A data structure for a multi-illumination model renderer.

Falby, John Stephen.

Monterey, California: U.S. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A DATA STRUCTURE FOR A
MULTI-ILLUMINATION MODEL RENDERER

by

John Stephen Falby

December 1986

Thesis Advisor: Michael J. Zyda

Approved for public release; distribution is unlimited

UNCLASSIFIED
ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S)	S MONITORING ORGANIZATION REPORT NUMBER(S)
a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) Code 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
a NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

TITLE (Include Security Classification)
A DATA STRUCTURE FOR A MULTI-ILLUMINATION MODEL RENDERER

PERSONAL AUTHOR(S)
Falby, John Stephen

a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM _____ TO _____	14 DATE OF REPORT (Year, Month Day) 1986 December	15 PAGE COUNT 118

SUPPLEMENTARY NOTATION

COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Multi-Illumination Model Renderer, Data Structure, implementation details

ABSTRACT (Continue on reverse if necessary and identify by block number)
The rendering of realistic computer images is important for many scientific, technical and commercial endeavors. Available literature provides the mathematical models to be utilized by a renderer. Lacking from the literature though are implementation details. In this study, we examine some of the existing illumination and shading models and present a data structure and initial design for a multi-illumination model renderer.

DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
a NAME OF RESPONSIBLE INDIVIDUAL Michael J. Zyda	22b TELEPHONE (Include Area Code) 408-646-2305 22c OFFICE SYMBOL 52Zk

D FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

A Data Structure For A Multi–Illumination Model Renderer

by

John Stephen Falby
Lieutenant Commander, United States Navy
B. A., The Citadel, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

December 1986

ABSTRACT

The rendering of realistic computer images is important for many scientific, technical and commercial endeavors. Available literature provides the mathematical models to be utilized by a renderer. Lacking from the literature though are implementation details. In this study, we examine some of the existing illumination and shading models and present a data structure and initial design for a multi-illumination model renderer.

TABLE OF CONTENTS

I. INTRODUCTION

The art of making computer-generated images appear more realistic is called rendering and is an ongoing area of research in the field of computer graphics. Realistic images are important to many scientific, technical and commercial applications of computer graphics. Research is centered primarily on modeling the way light interacts with objects made of various materials and textures, and how the color for each point of the display device is determined.

Available literature abounds with mathematical models for reflected light intensity calculations, called illumination models, and for color assignment computations, called shading models [Refs. 1-9]. However, implementation details are rarely discussed. The reason is economic. People make their livelihood from generating realistic computer-generated images and their implementation must remain secret if they are to maintain a competitive edge. The purpose of this study is to unveil some of that secrecy and present full and complete implementation details.

A. A MULTI-ILLUMINATION MODEL RENDERER

A renderer is a computer program that implements an illumination and shading model for use on a computer-generated image with the goal of making that image appear more realistic. A multi-illumination model (MIM) renderer, for

purposes of this study, is a renderer that incorporates multiple illumination and shading models and allows the user to select the models to be utilized in rendering a scene.

1. Inputs

A MIM renderer requires several inputs in order to render a scene. The scene itself must be defined and consists of a background and graphics objects. For purposes of this paper, a graphics object is a collection of polygonal surfaces. The color and specific properties of the surfaces must be known in order to model how light interacts with them. The light sources illuminating the scene are defined in terms of intensity, location and composition. The view position must be known in order to display the three-dimensional scene on the two-dimensional display device.

2. Output

The final output of our renderer is the set of pixel values useful for display on an RGB color monitor such as that found on a Silicon Graphics IRIS 2400 graphics workstation. The output necessary to effect the display of the shaded scene is either an index into a color map, or a full 24-bit RGB color specification.

B. METHODOLOGY

We begin by examining some of the currently known illumination and shading models with the goal of understanding and capturing the precise inputs

necessary to implement each model. Once we have compiled a list of the input data necessary, we present a data structure and beginning design for our MIM renderer.

C. ORGANIZATION

The organization of this study follows the well-trodden path found in the literature [Refs. 3, 6, 8: pp. 575-591, pp. 276-295, pp. 309-410]. We begin in Chapter Two by reviewing some basics about light. Chapter Three examines a simple illumination model, how it can be broken down to cover even simpler cases and how it can be expanded to other cases. Chapter Four examines various shading models. Chapter Five examines a recent illumination model that supports ray tracing. Chapter Six examines the precise data required to support some of the models discussed in previous chapters and how that data is obtained. Implementation details are presented in Chapter Seven. Known limitations of the data structure are presented in Chapter Eight. Chapter Nine outlines areas for future research and contains concluding remarks.

II. THE INTERACTION BETWEEN LIGHT RAYS AND OBJECTS

Before we can examine illumination models, the basic properties of light and its interaction with objects must be understood. We therefore review some basic properties of light.

A. LIGHT SOURCES

For purposes of modeling light behavior in computer graphics. light sources are generally classified as either point sources or distributed sources in the literature. A point source is small in size relative to the scene it is illuminating, as viewed from the scene, and is distant enough so that its light rays can be assumed to be parallel. A distributed source, on the other hand, is relatively large when viewed from the scene and its light rays can not be considered parallel. An additional consideration with respect to distributed light sources is that many rays from a single distributed source can illuminate the same point on an object, such as a neon light illuminating a chair. A special case of a distributed light source is the background, or ambient, illumination present in scenes. This illumination comes from the multiple reflections from objects in the scene whether visible from the viewing position or not. Hence the "source" is the whole scene. Ambient light is responsible for objects in shadow with respect to other light

sources being visible. Distributed light sources so far from the scene that their light rays illuminating the scene are nearly parallel are considered point sources.

A third type of light source is one that is small in relation to the scene as viewed from the scene, yet is so close that its rays cannot be assumed parallel. We call such light sources near point sources. The rays from a near point source are considered to emanate from a single point yet are not parallel. Therefore, each point on an object is illuminated by a single ray from the near point source.

B. THE INTENSITY OF INCIDENT LIGHT

Incident light is the light that illuminates an object or scene. As light travels outward from its source. its intensity decreases inversely as the square of the distance traveled. This well-known property of light is readily illustrated by the fact that objects closer to a light source appear brighter than objects that are farther away. The intensity of the light incident on an object is thus less than the intensity of the light source itself.

C. WHAT HAPPENS WHEN LIGHT STRIKES AN OBJECT?

Figure 2.1 illustrates the three possible results when a light ray strikes an object. The incoming light vector, $\vec{L}$, strikes the surface of the object at point P. Two of the results are illustrated by vectors $\vec{R}$ and $\vec{T}$. Vector $\vec{R}$ is the reflected vector while $\vec{T}$ is the transmitted vector. The third possibility is that all or part of the light is absorbed by the object. Absorbed light is converted to heat energy. This explains why stage performers can be sweating under stage lights while

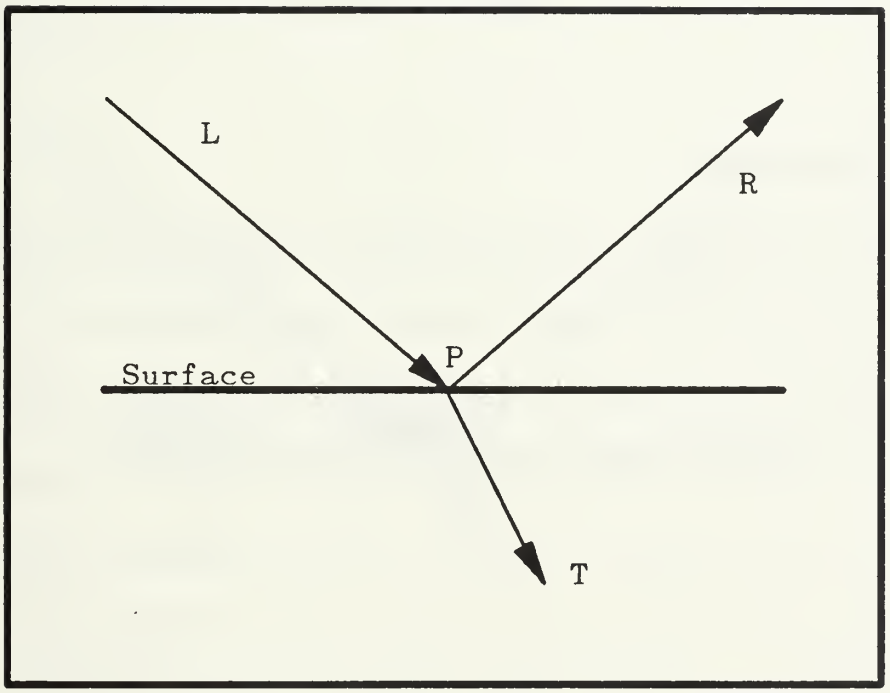


Fig. 2.1 - Reflection and Transmission of Incident Light

members of the audience sitting in the darkened auditorium feel chilly. If the surface at point P is colored and and the incident light is white, Figure 2.1 expands to Figure 2.2. The light vector $\vec{L}$ is represented by its three color components: red, green and blue. The surface of the object at point P is assumed to be blue and slightly translucent. The red and green components are totally absorbed and converted to heat energy while the blue component is both reflected and transmitted. We concern ourselves now with the reflected and transmitted rays.

1. Reflected Light

For reflected light, we have two types of reflection: diffuse reflection and specular reflection. Diffuse reflection results from the graininess of the surface of an object and is of equal intensity in all directions as illustrated in Figure 2.3. The intensity of the diffuse reflection of light from a point source by a perfect diffuser is determined from Lambert's cosine law. This law states that the intensity is proportional to the cosine of the incident angle (Figure 2.4). The incident angle, θ, is the angle between the light vector, $\vec{L}$, and the normal to the surface, $\vec{N}$. The cosine of the angle between the two vectors is equal to their dot product divided by the product of their length (Figure 2.5). Hence

$$\cos(\theta) = \frac{\vec{N} \cdot \vec{L}}{|\vec{N}||\vec{L}|}$$

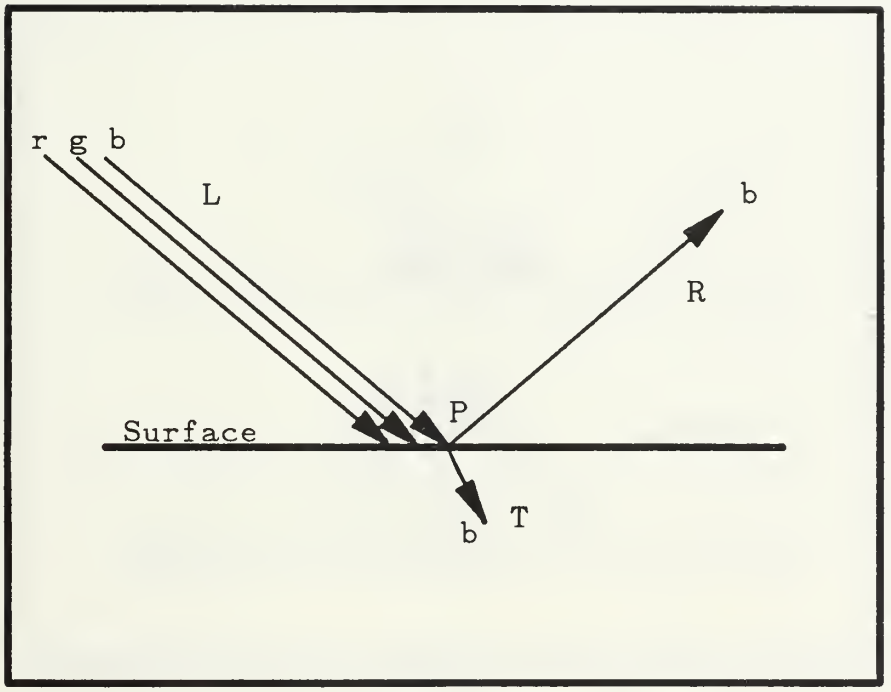


Figure 2.2

Reflection and Transmission Dependent on RGB Component

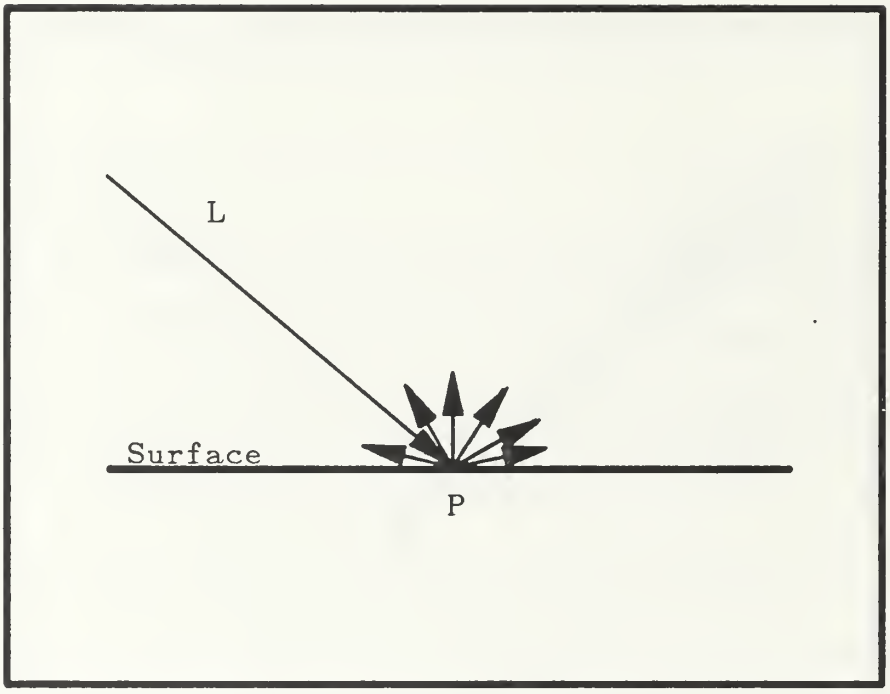


Figure 2.3 - Diffuse Reflection

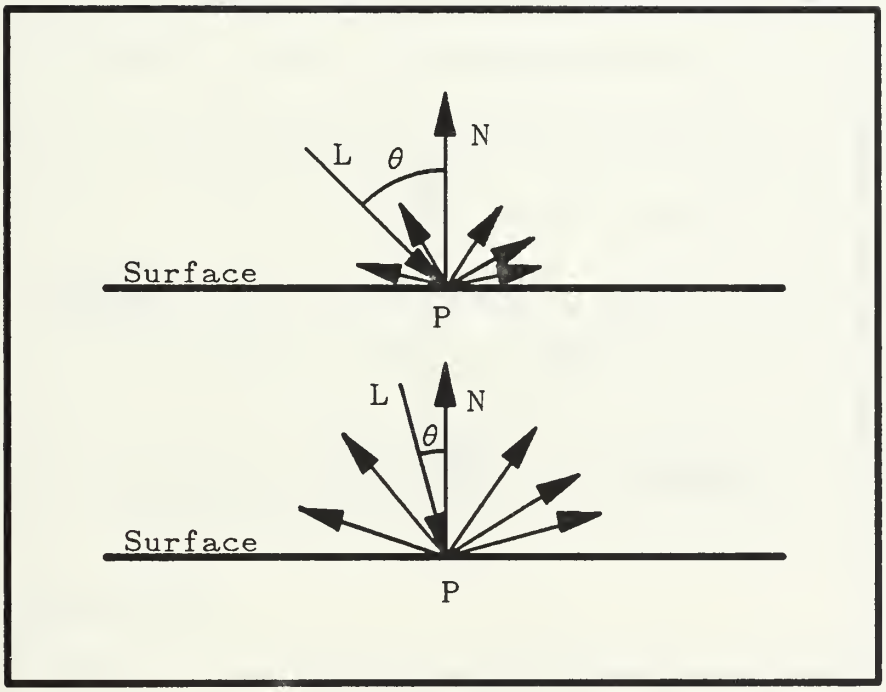


Figure 2.4
Intensity of Diffuse Reflection Proportional to Incident Angle

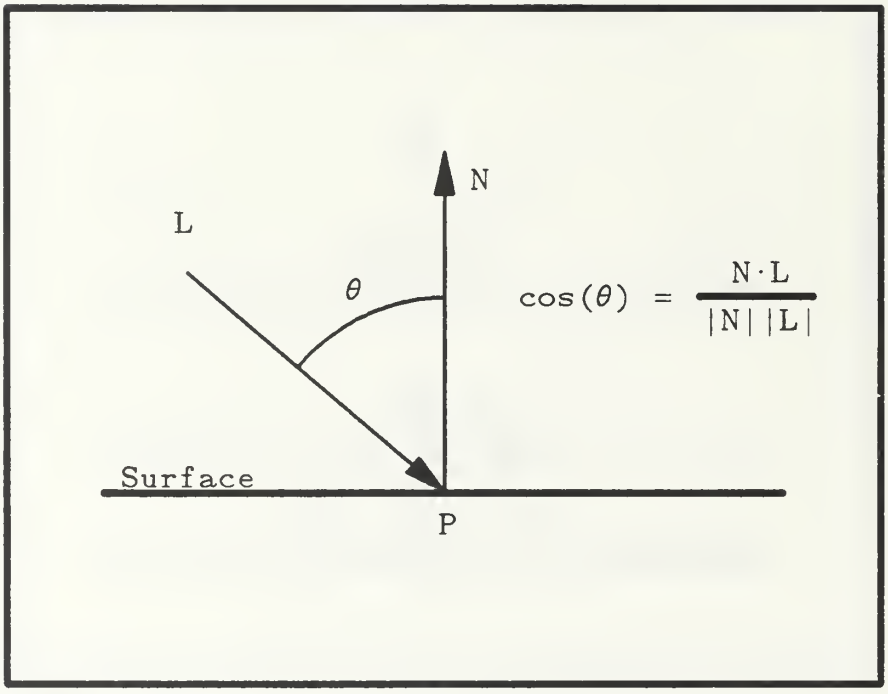


Figure 2.5 - Lambert's Cosine Law

If we let $\hat{L}$ be the unit light vector and $\hat{N}$ be the unit surface normal, then the above equation becomes

