 One Light and Many Blockers

The previous definitions can be applied individually to each convex light and each convex blocker to characterise each volume as illuminated or in shadow (umbra or penumbra) of a given blocker. With the same arguments than we used in the previous section on shadows of segments in the plane, we can also add that:

1. The penumbra created by a collection of convex blockers B_i illuminated by a convex light source L is at most the union of the penumbrae created by each blocker B_i.

2. The umbra created by a collection of convex blockers B_i illuminated by a convex light source L is at least the union of the umbrae created by each blocker B_i and at most the union of the umbrae of the pair-wise intersections of the penumbrae.

These volumes have been used in many algorithms [nish83] [camp90] [chin92] [poul92b] [bao93] to speed up shadow determination. These volumes are used by these algorithms to characterise the visibility of a light source from a surface element to shade. Ultimately the algorithm can identify the blocker or even the edges of the blocker that must be reprojected onto the light source to determine the visible portion of the light illuminating the surface element to shade.

Much work in radiosity has lately paid attention to the discontinuities in shading produced by shadows [heck91b] [heck92b] [heck92a] [camp91] [sale92] [lisc92] [lisc93] [tell92] [stew93] in order to produce surface meshes more suitable for proper radiance interpolation. The surfaces are intersected with the volumes. For a region on a surface that resides within a single volume, the radiance is sampled at a few surface elements and the radiance function is reconstructed for the entire region.

If the radiance over a light source is smooth (C^∞), the radiance over a planar receiver will also be smooth unless a change occurs in the visibility of the light source. Discontinuities in the radiance function are introduced by the presence of shadows. Heckbert [heck91b] identifies three types of radiance discontinuities (D^0, D^1 and D^2) on a receiver illuminated by a light source with a smooth radiance. A discontinuity is D^k at x if the radiance function is C^{k-1} continuous at x but not C^k. If the radiance on the light source is smooth, discontinuities of the radiance on the receiver will occur at special *visual events* in space that is when light vertices or edges and blocker edges align.

In Campbell and Fussell report's [camp91], only minimum and maximum extremal planes are considered to determine the illuminated, umbra and penumbra region(s) of a single light and a single blocker. However to produce accurate meshes within shadow regions, all discontinuities are of interest.

These regions have a similar interpretation in computing the aspect graph of polygonal scenes in computer vision [gigu90] [gigu91]. In an aspect graph, the projections of lines semantically identical (i.e. defining the visibility of a face) are regrouped for different views of a scene. Stewart and Ghali [stew93] construct a subset of the aspect graph. The σ-aspect graph, where σ is a polygonal light, determines the views in space such that the visible light polygon keeps the same structure.

A D^0 discontinuity occurs at the boundary of a shadow volume created by a directional or a point light source. It can also occur with a polygonal light in a situation such as illustrated in Figure 4.23. The blocker B intersects the rectangle receiver. The radiance on one side of the receiver is non-zero while the radiance on the other side is zero. A scene made of m edges and illuminated only by polygonal lights can produce $\Theta(m^2)$ D^0 discontinuities [lisc92] as $\Theta(m)$ edges can intersect $O(m)$ faces.

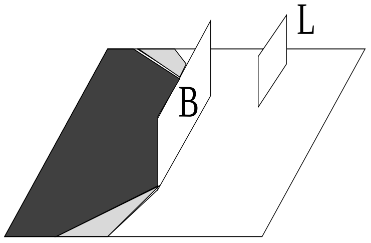


Figure 4.23: A D^0 discontinuity

D^1 and D^2 discontinuities occur at edge-vertex (EV) and edge-edge-edge (EEE) events when the blocker does not intersect the receiver but lies between the light and the receiver. An EV event produces a D^1 or a D^2 discontinuity. Figure 4.24 shows the change of visibility of a light. In a), when a vertex of light L and an edge of blocker B align, the light becomes visible,

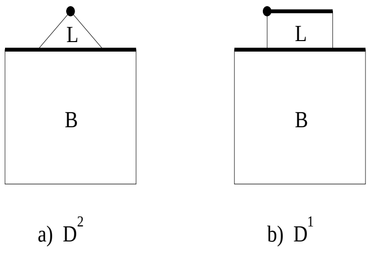


Figure 4.24: EV event

producing a D^2 discontinuity. After, the visible area of the light grows quadratically. In b), two edges of both light L and blocker B are parallel. When these edges align, light L becomes visible, producing a D^1 discontinuity. After, the visible area of the light grows linearly. A scene made of m edges can produce $O(m^2)$ EV events [heck91a]. An EEE event generally produces a D^2 discontinuity. The envelope of such an event is defined by a quadric ruled surface as it is determined by a set of lines adjacent to all three edges at the same time. Figure 4.25 shows the change of visibility of a light L when a light edge and two edges of blockers $B1$ and $B2$ align. After, the visible area of the light grows quadratically. A scene made of m edges can produce $O(m^3)$ EEE events [heck91a].

In radiosity, where a receiver with discontinuities can itself be considered as a secondary light source, a discontinuity D^k can propagate discontinuities D^{k+1} and D^{k+2} on other receivers. Since in the framework of this thesis we deal only with local illumination, we can concern ourselves only with D^0, D^1 and D^2 discontinuities.

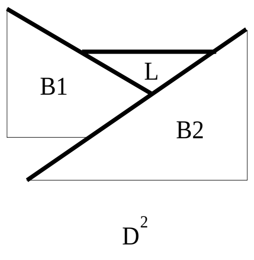


Figure 4.25: EEE event

It is interesting to note that a linear light in a 3D environment can also produce D^0, D^1 and D^2 discontinuities. In flatland [heck91a], a linear light can introduce only D^0 and D^1 discontinuities.

When the number of area light sources is increased, the shadows must be computed for each light source individually. Then, to determine the radiance at a given location, one must find the regions within which the surface element being shaded resides and must determine the portion of each light reaching the surface element.

4.3 Shadowing with Linear Light

In the previous chapter, we presented an analytical solution to shading surfaces illuminated by a linear light. This solution provides a good approximation of real illuminants like fluorescent and neon tubular lights. The proposed shading therefore gives a better impression of realism. However the shadows cast by objects illuminated by a linear light must also be properly computed in order to not destroy but even accentuate this realism.

These shadows can be generated by approximating the linear light by a series of collinear point lights located along the linear light. Unfortunately they will be prone to the same sampling and rasterisation problems we faced for the shading. Instead of producing a smooth gradient in the penumbrae along the direction of a linear light, discrete regions could appear, thus destroying the perception of a single linear light.

Nishita *et al.* [nish85] describe an algorithm to compute the shadows produced by linear lights in a scene made exclusively of convex polygons. Given a linear light source and a polygon casting a shadow on a planar surface, the polygon vertices are projected onto the supporting plane of the surface, taking as center of projection alternately each of the two end points of the light. Once the contour lines of the shadows from both end points are determined, the convex hull of these shadow polygons is computed and stored as the entire shadow region. The intersection of the contour lines is also stored as the umbra region. Later on during the scanline process, if a surface element is inside a penumbra region, the polygons producing this penumbra are projected back onto the light source and the shading is computed for the segments of the

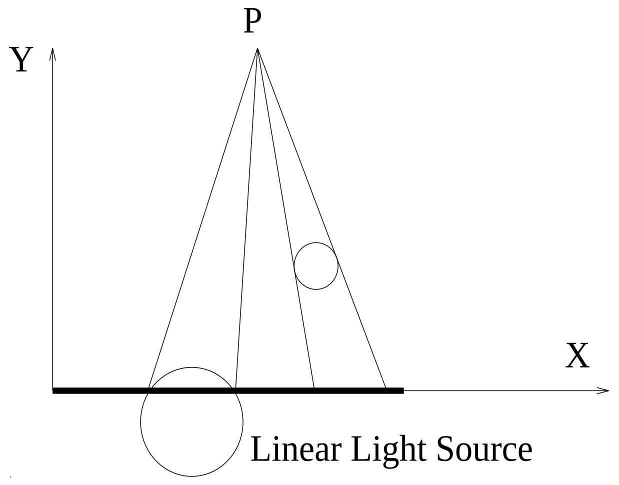


Figure 4.26: Visibility of the Light Source

light source that are visible from the surface element to shade. Otherwise if the surface element is illuminated, the full shading is computed without any extra shadowing test. Finally, if the surface element is in an umbra region, only the ambient reflection is considered.

Later, Nishita *et al.* [nish92] observed that some optimisation can be achieved for scenes with many aligned fluorescent lights on a ceiling. Rather than classifying each shadow region [nish85], they take advantage of the fact that several linear lights (segments) might lie along a single line and apply a scanline rendering technique to determine which portions of each linear light illuminate the surface element to shade. The shading is then computed for these visible linear light segments.

4.3.1 Primitive-based Shadowing

Instead of determining each shadow region, we decided to compute the visible portion of the linear light source from the point being shaded. In that sense, this approach to shadowing is basic but allows primitives other than polygons. When a surface element centered at point P has to be shaded, the light source is first *cut* by the plane tangent to the normal at P because no light can shine on P from an angle of more than $\frac{\pi}{2}$ of the normal at P. If a portion of the light source is visible from P, a triangle (the *light triangle*) is formed by joining the end points of the linear light and P. If an object intersects this triangle, its intersection casts a shadow on P. This intersection is projected onto the light and the shading is computed only for the *visible* segments of the light. In the case of the specular integral, the visible segments must also be *cut* such that for all segment i, $|\theta_i| < \frac{\pi}{2}$ and $|\theta_i + \gamma_i| < \frac{\pi}{2}$ in Equation 3.11 (*i.e.* $\vec{Re} \cdot L > 0$).

Let O be an object that may cast a shadow on P. The first test consists in intersecting the plane supporting the light triangle (the *light plane*) with object O. If the computations involved in intersecting O with the light plane are expensive, the simpler bounding volume of the object can be used instead to quickly reject the objects trivially not intersecting the plane. If it is determined the object does not intersect the light plane, then object O does not cast any shadow on P. Otherwise, the light triangle is transformed into the object coordinate system where more accurate tests can be performed. Afterwards, if indeed the object O intersects the light plane, it is transformed in such position that the light lies on the X axis with one end at the origin and the intersection point P lies on the plane $z = 0$ on the positive side of Y (see Figure 4.26). This involves only a series of rotations and translations, leaving the basic definition (shape) of the primitive unchanged. The remaining work consists in identifying the intersection between a primitive and the plane $z = 0$ and projecting in 2D the result of this intersection onto the X axis.

As an example, take a sphere that has been transformed by a series of scaling and shearing operations. The light triangle is first transformed in the sphere coordinate system where it is easier to determine if the sphere intersects the light plane. The intersection between the plane and a sphere is either null, a point or a circle. In the first two cases, no shadow is produced. For a circle, the two tangent points from P or the intersection between the circle and the X axis define the visible segments of the light source. Again, Figure 4.26 illustrates these two cases.

For simple objects like polygons, the transformation in the object coordinate system is not required and the transformation matrix bringing the light triangle onto the plane $z = 0$ and the linear light along the X axis needs to be computed only once per intersection point P. For other more complicated objects like cubic patches, finding the intersection between a light triangle and the primitive can be a very difficult task. Rather than not treating the shadowing of these primitives in the presence of a linear light, we offer an approximate solution based on distributed ray tracing. Rays are shot from P towards the linear light and intersected only with this primitive. The number of rays is determined as a function of the angle made when joining P with the two end points of the light. In a first pass, a regular set of rays are shot to roughly determine the edges of the intersection with the light plane. In a second pass, additional rays are used to refine the projection edges. At this stage, some random jittering is added to the ray directions, substituting aliasing by noise, which is very effective within the penumbra region. This technique has the advantage of being rather general to approximate the visibility of the linear light from P. In fact we rely on this approach for every primitive difficult to intersect with a plane. Unfortunately, it is not flawless as the edges of the intersection are basically determined from a limited set of rays and ultimately it has the same problems than any sampling technique. Although the results so far have been visually pleasing, we advocate the proper intersection of each primitive with the light plane. For instance, Fournier and Buchanan [four92c] subdivide a Bézier cubic patch into bilinear patches. These can efficiently be used instead of the patch itself for shadowing computation.

It is important to note here that as soon as we know that the whole light is hidden, the shadowing process is stopped. The intersection/projection scheme described in this section can be relatively expensive. It is therefore essential to eliminate the objects not intersecting the light triangle as quickly as possible. The next two sections describe two algorithms that we implemented to reduce the number of objects that are candidates for casting a shadow on the surface element to shade.

4.3.2 Light Triangle 3D Scan Conversion

We implemented our shading and shadowing algorithms into our local ray tracer. Our first acceleration scheme for shadowing is an extension of a standard ray tracing acceleration scheme. To reduce the number of objects that a ray must test intersection with, we subdivide the scene into regular cubes, also called voxels. Each voxel contains a list of objects intersecting this particular voxel. When a ray traverses a voxel, only the objects intersecting this voxel have to be tested. We use the regular grid traversal described in Amanatides and Woo [aman87]. Once an intersection point is found, a light triangle is formed. This light triangle is scan converted in the 3D grid of the scene. The objects intersecting a voxel intersected also by the light triangle are gathered. This possibly smaller set of objects is then tested and projected onto the light source as described in the previous section.

Although we implemented this technique with regular grid traversal, it can easily be adapted to various other space subdivisions like irregular grids, octrees, *kd*-trees and hierarchies of bounding volumes. While the scan conversion of light triangles through these structures might prove to be more complex, it might reduce further the number of structure elements visited and the number of candidates.

4.3.3 Linear Light Buffer

In our second scheme, a modified light buffer [hain86] for linear light sources has been implemented. In the traditional light buffer, a cube is placed around a point light source. As a preprocessing step, each object in the scene is projected onto the faces of the cube. Each face is subdivided into regions and each region contains a list of objects projecting onto it. Each list can be ordered by the distance to the light. When a surface element to shade has to be determined in shadow or illuminated, it is projected onto the light buffer and only the objects into its region need to be tested. These objects are tested in order from the light until we find an intersection or until the distance to the light of the next object is larger than the distance from the surface element to the light.

Our linear light buffer is represented by an infinitely long cylinder whose axis is oriented along the light. This cylinder is subdivided radially in sections. Each section has an ordered list of objects contained (at least in parts) within the limit of its angle. To improve the performance of the linear light buffer, we also divide it in three regions: the `left`, `center` and `right` side of the light source. This is illustrated in Figure 4.27. When an intersection point P is found, it

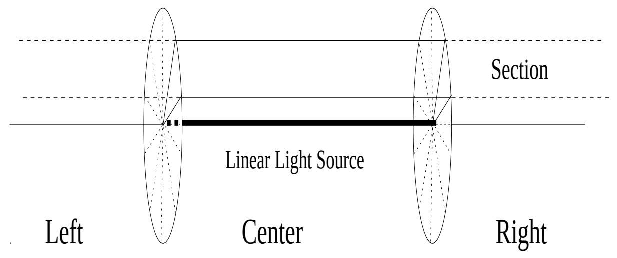


Figure 4.27: The Linear Light Buffer

is located within an angular section of the linear light buffer. If it is positioned in the `center` region, only the objects at least partly in this angular section and in the `center` region need to be tested. If P is in the `left` region, only the objects in the `left` and `center` regions need to be tested, and similarly for the `right` region.

4.3.4 Results of Shadowing with Linear Light

In the scan conversion of the light triangle in the 3D grid, no additional memory is needed. No preprocessing is required but the scan conversion process for each point to shade can be rather expensive. Table II and III illustrate our results for two test scenes.[1] Figures 4.29 and 4.28 show the rendered version of those two scenes. In Table II, the rendering times have been normalised by the time required to render the same scene with a grid of 5^3 voxels and without using the scan conversion of light triangles. In the tetrahedron scene, scan converting the light triangles into a grid resolution of 5^3 is 25% more expensive than no scan conversion. This is due to the facts that just a few voxels contain lots of small triangles and that identifying if a triangle intersects the light plane is a relatively cheap process. However when the grid resolution increases, each light triangle is scan converted in smaller voxels with less candidates per voxel. The number of triangles candidate is greatly reduced, which offsets the extra cost of scan conversion. The rendering cost is then reduced by as much as 76%. Notice that increasing the grid resolution does not always result in a speed up. In the tetrahedron scene, a grid of 50^3 is less cost effective than 25^3. The sphereflakes scene is more problematic and shows some instability with the algorithm. The bottom square is much larger than the spheres that are agglomerated in the center of the scene. Because of this situation, many spheres are divided in just a few voxels. The scan conversion is highly sensitive to variations in the grid resolution, as demonstrated by its unpredictability.

In the linear light buffer, the pointers to the objects for each region within each section of

[1]These two scenes, the sphereflakes and the tetrahedron, have been suggested by Haines [hain87] in an attempt to help benchmarking rendering techniques. The sphereflakes scene consists of 91 spheres defined recursively and a square. The tetrahedron scene is represented by 1024 tetrahedron recursively positioned in a pyramid-like structure. To test our algorithms, a linear light source is located above each scene.

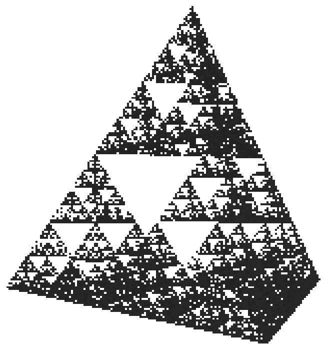


Figure 4.28: Tetrahedron

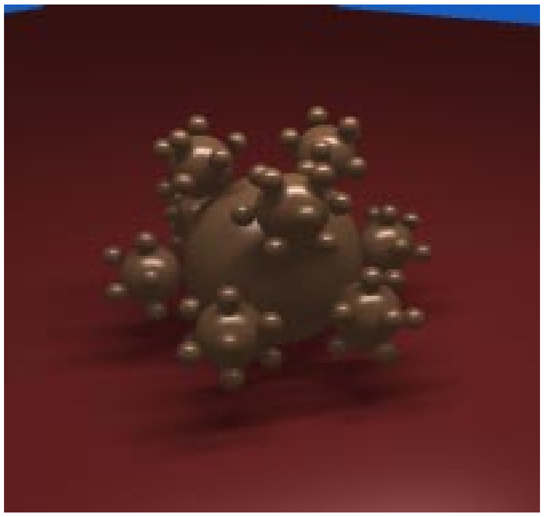


Figure 4.29: Sphereflakes

each linear light buffer need to be allocated. The assignment of an object to the appropriate sections and regions is done in preprocessing. During rendering, a surface element needs only a simple projection to determine the right section and region. An adaptive subdivision of the angular sections can be extended for the linear light buffer but this has not been done in our current implementation.

In Table III, the usefulness of the linear light buffer is evaluated. All the rendering times are normalised by the rendering time using no linear light buffer (a single section per linear light buffer) and no grid. It is interesting to note that unlike the scan conversion of the light triangle, this algorithm is very stable for both scenes. As the grid resolution increases, fewer objects are tested for intersection for each ray, thus reducing the whole rendering time. As the number of sections in the linear light buffer increases, the rendering time decreases steadily.

Table II: Light Triangle

Database	Light Triangle	Grid Resolution					
		5^3	10^3	15^3	20^3	25^3	50^3
tetrahedron	Inactive	1.0000	0.8926	0.8719	0.8657	0.8615	0.8642
	Active	1.2532	0.5239	0.2925	0.2689	0.2385	0.3740
sphereflakes	Inactive	1.0000	0.9059	0.9022	0.8863	0.8742	0.8629
	Active	1.2342	0.7969	0.9722	0.8245	0.8696	1.5742

Table III: Linear Light Buffer

Database	Grid Resolution	Linear Light Buffer Resolution						
		1	24	36	72	144	360	720
tetrahedron	1^3	1.0000	0.9732	0.9508	0.9425	0.9236	0.9120	0.9127
	5^3	0.1429	0.0455	0.0410	0.0359	0.0333	0.0318	0.0314
	10^3	0.1275	0.0303	0.0256	0.0206	0.0180	0.0166	0.0160
	15^3	0.1246	0.0418	0.0227	0.0177	0.0151	0.0138	0.0132
	20^3	0.1237	0.0384	0.0223	0.0168	0.0142	0.0128	0.0123
	25^3	0.1231	0.0341	0.0214	0.0163	0.0138	0.0122	0.0117
	50^3	0.1235	0.0534	0.0209	0.0159	0.0133	0.0118	0.0114
sphereflakes	1^3	1.0000	0.6656	0.6291	0.5751	0.5505	0.5332	0.5281
	5^3	1.0270	0.6834	0.6449	0.5916	0.5678	0.5471	0.5497
	10^3	0.9304	0.5809	0.5446	0.4876	0.4676	0.4479	0.4466
	15^3	0.9266	0.5782	0.5392	0.4858	0.4644	0.4449	0.4411
	20^3	0.9103	0.5606	0.5229	0.4796	0.4481	0.4280	0.4258
	25^3	0.8979	0.5486	0.5112	0.4688	0.4347	0.4156	0.4130
	50^3	0.8863	0.5366	0.4974	0.4579	0.4212	0.4031	0.3990

4.3.5 Conclusion on Shadowing with Linear Light

On a simple test scene, we observed that for a few primitives, shading and shadowing with a linear light source is equivalent in computation to using 10 point light sources. However using this few point light sources results in pictures showing discretisation artifacts within the shadow regions. For primitives as complicated as patches, the number of point light sources equivalent in computation is around 25.

An algorithm is introduced to handle correct shadowing with more complex primitives than polygons. This algorithm adds more flexibility in the primitives casting shadows from a linear light source at the cost of more expensive intersection computations. In order to reduce the additional cost of using this shadowing algorithm, we studied two techniques based on ray tracing acceleration techniques. The scan conversion algorithm has the advantage of requiring no additional memory. However the scan conversion process is rather expensive and it is difficult to evaluate the dimension of the grid subdivision that would provide a good speed up of the shadowing calculations. The linear light buffer requires additional memory for each linear light source. However its stability and the speed up observed made us choose this method for rendering many of our scenes.

For certain primitives, determining the visible portion of the linear light source is very complicated. A general approach based on shooting rays to determine the visible portion of the light is used. With such a process, the cost of shadow determination is very high and the tradeoff of both culling algorithms over no culling becomes even more worthwhile.

4.4 Shadowing with Area Light Sources

4.4.1 Sampling the Light

Most of the algorithms proposed to compute shadows from area light sources are based on sampling the light source to determine the visible part from the point to shade.

Cook *et al.* [cook84a] use distributed ray tracing where a set of rays are sent to the area light to sample the light radiance reaching the surface element to shade. Various techniques have been proposed to improve the sampling pattern of the light source [mitc91] [shir91]. Kok and Jansen [kok91] [kok92] devise some criteria to decide adaptively how many rays should be sent to the area light source. Ward [ward91] estimates the importance of the contribution of each light before sending samples. The low contributors are statistically approximated without shooting any ray. Similarly, Chen *et al.* [chen91] treat a light source only when its radiosity has a significant impact on the global illumination.

Cohen *et al.* [cohe85] propose the hemicube approach. A hemicube consists of five faces regularly subdivided in pixels. The hemicube could be positioned on the center of an area light source and therefore act just like the light buffer of Haines and Greenberg [hain86] with the exception that in the hemicube, only the closest object is kept for each hemicube pixel. Therefore an object is assumed to cover entirely a hemicube pixel. The hemicube has the advantage that it is simple and can make direct use of current hardware Z-buffers. However it suffers because of its regular subdivision. It can present strong aliasing and can miss features (shadows) if its resolution is not fine enough. Max and Troutman [max93] show how the hemicube can be improved upon by allowing a variation of the resolutions of the pixels on each face and by rotating the hemicube about the surface normal. Meyer [meye90] uses the hemicube information before shooting rays towards an area light. Pietrek [piet93] combines hemicube, ray casting and analytical techniques to quickly and accurately compute form factors. He also gives some criteria to decide which approach should be used in which situation based on the information contained in adjacent pixels in the hemicube. Sillion and Puech [sill89] replace the hemicube by a single plane with varying surface elements according to the solid angle they produce.

All these techniques suffer from sampling that is done at the center of a primitive (generally a polygon) to represent the shadowing over the entire surface.

4.4.2 Extended Rays

Instead of using sampling rays, Amanatides [aman84] extends the concept of a ray to a cone. By doing so, he can compute the shadows cast from a spherical light source. A single cone is sent to the light and the non-obstructed portions of the cone determine the portion of the light shining on a point. To reduce the cost of intersecting a cone with many objects, Genetti and Gordon [gene93] fit a cone over the light and recursively subdivide this cone into smaller cones until each cone is either considered fully illuminated, in shadow, or until the subdivision reaches a certain threshold.

Heckbert and Hanrahan [heck84] describe *beam tracing* where a polygonal *ray* is sent towards a polygonal light to determine the visible part of the light. Polygonal objects clip the beam as it progresses towards the light.

4.4.3 Characterising Regions

As we saw earlier, a shadow introduces discontinuities in the radiance function over a receiver. Baum *et al.* [baum91] use D^0 discontinuities produced at the intersection of polygons supported by non-parallel planes to create better meshes for radiosity computation. D^0 discontinuities also occur in the shadow (umbra) from directional and point light sources. The shadow volumes [crow77] from a point light source have been represented by a binary space partion (BSP) tree by Chin and Feiner [chin89]. Campbell and Fussell [camp90] create meshes for shadow discontinuities in radiosity by distributing a series of point light sources on a polygon emitter and generating the meshes (BSP) for each point light. This technique can lead to a large number of subdivisions as each light might need to be approximated by a large number of point light sources.

Nishita and Nakamae [nish83] compute the shadow with its umbra and penumbra (D^0, D^1 and D^2) regions cast by a convex polygonal blocker when illuminated by a convex polygonal light. They project every edge of blockers from every vertex of the light. The shadow is determined as the convex hull (if both light and blocker are convex polygons) of the shadow volumes from each light vertex while the umbra corresponds to the intersection of these shadow volumes. The reflected radiance is computed point by point at the rendering stage. If a point on a polygon is fully lit by the light, the radiance is computed directly without any further shadow test. If this point lies in an umbra region, only the ambient contribution is added. If the point is in penumbra of one or more polygons, the polygon(s) casting this shadow must be projected back onto the light and the radiance is computed only for the visible portion of the light. When the information about which polygon is casting a given penumbra is not provided, some techniques allow fast culling of the polygons [hain91] [woo93]. These techniques rely mostly on the plane equations formed by linking the point to shade to the edges of the light.

Bao and Peng [bao93] use a technique similar to [nish83] but rely on a BSP structure to order the polygons casting a shadow on the others. The light source is subdivided if it lies on both sides of the supporting plane of a scene polygon. Instead of considering only the planes formed by the scene edges and the light vertices, Chin and Feiner [chin92] also compute the planes formed by the light edges and the scene vertices. Thus they avoid the use of Weiler-Atherton [weil77] [athe78] clipping algorithm and construct similar umbra and penumbra BSP trees for each light. This latest approach has also been used by Campbell and Fussell [camp91] with more theoretical assertions about the boundary of each region.

This kind of approach characterises regions in space as illuminated, in umbra or in penumbra of a single blocker. However, they do not characterise regions that would be in umbra as a result of the combined action of two or more blockers. Therefore all the polygons for which a point is in penumbra must be projected back onto the light source to determine the portion of the light visible from the point to shade. Obviously, this can be an expensive process. As an example, assume a finely meshed object lies under an area light. Individually, every facet of the object produces a very small umbra but together they can cast a large umbra.

Campbell and Fussell [camp91] observe that the radiance within each penumbra region is not linear and simple linear interpolation of a few sample points do not provide the proper values for each point in a given regions. They propose to subdivide each region according to the radiance function. The maximum radiance points are found by numerical optimisation techniques and each region is further subdivided accordingly. Drettakis and Fiume [dret93] show how a few sample points can be used to reconstruct the radiance function in the absence of shadows.

Instead of characterising only the contour of the penumbra and umbra volumes, other researchers have studied the effect of the discontinuities within the penumbra regions. Most of the algorithms start by finding all the critical curves formed by a vertex of the light and an edge of the scene or by a vertex of the scene and an edge of the light. Heckbert [heck92a] finds the end points of each critical line lying on a given polygon, identifies the intersections between these line segments and subdivides each region with a Delaunay triangulation. A parametric surface is finally fit to each triangle to represent the radiance over the region. Lischinski *et al.* [lisc92] use the same scheme but represent the scene with a 3D BSP tree and store the critical curves (lines) on each polygon with a 2D BSP tree. A quadratic interpolating surface is fitted over the radiance and contributions of lights and reflectors (forming different meshes) are summed over a global mesh. Salesin *et al.* [sale92] present a solution that fits piecewise cubic interpolants to the radiance, thus preserving C^1 continuity. Finally, Lischinski *et al.* [lisc93] merge their discontinuity meshing algorithm with the hierarchical radiosity of Hanrahan *et al.* [hanr91] allowing for a fast solution of complex environments that produces images of higher quality due to the presence of more proper radiance discontinuities.

In all the previous approaches, the discontinuities produced by three edges are not computed. Teller [tell92] presents an algorithm to compute EV and EEE discontinuities in an antipenumbra by converting every segment in the Plücker coordinate system. In this 5D system, a directed line corresponds to a half-space delimited by a plane in the Plücker coordinates. A set of all directed segments forming the holes define a convex polytope. The intersection of an edge of the polytope with the Plücker quadric surface determines an end line of a discontinuity. The intersection of a face of the polytope with the Plücker surface defines a VE (planar) or a EEE (quadratic ruled surface) discontinuity. Stewart and Ghali [stew93] use a plane sweep approach starting at the light source. The intersections of scene edges and faces with the plane are recorded and processed in order, thereby constructing the aspect graph as seen from any point on the light.

4.5 Conclusion

Shadows provide important information and therefore play a role essential in our interpretation of the world. However as we saw in this chapter, properly and efficiently computing realistic shadows in computer graphics is an expensive process.

Characterising each shadow region in a scene has the advantage that it can be done once as preprocessing and can lead to important savings in a static environment. It can also allow computation of the radiance function over an entire region. However this process requires to compute all the regions, even if some shadows are not visible. These regions must be stored and queries about points in these regions have to be answered efficiently. Moreover, as the dimension of the light increases, the complexity of the regions grows, making the characterisations of regions much more difficult.

Sampling the light on a need-to-know basis is a simple process. However it is done at the risk of missing some blockers and can be expensive for large lights in complex environments. To avoid missing blockers, the blockers can be projected back onto the light, thus revealing the portion of the light as seen from the point to shade. This is done at the increased cost of doing the light clipping.

Intermediate techniques have been used to characterise regions as being in the shadow (umbra or penumbra) of individual blockers or fully illuminated. These techniques have been popular as they often offer a nice compromise between computing the shadows at every point and characterising every region.

In our approach to shadowing with linear light sources, we chose to recompute the intersection of a light triangle with a scene for each point to shade. This provided us with a simple algorithm adapted for each type of primitive in order to render soft shadows. Two culling techniques have also been developed to reduce the number of objects potentially casting a shadow on the point to shade. This decision seemed the most appropriate in a ray tracing renderer. However this decision must be made for each application domain.

In the next chapter, we will see how the shadow regions as well as the shading on the surfaces can be used to specify the illumination and surfaces characteristics within the modeling process.

Chapter 5

Lighting Design by Shading and Shadows

In the previous chapters, we reviewed various reflection models to compute the shading of many surfaces. We also presented algorithms to compute the shading and shadows of scenes illuminated by linear and polygonal lights. All these models and techniques provide tools to a user to create images of scenes with a higher degree of realism. However, the user must understand very well the reflection models and their relationships with the illuminants in order to produce the right shading effects in the scene she is designing.

In most modeling systems, lights are created and manipulated just like any geometric object. A user must then assign to each light an emitted power as a function of wavelength. Similarly, the user must provide values for the various surface parameters. At each step, the only way to observe the results of the changes is by rendering the scene. Depending on the scene complexity, the rendering technique and the quality needed for the intermediate image, a resulting image can be produced in times ranging from a fraction of a second to hours. With this feedback, the user can return to the modeler and again change the parameters to create better shading effects. This process is repeated until the image corresponds to what the user wants or until the user finally decides, often after many frustrating iterations, that the image is as good as it is going to get.

The task is not simple because the user must perform mentally an *inverse shading*. She can know which shading effect she wants on a surface, but must determine the values for each surface parameter and each light position and emitting power that will translate into the desired shading.