$$\cos(\theta) = \hat{\mathbf{N}} \cdot \hat{\mathbf{L}}$$

Therefore, the intensity of diffuse reflection is proportional to the dot product of the unit surface normal and the unit light vectors.

Whereas diffuse reflection is non-directional, specular reflection is directional. For a perfect reflector, it lies along the reflectance vector, $\vec{R}$, shown in Figure 2.6, and is equal in intensity to that of the incident vector, $\vec{L}$. Real objects are seldom perfect reflectors, however, and exhibit a spatial distribution about $\vec{R}$ forming a specular cone. The intensity within the cone is strongest along $\vec{R}$ and falls off to 0 at the edge of the cone. The size of the specular cone and intensities at various points within the cone depend on surface properties of the object, the intensities of the components making up the light and the incident angle, θ.

2. Transmitted Light

The light that passes through a translucent object is called transmitted light. Figure 2.7 illustrates the bending, or refraction, of the incident light vector, $\vec{L}$, resulting in the transmitted vector, $\vec{T}$, having a different direction. The direction of the transmitted vector obeys Snell's Law which is

$$\mathbf{r}_1 \times \sin(\theta_i) = \mathbf{r}_2 \times \sin(\theta_r)$$

where

- r_1 is the index of refraction for the medium through which the incident vector arrives.

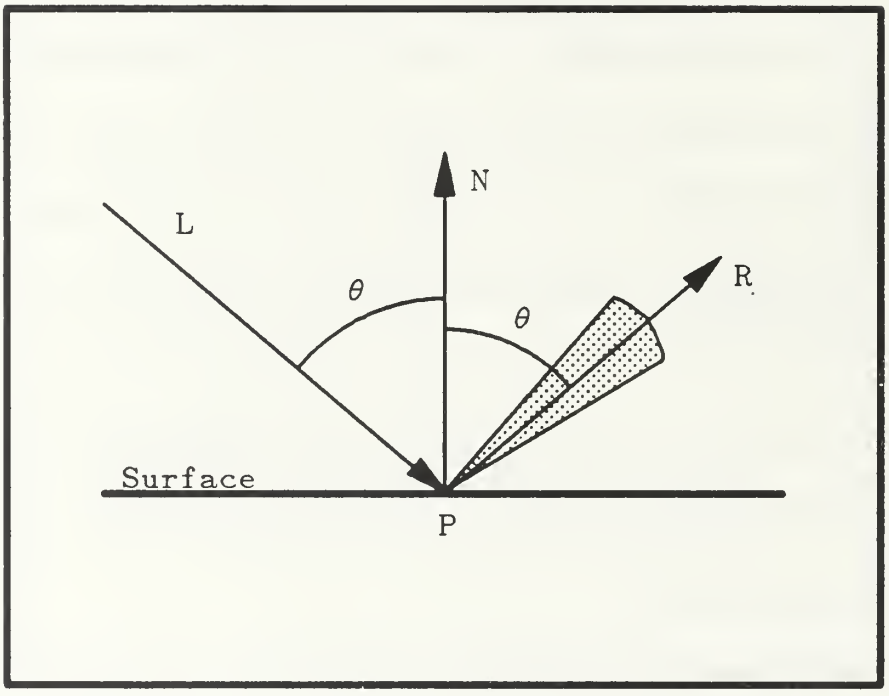


Figure 2.6 - Specular Reflection and Cone

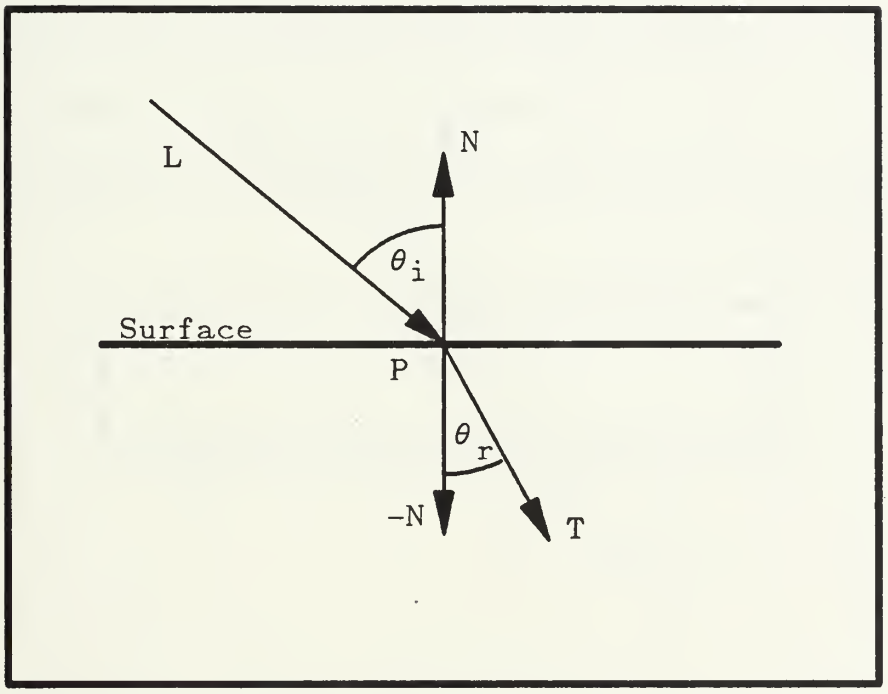


Figure 2.7 - Transmitted Light and Transmission

- r_2 is the index of refraction for the medium through which the transmitted vector passes.
- θ_i is the angle of incidence.
- θ_r is the angle of refraction.

Figure 2.8 illustrates what happens when the refracted vector passes completely through the intervening material. The incident vector is refracted upon entering the material in accordance with Snell's Law and again when it exits the object. The cumulative effect is that the resultant vector, $\vec{T}$, is parallel to the incident light vector, $\vec{L}$, but shifted. The magnitude of the shift depends on the index of refraction of the object and the distance the refracted ray must travel through the object.

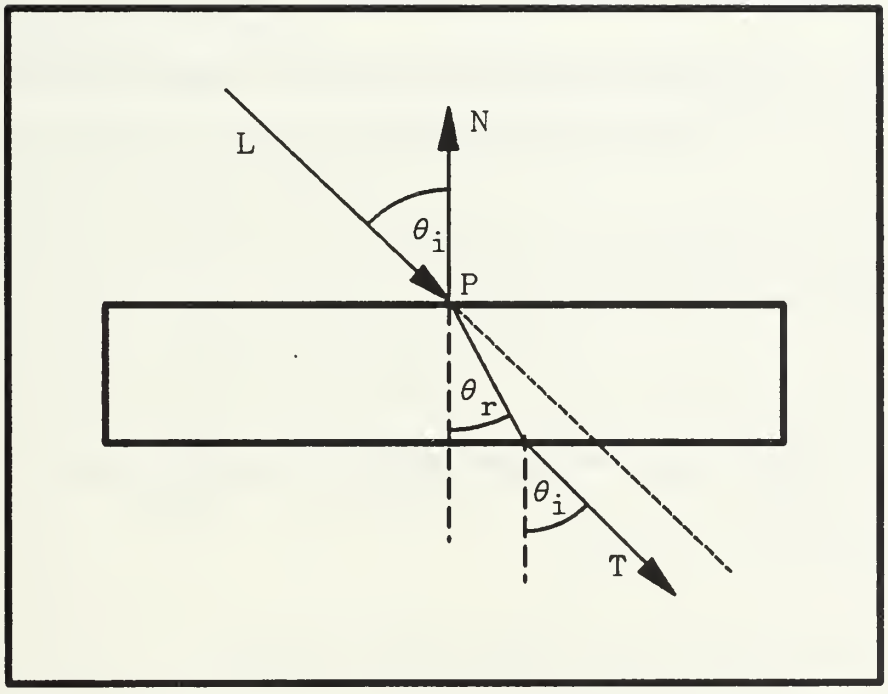


Figure 2.8 - Transmitted Light and Ray Shift

III. ILLUMINATION MODELS

As the number of existing illumination models is large, we limit ourselves to examining only a few. We first examine a model which calculates intensity due to diffuse and specular reflection from point sources. We then propose a model for reflected intensity from distributed light sources. These models are termed "local" in that they model only the interaction between an object and the light source while ignoring the interaction of light rays reflected between objects. We examine a "global" model proposed by Whitted [Ref. 9: pp. 343-349] that models the local illumination due to reflected light as well as the global illumination due to reflected and transmitted light in a later chapter. In all cases, we strive to understand not only the illumination model, but also what data is required by each to perform the calculation.

A. A LOCAL MODEL FOR POINT LIGHT SOURCES

Phong [Ref 7: pp. 311-317] proposed an illumination model that modeled the effects of diffuse and specular reflection. We examine a model based on Phong which is a merging of models in Hearn and Baker [Ref. 6: p. 280] and Rogers [Ref. 8: pp. 311-317]. The model is

$$I = k_d I_a + \frac{I_p}{d + d_0}[k_d(\vec{N} \cdot \vec{L}) + k_s(\vec{R} \cdot \vec{V})^n]$$

where

- I is the total intensity reflected from the point on the surface of an object.
- k_d is the diffuse coefficient.
- I_a is the intensity of ambient light.
- I_p is the intensity of the illuminating point source.
- d is the distance of the object from the view position.
- d_0 is an arbitrary constant to avoid division by 0 when d is small.
- $\vec{N}$ is the unit normal to the surface at the point.
- $\vec{L}$ is the unit incident light vector in the direction to the light source.
- k_s is the specular coefficient.
- $\vec{R}$ is the unit reflection vector of $\vec{L}$ at the illuminated point.
- $\vec{V}$ is the unit view vector in the direction of the viewer.
- n is Phong's specular exponent.

The model can be easily expanded to include multiple light sources and color [Refs. 6. 8: p. 379, p. 315]. The expanded model is

$$I = \sum_{i=r,g,b} \left[k_d I_{a_i} + \sum_{j=1}^{ls} \left(\frac{I_{P_{ij}}}{d + d_0} [k_{d_i}(\vec{N} \cdot \vec{L}_j) + k_s(\vec{R}_j \cdot \vec{V})^n] \right) \right]$$

where

- i varies through the red, green and blue components of light.
- j varies from 1 to the number of point light sources.
- ls is the number of point light sources.

In the simple model, the first term models the intensity of reflection due to ambient light, the second term models the intensity due to diffuse reflection of light incident from a point source, and the last term models the intensity of the specular reflection. Since the model is simply a linear sum, it can be broken down to model the intensity of illumination due to ambient light alone, the intensity of

illumination due to diffuse reflection alone, the intensity of illumination due to specular reflection alone, or any combination of these. We examine each term separately.

1. Ambient Term

The ambient term, $\mathbf{k_d I_a}$, models the intensity of reflection from a point on the surface of an object due to the ambient light source. The diffuse coefficient, $\mathbf{k_d}$, varies from 0.0 for no diffuse reflection to 1.0 for a perfect diffuser to account for the fact that all objects are not perfect diffusers. Rogers [Ref. 8: p. 313] and Foley and Van Dam [Ref. 3: p. 576] use a separate ambient coefficient which is different from the $\mathbf{k_d}$ of the diffuse term, while others, such as Whitted [Ref. 9: p. 344], delete it altogether. However, the reflection due to ambient light is a diffuse reflection and the diffuse coefficient can serve both the ambient and diffuse terms. Since different surfaces reflect different amounts of the ambient light, the modeling coefficient is retained.

Table 3.1 lists the input necessary for computing the ambient term. The object column lists the data that changes from object to object or from polygon to polygon within an object. The view column lists data that is either view-dependent or global to the scene. The light column lists data that is associated with each light source.

TABLE 3.1 - DATA REQUIRED BY AMBIENT TERM

OBJECT	VIEW	LIGHT
k_{d_r}	I_{a_r}	
k_{d_g}	I_{a_g}	
k_{d_b}	I_{a_b}	

2. Diffuse Term

The diffuse term, $\frac{k_d I_p(\vec{N} \cdot \vec{L})}{d + d_0}$, models the intensity contribution from diffuse reflection of incident light from the point source. I_p represents the intensity of the illuminating point source. As mentioned in the previous chapter, the intensity of light decreases inversely as the square of the distance traveled. Hence, this effect can be modeled by $\frac{k_d I_p(\vec{N} \cdot \vec{L})}{d^2}$. However, if the source is sufficiently far from the scene, the diffuse term offers no contribution. Hearn and Baker [Ref. 6: pp. 278-279] and Rogers [Ref. 8: p. 313] point out that experience has shown that a linear attenuation of $\frac{1}{d}$ works well asthetically in modeling the decrease in intensity over distance. In this case, the distance is taken to be in relation to either the view position or in relation to the object closest to the light source. If the latter approach is taken, the closest object is assigned a distance d of 0 while other objects are assigned a distance equal to their distance from that object. One result is that the object closest to the light source is illuminated by the full intensity of the light source while the objects further away are dimmer. We take d in relation to the view position. Therefore, the view position and the position of the illuminated point must be known. d_0 is arbitrarily chosen to prevent division by 0. It can also be varied to fine tune the resulting rendered scene.

Diffuse reflection obeys Lambert's Cosine Law. Hence, diffuse reflection from a point on a perfect diffuser is proportional to the cosine of the angle of

incidence, θ. The diffuse term models this by making use of

$$\cos\theta = \vec{N} \cdot \vec{L}$$

where $\vec{N}$ and $\vec{L}$ are the unit normal and incident light vectors, respectively. The position of the light source and the illuminated point must be known to determine $\vec{L}$. A unit surface normal, $\vec{N}$, is associated with each polygon of the graphics object. It is obtained by taking the cross product of two non-parallel edges of the polygon. Since most real objects are not perfect diffusers, the diffuse coefficient, k_d, is included to give the percentage of incident light that is diffusely reflected by the object. Table 3.2 summarizes the data necessary to compute the diffuse term.