In this chapter, we describe how we incorporate shading and shadow informations into the modeling process itself. By creating and directly altering these shading effects, the lights and surfaces are indirectly specified. Our modeler thus helps to determine where a light should be to produce a shadow here or a highlight there. It can also provide values for surfaces parameters to create a highlight of a certain size or to fit selected colours at surface points.

Our approach reduces the number of inverse shading tasks that a user must perform. It should lead to important savings in the number of modeling/rendering iterations. It also frees the user from knowing all the specifics of shading models and can prevent this user from having to enter *magic* numbers. Any user, whether experienced or not, should benefit from the extra information provided by our modeler.

Our new process does not preclude previous ways of defining and positioning light sources or specifying reflection parameters. In fact, one can still move the light sources directly, just like any other object in the scene, or specify surface characteristics by entering values for various shading parameters. Our techniques thus offer a new approach that enhances the process of lighting design and surface shading.

In the next sections, we quickly review some computer vision techniques for extracting information from shading features in an image. We see how difficult some of these problems are because of a lack of information (constraints). In a modeler, viewing parameters and the exact scene geometry are known. Because of this, many problems become much simpler to solve. We show how diffuse and specular reflections can be used to determine the geometric location of a light. We also describe a technique to define and position a light from the shadows it cast. Finally, we demonstrate how the modeler can fit the "best" values of the shading parameters to colour points applied on a surface.

5.1 A Parallel with Computer Vision

Computer graphics builds models to simulate reality. Once built, the user *renders* these models onto an image. In this sense, computer graphics models the causes (the interaction of light with matter) and renders the effects (colours) as an image. By way of contrast, computer vision is interested in analysing an image. It tries to isolate certain effects in an image in order to estimate the causes. While the two processes might seem to go in totally opposite directions, it is interesting to consider how advances in one might actually help the other.

An important research area in computer vision is analysing shading information to extract object shape, light orientation and surface characteristics from an image. There is an extensive literature on recovering shape from shading.[1] By integrating the shading gradient, it becomes possible to infer a surface that would produce the shading. The technique has several limitations because boundary conditions are often unavailable. In fact, it is difficult to determine what portion of a pixel colour is due to the surface at a boundary because we do not know the pixel coverage. Real surfaces are not generally ideal diffuse reflectors and interreflections between surfaces reduce the accuracy of the reconstructed surface. Much research still involves producing more general solutions, although shape from shading is a very difficult problem to solve without relying on additional assumptions.

Finding the light source direction from shading is a complementary problem to shape from shading. By assuming a scene is made of diffuse surfaces and a uniform isotropic distribution of surface normals, Pentland [pent82] applies a least-squares regression to the pixels intensities and gradients to determine the most likely light direction. However, when his assumptions are violated, it becomes very difficult to rely on his technique.

Highlight information has been used to determine light direction or local shape orientation. Babu *et al.* [babu85] study contours of constant intensity in an image to determine the orientation of planar surfaces under the illumination of a directional light source. Buchanan [buch87]

[1]For a collection of important results in this area, see [horn88] and [wolf92].

fits ellipses to the highlights to obtain the same information for planar surfaces illuminated by point light sources. One important requirement of most of these algorithms is the accurate identification of the highlight area. This area is often detected by some kind of thresholding technique. The unfortunate situation with thresholding is that different threshold values can lead to relatively different shape of the highlight and, therefore, to different shape and light recovery. Other techniques, such as the use of polarisation [wolf91], are promising, although they require the presence of polarising lenses on the cameras capturing the scene.

Much useful information can be extracted from the shadow areas in an image [walt75] [shaf85]. These areas provide additional information on the shape (profile) of the object casting a shadow and even on the shape of the object on which the shadow is cast. Moreover, they provide information on the direction and the shape of the light sources. However, shadows of dim lights will produce only very small changes in shading gradients. Similarly, the presence of several lights (possibly extended lights) and interreflections will create many shadows for each object. This amalgam of overlapping shadows will make the situation very difficult to interpret. Detecting shadows can be done in a manner similar to edge detection by applying various edge enhancing filters. For extended lights, the shadow edges are soft and the shadows must be detected based on changes in the gradients of the surface shading. Gershon [gers87] uses gradients in colour space to determine if a region corresponds to a shadow region or simply to a change of material.

While modeling a scene, a user has access to important information unavailable to computer vision, i.e. the geometry of the scene and the viewing parameters. To better understand a 3D scene, the user can move her view point around, display at the same time several views of the same scene, move objects, and remove hidden surfaces, all in real time. However, very few current applications use information about shading and shadows to improve the efficiency of the modeling step. In this chapter, we investigate how to define and manipulate light sources and to specify surface shading parameters.

5.2 Defining and Manipulating Light Sources

With the advent of high performance graphics hardware, it becomes possible to interactively create and manipulate more complex models with a higher degree of realism. The simple wire-frame models of yesterday can now be replaced by illuminated, smooth shaded, depth buffered, textured (filtered) and antialiased polygons, while still retaining real time interaction with the models.[2] Hanrahan and Haeberli [hanr90] demonstrate with their system how current graphics hardware can be used to *paint* textures and various other surface parameters (transparency, perturbation of surface normals, etc.) directly onto surfaces in a fully interactive system. This increase in rendering power allows us to better investigate light definition and manipulation from shadings and shadows as well as specify surface characteristics by their shading effects.

[2]As point of reference, the Silicon Graphics RealityEngine [akel93] is reported to have performance exceeding one million antialiased, texture-mapped triangles per second.

In the next three sections, we describe how light sources can be defined and manipulated from diffuse reflections, highlights and shadows. The advantages of each technique are demonstrated and their respective limitations are explained to give a better insight on the efficiency of each technique.

5.2.1 Lights from Diffuse Reflections

A diffuse surface appears equally bright from every angle. For most surfaces, the shading gradient provided by this model offers an important clue to understand the shape of a surface. Recall the diffuse contribution to the pixel radiance of a surface element illuminated by a directional light (Equation 3.3):

$$L_{pixel} = k_d(\vec{N} \cdot \vec{L})E_0.$$

In this equation, we assume that all the terms are constant except for the light direction $\vec{L}$. $\vec{N}$ is the surface normal at a given point.

This formulation tells us that for a given point on the surface specified as the *maximum intensity* of the diffuse reflection function, a unique directional light source can be determined when $(\vec{N} \cdot \vec{L}) = 1$, which occurs at

$$\vec{L} = \vec{N}.$$

The term *maximum intensity* is not properly correct if we think of it in the context of a complete shading model with diffuse and specular reflections. However if in a scene all is fixed except for the direction of the light, this maximum intensity will find the direction of the light that will produce the maximum irradiance on the surface element. It is interesting to note that other surface elements on this surface might also reach this maximum irradiance but will never surpass it.

The diffuse reflection as expressed above is independent of the viewer position. It is therefore possible to change the viewer position while the diffuse shading information remains the same. Unfortunately, this technique is limited to specifying only a direction along which a light source lies unless additional constraints are added to the system. Such constraints can exist in a modeling system. For instance, the user can constrain the light to reside on a plane (such as a ceiling in an indoor scene). By adding a point of diffuse maximum intensity, a direction is established. The intersection of this direction ray and the light plane[3] can be used to define a point light source or a vertex of an extended (linear or polygonal) light source.

While a point light defined by this technique still maximises the above diffuse expression, it does not correspond to maximising the diffuse radiance in the direction of the pixel for a surface illuminated by a point light. Instead, one needs to maximise the diffuse contribution of Equation 3.5

$$L_{pixel} = k_d f_{rd}(\vec{N} \cdot \vec{L})\frac{\Phi_p}{4\pi r^2}.$$

[3]Note that there might not be any intersection.

If all the terms are kept constant except for the light position, the maximum intensity of the above equation occurs at

$$\max\left(\frac{\vec{N}\cdot\vec{L}}{r^2}\right).$$

Without any additional constraint, this maximum occurs directly on the surface element. In many indoor scenes, illuminants have fixed definition (a point light, a linear light of a certain length or a polygonal light of a certain size) and thus must lie on the ceiling plane. By constraining the point light to lie on a plane, the maximisation can be expressed in Cartesian coordinates (see Figure 5.30) as

$$\max\left[\frac{x_p\sin\alpha - d\cos\alpha}{(d^2+x_p^2+y_p^2)^{\frac{3}{2}}}\right]$$

where x_p and y_p are the only variables. The local maximum can be obtained with a nonlinear optimisation algorithm.

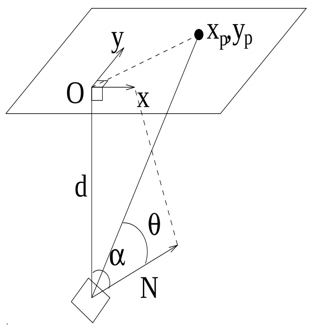


Figure 5.30: Maximising $\frac{(\vec{N}\cdot\vec{L})}{r^2}$

This time the surface element is not necessarily the portion of the surface with the largest irradiance. However, this point light does provide the maximum irradiance on the surface element while still lying on the plane. A similar approach can be applied to linear and polygonal illuminants if, for instance, the light can only be translated on the plane.

5.2.2 Lights from Highlights

Specular reflection models light that is reflected mostly around one direction. The light reflected this way corresponds to what we consider to be a highlight. Recall that the specular contribution to the pixel radiance of a surface element illuminated by a directional light (Equation 3.3)

$$L_{pixel} = k_s F_s(\vec{N}\cdot\vec{H})^n E_0,$$

where $\vec{N}$ is the surface normal at a given point, $\vec{H}$ is the bisector vector of the eye direction and the light direction, and n is the surface roughness coefficient.

In this equation, we assume that all of the terms are kept constant except for the light direction $\vec{L}$ (therefore $\vec{H}$ varies). This formulation tells us that for a given point on the surface, specified as the *maximum intensity* of the highlight, a unique directional light source can be determined when $(\vec{N} \cdot \vec{H}) = 1$, which occurs when $\vec{H} = \vec{N}$, and therefore

$$\vec{L} = 2\left(\vec{N} \cdot \vec{E}\right)\vec{N} - \vec{E},$$

where $\vec{E}$ is the eye direction.

Once again, we interpret the term *maximum intensity* to mean maximising the radiance function given above.

This simple relationship between the maximum intensity of the highlight and the light direction has been used in the past. Hanrahan and Haeberli [hanr90] mention how they specify a light direction by dragging the point of maximum intensity of a highlight on a sphere. This technique has also been implemented in two other modelers. A light modeler developed in 1983 at the New York Institute of Technology by Paul Heckbert manipulated highlights on a sphere. A light editor written by Richard Chuang around 1985 at Pacific Data Images was used to position highlights at specific locations on flying logos. A similar approach at LucasFilm was used to get a glare on a sword at a crucial moment in the movie *Young Sherlock Holmes.*

Our technique extends the basic approach by indirectly and interactively determining the surface roughness coefficient n in relation to the size of the highlight.

Once the maximum intensity point of the highlight has been selected, the user drags the cursor away from this point. At a new position on the surface, the surface normal is computed. This new point is used to determine the boundary of the highlight, i.e. where the specular term of the radiance reaches a preselected threshold t. To satisfy this threshold, n, the only free variable in $(\vec{N} \cdot \vec{H})^n = t$, is easily computed as

$$n = \frac{\log t}{\log(\vec{N} \cdot \vec{H})}. \tag{5.13}$$

While only these two points on a surface are necessary to orient a directional light source and establish the surface roughness coefficient, they give very little information about the complete shape of the highlight. To approximate the contour of the highlight, the pixel with the maximum intensity is used as a seed point and the neighbouring pixels covered by this surface are visited in a *boundary fill* fashion [fole90] until pixels on both sides of the threshold are identified or until the boundary of the visible surface is found. With this boundary fill algorithm, the second point is not guaranteed to be enclosed by the contour of the highlight determined from the original seed point. Figure 5.31 represents a shape where the point of maximum intensity and the threshold point would define two separate highlights from a unique directional light. If this happens, the second point is also used as a seed. Unfortunately, unless each pixel covered

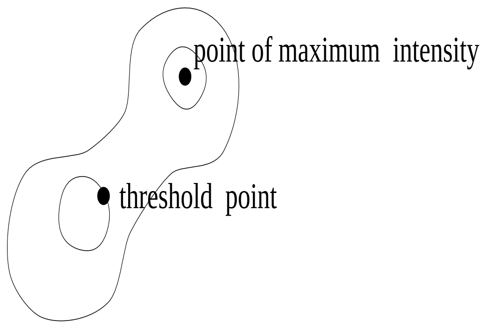


Figure 5.31: Peanut with two highlights from one light

by this surface is visited, some of the possible other highlights produced by this light on this surface might be missed. If the position of every highlight is necessary, every pixel covered by the surface must be checked. This is done only when requested by the user because this can lead to considerable increase in computation time, thus reducing the interaction speed.

Instead of using a screen based algorithm to trace the highlight boundary, we could have followed the boundary curve directly on the surface via numerical methods. This provides a more accurate definition of the highlight contour. However, because of its simplicity, speed and generality (it is applicable to any rasterisable object), the boundary fill algorithm proved quite satisfactory in our application.

When n has already been determined for a given surface, care must be taken in order to keep a unique value for n. If another highlight is created on this surface, as soon as the point with the maximum intensity is selected, the contour of this new highlight is computed with the previous value for n. However, this value for n and the position of the highlights are not fixed and can be interactively changed because some information is kept in a temporary frame buffer. In this frame buffer, each previously visited pixel contains information about its surface normal. The contour can therefore be scaled down (i.e. a smaller highlight defined by a larger value for n) very efficiently. If the contour is increased, only the unvisited pixels need to have their surface normals determined. It is also possible to move the highlight on the surface. This process is more expensive if the highlight is moved to a completely different location on the surface as many surface normals might need to be computed. On some graphics hardware, information on the surface normals can be obtained directly from the hardware, therefore allowing for even faster highlight manipulation.

The contour of the highlight that we define is an approximate representation of the highlight. In the fully rendered surface, this highlight may appear different because of the diffuse reflection contribution and the k_s factor that controls how much of the specular contribution translates into radiance. The highlight can appear different if a blocker resides between a point on the highlight and the light direction. Shadow rays can be shot towards the light to determine the presence of blockers and to modify the boundary fill algorithm with this information. Figure 5.32 shows a highlight produced by a directional light source over a patch of the teapot. The white segment within the highlight region represents the point of maximum intensity. This segment points towards the light direction.

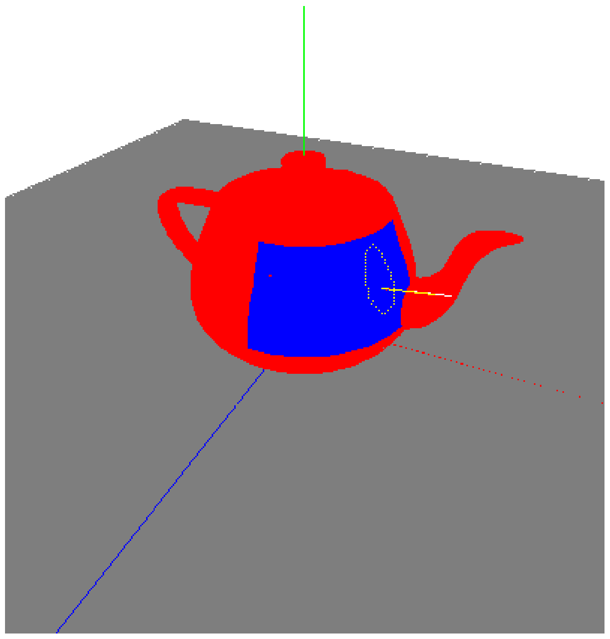


Figure 5.32: Creating a light by its highlight

The highlight information is dependent on the eye position. Therefore, if the eye is moved, every highlight in the scene must be recomputed. The points of maximum intensity will not be valid anymore and consequently every surface must be revisited to recover every highlight. This expensive process should be avoided as much as possible. This also means that a highlight computed in one window would have a different definition in another window for which a different projection is used. To avoid confusion and too much computing time, we decided to remove all highlight information when the viewing parameters are changed. The lights definitions and surfaces roughnesses are kept with the scene description. The highlights are recomputed only upon request from the user.

Another limitation of using highlight information to describe a light source is that a highlight determines only a direction. We therefore need more constraints to determine other types of light source. Some of these constraints are described in the previous section when the diffuse reflection approach was discussed. These are applicable here. Similar to the diffuse reflection used to manipulate a light, if a point light, a linear light or a polygonal light are constrained to lie on a given plane, the location of the light is controlled. This is solved with a nonlinear optimisation algorithm.

To represent highlights created by extended light sources, the contribution of each vertex of the light is not sufficient to determine the shape of the complete highlight (Figure 5.33). To display this information, the boundary fill algorithm has to compute the specular integral for a linear light (Equation 3.11) or a polygonal light (Equation 3.12) for each pixel visited. Such integrals are rather expensive to compute. In order to achieve real time, cheaper approximations based on precomputed tables help to reduce the cost. We did not investigate this approach in this thesis, but relied solely on the partial information provided by the light's vertices.

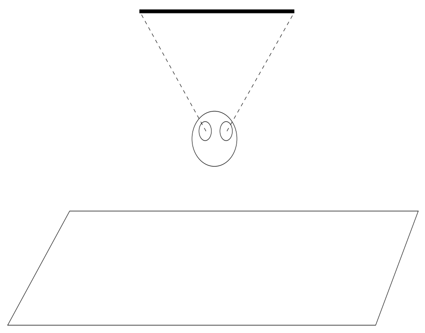


Figure 5.33: Incomplete highlight information

Highlight information can be very useful to specify a directional light source and a surface roughness coefficient. With extra constraints, they can even be used to define and control point, linear and polygonal light sources. Shadow information can also be used to define and manipulate light sources. The next section describes how this can be done.

5.2.3 Lights from Shadows

As seen in the previous chapter, shadows provide very important clues to understanding the geometry of a scene. In the context of this thesis, we use the fact that shadows can reveal important information about the nature of a light source. We define light sources by manipulating their *shadow volumes*.[4] These shadow volumes have the advantage of depending only on the lights and the objects' positions. Therefore, as opposed to the manipulation of lights by their highlights, the eye position can change without altering the description of the shadows; the shadows are consistent for every projection. This allows for multiple windows to be opened with different orthographic and perspective projections, as is common in most of the modeling systems.

[4]A shadow volume formed by a single object and a directional or a point light is the 3D volume within which every point is in the shadow of the object [crow77] [berg86]. For extended light sources (linear, polygonal), the shadow volume is the 3D volume within which every point is at least partly in the shadow of the object.

Some systems have been reported to display shadows in real time. Blinn [blin88] describes how an object scaled by zero in one dimension and coloured with a uniform darker shade can be used to simulate a *fake* shadow approximating a shadow from a directional light source. Most of the graphics hardware producing shadows in real time rely on shadow volumes [crow77]. In their extension of shadow volumes for objects defined as unions, intersections and differences of other objects (CSG objects), Jansen and van der Zalm [jans91] display shadows from directional and point light sources. Chin and Feiner include shadow polygons in the BSP tree representation of a scene. They display shadows from point light sources [chin89] and polygonal light sources [chin92]. Segal *et al.* [sega92] use textures created from the light source to simulate shadows.

While all of these techniques can display shadows in real time or near real time, none defines or manipulates lights from shadows. Some work in 3D user interfaces [hern92] describes how the fake shadows of Blinn [blin88] are projected onto three orthogonal planes (or fake mirrors when the object is fully shaded) to aid in transforming a model. This is part of a toolkit to manipulate an object from its fake shadow or fake reflection. This work is not concerned with dealing with the light sources and realism.

The shadow volume created by an object illuminated by a directional light source consists of a sweep of the object silhouette in the direction the light source shines. This silhouette can be analytically determined for simple primitives, computed for moderately complicated objects with algorithms like the one of Bonfigliolo [bonf86], sampled by studying the variation of surface normals at the vertices of a tessellated object or sampled using the information in a Z-buffer orthographic projection of this object.

Specifying the direction of a directional light is simply a question of choosing two arbitrary, although different, points in the scene. The second point is considered along the shadow cast by the first one. Figure 5.34 shows a cylinder illuminated by a directional light source. The white point on the top of the cylinder represents the first point selected by the user. The line segment originates from this point and intersects in 3D the plane underneath where the cursor points. For some primitives, computing the exact silhouette can be an expensive process. Depending on the application domain, an approximation of the shadow volume can be sufficient, allowing for real time computation and display while still providing important information about the real shadows. In the case of the cylinder of Figure 5.34, each polygon vertex forming the cylinder is simply projected in the direction the light shines.

In our modeler, we use four different representations for the shadow volumes. The first one consists of (1) simple line segments originating from points on the blocker (not necessarily all on the silhouette). This representation is fast and allows a user to easily select a specific line segment. The backfacing faces of the shadow volumes and the objects behind the shadow volumes are still visible. The shadow volumes can also be displayed as (2) opaque or (3) semi-transparent volumes. It is then easier to see the shadow volumes as 3D objects and determine the parts of surfaces that are intersecting the shadow volume. It also allows a user to easily select a frontfacing face of the shadow volume. The last representation uses (4) the graphics hardware to display only the intersection of the shadow volumes with the objects in the scene. Therefore the shadow volumes disappear and only the shadows that they produce are displayed. This is how we see shadows but this does not allow for a simple manipulation of the volumes creating these shadows.

To move a shadow volume once it is defined, a user needs to select a point on the shadow volume. The point on the object casting this shadow is then identified. By dragging the cursor to a new location, a new direction is computed, the direction of a directional light source.

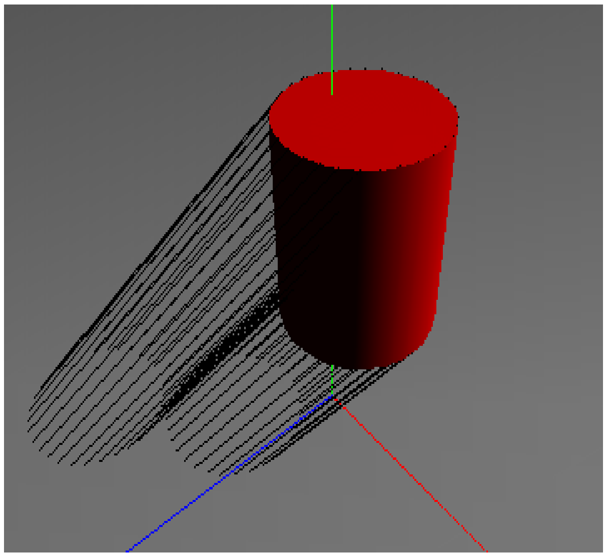


Figure 5.34: Creating a directional light by its shadow

We use the fact that a directional light source can be interpreted as a point light source located at infinity to explain how we create a point light source from a shadow volume cast from a directional light. Figure 5.35 illustrates the process of going from a directional light source (Figure 5.35a) to a point light source (Figure 5.35b) by modifying its shadow volume.

A point sn_1 on the shadow volume is selected. The point sn_2 on the silhouette casting shadow on the point sn_1 is identified. This *shadow segment* $[sn_1, sn_2]$ will now be considered as nailed (not moving) and the point light source will reside on the supporting line of this segment. By selecting another point s_1 on the shadow volume, the point s_2 casting this shadow on this point is identified. The nailed segment $[sn_1, sn_2]$ and the point s_2 define a plane $(sn_1 - sn_2 - s_2)$. By moving the cursor away from s_1, a point s'_1 on this plane is located. s'_1 is considered on the shadow cast by s_2. The two supporting lines of segments $[sn_1, sn_2]$ and $[s'_1, s_2]$ intersect at a unique point p_i. The point light source is therefore moved to p_i as shown in Figure 5.35b.

Once a point light source is created, it can be manipulated in the scene by manipulating its shadow volume. This can be done by fixing any shadow segment as previously described, or, if

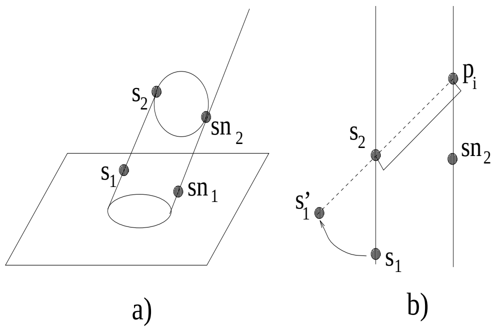


Figure 5.35: Going from a directional light source to a point light source

no shadow segment is nailed, by adding a new constraint to the system. Once such assumption is that the distance d from the point light p_i to the point s_2 casting a shadow is constant. Combinations of these two actions are sufficient to position almost any point light source in a scene.

In some rare configurations of a scene, some positions might not be accessible by manipulating only these shadow volumes. For instance, assume that a scene is made of a single flat polygon and of a directional light parallel to the plane of the polygon. In such a situation, the light would never be able to escape the plane of the polygon. To avoid this kind of situation, the scene is bounded by a large box and shadows cast on these walls can always be manipulated to avoid the problem above.

It is important to note that the point s_2 might not lie on the boundary of the shadow volume while the point light source is moved around. However the real shadow volume is always displayed so the user has a direct view of the altered shadow.

To create extended light sources like linear or polygonal lights, new point light sources are needed to define the vertices of the light source. The shadow volumes of each light vertex are handled just like any point light source. For polygonal light sources with more than three vertices, each light vertex must reside on the light plane. This extra constraint is added to the system to create the shadow volume of a new light vertex because only a direction (two points) is necessary to determine the 3D location of the light vertex.

As we saw in the previous chapter, shadows of extended light sources are formed by the umbra and penumbra volumes. Several techniques can be used to compute these volumes. In our implementation, we compute the umbra as the intersection of each *vertex shadow volume* (one shadow volume per light vertex); the penumbra is the difference between the whole shadow and the umbra. Some problems occur when neither the blocker or the light are limited to being convex. It can be shown however that if both the light and the blocker are divided into convex elements, the whole shadow cast by this blocker from this light is the *union* in 3D of all the shadow volume convex hulls as:

For now on, assume a polygonal convex light and a convex object.

```
For each (convex light element)
    For each (convex blocker element)
        Compute the convex hull of the shadow
            volumes created by these two elements
Compute the 3D union of all these convex hulls
```

To compute efficiently the shadow volume (umbra and penumbra) cast by a convex object illuminated by a convex light, we do the following. We assume the blocker does not intersect the light plane. We first find a plane separating the light from the blocker. If there is no such plane, the light intersects the blocker (remember that the light and the blocker are both convex). The portion of the light inside the blocker is simply disregarded. Finding a separating plane can be achieved in $O(n \log n)$ where n is the number of faces in the light and the blocker. Then this plane is displaced away from the light until the light and blocker are on the same side of the plane. For each light vertex and blocker vertex, their supporting line is intersected with the plane. A simple 2D convex hull algorithm (we used the algorithm of Graham reported in Sedgwick's book [sedg90]) is then applied to these points. The points on this 2D convex hull are associated with lines in 3D. Each line belongs to a vertex shadow volume. The 3D convex hull of these vertex shadow volumes is delimited by planes supported by the same lines of the vertex shadow volumes than the ones obtained by the 2D convex hull on the separating plane.

Computing the umbra region (i.e. the intersection of each vertex shadow volume) cannot generally be performed in 2D. Figure 5.36 shows an example where using only the information in the 2D projection plane would fail to identify the umbra region (showed in hatched). To

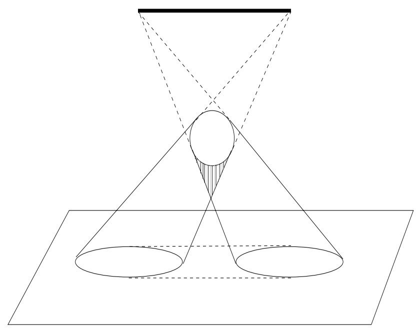


Figure 5.36: Hatched umbra region is undetected in the projection domain

compute the umbra volume, we intersect each shadow polygon[5] of a light-vertex shadow volume with every other vertex shadow volumes formed by the other vertices of a single light. This process can be very expensive as it is $O((ps)^2)$ where p is the number of vertices of the light and s is the number of shadow polygons forming the shadow volume. Algorithms to compute the intersection of the supporting planes of each light-vertex shadow volume can be used. This can be achieved in $O(n \log n)$ [prep85] where n is the number of supporting planes of each shadow polygon.

Although all these techniques give a good impression of the final shadow, they do not incorporate umbra volumes resulting from the union of penumbra volumes from different blockers. The various algorithms provided by Teller [tell92] and Stewart and Ghali [stew93] could be used to identify all the umbra volumes but accurately computing these extra volumes will be at the cost of a much slower general interaction.

5.2.4 Results

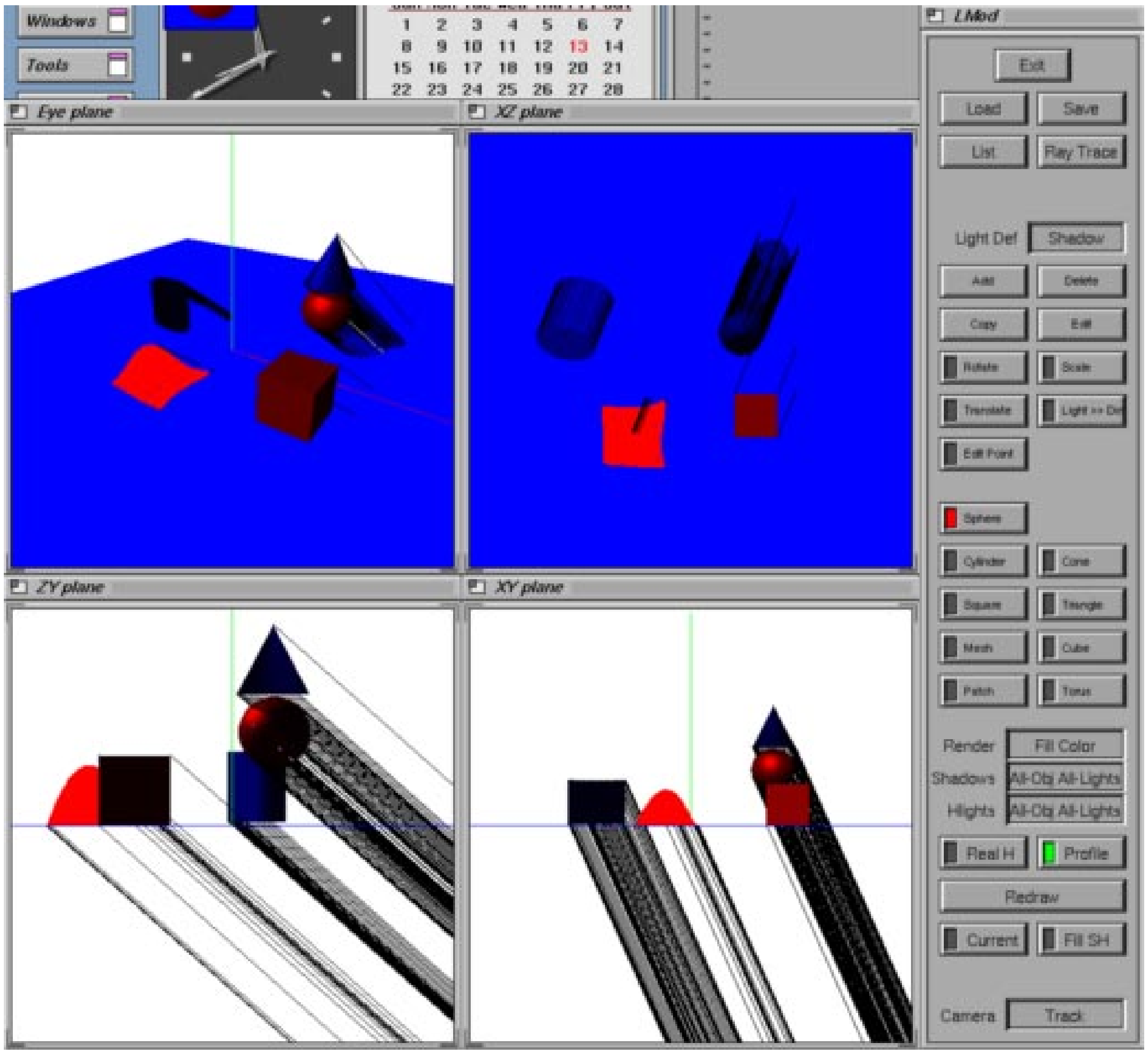
A modeler has been implemented in order to test the techniques presented in this chapter. The modeler operates on primitives such as conics (sphere, disk, cone, cylinder), squares, cubes, triangular meshes and Bézier patches. Figure 5.37 shows a global view of the modeler itself.