3. Specular Term

$\frac{k_s I_p(\vec{V} \cdot \vec{R})^n}{d+d_0}$ is the specular term based on Phong and models the intensity due to specularly reflected light. Table 3.3 lists the data required for computing the specular term. $\vec{V} \cdot \vec{R}$ gives the cosine of the angle α between the unit view vector, $\vec{V}$, and the unit reflection vector, $\vec{R}$ (Figure 3.1). This models the falloff in intensity of the specularly reflected light as α increases. However, $\vec{V} \cdot \vec{R}$ models a constant-size specular cone and falloff in intensity while the specular cones and falloff rates for different objects differ depending on the surface material. In general, the spatial distribution, and hence the diameter of the specular cone, is small for glossy surfaces and large for dull surfaces. Also, the falloff in intensity as $\vec{V}$ moves away from $\vec{R}$ is more rapid than that modeled by

TABLE 3.2 - DATA REQUIRED BY DIFFUSE TERM

OBJECT	VIEW	LIGHT
k_{d_r}	d_O	I_{p_r}
k_{d_g}	Position of Viewpoint	I_{p_g}
k_{d_b}		I_{p_b}
$\vec{N}$		Position of Light
Position of Illuminated Point		

TABLE 3.3 - DATA REQUIRED BY SPECULAR TERM

OBJECT	VIEW	LIGHT
k_{s_r}	d_O	I_{p_r}
k_{s_g}	Position of Viewpoint	I_{p_g}
k_{s_b}		I_{p_b}
$\vec{N}$		Position of Light
n		
Position of Illuminated Point		

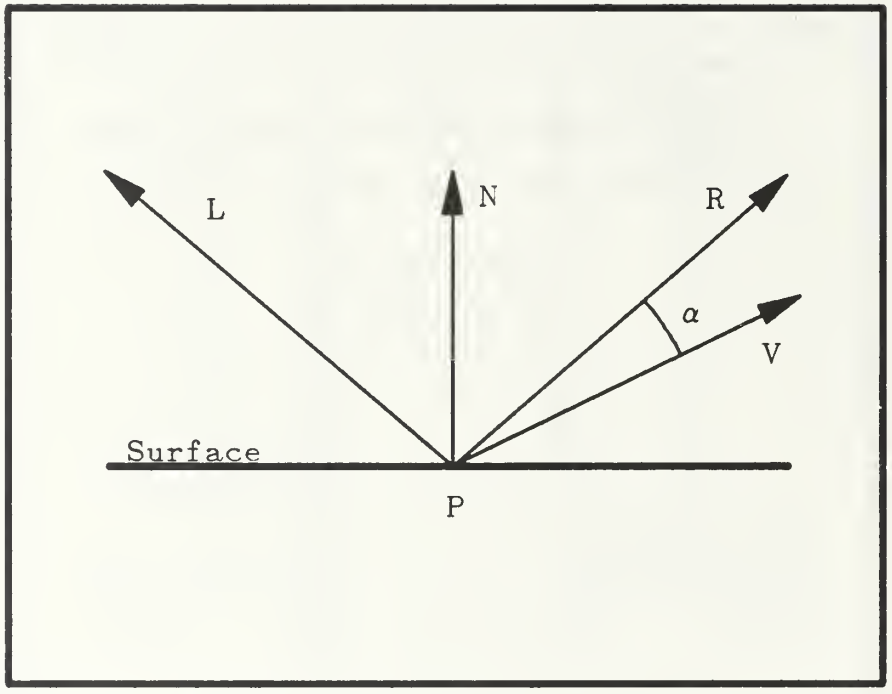


Figure 3.1 - Angle Between View and Reflection Vectors

$\vec{V} \cdot \vec{R}$. Therefore, Phong included the specular exponent, n. For very shiny surfaces n is made very large, greater than 200, while for dull surfaces n is made smaller.

The unit reflection vector, $\vec{R}$, may be obtained directly from the unit light vector, $\vec{L}$, and unit normal vector, $\vec{N}$. Referring to Figure 3.2, we first obtain $\vec{L}'$ from

$$\vec{L}' = \frac{\vec{L}}{|\vec{L} \cdot \vec{N}|}$$

$\vec{L}'$ is then the vector in the same direction as $\vec{L}$, but with a length equal to the distance from the illuminated point, P, and the intersection of $\vec{L}$ with a line parallel to the tangent at P and passing through the head of $\vec{N}$. We now obtain $\vec{R}'$ by

$$\vec{R}' = \vec{L}' + 2\vec{N}$$

As illustrated in Figure 3.3, $\vec{R}'$ makes an angle with $\vec{N}$ which is equal to the angle of incidence, θ. Therefore, $\vec{R}'$ is in the same direction as $\vec{R}$. We may now determine $\vec{R}$ by

$$\vec{R} = \frac{\vec{R}'}{|\vec{R}'|}$$

The view vector is obtained from the view position and the position of P. $\frac{1}{d + d_0}$ again models the decrease in intensity of the incident light over distance. k_s is the specular coefficient and models the fact that only a percentage of the

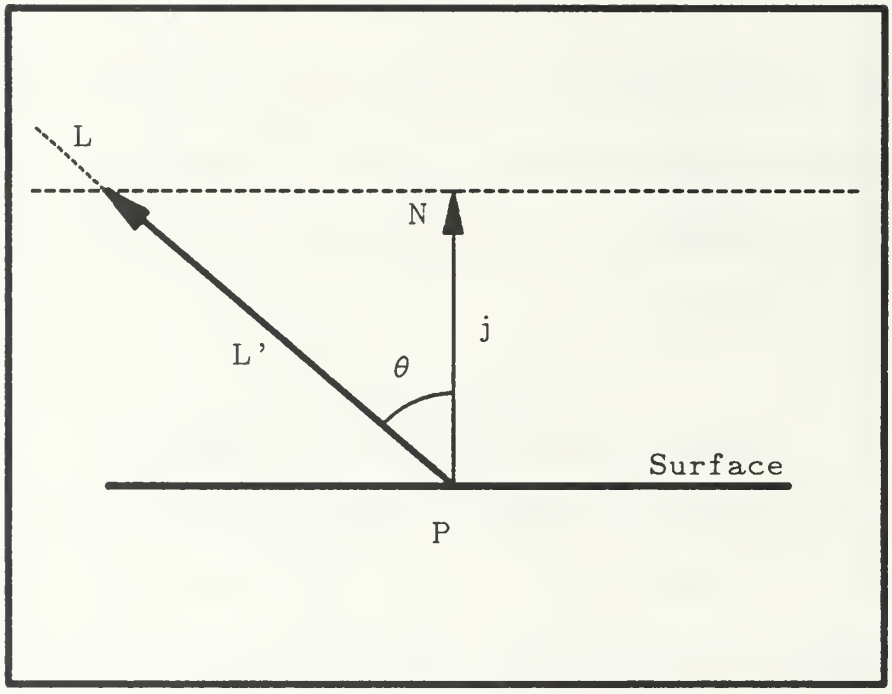


Figure 3.2 - Obtaining L'

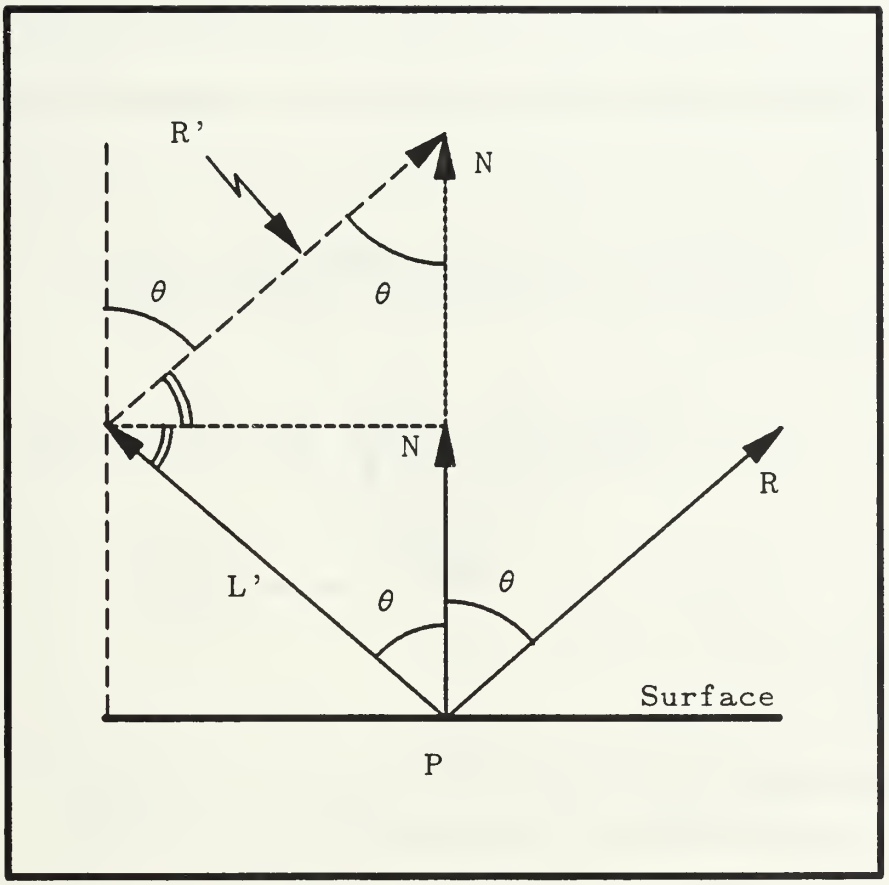


Figure 3.3 - Determining the Reflection Vector

incident light is reflected specularly. It varies from 1.0 for total specular reflection to 0.0 for no specular reflection.

B. A LOCAL MODEL FOR DISTRIBUTED LIGHT SOURCES

The above model for local reflection due to a point source can be easily expanded to model the local reflection due to distributed light sources. A point source illuminates a point with one light ray. As illustrated by Figure 3.4, a distributed light source illuminates a point with infinitely many rays. To model this, we sample light rays emanating from across the face of the light source. Our expanded model is

$$I = k_d I_a + \sum_{j=1}^{ls} \sum_{i=1}^{m} \left(\frac{I_{d_j}}{m(d_{i,j} + d_0)} \left[k_d(\vec{N} \cdot \vec{L}_{i,j}) + k_s(\vec{R}_{i,j} \cdot \vec{V})^n \right] \right)$$

where

- ls is the number of point light sources.
- j varies from 1 to ls.
- m is the number of light rays sampled.
- i varies from 1 to m.
- I_{d_j} is the intensity of the j^{th} distributed light source.
- $d_{i,j}$ is the distance of the point on the object from the point on the j^{th} distributed light source originating the i^{th} ray.

The distance, $d_{i,j}$, is considered with respect to the point on the j^{th} distributed light source originating the i^{th} ray. Since the light source is not far enough away to be considered a point source, the distance is computable. The sample rate, m.

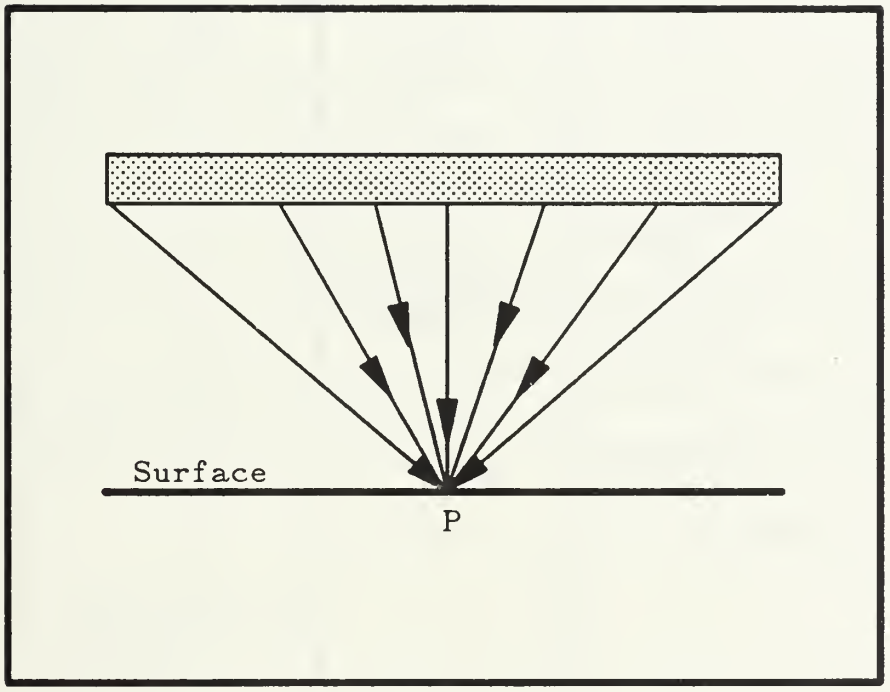


Figure 3.4 - Distributed Illumination

is a function of the distance of the illuminated point from the light source and the dimensions of the light source. The intensity of the distributed source must be divided by the sampling rate to avoid the intensity of the reflected light being greater than that of the light source itself. This model is approximately **m** times as expensive computationally as the local model for point sources.

The data required to compute the intensity of a point by this model is listed in Table 3.4. The data required under the **OBJECT** and **VIEW** columns is the same as that for point sources. The amount of data required under the **LIGHT** column is increased. In addition to the intensity and position of the light source. we must know the dimensions of the light source. The position of the source is taken to be the position of the center of the source. We also limit ourselves to rectangular or circular sources for ease of computing the locations on the surface generating the sampled rays. Therefore, the type of geometry, circular or rectangular, is given.

TABLE 3.4 - DATA REQUIRED BY DISTRIBUTED MODEL

OBJECT	VIEW	LIGHT
k_{d_r}	I_{a_r}	I_{d_r}
k_{d_g}	I_{a_g}	I_{d_g}
k_{d_b}	I_{a_b}	I_{d_b}
k_{s_r}	d_0	Position of Light
k_{s_g}	Position of Viewpoint	Dimensions of the Light Source
k_{s_b}		Geometry of the Light Source
$\vec{N}$		
n		
Position of Illuminated Point		

IV. SHADING MODELS

Color assignment computations are performed by a shading model. Such a model may or may not make use of an illumination model. The most simple case in which an illumination model is not used is when the color of the object is provided in the object data base and the shader accepts this value. The result is that the object is drawn with the predefined shade each time it is displayed. This results in constant shading no matter what the view or light positions happen to be. We are interested in shading models that utilize the local illumination models previously discussed. We therefore examine models due to Gouraud [Ref. 4] and Phong [Ref. 7: pp. 311-317].

A. GOURAUD SHADING

If a shading model utilizes the local illumination models previously discussed to calculate the intensity of each visible point in the scene, the resulting rendered objects appear faceted. This results from the sharp edges and angles between the polygons making up the graphics object. Gouraud [Ref. 4] developed a technique that results in smoother shading.

1. Gouraud's Method

Gouraud's technique utilizes a scan line algorithm to render an object. The first step is to determine the intensity of light reflected from each of the

vertices of the polygonal surface. This is done by utilizing the normal of a vertex when applying the illumination model instead of the surface normal. The vertex normal is the average of the surface normals for all polygons sharing that vertex. After the intensity at each vertex is determined, a bilinear interpolation is applied to the intensities at the vertices to determine the intensity of a point.

2. Determining the Vertex Normal

Figure 4.1 illustrates how the normal at vertex **A** is obtained. Point **T** is an inside point for the object comprised of three surfaces. The edge vectors containing vertex **A** as an endpoint are $\overline{AB}$, $\overline{AC}$, and $\overline{AD}$. The coordinates of the points are: **A** (a_1, a_2, a_3), **B** (b_1, b_2, b_3), **C** (c_1, c_2, c_3), **D** (d_1, d_2, d_3). The direction of the surface normal at vertex **A**, N_A, is determined by taking the average of the cross products of all the edge vectors which contain vertex **A** as an endpoint. Thus,

$$N_A = (\overline{AB} \times \overline{AD}) + (\overline{AD} \times \overline{AC}) + (\overline{AC} \times \overline{AB})$$

This is in counterclockwise order as viewed from the "outside" of the object, that is, the side away from the inside point, **T** . The vector equations for the edges are:

$$\overline{AB} = (b_1-a_1)i + (b_2-a_2)j + (b_3-a_3)k$$

$$= e_1i + e_2j + e_3k$$

$$\overline{AB} = (c_1-a_1)i + (c_2-a_2)j + (c_3-a_3)k$$

$$= f_1i + f_2j + f_3k$$

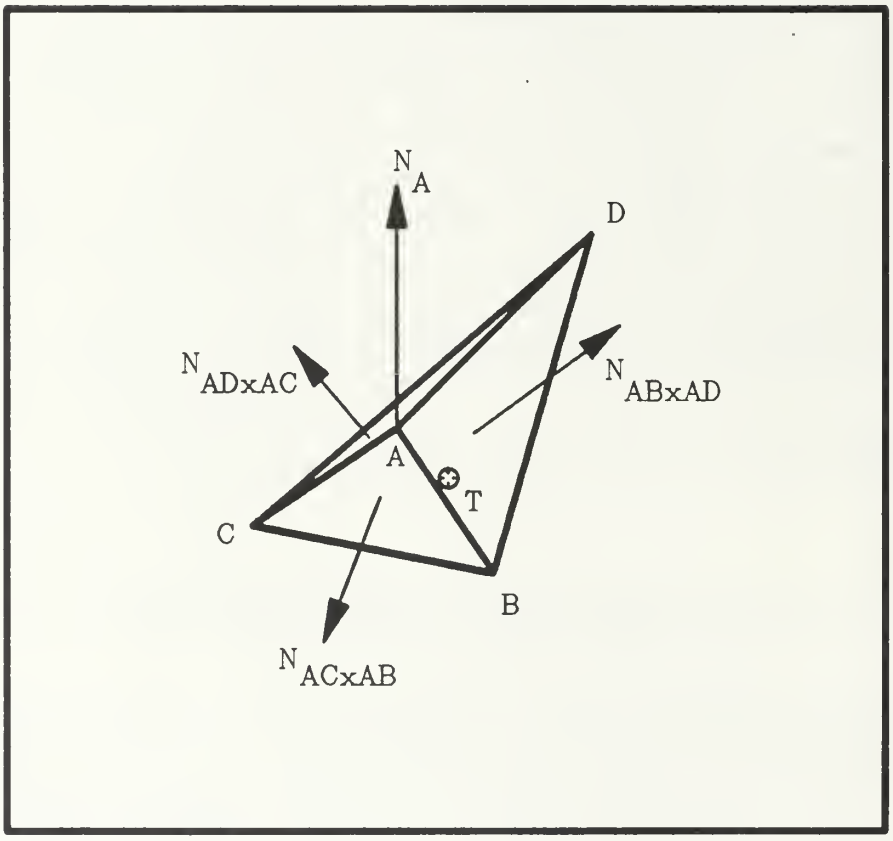


Figure 4.1 - Determining the Vertex Normal

$$\overline{AB} = (d_1-a_1)i + (d_2-a_2)j + (d_3-a_3)k$$

$$= g_1i + g_2j + g_3k$$

The cross products of the edge vectors with A as an endpoint are:

$$\overline{AB}\times\overline{AD} = [(e_2)(g_3)-(e_3)(g_2)]i + [(e_3)(g_1)-(e_1)(g_3)]j + [(e_1)(g_2)-(e_2)(g_1)]k$$

$$= m_1i + m_2j + m_3k$$

$$\overline{AD}\times\overline{AC} = [(g_2)(f_3)-(g_3)(f_2)]i + [(g_3)(f_1)-(g_1)(f_3)]j + [(g_1)(f_2)-(g_2)(f_1)]k$$

$$= n_1i + n_2j + n_3k$$

$$\overline{AC}\times\overline{AB} = [(f_2)(e_3)-(f_3)(e_2)]i + [(f_3)(e_1)-(f_1)(e_3)]j + [(f_1)(e_2)-(f_2)(e_1)]k$$

$$= p_1i + p_2j + p_3k$$

The average of the cross products gives the direction of the vertex normal at A.

$$N_A = (m_1 + n_1 + p_1)i + (m_2 + n_2 + p_2)i + (m_3 + n_3 + p_3)i$$

$$= t_1i + t_2j + t_3k$$

Dividing by the number of normals averaged does not yield the unit vertex normal. To obtain the unit vertex normal, the average normal must be divided by its magnitude. The unit vertex normal at A , $\hat{N}_A$, is then

$$\hat{N}_A = \frac{N_A}{|N_A|}$$

$$= \frac{t_1i + t_2j + t_3k}{\left(t_1^2 + t_2^2 + t_3^2\right)^{\frac{1}{2}}}$$

The vertex normal as derived above points in the correct direction because all

cross products were taken in a counterclockwise fashion as viewed from outside the graphics object.

3. Determining the Intensity at a Point

Figure 4.2 illustrates the steps to determine the intensity of a point on the surface of an object. To determine the intensity of point P, the illumination model is applied directly to each vertex of the polygon to determine its intensity. A scan line is then generated passing through P and intersecting two edges of the polygon, $\overline{KL}$ at point A and $\overline{MN}$ at point B . The intensity at A is then determined by interpolation.

$$I_A = \frac{|\overline{AL}|}{|\overline{KL}|} I_K + \frac{|\overline{KA}|}{|\overline{KL}|} I_L$$

where

- I_A is the intensity at point A.
- I_K is the intensity at vertex K .
- I_L is the intensity at vertex L .

The intensity at B, I_B, is similarly obtained.

$$I_B = \frac{|\overline{BN}|}{|\overline{MN}|} I_M + \frac{|\overline{MB}|}{|\overline{MN}|} I_N$$

The intensity at P , I_P , is then obtained by interpolation.

$$I_P = \frac{|\overline{PB}|}{|\overline{AB}|} I_A + \frac{|\overline{AP}|}{|\overline{AB}|} I_B$$

This calculation can be performed incrementally along the scan line.

$$I_{P_i} = I_{P_{i-1}} + (\Delta I \Delta D)$$

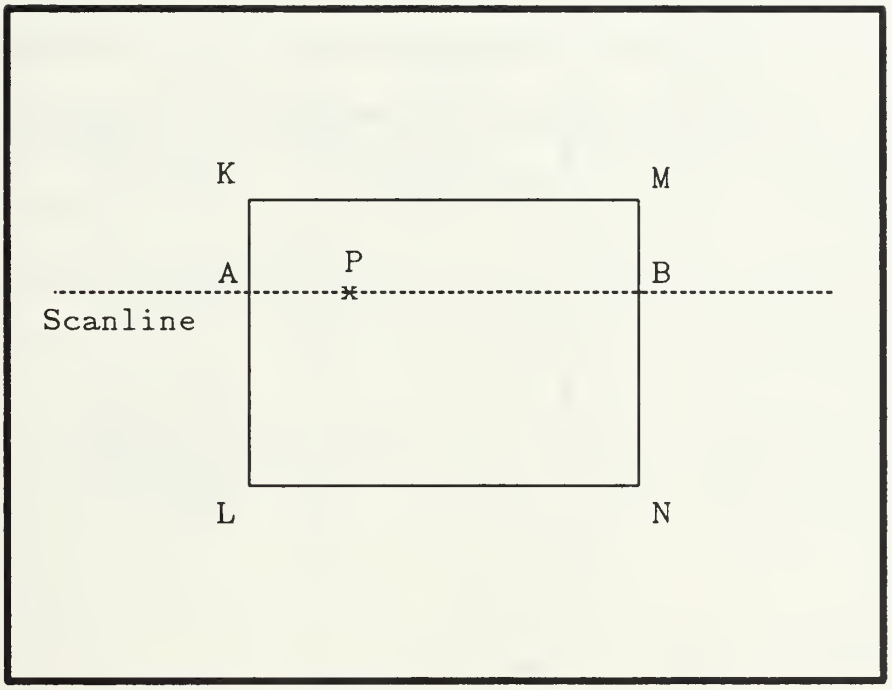


Figure 4.2 - Interpolation

where

- I_{P_i} is the intensity of the i^{th} point on the scan line.
- $I_{P_{i-1}}$ is the intensity of the previous point on the scan line.
- i is the index where $0 < i <$ number of pixels on the scan line.
- ΔI is the change in intensity between **A** and **B** per unit distance.
- ΔD is the distance between two adjacent points of the object on the scan line.

4. Performance

Gouraud shading produces smoother shading than constant shading. Yet there exist three problems [Refs. 6, 8: pp. 290-291, pp. 324-325]. One is the evidence of Mach banding due to the sharp edges between polygons. Another is that two adjacent polygons may be on different planes but still have vertex normals that are the same, resulting in the surface appearing flat. This can be overcome by assigning normals to the common vertices manually such that a vertex has one normal when associated with one polygon, and another normal when associated with another polygon. Another fix involves the addition of polygons near the crease to manually smooth the angle at the crease and thus ensure that the normals at the vertices are not the same. However, this can be time consuming and difficult. The third problem is that the shape of specular highlights are greatly influenced by the shape of the underlying polygons. Therefore, an illumination model consisting of the ambient and diffuse terms, or just the diffuse term, of the local models previously discussed is usually used for Gouraud shading. Table 4.1 summarizes the data necessary to

TABLE 4.1 - DATA REQUIRED BY GOURAUD SHADING

OBJECT	VIEW	LIGHT
Polygon Vertices	Viewing Parameters	
Position of Illuminated Point		
Position Scanline Intersects Polygon Edges		

perform the shading calculations without considering the data required by the illumination model.

B. PHONG SHADING

The problems noted above are mostly solved by a technique developed by Phong [Ref. 7]. Instead of interpolating intensity values across the scan line. Phong shading interpolates the surface normal vector for each point and then applies the illumination model. This method is more expensive computationally than Gouraud shading but produces more realistic images.

1. Phong's Method

The first step in determining the intensity of point **P** (Figure 4.2) is the calculation of the unit normal vectors for all vertices of the polygonal surface as described above. A scan line is then generated. The unit surface normals at the points where the scan line crosses edges of the polygon are determined. The scan line intersects $\overline{KL}$ at **A** and $\overline{MN}$ at **B** . The unit normal at **A**, N_A. is determined by interpolation.

$$N_A = \frac{|\overline{AL}|}{|\overline{KL}|} N_K + \frac{|\overline{KA}|}{|\overline{KL}|} N_L$$

where

- N_A is the unit surface normal at point **A**.
- N_K is the unit surface normal at vertex **K**.
- N_L is the unit surface normal at vertex **L**.

The unit surface normal at **B** , N_B , is similarly obtained.

$$N_B = \frac{|\overline{BN}|}{|\overline{MN}|} N_M + \frac{|\overline{MB}|}{|\overline{MN}|} N_N$$

The unit surface normal at **P** , N_P , is then obtained by interpolation.

$$N_P = \frac{|\overline{PB}|}{|\overline{AB}|} N_A + \frac{|\overline{AP}|}{|\overline{AB}|} N_B$$

This calculation can be performed incrementally along the scan line.

$$N_{P_i} = N_{P_{i-1}} + (\Delta N \Delta D)$$

where

- N_{P_i} is the unit surface normal of the i^{th} point on the scan line.
- $N_{P_{i-1}}$ is the unit surface normal of the previous point on the scan line.
- **i** is the index where $0 <$ **i** $<$ number of pixels on the scan line.
- **ΔN** is the change in the unit surface normals between **A** and **B** per unit distance.
- **ΔD** is the distance between two adjacent points of the object on the scan line.

The final step is to apply the illumination model for point sources, discussed in Chapter III, to the point **P** utilizing the unit surface normal at **P** .

2. Performance

Phong shading corrects or reduces the problems associated with Gouraud shading. However, it still has slight Mach banding which can be worse than that for Gouraud shading for certain cases, such as spheres [Ref. 8: p. 326]. When these cases are avoided, the resulting rendered image is more realistic in appearance than the same image rendered with Gouraud shading. The price is the increased computational expense due to applying the illumination model at each displayed point instead of just at the vertices. Phong shading can utilize the local

illumination models previously discussed with ambient, diffuse and specular terms. Table 4.2 summarizes the data required by Phong's shading model without considering the data required by the illumination model.

TABLE 4.2 - DATA REQUIRED BY PHONG SHADING

OBJECT	VIEW	LIGHT
Polygon Vertices	Viewing Parameters	
Position of Illuminated Point		
Position Scanline Intersects Polygon Edges		

V. RAY TRACING

A. INTRODUCTION

In the real world, a viewed object can be illuminated by rays which have been reflected or refracted by one or more surfaces as well as by direct rays from the light source. In Figure 5.1, point P is illuminated by three rays: one refracted by object A, one reflected by object B and one from the light source L. Ray tracing is a method which models this interaction between light and objects. As the name suggests, rays originating from some point are traced as they travel through the scene. When the ray encounters an object, the appropriate reflection and refraction are calculated and each of these is then traced. One technique originates rays at the light source while another technique originates rays from the view position. The first technique traces many rays which do not reach the view position. Therefore, the second technique is more efficient and is most often used.

The models previously discussed were local in that they modeled only the interaction of the light rays with the point illuminated. An illumination model proposed by Whitted [Ref. 9: pp. 343-349] models not only the local effects, but also the global interaction between light rays and all the objects within the scene. Whitted's model utilizes a ray tracing visible surface algorithm that provides the required global data. We examine the illumination model, the visible surface

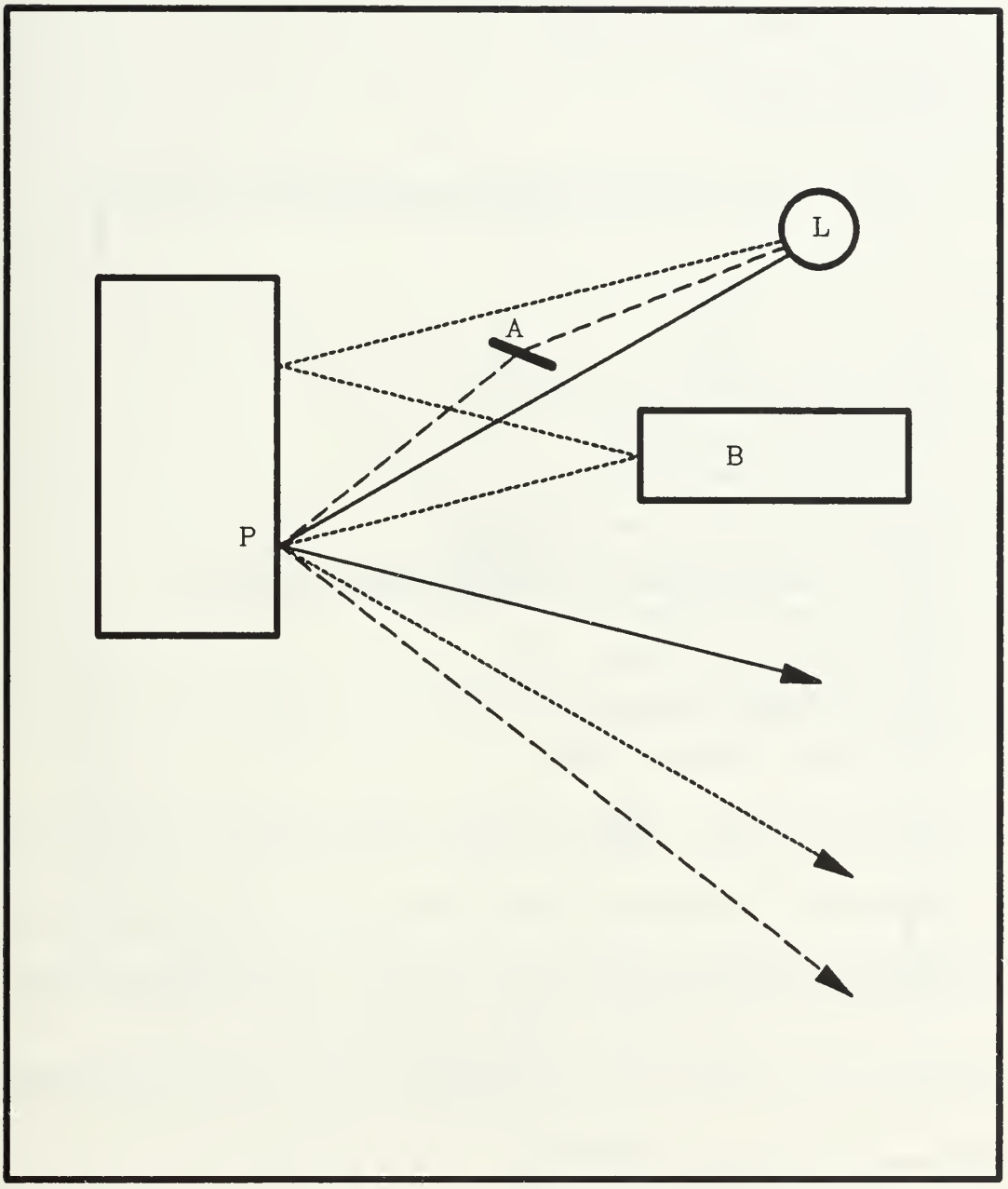


Figure 5.1 - Multiple Rays Illuminating an Object

algorithm, the shading model and the data required to calculate the color assignment for each pixel in the scene.