The code is written in C under GL of Silicon Graphics. The graphics hardware allows for real time surface removal, Phong shading, transparency and shadow volumes, which is very useful to model scenes and create and manipulate shadow volumes. Unfortunately, this real time shading can lead to some minor difficulties when creating highlights, because the threshold t must be adjusted to the hardware shading implementation.

Figures 5.38 to 5.40 show a cone under a triangular light source. At first, no convex hull is applied. In this image (Figure 5.38), it is easier to associate each shadow with a light vertex. Once the convex hull is applied (Figure 5.39), only the segments on the silhouette of the penumbra are displayed. Notice the umbra region just under the cone, within the penumbra volume. The shape of the umbra was difficult to extract from the previous representation. In Figure 5.40, the umbra and penumbra volumes are filled with a semi-transparent mask. All these static images do not give full credit to the representations into the modeling process. As soon as the point of view is rotated, the lines provide more 3D information.

In the first part of this chapter, we described how shading effects like diffuse reflection, highlights and shadows can be used to define and manipulate light sources. These effects can be rendered in real time with existing graphics hardware. However, we also used some simple line drawing representations (highlight contour, shadow volume sampled as line segments) in order to ease the manipulation of these effects. Obviously, these representations do not provide the same perceptual information than a fully rendered image. In the next sections, we will present a system to interactively determine surface characteristics as they appear in the final

[5]The silhouette of the object can be discretised. Each point cast its shadow in one direction. Two consecutive points on this silhouette and their shadow direction define a quadrilateral with two of its vertices at infinity.



Light Def Shadow
Render Fill Color
Shadows All-Obj All-Lights
Hlights All-Obj All-Lights
Camera Track

Figure 5.37: Global view of the modeler

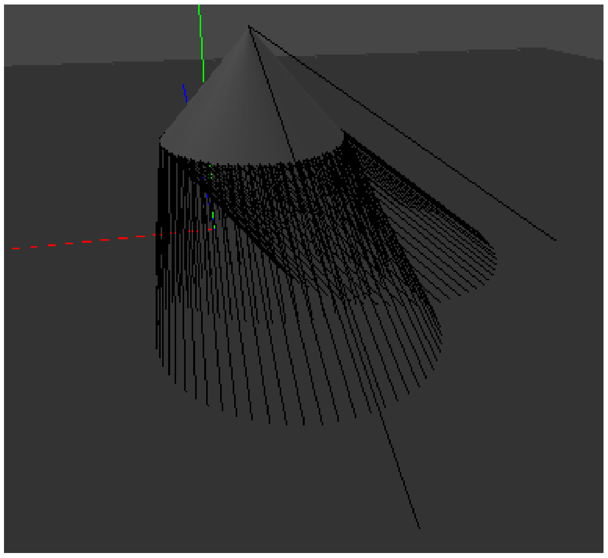


Figure 5.38: Cone under a triangular light: No convex hull

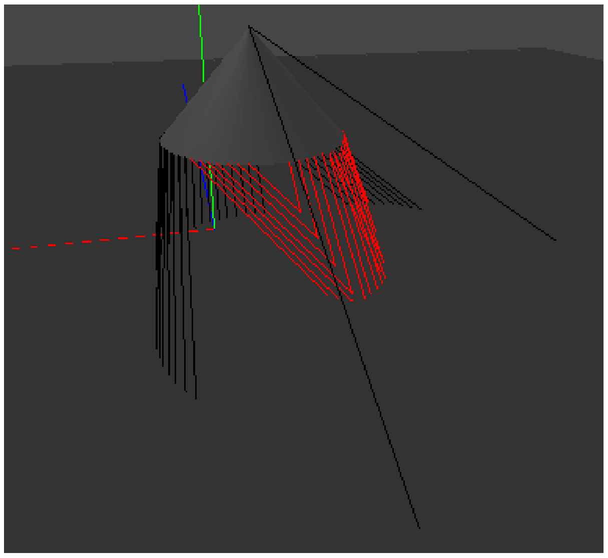


Figure 5.39: Cone under a triangular light: Convex hull applied

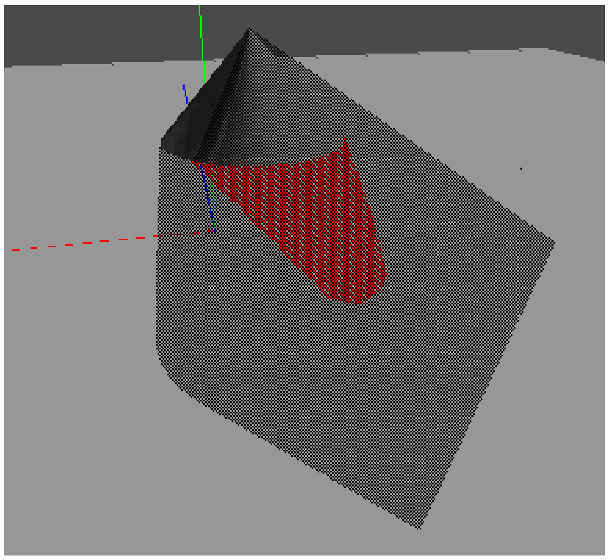


Figure 5.40: Cone under a triangular light: Convex hull with filled shadows

image. This is achieved by *painting* colours on the surfaces. The system finds the *best* values for the various surface parameters in order to produce these colours at their exact locations. We saw how the surface roughness could be determined by defining the size of a highlight but could not specify the final look of the highlight. In fact, while the final shape of the highlight follows its representation, if the surface has a very low specular contribution, this entire highlight might be invisible in the fully shaded image. This new painting system will allow us to control the colour of this highlight.

5.3 Defining Surface Characteristics

When drawing an image, an artist first sketches rough line drawings of his ideas. Once satisfied with the story, she starts refining the line drawings by adding more details and correcting the first drafts. Finally, when the line drawings express enough her feelings, the artist is ready for the last step, selecting the right colours and colouring.

While these steps are not mutually exclusive, we can generalise the process as:

- rough line drawings to convey spatial relationships
- correction and fine line drawings to provide fuller details in the image
- colour selection and filling for the final picture.

With the advent of computer graphics, new tools were developed, opening a new era for cartoon animation and special effects in cinema. These tools have been widely spread in logo-type animations but also their impacts were recognised in cartoon computer animations like *Tony de Peltrie* by Lachapelle-Bergeron-Robidoux-Langlois, *Luxo Jr., Red's Dream, Tin Toy* (Academy Award) and *Knick Knack* from Pixar, and in *Locomotion* from Pacific Data Images. These tools also opened new possibilities for special effects as in sequences from *The Wrath of Khan* by Lucasfilm and in sequences from *The Abyss, Terminator II* and *Jurassic Park* by Industrial Light and Magic.

Unfortunately, even though many software packages are now available, they still require a good knowledge of their shading techniques in order for a user to produce the exact results one originally thought of. This is true in each step of the drawing process described above, but becomes particularly true as the latest steps are reached. Getting a certain shading is very important in fully computer generated images. The appearance of a metallic surface is different than one of plastic. A user must therefore be able to manipulate the surface parameters to approximate well enough the reflection from the real surfaces. While a user has the possibility to alter the shading of the surrounding surfaces in order to improve the overall look of a final image, it becomes much more difficult to use the same tricks in special effects that normally require to merge computer generated objects with real objects. When two surfaces, a real one and a computer generated one, are next to each other and must look the same, it is essential that the simulated shading approximates visually the real shading of the surface. Within the limits of our simple reflection models, it becomes very difficult to find the "best" approximation.

A typical surface in computer graphics has several attributes. They include surface colour, ratio of diffuse versus specular reflection, surface roughness, transparency, index of refraction, etc. The number of attributes depends on the rendering system, the object representation and desired effects, whether the shading, reflection, refraction and shadowing models are physically based or pure hacks.

To evoke the right surface in painting, an artist must mix basic colours and apply them to specific locations on the canvas. A painter has the freedom to apply any colour at any given point in a scene. The realism of the painting relies on the skills of the artist in reproducing with colours what she perceives on the shading effect. Creating the right surface in computer graphics is difficult because of the possibly large number of parameters involved to provide at least the flexibility to create some basic shading effects. It is similar to painting in the sense that colours can be used to set certain parameters. It is different because mathematical models are given to approximate the behaviour of light and an artist is limited by their rules.

In the next sections, we will investigate how our knowledge of current shading properties can be used to free a computer graphics artist from knowing every surface parameter in order to produce a given shading effect on a surface. The system in itself is relatively simple to understand. An artist selects points on a surface and assigns colours to each of these points. A unique value for each shading parameter is provided and the remaining parts of the surface are shaded according to the values of these parameters.

5.3.1 Painting Systems

Since we present our system as a variation on a painting system, we describe in this section the evolution and state of the art in computer painting systems.

Since the beginning of computers, computer scientists have been interested in the large benefits computers can offer for interactively creating pictures. They are fast, precise, allow to undo previous commands and automate often very monotonous tasks. Sutherland in his Ph.D. thesis [suth63] presents many innovative ideas on how computers could be used as a drafting tool. Although his system basically deal only with line drawing representations of objects, many of his concepts have been extended to today's painting systems.

The typical painting system is 2D as it tries to simulate the general environment of real painters. The pencils are substituted by a mouse, a tablet or a light pen, the canvas by a computer screen, the paint by a palette of colours, the brushes by various shapes for the cursor, the eraser by a cursor removing colours, etc. The strong analogy between the tools of the artists and the tools offered in painting systems explains the general simplicity of the painting systems. They can also offer characteristics unique to computers. For example, it becomes possible to undo a command, remove paint, fill regions with a colour or a pattern, replace a colour by another with basic commands. Smith [smit82] describes many of standard painting system features. To give a look and feel closer to real paint and brushes, some systems use information about brush definition and movement [hobb85][fish85][stra86]. Others simulate textures closer to real paint [guo91][smal91][cock92].

The use of digitised images into a computer extends the power of painting systems. One can capture any image, apply various transformations to it (scaling and filtering are just two examples of a large class of image processing algorithms) and incorporate it in a painting. Haeberli [haeb90] presents such a painting system which samples a given (possibly real) image as the brush moves over it and redraws each region according to a painting style. He can simulate styles like cubism and impressionism. Cabral and Leedom [cabr93] use vector fields to produce similar effects on images.

Although 2D painting systems are the most popular, it is easy to see how researchers extended them to 2D$\frac{1}{2}$ (like in cel-based animation) by using an α-channel to composite together several images. By filtering and shading the edges in a 2D image, Williams [will91] creates an impression of pseudo-3D. He also presents a model for painting in 3D [will90] where the brush (cursor) is moved in the 3D space to apply paint (colour) to 3D volume elements.

Hanrahan and Haeberli [hanr90] describe a highly interactive 3D painting program. Based on the advances in workstation graphics hardware, they show how they achieve in real time the painting of various surface attributes directly onto the 3D surfaces. Examples of attributes they use include light and surface colours (ambient, diffuse and specular), surface roughness, bump and displacement mappings as well as some investigations on transparency. While their technique is a good step forward into more user-friendly mapping of surface attributes, it still requires the user to execute all the work of choosing colours and properties via standard techniques (entering numbers or moving sliders). This forces the user to understand the impacts of various shading parameters to obtain the desired effect. Seeing the variation of shading directly in the image as a particular slider moves is helpful but if one does not know which sliders have to move at which position, finding the right combination for a given shading effect can be a frustrating task.

In the next section, we will present a new kind of painting system. This system covers the middle ground between a traditional painting system where the user has full control of which colour is assigned to which pixel, and shading parameters selection where the user is given a shading effect by directly changing values for the various shading parameters. In this new system, a user does not have to understand the underlying mathematical shading model but still can control the shading to reach her goal. In the next sections, we will present the interface of our painting system and then explain the numerical methods used to provide the "best" values for the surface parameters.

5.3.2 Painting Scenario

The interface of our painting system is relatively simple. A user selects colours and applies them to points in the 3D scene. A colour point is represented by a small disk aligned on the surface along the normal at this point. The center of the disk indicates the 3D location of the colour point. The disk is painted with the selected colour but is not shaded. It is surrounded by a small ring of opposite colour [naim85] to easily detect and manipulate it on a surface. These colour points can be moved on the surface, deleted, their colour can be modified and the size of each disk can be scaled up or down. A larger disk is convenient to see properly the colour of each point and to manipulate it. Since colour is context sensitive, it is important to be able to increase the colour disk in order to better perceive this colour. A smaller disk occludes less of the underneath surface, allowing to better see the shading gradient around the colour point. This can be important to see small highlights. Figure 5.41 shows some colour points (represented by their disks) on a surface.

Our system is similar to the system presented by Hanrahan and Haeberli but limited only to colours on selected points on surfaces. However it goes a step beyond by trying to fit the "best" values to the shading parameters of this surface. By best values, we mean that the system attempts to optimise certain functions (under-constrained system) or to fit values (over-constrained system) so that the coloured points will remain as close as possible to their assigned colours when the full rendering is completed.

To select colours, we extended a basic colour tool provided by Haeberli. Three sliders indicate the proportion of each parameter in a given colour space. Five colour spaces are currently available: RGB, CMY, HSV, HLS and YIQ.[6] It is possible to convert each parameter

[6]It is not clear which colour space is the most appropriate in our context. We found the HLS colour space

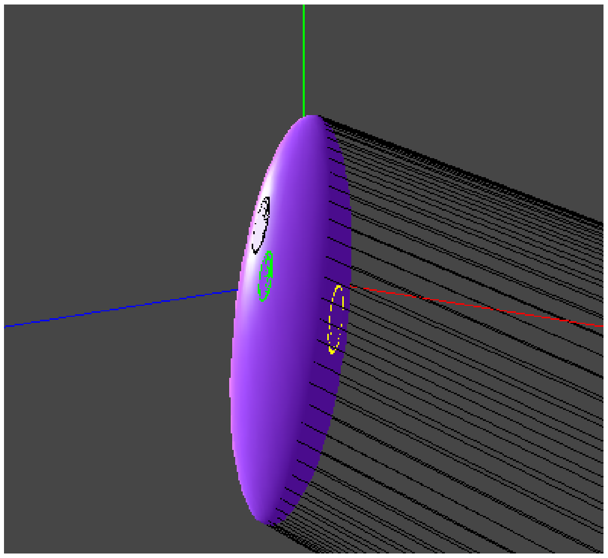


Figure 5.41: Colour points on a surface

from any to any of these colour spaces. The colour corresponding to the positions of the three sliders is displayed in the rightmost rectangle. Figure 5.42 shows the colour tool. This tool allows one to sample the colour of any pixel on the screen. This is a very useful feature to approximate the shading of real objects. Assume a real image is displayed on the monitor. The shape of the real object can be approximated in the modeler. Then, the colour of strategic points on the real surface can be selected and applied to the same points on the model in order to approximate the shading of the real surface. The colour tool has been extended to provide a range for a selected colour to relax the constraints (this is explained in the next section). It has also been modified to communicate directly with our modeler.

However useful our basic colour tool is, it could still be improved upon. Schwarz *et al.* [schw84] [schw87] compare various colour spaces and colour selection tasks. Their results should be integrated in our colour tool. However, as this goes beyond the scope of this thesis, we simply refer the reader to [wysz82] and [hall89] for more information about this topic.

For some combination of colour points, no values for the surface attributes will satisfy every constraint. When this happens, the system performs a rough and/or a finer investigation of altering the latest colour. Colours for which a converging solution can be found are indicated in our colour tool. The user can then select one of these new colours or decide to move the point to another location on the surface and let the system estimate if this new position leads to a possible solution.

to be useful for points within mostly the diffuse region of a surface. However because of our frequent use of the RGB colour space in other applications, we more often rely on this space.