B. THE ILLUMINATION MODEL

Whitted's model is

$$I = I_a + k_d \sum_{j=1}^{ls} (\vec{N} \cdot \vec{L}_j) + k_s S + k_t T$$

where

- I is the total intensity at the point.
- I_a is the ambient intensity.
- k_d is the diffuse coefficient.
- $\vec{N}$ is the unit surface normal at the point.
- $\vec{L}_j$ is the unit light vector to the j^{th} light source.
- k_s is the specular coefficient.
- S is the global specular term.
- k_t is the tranmission coefficient.
- T is the global transmission term.

As pointed out previously, Whitted drops the diffuse coefficient for the ambient term. He retains the diffuse term utilized by Phong [Ref. 7]. He makes no mention of including the local effects of specular reflection. This can easily be added by including the local specular term $\sum_{j=1}^{ls} \frac{k_s I_j (\vec{R} \cdot \vec{V})}{d_j + d_0}$ in the model. We discuss the model though as presented by Whitted.

Figure 5.2 illustrates the specular and transmission terms. The intensity reflected back to the view position due to specular reflection is a percentage (k_s) of

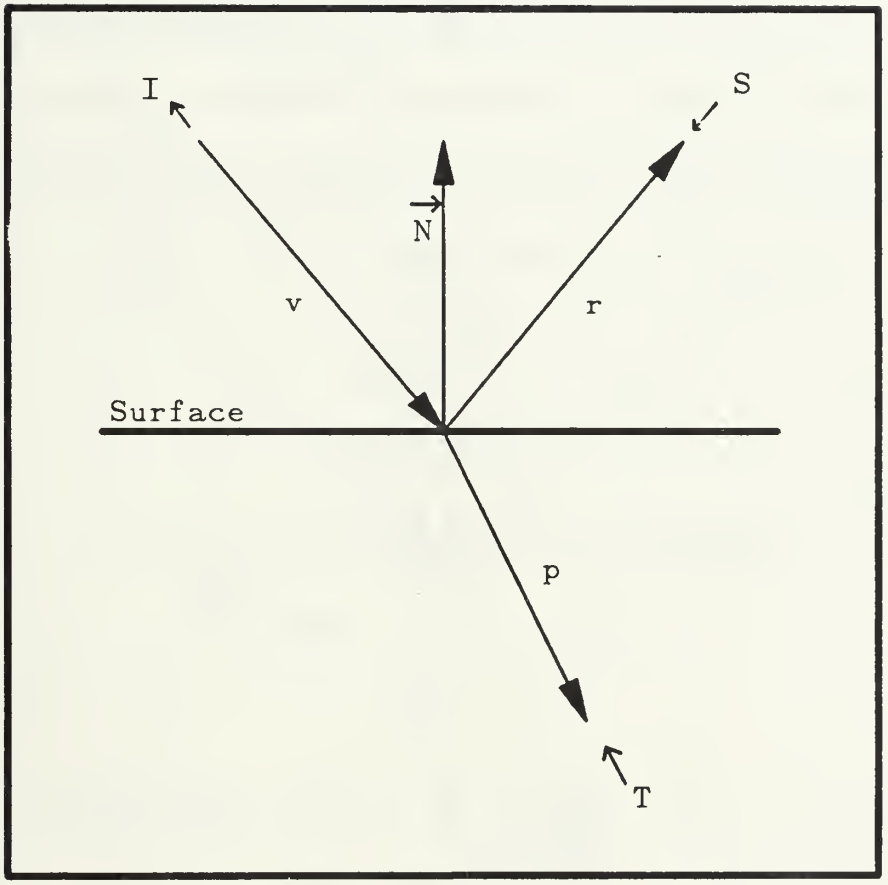


Figure 5.2 - Ray Tracing

the intensity of light, S, inbound along the $-\vec{r}$ direction and reflected back in the $-\vec{v}$ direction. $\vec{v}$ is the ray in the direction from the view position to the intersected point as distinguished from $\vec{V}$ which is the unit vector in the direction from the illuminated point to the view position. $\vec{r}$ is the reflection ray of $\vec{v}$ as distinguished from $\vec{R}$, the unit reflection vector of the incident light vector. The intensity passed in the $-\vec{v}$ direction due to transmission is a percentage (k_t) of the intensity of light, T, inbound along the $-\vec{p}$ direction and refracted back along $\vec{v}$ to the view position. $\vec{r}$ and $\vec{p}$ are functions of the unit normal vector, $\vec{N}$, and the unit light vector, $\vec{L}$ and given by (See Figure 5.3)

$$\vec{v}' = \frac{\vec{v}}{|\vec{v} \cdot \vec{L}|}$$

$$\vec{r} = \vec{v}' + 2\vec{N}$$

$$\vec{p} = k_f(\vec{N} + \vec{v}') - \vec{N}$$

$$k_f = \left((k_n|\vec{v}'|)^2 - |\vec{v}' + \vec{N}|^2\right)^{-\left(\frac{1}{2}\right)}$$

$$k_n = \frac{\eta_2}{\eta_1}$$

$\vec{v}'$ is a vector in the same direction as $\vec{v}$ but with a length equal to the distance from the the illuminated point and the intersection of a line passing through the head of the unit surface normal which is parallel to a line tangent to the illuminated point. η_1 and η_2 are the refraction coefficients for the mediums $\vec{v}$ and $\vec{p}$ pass through respectively. If the denominator of the Fresnel coefficient, k_f, is imaginary, total internal reflection occurs and the transmission coefficient is 0.

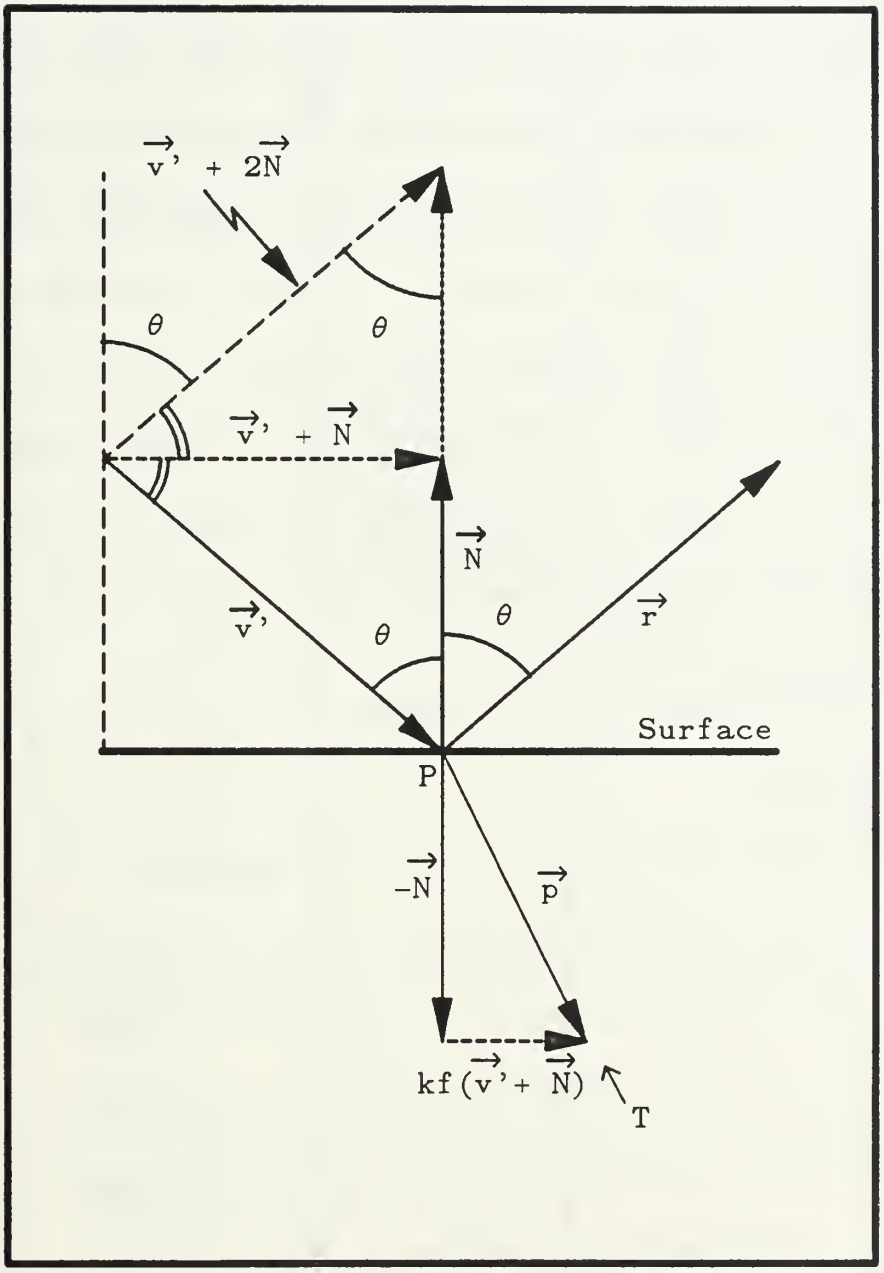


Figure 5.3 - Determining the r and p Directions

The specular component does not spread out the specular reflection exactly as does Phong's. One of two methods is applied. If S is due to light from a point source, Phong's specular term is used. However, the specular term cited by Whitted [Ref. 9: p. 344] is not the same as that cited by Phong [Ref. 7]. It is Phong's specular term only insofar as it utilizes Phong's specular exponent. If S is not due to a point source. a random perturbation is added to the surface normal to simulate the graininess, or roughness, of the surface. For glossy surfaces, the perturbation has a small variance. while for rougher surfaces the variance is increased. Whitted points out that too large a variance results in aliasing artifacts appearing in the rendered scene. "Too large" is not defined in Whitted's paper. The data required by Whitted's illumination model is listed in Table 5.1.

C. THE VISIBLE SURFACE PROCESSOR

1. Overview

The visible surface processor is a ray tracing algorithm. As such, it traces rays as they interact with the objects in the scene. Since the model requires only the rays which reach the viewer. rays are originated from the view position. A ray is traced from the view position through each pixel position on the projection plane towards the scene. This initial ray is $\bar{v}$. Intersection tests, described in a later section of this chapter, are performed to determine if the ray intersects any objects in the scene. If it does, the closest intersected object to the source of the ray is determined. From the intersected point two rays are generated in the $\bar{r}$ and

TABLE 5 - DATA REQUIRED BY RAY TRACING

OBJECT	VIEW	LIGHT
k_{d_r}	I_{a_r}	I_{d_r}
k_{d_g}	I_{a_g}	I_{d_g}
k_{d_b}	I_{a_b}	I_{d_b}
k_{s_r}	d_0	I_{p_r}
k_{s_g}	Viewing Parameters	I_{p_g}
k_{s_b}	Index of Refraction for Global Medium	I_{p_b}
k_{t_r}		Position of Light
k_{t_g}		
k_{t_b}		
$\vec{N}$		
n		
Position of Illuminated Point		
Index of Refraction		
Bounding Sphere Radius		

$\vec{p}$ directions. Theses rays are then traced in the same fashion as $\vec{v}$. The process continues with new rays being generated for the closest point of intersection of a ray with an object until no ray intersects an object, indicating that all rays have left the scene, or maximum allocated storage is reached. Figure 5.4 illustrates the full trace of an initial ray, $\vec{v}$. The resulting path is easily represented in tree form, as in Figure 5.5. The branches are the generated rays. The interior node contains the information from the definition of the intersected object required to calculate the local terms of the illumination model. The leaf nodes have an intensity of 0 since the branch either left the scene or maximum allocated storage was exceeded. After being built by the visible surface processor, the tree is traversed by the shader to calculate the intensity of the pixel $\vec{v}$ passed through.

2. Mechanics

$\vec{v}$ is determined from the xyz coordinates of the view point and the point on the projection plane which corresponds to the center of a display device pixel. Intersection tests are discussed in the next section. The object first intersected by a ray is determined by comparing the distances between the xyz coordinate of the point originating the ray and the xyz coordinate of all intersected points. The object whose intersection distance is least is the closest. Interior nodes contain, at a minimum, the xyz coordinates of the intersected point and a pointer to the intersected surface along with data required for the tree representation. The shader traverses the tree in either inorder or postorder traversal, calculating the intensity of interior nodes by applying the illumination model. The intensity of

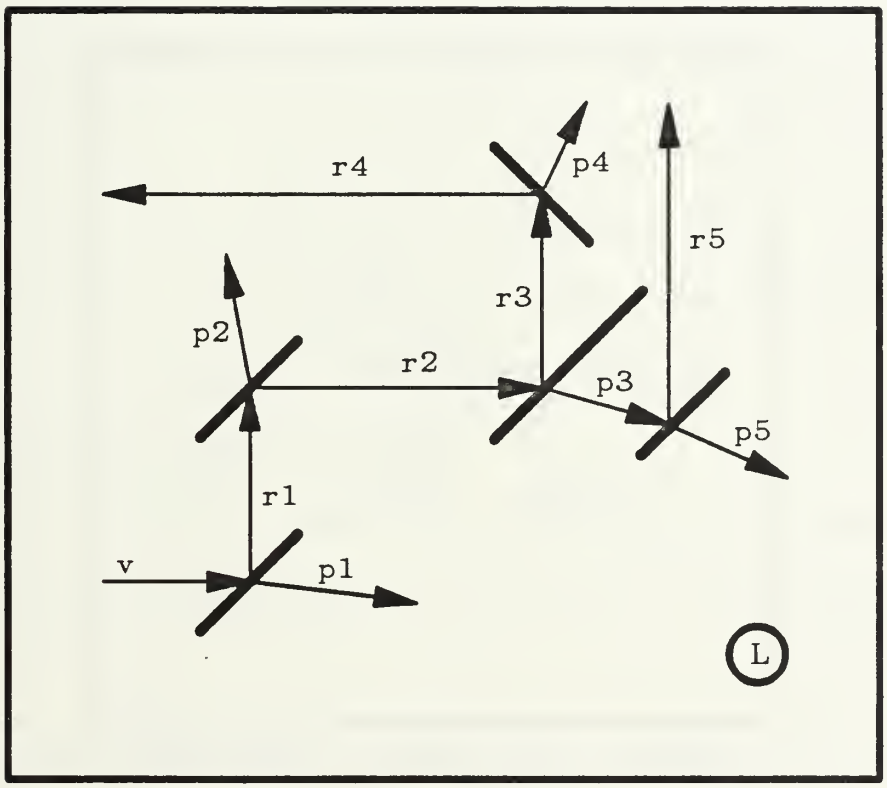


Figure 5.4

Obtaining Global Data Reflection and Transmission Rays

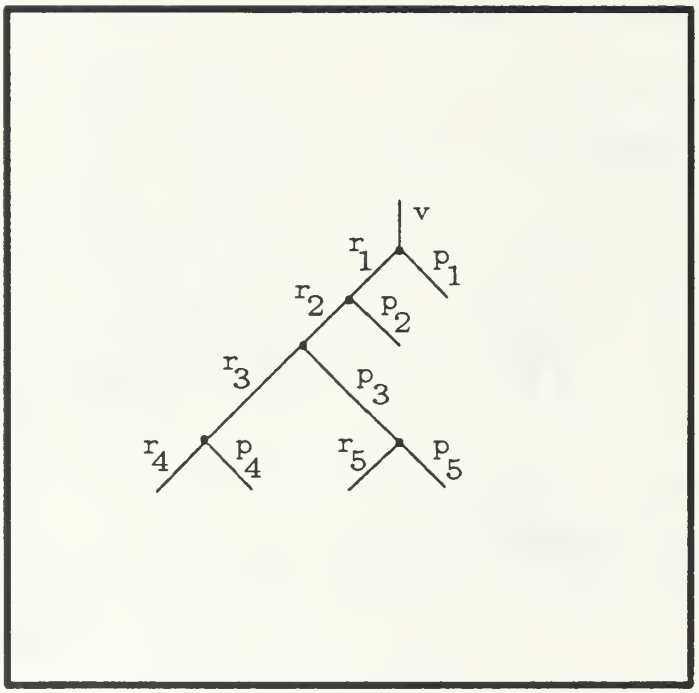


Figure 5.5 - Glabal Data Tree

all leaf nodes is 0. The global specular term, S, of the illumination model for an interior node is provided by that node's $\vec{r}$ branch while the global transmission term, T, is provided by the $\vec{p}$ branch. The root node's intensity is assigned to the pixel through which $\vec{v}$ passed.

At the closest intersection of a ray with an object, an additional ray is generated in the direction of each light source, as well as $\vec{r}$ and $\vec{p}$. These rays are termed shadow feelers and serve to determine if the intersected point lies in shadow with respect to a particular light source. In Figure 5.6. all light feelers except l_4 reach the light source L before intersecting an object. l_4 intersects an object first. Therefore, the contribution of the light source to the local terms of the illumination model for the originating point is attenuated.

3. Determining Object Intersection

Whitted points out that ray tracing algorithms typically spend 75 to 95 per cent of their time determining object intersection. In order to determine if a ray intersects an object, it must be determined if the ray intersects any of the polygons making up the object. To determine if a ray intersects a polygon, it must first be determined if the ray intersects the plane containing the polygon. If so, a bounding test is performed to see if the point of intersection lies withing the polygon boundaries. This is expensive since most graphics objects are made from polygonal approximations made up of many polygons.

The use of bounding volumes can decrease the number of tests required to determine if a ray intersects an object. The bounding volume used is the

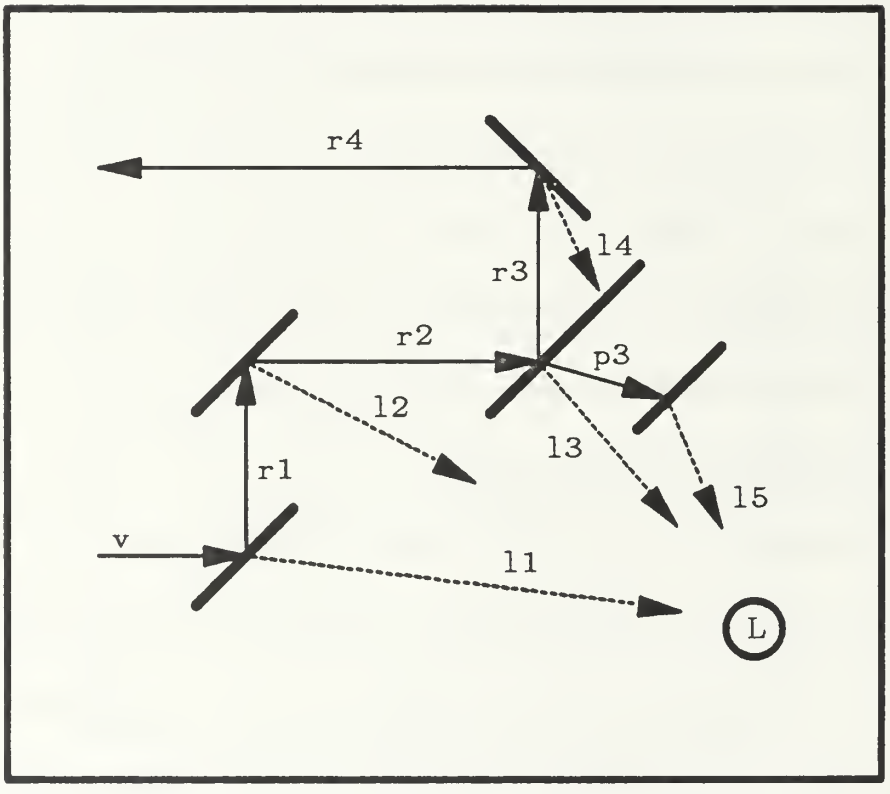


Figure 5.6 - Light Feelers

minimum size required to completely contain the object being bounded. When a ray does not intersect an object's bounding volume, no further intersection testing needs to be done for that object. For a bounding box, which is more efficient in terms of empty space between the bounding box and bounded object for some objects, an intersection test must be performed for at least three of its sides. For a bounding sphere, only one test needs to be made. The minimum distance between the ray and the center of the bounding sphere is determined. This distance is compared to the radius of the sphere. If the radius is less than the minimum distance, the ray does not intersect the object and no further tests need be done for the bounded object. Whitted has a discussion of intersection tests for objects consisting of bicubic patches, which we do not cover here [Ref. 9: p. 346].

D. ANTI-ALIASING

In order to decrease the effects of aliasing, Whitted defines a pixel as a square. Rays are projected out from each corner of the pixel square. If the resulting intensities are equal, or nearly so, their average is assigned to the pixel. If they are not nearly equal, the pixel square is subdivided and the process repeated for each resulting subsquare. This procedure is repeated until the computer runs out of resolution or an adequate amount of information about the pixel has been recovered.

Small objects can be missed if care is not taken. Each object must have a bounding sphere whose radius is above a certain minimum. The minimum is

determined based on pixel square size and object distance from the viewer. If a bounding volume is intersected, but not the bounded object, the pixel square is subdivided and the algorithm repeated until the object is intersected. In the case of rays being reflected from curved surfaces, the resolution of the of the computer can be reached before the object is found.

VI. DATA REQUIRED TO SUPPORT THE MODELS

Having presented some illumination and shading models, we review the data required to support them. The data is grouped into three categories: **OBJECT** (Table 6.1), **VIEW** (Table 6.2) and **LIGHT** (Table 6.3). Under each category, we review why each piece of data is required and how it is obtained.

A. OBJECT DATA

Data in this category is associated with each graphics object as a whole or with the individual polygons making up the graphics object. We examine each piece of data.

1. Polygon Vertices

A graphics object is a collection of polygons. Each polygon is defined by three or more vertices. Each vertex is defined by an xyz coordinate. Polygon vertices are provided in the object data base.

2. Coefficients

Each polygon has associated with it a specular, diffuse and transmission coefficient. The coefficients model the fact that only a percentage of the incident light is reflected diffusely, reflected specularly, or transmitted. The percentage for each is dependent on the wavelength of the incident light. Therefore,

TABLE 6.1 - OBJECT DATA

Polygon Vertices

Coefficients

- Diffuse $k_{d_{rgb}}$
- Specular $k_{s_{rgb}}$
- Transmission $k_{t_{rgb}}$

Unit Surface Normal - N

Unit Vertex Normals - N_v

Position of the Illuminated Point - P

Scan Line Intersections with Polygon

Specular Exponent - n

Inside Point - IP

Index of Refraction - η

Bounding Sphere Radius and Center

TABLE 6.2 - VIEW DATA

Ambient Intensity - $I_{a_{rgb}}$
Viewing Parameters
Index of Refraction for Global Medium
Constant to Prevent Division by Zero

TABLE 6.3 - LIGHT DATA

Intensity of the Light Source - $I_{ls_{rgb}}$
Light Source Type
Light Source Position
Light Source Geometry
Light Source Dimensions

each coefficient has a red, green and blue component. The coefficients are provided in the object data base and vary from 0.0 to 1.0 in value.

3. Unit Surface Normal

The unit surface normal is utilized by a shader that does not use intensity or vertex normal interpolation. Ray tracing may use it, depending on the implementation, as well as a shader applying a local illumination model at each point without interpolation. It consists of the **i**, **j**, and **k** coefficients for the vector. It is obtained by either direct calculation or from the object data base.

4. Unit Vertex Normal

The unit vertex normal is associated with each vertex of a polygon. It is utilized by both Gouraud and Phong shading, and may be utilized in ray tracing depending on the implementation. The vertex normals can be an average of the unit normals of the surrounding polygons, or they can be equal to the unit surface normal of the polygon itself, such as when a cube is being rendered. When a cube is rendered, the vertex normal must not be obtained by averaging if the sharp edges are to be retained. Therefore, the unit vertex normal is provided in the object data base. It consists of the **i**, **j**, and **k** coefficients for the vector equation.

5. Position of the Illuminated Point

The position of the illuminated point is expressed in terms of an xyz coordinate in the world coordinate system. The local shading models obtain the coordinates from the scan line algorithm implemented by the renderer. Ray tracing obtains the position of the point from intersection calculations performed

by the visible surface processor. The coordinates of the point are utilized in determining the unit light, surface normal, view and reflection vectors as well as color assignment by the shader.

6. Position Scan Line Crosses Polygon Edges

Both Phong and Gouraud shading utilize the points the scan line crosses the edges of a polygon in their calculations. Ray tracing may also make use of them if the implementation utilizes unit vertex normal interpolation to arrive at the unit surface normal at the point of ray intersection with a surface. The position is provided by the scan line algorithm implemented in the renderer or on the target system.

7. Phong's Specular Exponent

Phong's specular exponent models the rapid falloff in intensity of specularly reflected light as the view vector deviates from the reflection vector. Its value is dependent on the surface properties of the surface being modeled. A glossy surface has a value of 200 or greater while less shiny surfaces have a smaller value. It is obtained from the object data base.

8. Inside Point

The inside point of the graphics object is a point in the interior of the object. It is used in the determination of the outward facing surfaces of the polygons. Some graphics objects are actually made from two or more subobjects, such as an object made up of cylinders and spheres. Such graphics objects do not have an inside point common to the entire graphics object. Therefore, the inside

point of each subobject making up the graphics object must be known. It is an xyz coordinate in the world coordinate system and is provided in the object data base.

9. Index of Refraction

The index of refraction is utilized in calculating the global transmission term of Whitted's illumination model for ray tracing. It is provided in the object data base.

10. Bounding Sphere Center and Radius

The center and radius of the bounding sphere are utilized by the visible surface processor in conjunction with ray tracing in an attempt to reduce the number of intersection tests between rays and graphics objects. The center and radius must be such that the bounding sphere completely encloses the graphics object. Since a graphics object can be built from various subobjects, we require that a bounding sphere also be associated with each subobject with center being equal to the inside point of the subobject. If we adopt this strategy. the number of polygons of the graphics object requiring intersection test will be reduced to those of the subobjects whose bounding spheres are intersected. The center is already provided as the inside point of a subobject in the object data base. The radius is also provided in the object data base and is in world coordinate units.

B. VIEW DATA

The data defining the global context for the scene is termed view data. View data includes any data which is required by all the graphics objects in the scene. Table 6.2 lists the view data. We examine each item.

1. Viewing Parameters

In order to determine which points on an object correspond to pixels on the display device, the viewing parameters of the scene must be known. They consist of an xyz coordinate in the world coordinate system that is the view position, a projection plane that is usually expressed as a distance from the view position to the projection plane in world coordinate units, and a set of clipping planes. The specific requirements vary depending on the implementation. The viewing parameters are either determined by the implementation of the renderer, or are provided in the view data base.

2. Constant to Prevent Division by Zero

The constant to prevent division by zero, d_0, is utilized during calculation of the local diffuse and specular terms of the illumination models presented. It has a minimum value equal to the smallest positive non-zero value representable on the target computer system. It can also be utilized to fine tune the rendering of the scene, thus it may have values larger than the minimum. Therefore, it is provided in the view data base with a check being required in the renderer to ensure it is not less than the target system's minimum.

3. Refraction Index for the Global Medium

The refraction index of the global medium through which the initial view vector passes in support of ray tracing must be known. It is utilized in calculating the global transmission term of Whitted's illumination model for ray tracing. This must be provided in the view data base.

4. Ambient Light Intensity

Ambient light is the background light due to multiple reflections of light from the objects in the scene. It is broken down into its red, green and blue components and is expressed as a percentage of the maximum color intensity value for the target computer system. It varies in value from 0.0 to 1.0 for maximum displayable intensity. Since ambient light is global to the scene, it is provided in the view data base.

C. LIGHT DATA

The data associated with each light source illuminating the scene is termed light data. Table 6.3 lists the light data required by the illumination and shading models presented. We examine each item.

1. Light Source Intensity

The intensity of the red, green and blue components of the illuminating light source must be provided in the light data base. It is utilized in all illumination model intensity calculations other than those consisting of just the ambient term. The intensities of the components varies from 0.0 to 1.0.

2. Light Source Type

A light source can be a point source or a distributed source. The type is provided in the light data base.

3. Light Source Position

The position of a light source is used in determining the unit light vector and unit reflection vector. For a distributed light source, the position corresponds to the center of the light source. The position is provided in the light data base and consists of an xyz coordinate in the world coordinate system.

4. Light Source Geometry

The geometry of a distributed light source is used in determining the ray sampling rate and position on the light source originating each individual ray in support of the local illumination model for distributed light sources. For simplicity, we restrict the geometry to either rectangular or circular. The geometry of a distributed light source is provided in the light data base.

5. Light Source Dimensions

The dimensions of a distributed light source must be known in order to determine the sampling rate and to determine the position on the light source originating individual rays. The dimensions are length and width for rectangular sources and radius for circular sources. Dimensions are in world coordinate units and are provided in the light data base.

VII. IMPLEMENTATION DETAILS

We present the data structure to be utilized by the multi-illumination model renderer for representation of the external data required to render a scene. Our implementation is in the C programming language and makes use of dynamic arrays by use of the **calloc** command. This provides maximum flexibility while making efficient use of memory. We use the name of the pointer to the dynamic array to refer to the array.

A. PICTURE

The primary data structure is the record **PICTURE** (Figure 7.1). The number of objects, **NUM_OBJECTS**, can change from scene to scene. Therefore, a dynamic array **OBJECTS** is utilized to store the data for each object. Each element of **OBJECTS** is a record. **OBJECT_REC**. **NUM_OBJECTS** is stored to serve as an upper bound for indexing into **OBJECTS**. The number of lights, **NUM_LIGHTS**, also can vary between scenes. So a dynamic array, **LIGHTS**, indexed from **0** to **NUM_LIGHTS−1**, is utilized to store light source data. Each element of **LIGHTS** is a **LIGHT_REC**.