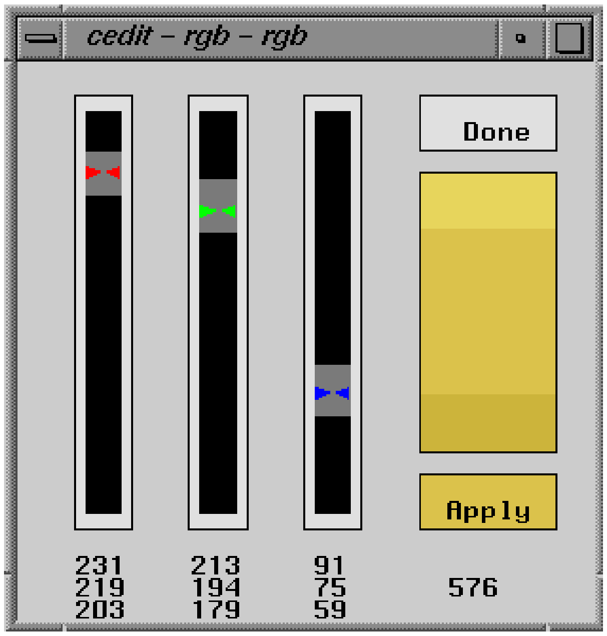


Figure 5.42: Colour tool of Haeberli

Now that we have a better feel of how the user interface of our painting system operates, we will present in the next sections the numerical algorithms used to find the "best" values for the surface parameters as determined by the colour points. We will see how each colour point can be interpreted as a system of equations, a constraint in an optimisation problem or a sample in a fitting problem.

5.3.3 Painting: Solving a System of Equations

Assume one has a 3D scene with all the geometry known and all the light sources positioned with their power fixed. This is a situation that often occurs in scene design. First the designer builds the geometry with temporary surface attributes. These attributes are mainly used to differentiate the various objects in a scene. They are subsequently modified once the scene is closer to completion. In indoor scenes, it is also often the case where the light sources have fixed positions and power within the scene geometry (light bulbs of 100 Watts hanging at precise locations from the ceiling). The lights can also be positioned with the techniques we described in the previous sections.

When a surface element is shaded, the radiance reflected to the pixel is function of the scene geometry and of the surface parameters. The general reflection model of Equation 2.2 applied to one directional light:

$$L_{pixel} = k_a L_{ia} + k_d \int_{\omega} (\vec{N} \cdot \vec{L}) L \, d\omega + k_s \int_{\omega} F_s (\vec{N} \cdot \vec{H})^n L \, d\omega$$

We can express this function in each colour channel (r,g and b)[7] as

$$\begin{aligned} L_{pixel}(r) &= k_a(r) L_{ia}(r) + \\ & \quad k_d(r) \int_{\omega} L(r)(\vec{N} \cdot \vec{L}) \, d\omega + \\ & \quad k_s(r) \int_{\omega} F_s(r) L(r) (\vec{N} \cdot \vec{H})^n \, d\omega \\ L_{pixel}(g) &= k_a(g) L_{ia}(g) + \\ & \quad k_d(g) \int_{\omega} L(g)(\vec{N} \cdot \vec{L}) \, d\omega + \\ & \quad k_s(g) \int_{\omega} F_s(g) L(g) (\vec{N} \cdot \vec{H})^n \, d\omega \\ L_{pixel}(b) &= k_a(b) L_{ia}(b) + \\ & \quad k_d(b) \int_{\omega} L(b)(\vec{N} \cdot \vec{L}) \, d\omega + \\ & \quad k_s(b) \int_{\omega} F_s(b) L(b) (\vec{N} \cdot \vec{H})^n \, d\omega \end{aligned} \tag{5.14}$$

If we consider $k_a(\lambda) L_{ia}(\lambda)$ and $F_s(\lambda)$ as constants then we have a reflection model which approximates the hardware SGI GL reflection model for a directional light.

If the value for each variable is known, the reflected radiance is easily computed. In our painting system, we attempt to solve the *inverse* problem. Therefore our system must find values for the surface characteristics that would satisfy the colour points if the full shading were performed. In the above reflection model, for a given point, all the surface attributes are independent in red, green and blue. Therefore without lost of generality, we can consider solving the problem in the red channel. The approach is identical for the green and blue channels.

For a given colour point, the known values in the diffuse reflection can be summed for all m lights as:

$$L_d(r) = \sum_{i=1}^{m} \int_{\omega_i} (\vec{N} \cdot \vec{L_i}) L_i(r) \, d\omega_i \qquad \text{for } \vec{N} \cdot \vec{L_i} > 0.$$

And similarly for the specular reflection:

$$L_s(r) = \sum_{i=1}^{m} \int_{\omega_i} (\vec{N} \cdot \vec{H_i})^n L_i(r) \, d\omega_i \qquad \text{for } \vec{N} \cdot \vec{L_i} > 0 \text{ and } \vec{N} \cdot \vec{H_i} > 0.$$

Generally speaking, $L_d(r)$ and $L_s(r)$ can be computed for any other type of light source, whether it is a point light, a linear light or an area light.

Each colour point contributes to a new equation in each of three channels. If there are as many independent equations as variables, the system of equations can be solved and values

[7]A colour channel consists in the convolution of light spectrum with a wavelength response curve.

identified for each surface attributes. Looking at Equation 5.14, if F_s is constant, we can therefore handle these terms as a single variable $(k_a(r), k_d(r), k_s(r))$ assuming they capture both the surface colour and proportion of ambient, diffuse and specular reflection, respectively. With the shading model of Equation 5.14, a system of three variables can be determined with only three colour points in the following form

$$\begin{bmatrix} 1 & L_{d1} & L_{s1} \\ 1 & L_{d2} & L_{s2} \\ 1 & L_{d3} & L_{s3} \end{bmatrix} \begin{bmatrix} k_a \\ k_d \\ k_s \end{bmatrix} = \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix}$$

and if a solution to this system exists, a unique value for k_a, k_d and k_s can be computed. Unfortunately, it is unlikely that a user will be able to provide the exact colours that would lead to a solution. We need to transform the problem such that it would lead to a solution. The next two sections will present two interpretations of this problem and their solutions.

5.3.4 Painting: An Optimisation Problem

We can look instead at this inverse problem as an optimisation problem. In this scheme, each colour point is given as a volume in the colour space of acceptable colours, thus introducing two constraints. Additional constraints are associated with the variables in the shading model. For instance, no variable can be negative. Moreover, some shading model can put an upper limit on the values of some of their parameters. The combination of all the constraints on a surface can lead to no solution, a unique solution (rare) or an infinity of solutions. In the latter, to provide the user with a unique solution, we need to minimise (or maximise) an objective function. Several objective functions are possible and each can lead to different behaviours of the system. It is also possible to use different objective functions depending on the number of colour points and their locations. For instance, one can decide to maximise the ambient term for the first colour point on a surface and to maximise the diffuse term when the next colour point is added. Or one can decide to use a series of objective functions like first maximising the diffuse term, bounding the value within a small range of this local maximum and then maximising the specular term.

In our current system, the choice of objective functions is based on the number of colour points and their location. Generally, we proceed as follow. We first minimise the ambient term, bound this minimum and then maximise the diffuse term. This way, shadow areas will be as dark as the user wants and the user will have the most control on the diffuse reflection. We only change the objective function if the first colour point is located in a shadow area. Then the diffuse reflection is not influenced by this colour and maximising it would lead to the maximum value allowable for the diffuse reflection. So instead, we minimise the diffuse term.

We found that this combination of objectives leads to a behaviour of our system that is intuitive and simple to understand and use. However it is possible to *personalise* the behaviour of the system. We can provide the user with a library of objective functions. The user can then interactively select the objectives and combine them. She could also decide which objectives to use depending on the number of points and their locations. The user would then need to understand the three types of reflections but nothing more about the specifics of the shading model.

The shading model of Equation 5.14 can be solved by a simple constrained linear optimisation algorithm. These algorithms have the advantage of converging to the global minimum. However most of the even slightly more sophisticated reflection models will often introduce nonlinear constraints.

Look at the following reflection model used in our local ray tracer [aman92]:

$$\begin{array}{rclclcl} L_{pixel}(r) & = & k_a S(r) & + & k_d S(r) L_d(r) & + & k_s L_s(r)\,[M S(r) + (1-M)] \\ L_{pixel}(g) & = & k_a S(g) & + & k_d S(g) L_d(g) & + & k_s L_s(g)\,[M S(g) + (1-M)] \\ L_{pixel}(b) & = & k_a S(b) & + & k_d S(b) L_d(b) & + & k_s L_s(b)\,[M S(b) + (1-M)] \end{array} \quad (5.15)$$

In this model, $S(\lambda)$ is a simple function of the surface colour. This model simulates a linear dielectric-conductor ratio ($M \in [0,1]$) for the specular reflection. When $M = 1$, the specular reflection is function of the light colour only. It is more complex than the previous reflection model. Also notice that the coefficients of ambient, diffuse and specular reflection (k_a, k_d, k_s) are the same for all three primaries. Therefore we cannot treat the solution independently in each channel. The constraints in this model are nonlinear. Some variables could be separated to form a system of independent equations but at the cost of introducing several quadratic terms. We preferred not to do so in order to experiment with the more general problem of solving an optimisation problem with nonlinear constraints.

We input these constraints and their gradients into FSQP [zhou93], a general algorithm to solve constrained nonlinear optimisation problems. FSQP is based on Sequential Quadratic Programming with two types of line searches in the solution domain: a monotonic search along an arc and a nonmonotonic search along a straight line. The interested reader should refer to the original report [zhou93] for a detailed version of the algorithms used. This algorithm performs very well (locally superlinear convergence) for a smooth objective function and smooth constraints. Minimising the ambient reflection or maximising the diffuse reflection generally satisfies the smoothness desired for an objective function. The specular reflection can generate a solution within a very narrow domain. If the initial guess is infeasible for some constraints, FSQP will attempt to find a feasible starting value but can fail to find one. If this happens, no optimisation can be performed.

To avoid this situation, we developed careful initial guesses based on our knowledge of the domains of each variable and within which region each colour point resides. Consider the following example with only two colour points, one in shadow and one in the diffuse area. Assume $k_a = 1$ and suppose we want to study the domain of possible values for k_d. We can use the colours to determine the limits such that

$$\begin{array}{rclclcll} D_l(r) & \leq & k_a S(r) & + & k_d & S(r)\cdot & L_d(r) & \leq D_t(r) \\ D_l(r) & \leq & A_l(r)(1+\Delta A(r)\cdot x) & + & k_d & A_l(r)(1+\Delta A(r)\cdot x) & L_d(r) & \leq D_t(r) \end{array} \quad (5.16)$$

where	$D_l(r)$	is the lower colour red for the diffuse colour point
	$D_t(r)$	is the top colour red for the diffuse colour point
	$A_l(r)$	is the lower colour red for the ambient colour point
	$A_l(r)(1+\Delta A(r))$	is the top colour red for the diffuse colour point
	$x \in [0,1]$	

Looking at the variation on k_d, we obtain

$$\left(\frac{D_l(r)}{A_l(r)(1+\Delta A(r)\cdot x)} - 1\right) \Big/ L_d(r) \le k_d \le \left(\frac{D_t(r)}{A_l(r)(1+\Delta A(r)\cdot x)} - 1\right) \Big/ L_d(r)$$

On a graph, this expression is represented in Figure 5.43. $k_{dl}(r)$ represents the lowest value for

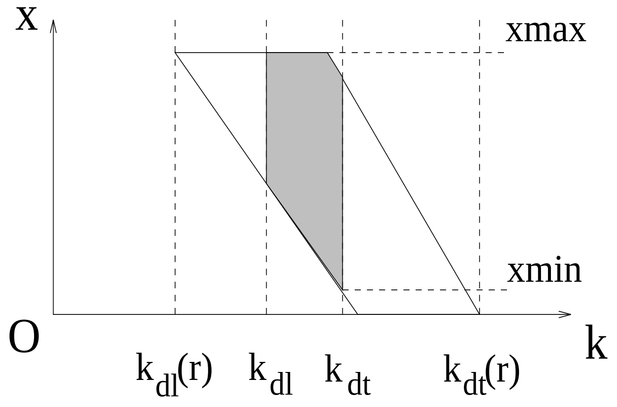


Figure 5.43: Domain of k_d

k_d for the red channel while $k_{dt}(r)$ is the top one. Each channel (r, g, b) creates a trapezoid. The intersection of these three trapezoids determines the limits for $k_d \in [k_{dl}, k_{dt}]$ and also for $S(c) = A_l(c)(1+\Delta A(c)\cdot x)$ with $x \in [xmin, xmax]$ and c is a colour channel.

When a colour point in a specular region is added, the same kind of analysis, although involving more parameters, must be performed to determine an initial guess that satisfies the constraints.

When FSQP cannot generate a valid initial guess and our investigation of each variable boundaries fails to identify a feasible initial guess, there is always the possibility that no solution exists that would satisfy all the constraints. When this happens, the system shows which constraints are violated (based on our study of the domain of each variable) and the user can modify the colour point, its boundaries or its position to allow for a different search of an initial guess.

When the colour points define more constraints than variables, the new constraints can only contribute to narrow down the range in which to search for a solution. This can be achieved in a simple manner by reducing the acceptable range of the colour points already located on this surface. In our experience, adding more colour points often only contributes to introduce new constraints that can be violated and therefore eliminates the possibility of providing a solution.

5.3.5 Painting: A Fitting Problem

We can interpret differently the colour points when more points than variables are input in the system. This becomes a typical problem of fitting the "best" approximation for each variable subjected to nonlinear constraints. We use a nonlinear least-squares fitting algorithm, *lmder*, taken from MINPACK. *lmder* is based on the Levenberg-Marquardt algorithm.

In a typical least-squares fitting, each variable is free. However in our shading models, each variable is constrained within a certain domain. To retain this concept of boundaries, we introduce penalty functions. A variable $f(x)$ is replaced by:

$$f(x) = \begin{cases} f(x) \cdot e^{bl-x} & \text{if } x < bl \\ f(x) & \text{if } bl \leq x \leq bu \\ f(x) \cdot e^{x-bu} & \text{if } bu < x \end{cases}$$

where bl and bu are the lower and upper boundaries on x, respectively. Any steep function could replace the exponential here. The steeper the function, the higher the penalty will be if a variable goes beyond its domain.

We also modify our system in order to provide different weights for different kinds of colour points. From the shading Equations 5.14 and 5.15, one can observe that a colour point within a shadow region on a surface can directly influence very few variables. Therefore we can increase by a large factor (100 in our system) the weight of these points. This ensures that the points in the other regions do not push the ambient term outside its limits. Similarly, the diffuse reflection is usually well controlled by one or two colour points. A smaller factor (10 in our system) keeps their contributions important, although less than the ambient ones. Finally, in most cases, the designer uses more colour points to finely control the variation of colours in the highlights. Each colour point in the highlight region will keep its unit contribution. By choosing appropriate weights, it is possible to approximate the behaviour of the objective functions. If done properly, a user will not notice the passage from an optimisation problem to a least-squares fitting one.

However, unlike the optimisation algorithm, the least-squares fitting does not guarantee that every colour point will retain its colour in the final rendering. The weights and penalties do help to keep the final colours as close as possible to the original ones. This technique offers a nice alternative to the optimisation when the user is not able to find an initial feasible guess or when the behaviour of the objective function does not correspond exactly to what the user expects of the final shading.

5.3.6 Results

A large advantage of this combination of optimisation and least-squares fitting relies to a certain degree on the fact that most of their use is invisible to the user. By this we mean that the user does not need to know how many variables there are in the reflection model and their contributions to the final shading of a surface. In fact, several different shading models could be used within the modeler and the user would have never to request one over another. The system could adapt itself to the demands of the user when she insists on certain colour points to be placed at specific locations.

This advantage can also be interpreted as a disadvantage. A poor choice of objective functions in the optimisation part or a poor combination of weights in the least-squares fitting could lead to strange behaviours of the system that the user could not understand because everything is hidden.

We found however that for the functions and weights we used, the behaviour of our system appeared intuitive, predictable and lead quickly to the desired shading. If the first colour points are placed mainly in the shadowed and diffuse regions of a surface, the user gets high control on the final shading with only a few points. Once these aspects of a surface are satisfying, the user can finely tune the look of the highlights with more colour points. In many of our test scenes, we found that three to four colour points often provide enough control to quickly produce a close approximation to the right values associated with the surface parameters.