The view data, being global to the scene, is also contained in **PICTURE**. The refraction index of the global medium, **GLOBAL_REFRAC**, is stored as a floating point number. The constant to prevent division by zero, **NO_ZERO**, is also stored

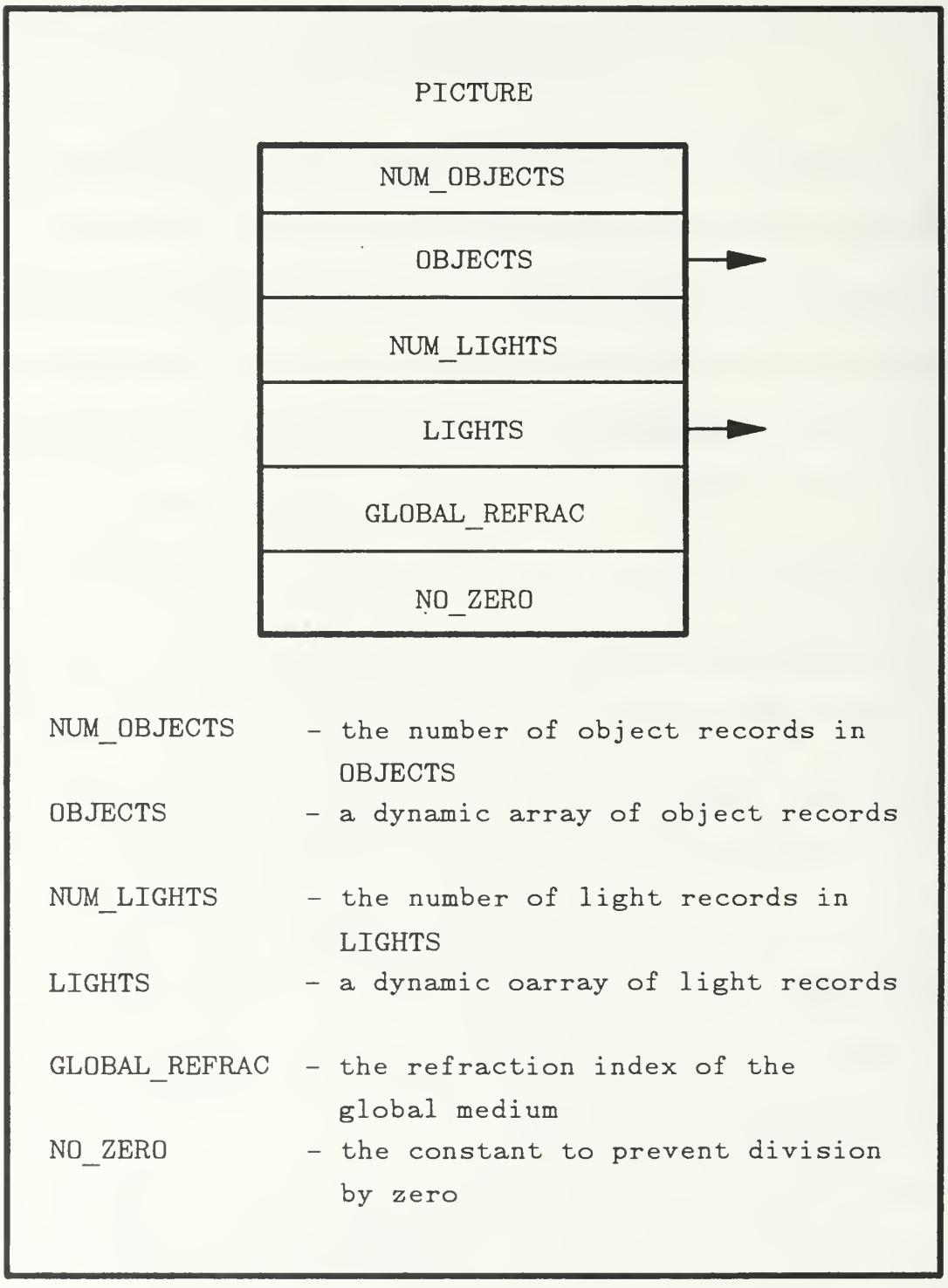


Figure 7.1 - PICTURE Data Structure

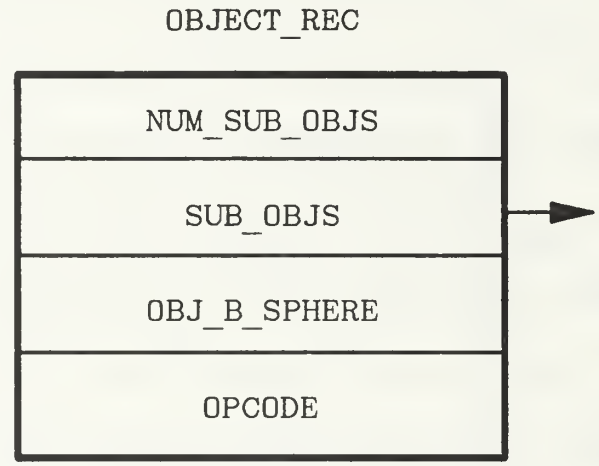
as a floating point number. The multi-illumination model renderer must perform a test when reading in this value from file to ensure that it is above the minimum value allowed for the target system. The last item of global data is the ambient intensity. It is stored in **AMBIENT** as a tuple of floating point numbers.

B. OBJECT_REC

Each **OBJECT_REC** contains the data required by the multi-illumination model renderer that is associated with each indivdual object. Figure 7.2 illustrates an **OBJECT_REC**. Each graphics object is made up of one or more subobjects. The dynamic array **SUB_OBJ** contains the data for each subobject. **NUM_SUB_OBJS** contains the number of subobjects in the object. Each element of **SUB_OBJ** is a **SUB_OBJ_REC**. Associated with each object is a bounding sphere, defined in **OBJ_B_SPHERE**, a record consisting of the xyz coordinate of the sphere's center and the sphere's radius. The data item **OP_CODE** is decoded by the multi-illumination model renderer to indicate certain information. Current usage consists of a digit indicating whether or not vertex normals are stored and a digit indicating whether or not surface normals are stored. In both cases a 0 indicates not stored while a 1 indicates stored.

C. SUB_OBJ_REC

Since intersection testing takes the lion's share of time in ray tracing, we include a bounding sphere, **SUB_B_SPHERE**, for each subobject. This reduces the number of polygons for which an intersection test must be performed if the



NUM_SUB_OBJS - the number of subobject records in SUB_OBJS

SUB_OBJS - a dynamic array of subobject records

OBJ_B_SPHERE - the bounding sphere for the object consisting of a center and a radius

OPCODE - indicates whether or not surface and vertex normals are stored

Figure 7.2 - OBJECT_REC Data Structure

object's bounding sphere is intersected. The center of the bounding sphere is also the interior point for the subobject.

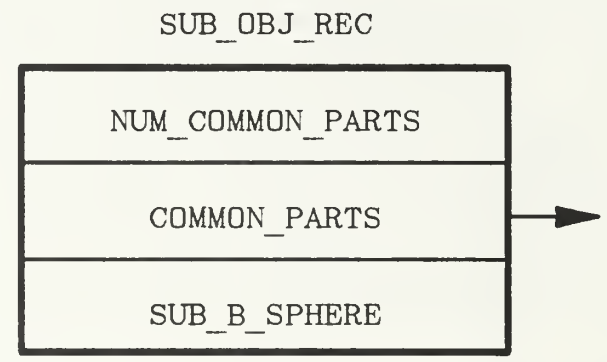
Each subobject is a collection of polygons. Associated with each polygon are attributes such as specular exponent and diffuse coefficient. In order to conserve memory and make the renderer more efficient, polygons sharing the same attributes are grouped together. The dynamic array **COMMON_PART** contains one or more groupings of polygons in the structure **COMM_PART_REC**. The number of such groupings is stored as **NUM_COMM_PARTS**. Figure 7.3 illustrates the structure **SUB_OBJ_REC**.

D. COMM_PART_REC

The polygons in each grouping share common attributes. The attributes are diffuse coefficient **kd**, specular coefficient **ks**, transmission coefficient **kt**, refraction index **REFRAC** and specular exponent **SPEC_EXP**. The coefficients are stored as tuples of floating point numbers representing the coefficients for the red, green and blue components of light. The polygons sharing these attributes are stored in the dynamic array **POLYGONS**. The number of polygons in the array is **NUM_POLYGONS**. Figure 7.4 illustrates the structure **COMM_PART_REC**.

E. POLYGON_REC

Figure 7.5 illustrates the structure **POLYGON_REC**. The vertices of the polygon are stored in the dynamic array **VERTICES**, each entry of which is an xyz coordinate. A dynamic array **VERT_NORMALS** holds the vertex normals if

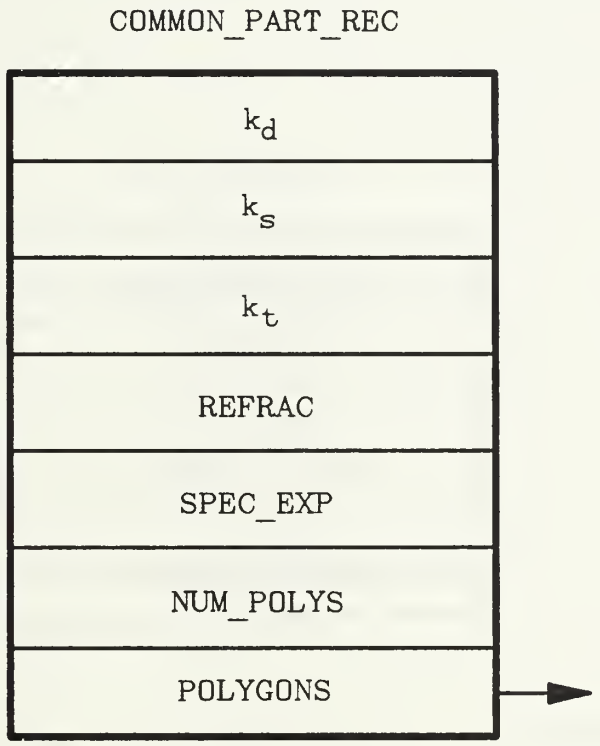


NUM_COMMON_PARTS - the number of common part records in COMMON_PARTS

COMMON_PARTS - a dynamic array of common part records

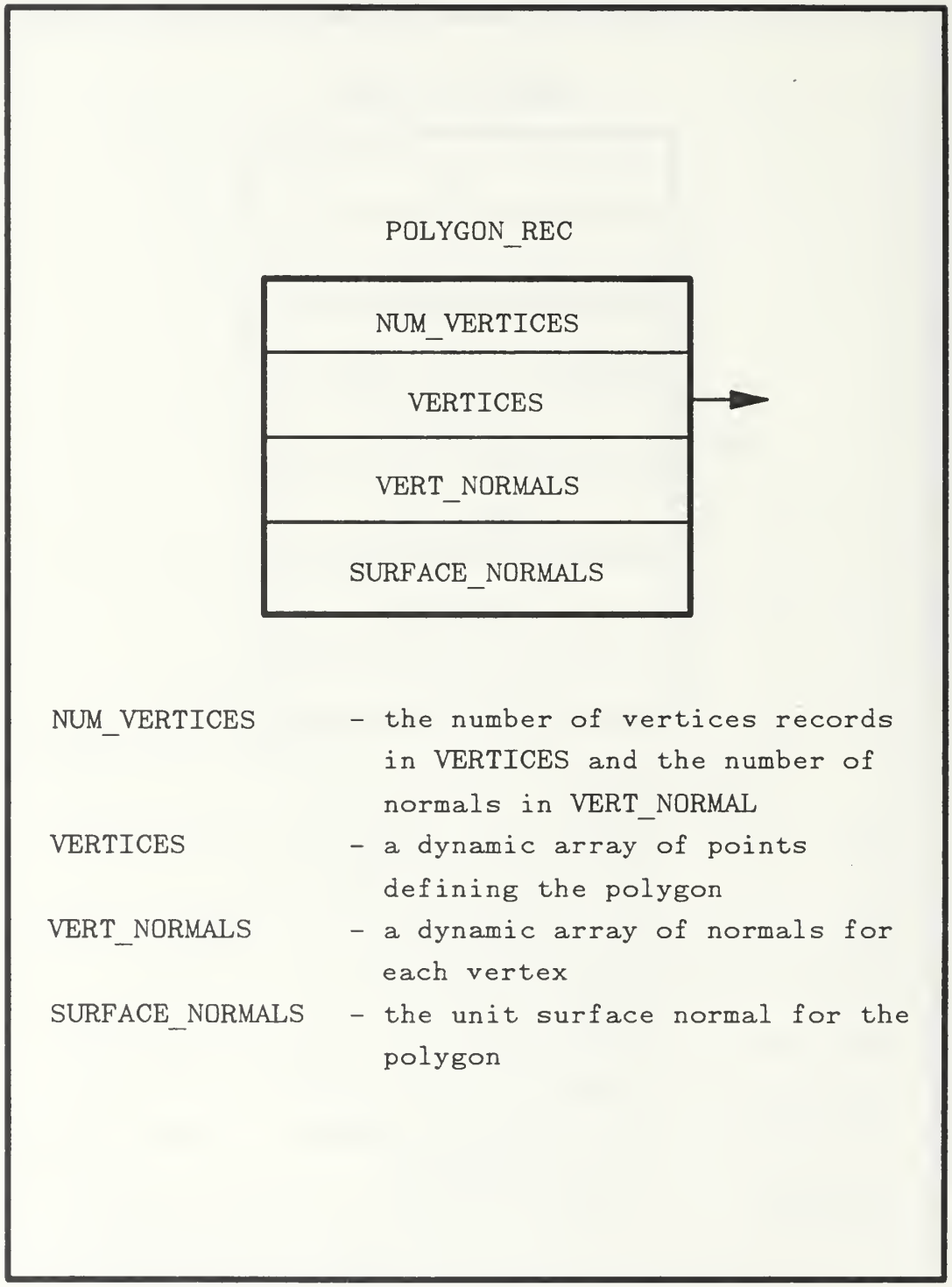
SUB_B_SPHERE - the bounding sphere for the subobject consisting of a center and radius

Figure 7.3 - SUB_OBJ_REC Data Structure



k_d	- the diffuse coefficient
k_s	- the specular coefficient
k_t	- the transmission coefficient
REFRAC	- the refraction index for the common attribut part
SPEC_EXP	- Phong's specular exponent for the common attribut part
NUM_POLYS	- the number of polygon records in the array POLYGONS
POLYGONS	- a dynamic array of polygon records

Figure 7.4 - COMMON_PART_REC Data Structure



NUM_VERTICES - the number of vertices records in VERTICES and the number of normals in VERT_NORMAL