It is important to provide the user with adequate feedback when colour points are added or moved on the surface. The real time hardware rendering with the SGI shading model allows one to see directly the surface change as the colour points are altered. Unfortunately, this is not possible for more sophisticated shading models unless they can somehow be converted or approximated by the SGI shading model. By keeping some information (3D location, surface normal, illuminant irradiance) for each pixel covered by the surface the user is currently working on, the entire surface can be reshaded efficiently when the user needs it.

5.3.7 Inverse Shading in Global Illumination

The problem of *inverse* shading has just recently been addressed in two different approaches applied to a different facet of the problem that we address in this thesis. Schoeneman *et al.* [scho93] describe how time consuming, tedious and often counter-intuitive selecting lights and surface reflectances can be in a system solving global illumination in a diffuse environment. This is true even for experienced users. They apply the inverse shading to identify the lights' intensities and colours that would closely match a target provided by a user. The user paints colours onto the scene with a tool like a spray can. By assuming only ideal diffuse surfaces and treating only the direct illumination, they devise clever incremental updates of the matrices and vectors that are then used in a constrained least-squares. This way, they can handle efficiently scenes with as many as 19,000 polygons and 12 lights. Even though they increase the weight of the painted scene vertices compared to the unpainted ones, their system cannot guarantee that the final colour will be close to the painted one because it is also function of the surface reflectance associated with each surface element. Also, since only the lights are modified, it is likely that other surfaces will have their appearance changed after each application of paint.

Kawai *et al.* [kawa93] also apply inverse shading to a radiosity solver. They use unconstrained nonlinear optimisation (Broyden-Fletcher-Goldfarb-Shanno) to find a local minimum of a possibly complex objective function. It includes physical terms (light source emission, directionality and distribution as well as element reflectivity) and terms based on human perception (impression of clearness, pleasantness, and privacy based on the scene brightness). In order to reduce the number of free variables, the user must select the active ones and impose constraints on them. Their system can provide complete results within a minute or two on a scene of a small conference room. They report however that their system can require "unintuitive tweaking" when the psychophysical properties of lighting are not accounted for.

5.3.8 Extensions

In our experience and as indicated Kawai *et al.* [kawa93], adding more variables to solve for usually has the effect of enlarging the domain of possible solutions. Unfortunately, depending on the behaviour of the new variables, they can introduce more local minima. When this happens, it becomes more difficult to determine if a given local minimum corresponds to a visually satisfying result for the objective function. Some techniques are available to start searches at various locations and therefore examine different local minima. This is in fact what Kawai *et al.* [kawa93] do.

We believe that some additional variables could be considered to extend our current solution. The first one is the roughness coefficient. We did not consider it because it was available by the way that we define and manipulate the highlights. Other variables from more sophisticated reflection models could be investigated. Some examples include transparency, anisotropy, diffraction, polarisation, layered surfaces, etc. Similarly to Schoeneman *et al.* [scho93] and Kawai *et al.* [kawa93], we could also consider altering the lights intensities and colours and apply our techniques in a radiosity solver. We could even attempt to define and manipulate the lights from the colour points.

However, with all those possibly more complex shading effects and with a growing number of free variables, developing efficient objective functions that keep the behaviour of our system intuitive and powerful becomes a task as difficult as solving for the constraints.

5.4 Conclusion

In this chapter, we investigated using lighting effects, i.e. shading, highlights and shadows, to define the lights themselves and specify their location as well as identify the surface shading parameters. We showed some inherent limitations with these approaches but also demonstrated a powerful new technique. This technique allows a user to manipulates interactively shading, highlights and shadows, which can be very important when designing a scene. In previous modeling systems, these effects were not under the direct control of the user. Therefore a user needed to iterate between rendering the entire scene and modifying the lights. This process can be expensive depending of the quality of the rendering required. Incorporating shading, highlights and shadows in the modeling process adds more information on the geometry of the scene and its illumination which should help the user to better understand the scene even before rendering it.

Our system, although simple, gives direct information to the user on the lighting effects she is constructing during the modeling process. These lighting effects are the objects being directly manipulated. This direct manipulation is crucial because getting the desired shading effect by manipulating the causes is generally more difficult than manipulating the effects themselves.

We foresee that, as the graphics hardware improves and as CPUs become faster, more and more effects available only at the rendering stage will become an inherent part of the modeling stage itself. Real time Phong shading is now common with high-end modelers. These improvements will lead us to investigate more intuitive ways of defining and controlling these special effects. Although the separation between computer graphics and computer vision is still strong, we believe this will lead us to more computer vision in computer graphics for greater benefits of realism and potentially more computer graphics in computer vision for better scene analysis of natural phenomena.

Chapter 6

Conclusion

What we see is the result of interaction between light and matter. The intensities reaching our eyes are interpreted to help us understand the world we live in. We can identify objects, their surface properties, their relative positions in 3D space, their motion, etc. By improving our understanding of light and its interaction with matter, we can also improve our understanding of the matter itself.

3D computer graphics is one technique used to represent a 3D synthetic world by projecting this world onto an array of pixels. The simulation of light transport and its interaction with virtual objects determine the colour of each pixel. One active area of research in computer graphics consists in simulating as accurately as possible our real world. This search for realism is important because our visual system was developed to understand the world we live in. So by producing realistic computer generated pictures, we can expect people to better understand an artificial world.

In this thesis, we focused our research onto one aspect of realism: the local illumination. In local illumination, we are interested in shading a surface directly illuminated by light sources. Typically in computer graphics, people were using directional and point light sources as illuminants. They are satisfactory for distant or small lights but lack realism and flexibility when a higher dimensional light is needed. For the few people using linear or polygonal light sources, only diffuse reflection was computed analytically while specular reflection was simply approximated via sampling. Assuming the popular Phong specular expression, we presented an analytical solution to the diffuse as well as to the specular reflections from a linear light source. Our solution is suitable for various extensions on light emission and has also been extended by Tanaka and Takahashi [tana91b] for computing the reflection from a polygonal light source.

The increase in realism in the shading of surfaces illuminated by linear and area light sources comes unfortunately with a higher complexity for computing shadows. We gathered several properties of shadows to form a theoretical foundation. These results help us to better understand the cost of correctly computing shadows. With this in mind, we presented two algorithms to reduce the number of possibly occluding candidates for a linear light source and discussed the algorithms currently used for shadowing with polygonal lights.

While the first part of this thesis addressed mainly rendering issues, the second part approached the generation of realistic shading from a modeling standpoint. When modeling a scene, a designer traditionally moves the objects in the virtual world, moves the lights and inputs the surface characteristics by assigning values to various shading parameters. Unfortunately, all along this process, little feedback is given until the end of the rendering process. As the rendering process becomes more expensive due to the desire of achieving a stronger realism, the number of iterations between modeling and rendering increases, often leading to frustrating situations. By knowing well the shading model used, an experienced designer can reduce this number of iterations but this knowledge will not always be applicable to systems with different shading models.

In the second part of this thesis, we presented a shift from the traditional modeling systems. We introduced rendering issues directly in the modeling process. It therefore becomes possible to indirectly alter the causes by manipulating the effects. We investigated defining and positioning light sources by moving on a surface a point of highest diffuse or specular intensity or by moving the shadow volumes cast by an object. We also showed how the specular exponent can be specified by selecting the size of a highlight and how various surface properties can be identified by painting colours onto points on a surface. With all these techniques, a designer should achieve a given shading effect in much less time than previously and this, without any special knowledge of the shading model.

Creating the *right* picture with today's computer rendering technology is an art. The designer must face intrinsic limitations due to the modeling process, the rendering algorithms and, very important but too often neglected, the user interface used as an intermediary between what is wanted and how to do it. This research alleviates the burden of the rendering task by extending the types of light sources available and by introducing rendering issues directly in the modeling process. We expect these results to have a direct impact on tomorrow's rendering technology.

Bibliography

[akel93] Kurt Akeley. "RealityEngine graphics". *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 109–116, August 1993.

[aman84] John Amanatides. "Ray Tracing with Cones". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 129–135, July 1984.

[aman87] John Amanatides and Andrew Woo. "A fast voxel traversal algorithm for ray tracing". *Eurographics '87*, pp. 3–10, August 1987.

[aman92] John Amanatides, John Buchanan, Pierre Poulin, and Andrew Woo. "Optik Users' Manual — Version 2.6". Technical Report Imager 1992–1, University of British Columbia, August 1992.

[athe78] P. Atherton, K. Weiler, and D. Greenberg. "Polygon Shadow Generation". *Computer Graphics (SIGGRAPH '78 Proceedings)*, Vol. 12, No. 3, pp. 275–281, August 1978.

[aupp93] Larry Aupperle and Pat Hanrahan. "A hierarchical illumination algorithm for surfaces with glossy reflection". *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 155–162, August 1993.

[babu85] Mohan D.R. Babu, Chia-Hoang Lee, and Azriel Rosenfeld. "Determining Plane Orientation from Specular Reflectance". *Pattern Recognition*, Vol. 18, No. 1, pp. 53–62, January 1985.

[bao93] Hujun Bao and Qunsheng Peng. "Shading models for linear and area light sources". *Computers and Graphics*, Vol. 17, No. 2, pp. 137–145, March/April 1993.

[barb92] Christopher G. Barbour and Gary W. Meyer. "Visual cues and pictorial limitations for computer generated photo-realistic images". *The Visual Computer*, Vol. 9, No. 3, pp. 151–165, December 1992.

[baum91] Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James M. Winget. "Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 51–60, July 1991.

[beck93] Barry G. Becker and Nelson L. Max. "Smooth transitions between bump rendering algorithms". *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 183–190, August 1993.

[berg86] P. Bergeron. "A General Version of Crow's Shadow Volumes". *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, pp. 17–28, September 1986.

[bish86] G. Bishop and D.M. Weimer. "Fast Phong Shading". *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, No. 4, pp. 103–106, August 1986.

[blin77] James F. Blinn. "Models of Light Reflection For Computer Synthesized Pictures". *Computer Graphics (SIGGRAPH '77 Proceedings)*, Vol. 11, No. 2, pp. 192–198, July 1977.

[blin88] James F. Blinn. "Jim Blinn's Corner: Me and my (fake) shadow". *IEEE Computer Graphics and Applications*, Vol. 8, No. 1, pp. 82–86, January 1988.

[bonf86] L. Bonfigliolo. "An Algorithm for Silhouette of Curved Surfaces based on Graphical Relations". *Computer-Aided Design*, Vol. 18, No. 2, pp. 95–101, March 1986.

[borg91] Carlos F. Borges. "Trichromatic approximation for computer graphics illumination models". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 101–104, July 1991.

[buch87] Craig Stuart Buchanan. "Determining Surface Orientation from Specular Highlights". M.Sc. Thesis, Department of Computer Science, University of Toronto, August 1987.

[cabr87] Brian Cabral, Nelson Max, and Rebecca Springmeyer. "Bidirectional Reflection Functions from Surface Bump Maps". *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol. 21, No. 4, pp. 273–281, July 1987.

[cabr93] Brian Cabral and Leith Casey Leedom. "Imaging vector fields using line integral convolution". *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 263–270, August 1993.

[camp90] A.T. Campbell and Donald S. Fussell. "Adaptive Mesh Generation for Global Diffuse Illumination". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 155–164, August 1990.

[camp91] A.T. Campbell and Donald S. Fussell. "An analytic approach to illumination with area light sources". Technical Report TR-91-25, Department of Computer Science, University of Texas at Austin, August 1991.

[chen91] Shenchang Eric Chen, Holly E. Rushmeier, Gavin Miller, and Douglass Turner. "A progressive multi-pass method for global illumination". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 165–174, July 1991.

[chin89] Norman Chin and Steven Feiner. "Near Real-Time Shadow Generation Using BSP Trees". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 99–106, July 1989.

[chin92] Norman Chin and Steven Feiner. "Fast object-precision shadow generation for area light sources using BSP trees". *Computer Graphics Special Issue (1992 Symposium on Interactive 3D Graphics)*, Vol. 26, pp. 21–30, March 1992.

[cock92] Tunde Cockshott, John Patterson, and David England. "Modelling the texture of paint". *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings)*, Vol. 11, No. 3, pp. 217–226, September 1992.

[cohe85] M.F. Cohen and D.P. Greenberg. "The Hemi-Cube: A Radiosity For Complex Environments". *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, pp. 31–40, July 1985.

[cohe93] Michael F. Cohen and John R. Wallace. *Radiosity and realistic image synthesis.* Academic Press, 1993.

[cook82] R.L. Cook and K.E. Torrance. "A Reflectance Model for Computer Graphics". *ACM Transactions on Graphics*, Vol. 1, No. 1, pp. 7–24, January 1982.

[cook84a] R.L. Cook. "Shade trees". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 223–231, July 1984.

[cook84b] Robert L. Cook, Thomas Porter, and Loren Carpenter. "Distributed Ray Tracing". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 137–145, July 1984.

[crow77] Franklin C. Crow. "Shadow Algorithms for Computer Graphics". *Computer Graphics (SIGGRAPH '77 Proceedings)*, Vol. 11, No. 2, pp. 242–248, July 1977.

[dias91] Maria Lurdes Dias. "Ray tracing interference color". *IEEE Computer Graphics and Applications*, Vol. 11, No. 2, pp. 54–60, March 1991.

[dret93] George Drettakis and Eugene Fiume. "Accurate and consistent reconstruction of illumination functions using structured sampling". *Computer Graphics Forum (EUROGRAPHICS '93 Proceedings)*, Vol. 12, No. 3, September 1993.

[firb85] P.A. Firby and D.J. Stone. "Interference in Computer Graphics". *Computer Graphics Forum (Eurographics '85 Proceedings)*, Vol. 4, No. 3, pp. 209–216, September 1985.

[fish85] Kenneth P. Fishkin and Brian A. Barsky. "Algorithms for brush movement". *The Visual Computer*, Vol. 1, No. 4, pp. 221–230, December 1985.

[fole90] J.D. Foley, A. van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley Publishing Company, second edition, 1990.

[four92a] Alain Fournier. "Filtering Normal Maps and Creating Multiple Surfaces". Technical Report Imager 92/1, Imager, Computer Science, University of British Columbia, 1992.

[four92b] Alain Fournier. "Normal distribution functions and multiple surfaces". *Graphics Interface '92 Workshop on Local Illumination*, pp. 45–52, May 1992.

[four92c] Alain Fournier and John Buchanan. "Chebyshev polynomials for boxing and intersections". To be submitted to ACM Transactions on Graphics, 1992.

[gart90] Judith A. Gartaganis and John Tartar. "Wave-based spectrum rendering". Technical Report TR 90-13, Department of Computer Science, University of Alberta, May 1990.

[gene93] Jon Genetti and Dan Gordon. "Ray tracing with adaptive supersampling in object space". *Proceedings of Graphics Interface '93*, pp. 70–77, May 1993.

[gers87] Ron Gershon. *The use of color in computational vision.* Ph.D. thesis, Dept. of Computer Science, University of Toronto, 1987.

[gigu90] Ziv Gigus and Jitendra Malik. "Computing the aspect graph for line drawings of polyhedral objects". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 2, pp. 113–122, February 1990.

[gigu91] Ziv Gigus, John Canny, and Raimund Seidel. "Efficient computing and representing aspect graphs of polyhedral objects". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 6, pp. 542–551, June 1991.

[glas89] Andrew S. Glassner. "How to Derive a Spectrum from an RGB Triplet". *IEEE Computer Graphics and Applications*, Vol. 9, No. 4, pp. 95–99, July 1989.

[guo91] Qinglian Guo and T.L. Kunii. "Modeling the diffuse painting of sumie". *IFIP Modeling in Computer Graphics*, 1991.

[haeb90] Paul Haeberli. "Paint By Numbers: Abstract Image Representations". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 207–214, August 1990.

[hain86] Eric A. Haines and Donald P. Greenberg. "The Light Buffer: A Ray Tracer Shadow Testing Accelerator". *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, pp. 6–16, September 1986.

[hain87] Eric Haines. "A Proposal for Standard Graphics Environments". *IEEE Computer Graphics and Applications*, Vol. 7, No. 11, pp. 3–5, November 1987.

[hain91] Eric Haines and John Wallace. "Shaft culling for efficient ray-traced radiosity". *Eurographics Workshop on Rendering*, 1991.

[hall83] R.A. Hall and D.P. Greenberg. "A Testbed for Realistic Image Synthesis". *IEEE Computer Graphics and Applications*, Vol. 3, No. 8, pp. 10–20, November 1983.

[hall89] R. Hall. *Illumination and Color in Computer Generated Imagery.* Springer-Verlag, 1989.

[hall93] David E. Hall and Holly E. Rushmeier. "Improved explicit radiosity method for calculating non-Lambertian reflections". *The Visual Computer*, Vol. 9, No. 5, pp. 278–288, March 1993.

[hanr90] Pat Hanrahan and Paul Haeberli. "Direct WYSIWYG Painting and Texturing on 3D Shapes". *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 215–223, August 1990.

[hanr91] Pat Hanrahan, David Salzman, and Larry Aupperle. "A rapid hierarchical radiosity algorithm". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 197–206, July 1991.

[hanr93] Pat Hanrahan and Wolfgang Krueger. "Reflection from layered surfaces due to subsurface scattering". *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 165–174, August 1993.

[he91] Xiao D. He, Kenneth E. Torrance, Francois X. Sillion, and Donald P. Greenberg. "A comprehensive physical model for light reflection". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 175–186, July 1991.

[heck84] Paul S. Heckbert and Pat Hanrahan. "Beam Tracing Polygonal Objects". *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, pp. 119–127, July 1984.

[heck91a] Paul S. Heckbert. *Simulating global illumination using adaptive meshing.* Ph.D. Thesis, Computer Science Division (EECS), University of California, Berkeley, June 1991.

[heck91b] 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, July 1991.

[heck92a] Paul Heckbert. "Discontinuity meshing for radiosity". *Eurographics Workshop on Rendering*, pp. 203–216, 1992.

[heck92b] Paul S. Heckbert. "Radiosity in flatland". *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings)*, Vol. 11, No. 3, pp. 181–192, September 1992.

[hern92] Kenneth P. Herndon, Robert C. Zeleznik, Daniel C. Robbins, D. Brookshire Conner, Scott S. Snibbe, and Andries van Dam. "Interactive shadows". *Symposium on User Interface Software and Technology*, pp. 1–6, November 1992.

[hobb85] John Douglas Hobby. *Digitized Trajectories.* Ph.D. Thesis, Stanford University, 1985.

[horn88] Berthold K. P. Horn and M.J. Brooks, editors. *Shape from Shading.* MIT Press, 1988.

[houl91] Caroline Houle. "Light Source Modelling". M.Sc. Thesis, Department of Computer Science, University of Toronto, 1991.

[imme86] D.S. Immel, M.F. Cohen, and D.P. Greenberg. "A Radiosity Method for Non-Diffuse Environments". *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, No. 4, pp. 133–142, August 1986.

[jans91] Frederik W. Jansen and Arno N.T. van der Zalm. "A shadow algorithm for CSG". *Computers and Graphics*, Vol. 15, No. 2, pp. 237–247, 1991.

[kauf81] J.E. Kaufman and H. Haynes. *IES Lighting Handbook*. Illuminating Engineering Society of North America, 1981.

[kawa93] John K. Kawai, James S. Painter, and Michael F. Cohen. "Radioptimization — Goal based rendering". *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 147–154, August 1993.

[kok91] Arjan Kok and Frederik Jansen. "Source selection for the direct lighting component in global illumination". *Eurographics Workshop on Rendering*, 1991.

[kok92] Arjan J.F. Kok and Frederik W. Jansen. "Adaptive sampling of area light sources in ray tracing including diffuse interreflection". *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings)*, Vol. 11, No. 3, pp. 289–298, September 1992.

[krin47] E.L. Krinov. *Spectral reflectance properties of natural formations*. Laboratoria Aerometodov, Akad. Nauk SSSR, Moscow, 1947.

[lewi93] Robert Lewis. "Making shaders more physically plausible". *Eurographics Workshop on Rendering*, pp. 47–62, 1993.

[lisc92] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. "Discontinuity meshing for accurate radiosity". *IEEE Computer Graphics and Applications*, Vol. 12, No. 6, pp. 25–39, November 1992.

[lisc93] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. "Combining hierarchical radiosity and discontinuity meshing". *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 199–208, August 1993.

[max93] Nelson Max and Roy Troutman. "Optimal hemicube sampling". *Eurographics Workshop on Rendering*, pp. 185–200, 1993.

[meye88] G. W. Meyer. "Wavelength Selection for Synthetic Image Generation". *Computer Vision, Graphics and Image Processing*, Vol. 41, pp. 57–79, 1988.

[meye90] Urs Meyer. "Hemi-Cube Ray-Tracing: A Method for Generating Soft Shadows". *Eurographics '90*, pp. 365–376, September 1990.

[meye91] Gary Meyer and Richard Hale. "A Spectral Database for Realistic Image Synthesis". *Proceedings of Graphics Interface '91*, pp. 47–52, June 1991.

[mitc91] Don P. Mitchell. "Spectrally optimal sampling for distribution ray tracing". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 157–164, July 1991.

[mora81] Hans P. Moravec. "3D Graphics and the Wave Theory". *Computer Graphics (SIGGRAPH '81 Proceedings)*, Vol. 15, No. 3, pp. 289–296, August 1981.

[musg89] F. Kenton Musgrave. “Prisms and rainbows: a dispersion model for computer graphics”. *Proceedings of Graphics Interface '89*, pp. 227–234, June 1989.

[naim85] Avi Naiman. “Color spaces and color contrast”. *The Visual Computer*, Vol. 1, No. 3, pp. 194–201, November 1985.

[nico77] F.E. Nicodemus, J.C. Richmond, J.J. Hsia, I.W. Ginsberg, and T. Limperis. “Geometrical Considerations and Nomenclature for Reflectance”, October 1977.

[nish83] Tomoyuki Nishita and Eihachiro Nakamae. “Half-Tone Representation of 3-D Objects Illuminated by Area or Polyhedron Sources”. *Proc. of IEEE Computer Society's Seventh International Computer Software and Applications Conference (COMPSAC83)*, pp. 237–242, November 1983.

[nish85] T. Nishita, I. Okamura, and E. Nakamae. “Shading Models for Point and Linear Sources”. *ACM Transactions on Graphics*, Vol. 4, No. 2, pp. 124–146, April 1985.

[nish92] T. Nishita, S. Takita, and E. Nakamae. “A Shading Model of Parallel Cylindrical Light Sources”. *Visual Computing (Proceedings of CG International '92)*, 1992.

[peer93] Mark S. Peercy. “Linear color representations for full spectral rendering”. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 191–198, August 1993.

[pent82] Alex P. Pentland. “Finding the illuminant direction”. *Journal of Optical Society of America*, Vol. 72, No. 4, pp. 448–455, April 1982.

[phon75] Bui-T. Phong. “Illumination for Computer Generated Pictures”. *Communications of the ACM*, Vol. 18, No. 6, pp. 311–317, June 1975.

[pico92] Kevin P. Picott. “Extensions of the linear and area lighting models”. *IEEE Computer Graphics and Applications*, Vol. 12, No. 2, pp. 31–38, March 1992.

[piet93] Georg Pietrek. “Fast calculation of accurate form factors”. *Eurographics Workshop on Rendering*, pp. 201–220, 1993.

[poul90a] Pierre Poulin and John Amanatides. “Shading and Shadowing with Linear Light Sources”. *Eurographics '90*, pp. 377–386, September 1990.

[poul90b] Pierre Poulin and Alain Fournier. “A Model for Anisotropic Reflection”. *Computer Graphics (SIGGRAPH '90 Proceedings)*, Vol. 24, No. 4, pp. 273–282, August 1990.

[poul91a] Pierre Poulin. “An extended shading model for linear light sources”. *Proceedings of the 1991 Western Computer Graphics Symposium*, pp. 31–35, April 1991.

[poul91b] Pierre Poulin and John Amanatides. “Shading and shadowing with linear light sources”. *Computers and Graphics*, Vol. 15, No. 2, pp. 259–265, 1991.

[poul92a] Pierre Poulin. “Separate functions for local illumination”. *Graphics Interface '92 Workshop on Local Illumination*, pp. 37–43, May 1992.

[poul92b] Pierre Poulin and Alain Fournier. “Lights from highlights and shadows”. *Computer Graphics Special Issue (1992 Symposium on Interactive 3D Graphics)*, Vol. 26, pp. 31–38, March 1992.

[poul92c] Pierre Poulin and Alain Fournier. “Lights from highlights and shadows”. *Proceedings of the 1992 Western Computer Graphics Symposium*, pp. 141–145, April 1992.

[prep85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer Verlag, Berlin, Germany, 1985.

[raso91] Maria Raso and Alain Fournier. “A Piecewise Polynomial Approach to Shading Using Spectral Distributions”. *Proceedings of Graphics Interface '91*, pp. 40–46, June 1991.

[rush90] Holly E. Rushmeier and Kenneth E. Torrance. “Extending the Radiosity Method to Include Specularly Reflecting and Translucent Materials”. *ACM Transactions on Graphics*, Vol. 9, No. 1, pp. 1–27, January 1990.

[sale92] D. Salesin, D. Lischinski, and T. DeRose. “Reconstructing illumination functions with selected discontinuities”. *Eurographics Workshop on Rendering*, pp. 99–112, 1992.

[schl93] Christophe Schlick. “A customizable reflectance model for everyday rendering”. *Eurographics Workshop on Rendering*, pp. 73–83, 1993.

[scho93] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenberg. “Painting with light”. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pp. 143–146, August 1993.

[schw84] M.W. Schwarz, J.C. Beatty, W.B. Cowan, and J.F. Gentleman. “Towards an effective user interface for interactive colour manipulation”. *Graphics Interface '84 Proceedings*, pp. 187–196, 1984.

[schw87] Michael W. Schwarz, William B. Cowan, and John C. Beatty. “An experimental comparison of RGB, YIQ, LAB, HSV, and opponent color models”. *ACM Transactions on Graphics*, Vol. 6, No. 2, pp. 123–158, April 1987.

[sedg90] Robert Sedgewick. *Algorithms in C.* Addison-Wesley, 1990.

[sega92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. “Fast shadows and lighting effects using texture mapping”. *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, No. 2, pp. 249–252, July 1992.

[shaf85] Steven A. Shafer. *Shadows and Silhouettes in Computer Vision.* Kluwer Academic Publishers, 1985.

[shin87] Mikio Shinya, Tokiichiro Takahashi, and Seiichiro Naito. “Principles and Applications of Pencil Tracing”. *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol. 21, No. 4, pp. 45–54, July 1987.

[shir91] Peter S. Shirley. *Physically Based Lighting Calculations for Computer Graphics.* Ph.D. Thesis, Computer Science Department, University of Illinois at Urbana-Champaign, 1991.

[sill89] Francois Sillion and Claude Puech. "A General Two-Pass Method Integrating Specular and Diffuse Reflection". *Computer Graphics (SIGGRAPH '89 Proceedings)*, Vol. 23, No. 3, pp. 335–344, July 1989.

[sill91a] Francois Sillion. "The state of the art in physically-based rendering and its impact on future applications". *Eurographics Workshop on Rendering*, 1991.

[sill91b] Francois X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. "A global illumination solution for general reflectance distributions". *Computer Graphics (SIGGRAPH '91 Proceedings)*, Vol. 25, No. 4, pp. 187–196, July 1991.

[smal91] David Small. "Simulating Watercolor by Modeling Diffusion, Pigment, and Paper Fibers". *Proceedings of SPIE '91*, February 1991.

[smit82] Alvy Ray Smith. "Paint". *Tutorial: Computer Graphics*, pp. 501–515, 1982.

[smit89] B.E. Smits and G.M. Meyer. "Newton colors: Simulating interference phenomena in realistic image synthesis". *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, 1989.

[stew93] A. James Stewart and Sharif Ghali. "An output sensitive algorithm for the computation of shadow boundaries". *Proceedings of the fifth Canadian Conference on Computational Geometry*, pp. 291–296, August 1993.

[stra86] S. Strassmann. "Hairy Brushes". *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, No. 4, pp. 225–232, August 1986.

[suth63] I.E. Sutherland. *Sketchpad: A man-machine graphical communication system.* SJCC. Spartan Books, 1963.

[tana90] Toshimitsu Tanaka and Tokiichiro Takahashi. "Cross Scanline Algorithm". *Eurographics '90*, pp. 63–74, September 1990.

[tana91a] Toshimitsu Tanaka and Tokiichiro Takahashi. "Precise rendering method for edge highlighting". *Scientific Visualization of Physical Phenomena (Proceedings of CG International '91)*, pp. 283–298, 1991.

[tana91b] Toshimitsu Tanaka and Tokiichiro Takahashi. "Shading with Area Light Sources". *Eurographics '91*, pp. 235–246, September 1991.

[tell92] Seth J. Teller. "Computing the antipenumbra of an area light source". *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, No. 2, pp. 139–148, July 1992.

[torr67] K.E. Torrance and E.M. Sparrow. "Theory for Off-Specular Reflection from Roughened Surfaces". *Journal of Optical Society of America*, Vol. 57, No. 9, 1967.

[verb84] C.P. Verbeck and D.P. Greenberg. "A Comprehensive Light-Source Description for Computer Graphics". *IEEE Computer Graphics and Applications*, Vol. 4, No. 7, pp. 66–75, July 1984.

[walt75] David Waltz. "Understanding Line Drawings of Scenes with Shadows". *The Psychology of Computer Vision*, pp. 19–91. Mc-Graw Hill, New York, 1975.

[ward91] Gregory Ward. "Adaptive shadow testing for ray tracing". *Eurographics Workshop on Rendering*, 1991.

[ward92a] Gregory J. Ward. "Measuring and modeling anisotropic reflection". *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, No. 2, pp. 265–272, July 1992.

[ward92b] Gregory J. Ward. "Towards more practical reflectance measurements and models". *Graphics Interface '92 Workshop on Local Illumination*, pp. 15–21, May 1992.

[warn83] D.R. Warn. "Lighting Controls for Synthetic Images". *Computer Graphics (SIGGRAPH '83 Proceedings)*, Vol. 17, No. 3, pp. 13–21, July 1983.

[weil77] K. Weiler and K. Atherton. "Hidden surface removal using polygon area sorting". *Computer Graphics (SIGGRAPH '77 Proceedings)*, Vol. 11, No. 2, pp. 214–222, July 1977.

[west92] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. "Predicting reflectance functions from complex surfaces". *Computer Graphics (SIGGRAPH '92 Proceedings)*, Vol. 26, No. 2, pp. 255–264, July 1992.

[will90] Lance Williams. "3D Paint". *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, Vol. 24, No. 2, pp. 225–233, March 1990.

[will91] Lance Williams. "Shading in Two Dimensions". *Proceedings of Graphics Interface '91*, pp. 143–151, June 1991.

[wolf90] Lawrence B. Wolff and David J. Kurlander. "Ray Tracing with Polarization Parameters". *IEEE Computer Graphics and Applications*, Vol. 10, No. 6, pp. 44–55, November 1990.

[wolf91] Lawrence B. Wolff. *Polarization Methods in Computer Vision.* Ph.D. thesis, Columbia University, 1991.

[wolf92] Lawrence Wolff, Steven A. Shafer, and Glenn Healey, editors. *Radiometry.* Physics-based Vision: Principles and Practice. Jones and Bartlett, 1992.

[woo90] Andrew Woo, Pierre Poulin, and Alain Fournier. "A Survey of Shadow Algorithms". *IEEE Computer Graphics and Applications*, Vol. 10, No. 6, pp. 13–32, November 1990.

[woo93] Andrew Woo. "Efficient shadow computations in ray tracing". *IEEE Computer Graphics and Applications*, Vol. 13, No. 5, pp. 78–83, September 1993.

[wysz82] G. Wyszecki and W.S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae.* Wiley, 1982.

[yuan88] Ying Yuan, Tosiyasu L. Kunii, Naota Inamoto, and Lining Sun. "GemstoneFire: adaptive dispersive ray tracing of polygons". *The Visual Computer*, Vol. 4, No. 5, pp. 259–270, November 1988.

[zhou93] Jian L. Zhou and Andre L. Tits. "User's guide for FSQP version 3.2: A Fortran code for solving constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality and linear constraints". Technical Report TR-92-107r2, Electrical Engineering Department and Institute for Systems Research, University of Maryland at College Park, March 1993.