VERTICES - a dynamic array of points defining the polygon

VERT_NORMALS - a dynamic array of normals for each vertex

SURFACE_NORMALS - the unit surface normal for the polygon

Figure 7.5 - POLYGON_REC Data Structure

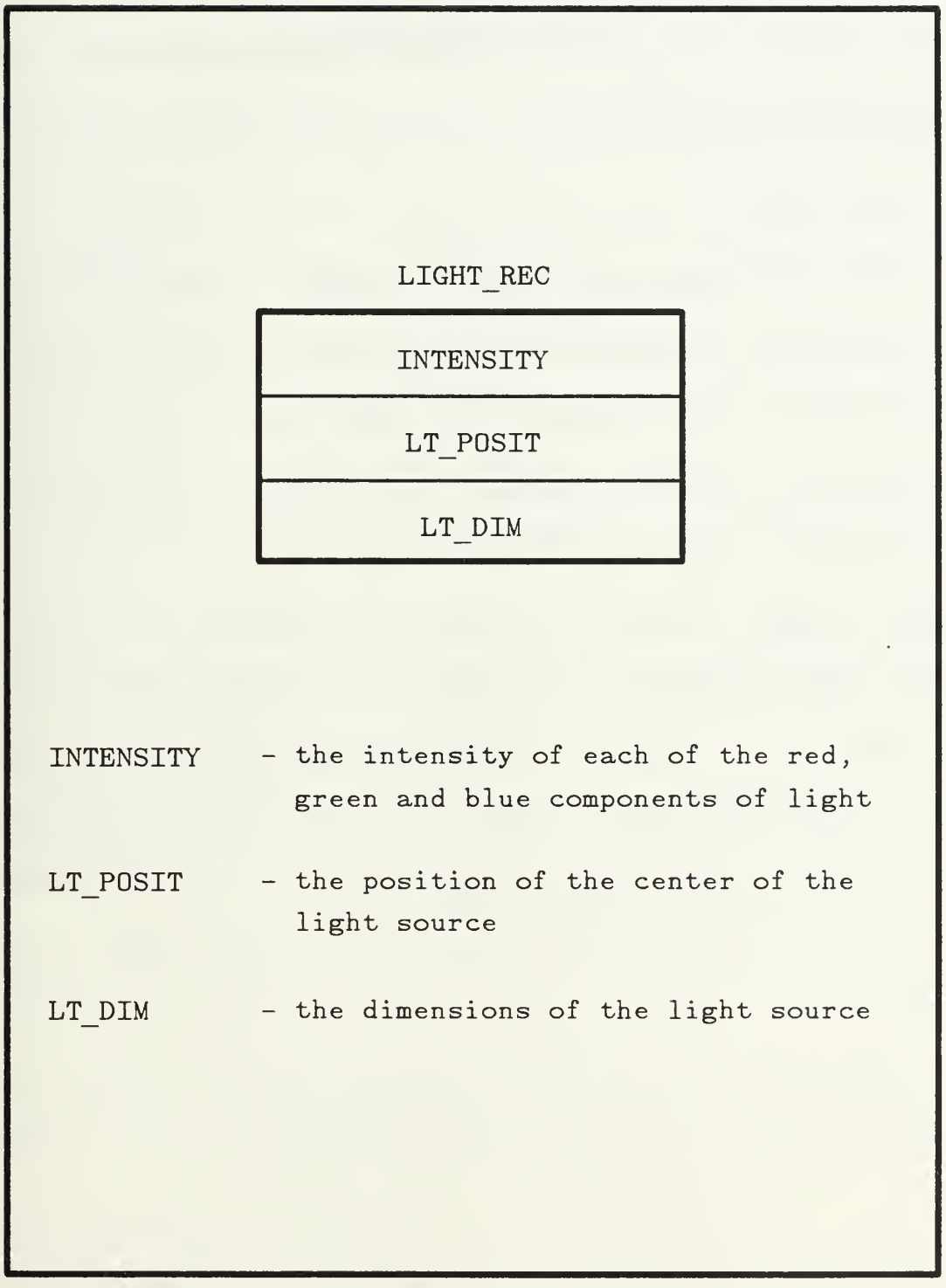


Figure 7.6 - LIGHT_REC Data Structure

they are stored. **NUM_VERTICES** gives the number of vertices and, if stored, vertex normals in their respective arrays. The surface normal is stored in **SURF_NORMAL**. All normals are represented as tuples consisting of the **i**, **j** and **k** coefficients for the vector equation.

F. LIGHT_REC

The data associated with a light source is stored in a **LIGHT_REC** (Figure 7.6). **INTENSITY** is a tuple of floating point numbers representing the intensity of the red, green and blue components of the light source. **LT_POSIT** is the xyz coordinate of the position of the center of the light source. **LT_DIM** stores the dimensions of the light source and is a record of two floating point numbers, **DIM1** and **DIM2**. If **DIM2** is zero, the source is a distributed source with radius given by **DIM1**. If **DIM1** is zero, the source is a point source. If **DIM1** is non-zero and **DIM2** is zero, the source is a circularly distributed light source with radius equal to **DIM1**. If both **DIM1** and **DIM2** are non-zero, the source is a rectangular light source with length of **DIM1** and heigth of **DIM2**.

VIII. LIMITATIONS

The data structure presented provides flexibility, good memory management and, in the case of ray tracing, a potential time savings due to bounding spheres being defined for all subobjects. Certain limitations, however, do exist. One limitation involves the inside point. One common inside point is used to determine which face is facing outward for all polygons in the subobject. Therefore, subobjects must not contain surfaces which are so concave that no point can be found which is inside with respect to all the polygons in the subobject. The handle of a teapot or the inside surface of a cylinder are examples of such subobjects. These subobjects must be further broken down until an inside point can be found for each part. An alternative to this procedure is to define an inside point for each polygon. The **POLYGON_REC** must then contain an **INSIDE_POINT** field and the center of the bounding sphere for the subobject would no longer be considered the inside point for all polygons in the subobject. This solution can be expensive in terms of memory space but is no more expensive than storing the surface normal for each polygon. Another solution which is potentially less expensive is to include the inside point as an additional attribute for the common parts of each subobject.

Another limitation is that the data structure requires all objects to be comprised of polygons. Objects created from revolutions, splines and patches must

be broken down into their polygonal representations. [Ref. 10] is an example of such a method for spline patches.

Defining a bounding sphere for each subobject can result in more intersection tests in some cases, such as objects with subobjects consisting of one polygon. The intersection test for the bounding sphere becomes redundant. One solution to this problem is to include another field in **OPCODE** which indicates whether or not the bounding spheres for the subobjects are to be utilized during intersection testing.

IX. AREAS OF FUTURE RESEARCH AND CONCLUSIONS

A. AREAS OF FUTURE RESEARCH

Having examined various illumination and shading models and presenting a data structure to be utilized by them, the next step is to examine the actual implementation of one or more of the models. The most promising candidate for initial implementaion is ray tracing for several reasons. The method requires no other algorithms for hidden surface elimination, inclusion of shadows and transparency, clipping or identification of the illuminated point and its position (usually via a scan line algorithm). The algorithm breaks down naturally into the computationally expensive building of the global tree for each view ray and the traversing of the tree to arrive at the intensity for the pixel. Rays can be generated by one processor and then handed off to other processors for building and traversing of the tree to arrive at the intensity for the root ray. This has potential for greatly decreasing the time necessary to render a scene.

Another area of research is the implementation of the non-ray tracing methods. The obtaining of the prerequisite data that can not be provided in the data base requires study. How much of this data can be obtained from the target system and how much of it must be obtained from software implementations needs to be determined. The portability issue also needs to be considered.

A skeletal modeling program has been implemented in support of the proposed MIM renderer. The modeler must be completed to provide the necessary data to a file for use by the renderer.

B. CONCLUSIONS

We have reviewed a number of illumination and shading models, presenting the methods, the data required by each model and how each piece of data is obtained. We then presented a data structure for representing the data that is not system dependent and that is utilized often during the process of performing the required calculations to render the scene. The data structure is general enough that it can be utilized by a multi-illumination model renderer that implements the models presented. It makes efficient use of memory, provides potentially greater efficiency for the ray tracing model. and is general enough to be utilized by all the models presented.

Appendix A provides the C code declarations and definitions for the data structure presented. Appendix B contains an extension of the data structure that is implemented on a skeletal modeling program in support of the MIM renderer. The major difference is the extensive use of pointers. Finally. Appendix C provides a sample file of data for a simple scene that is the output of the final modeling program and the input to the MIM renderer.

APPENDIX A – DATA STRUCTURE DEFINITION

```
/*****************************************************************
**      MAXCODES
**
**      maximum number of opcodes utilized by renderer
*****************************************************************/
#define             MAXCODES        2


/*****************************************************************
**      rgb
**
**      structure to hold the percentage of maximum intensity for
**      the red, green and blue components of light
**
**              red     - percentage of full red intensity
**              green   - percentage of full green intensity
**              blue    - percentage of full blue intensity
*****************************************************************/
typedef struct {
        float           red;
        float           green;
        float           blue;
} rgb;
```

```
/*****************************************************************************
**      point
**
**      structure to hold the xyz coordinates of a point
**
**              x_coord       - x coordinate of the point
**              y_coord       - y coordinate of the point
**              z_coord       - z coordinate of the point
*****************************************************************************/
typedef struct {
        float           x_coord;
        float           y_coord;
        float           z_coord;
} point;


/*****************************************************************************
**      dim_rec
**
**      structure to hold the dimensions of a light source
**
**              dim1  - first dimension
**                              - if 0, source is a point source
**                              - else, source is a distributed source
**              dim2  - second dimension
**                              - if not 0, source is rectangular
**                              - else, if dim1 not 0, source is circular
**                              - else, source is point source
*****************************************************************************/
typedef struct {
        float           dim1;
        float           dim2;
} dim_rec;
```

```
/*****************************************************************
**      b_sphere
**
**      structure to hold the definition of a bounding sphere
**
**              center          - xyz coordinates of the bounding sphere's center
**              radius          - the radius of the bounding sphere
*****************************************************************/
typedef struct {
        point           center;
        float           radius;
} b_sphere;


/*****************************************************************
**      norm_rec
**
**      structure to hold the coefficients for the vector equation
**      of a normal vector
**
**              i_coef          - coefficient for the i term
**              j_coef          - coefficient for the j term
**              k_coef          - coefficient for the k term
*****************************************************************/
typedef struct {
        float           i_coef;
        float           j_coef;
        float           k_coef;
} norm_rec;
```

```
/*****************************************************************************
**      polygon_rec
**
**      structure to hold data associated with a polygon
**
**              num_vertices        - the number of vertices defining the polygon
**              *vertices           - pointer to dynamic array of vertices
**              *vert_normals       - pointer to dynamic array of vertex normals
**              surface_normal      - the unit surface normal for the polygon
*****************************************************************************/
typedef struct {
        int             num_vertices;
        point           *vertices;
        norm_rec        *vert_normals;
        norm_rec        surface_normal;
} polygon_rec;


/*****************************************************************************
**      common_part_rec
**
**      structure to hold data associated with polygons which share the
**      common attributes: diffuse, specular and transmission coefficients
**                         refraction index
**                         specular exponent (Phong's)
**
**              kd              - diffuse coefficient
**              ks              - specular coefficient
**              kt              - transmission coefficient
**              refrac          - refraction index
**              spec_exp        - specular exponent
**              num_polys       - number of polygons sharing these attributes
**              *polygons       - dynamic array of polygons sharing attributes
*****************************************************************************/
typedef struct {
        rgb                     kd, ks, kt;
        float                   refrac;
        int                     spec_exp;
        int                     num_polys;
        polygon_rec             *polygons;
} common_part_rec;
```

```
/*****************************************************************
**      sub_obj_rec
**
**      structure to hold data associated with each subobject
**
**              num_common_parts        - number of parts sharing
**                                        common attributes
**              *common_parts           - dynamic array of common parts
**              sub_b_sphere            - bounding sphere for the subobject
*****************************************************************/
typedef struct {
        int                     num_common_parts;
        common_part_rec         *common_parts;
        b_sphere                sub_b_sphere;
} sub_obj_rec;


/*****************************************************************
**      object_rec
**
**      structure to hold data associated with each object
**
**              num_sub_objs    - number of subobjects in object
**              *sub_objs       - dynamic array of subobjects
**              obj_b_sphere    - bounding sphere for the object
**              opcode[ ]       - array of MAXCODES characters
**                                decoded by renderer
*****************************************************************/
typedef struct {
        int             num-sub_objs;
        sub_obj_rec     *sub_objs;
        b_sphere        obj_b_sphere;
        char            opcode[MAXCODES];
} object_rec;
```

```
/*****************************************************************************
**      light_rec
**
**      structure to hold data associated with light sources
**
**              intensity       - intensity of light source
**              lt_posit        - xyz position of light source
**              lt_dim          - dimensions of light source
*****************************************************************************/
typedef struct {
        rgb             intensity;
        point           lt_posit;
        dim_rec         lt_dim;
} light_rec;


/*****************************************************************************
**      picture
**
**      structure to hold all data associated with the scene to be rendered
**
**              num_objects         - number of objects in scene
**              *objects            - dynamic array of objects
**              num_lights          - number of lights illuminating scene
**              *lights             - dynamic array of lights
**              global_refrac       - refraction index of global medium
**              no_zero             - constant to prevent division by zero
**                                        - for Silicon Graphics IRIS 2400
**                                          minimum is approximately 1.8e-38
*****************************************************************************/
typedef struct {
        int             num_objects;
        object_rec      *objects;
        int             num_lights;
        light_rec       *lights;
        float           global_refrac;
        float           no_zero;
} picture;
```

```
/*****************************************************************
**      MAXCODES
**
**      maximum number of opcodes utilized by modeler
*****************************************************************/
#define             MAXCODES        2


/*****************************************************************
**      point
**
**      structure containing xyz coordinates of a point
**
**              x       - x coordinate
**              y       - y coordinate
**              z       - z coordinate
*****************************************************************/
typedef struct{
        float   x;
        float   y;
        float   z;
} point;
```

```
/*****************************************************************************
**      norm_rec
**
**      structure containing coefficients for vector equation
**
**              i_coef          - i coefficient
**              j_coef          - j coefficient
**              k_coef          - k coefficient
*****************************************************************************/
typedef struct {
        float   i_coef;
        float   j_coef;
        float   k_coef;
} norm_rec;


/*****************************************************************************
**      poly_rec
**
**      structure containing data defining a polygon
**
**              num_points          - number of points defining polygon
**              *poly_pts           - dynamic array of points defining polygon
**              vert_normals        - dynamic array of coefficients of unit
**                                    vertex normal for each point
**              *next_poly_rec      - pointer to next poly_rec
*****************************************************************************/
struct poly_rec {
        int                         num_points;
        point                       *poly_pts;
        norm_rec                    vert_normals;
        struct poly_rec             *next_poly_rec;
};
```

```
/*****************************************************************
**      comn_atr_rec
**
**      structure containing data for polygons sharing common attributes
**
**              kd              - diffuse coefficient
**              ks              - specular coefficient
**              kt              - transmission coefficient
**              refrac          - refraction coefficient
**              spec_exp        - specular exponent
**              c_index         - index into a colormap, an array of
**                                rgb color intensities
**              num_polys       - number of polygons
**              *first_poly     - pointer to first poly_rec in linked list
**              *last_poly      - pointer to last poly_rec in linked list
*****************************************************************/
typedef struct {
        rgb                     kd;
        rgb                     ks;
        rgb                     kt;
        float                   refrac;
        int                     spec_exp;
        int                     c_index;
        int                     num_polys;
        struct poly_rec         *first_poly;
        struct poly_rec         *last_poly;
} comn_atr_rec;
```

```
/*****************************************************************************
**      sub_obj_rec
**
**      structure to hold data associated with each subobject
**
**              num_comn_atr_parts      - number of parts sharing
**                                        common attributes
**              interior_point          - xyz coordinates of interior point
**              *comn_part              - dynamic array of common parts
**              *first_poly             - pointer to first polygon in subobject
**              *last_poly              - pointer to last polygon in subobject
*****************************************************************************/
typedef struct {
        int                     num_comn_atr_parts;
        point                   interior_point;
        comn_atr_rec            *comn_part;
        struct poly_rec         *first_poly;
        struct poly_rec         *last_poly;
} sub_obj_rec;


/*****************************************************************************
**      object_rec
**
**      structure holding data associated with an object
**
**              num_sub_objects         - number of subobjects in the object
**              *sub_obj                - dynamic array of subobjects
*****************************************************************************/
typedef struct {
        int             num_sub_objects;
        sub_obj_rec     *sub_obj;
} object_rec;
```

APPENDIX C – SAMPLE DATA FILE

```
* VIEW PARAMETERS GO HERE*
NO_ZERO
1.800000e-38
GLOBAL_REFRAC
1.000000
NUM_LIGHTS
2
INTENSITY
1.000000
0.500000
0.000000
LT_POSIT
-100.000000
200.000000
100.000000
LT_DIM
0.000000
0.000000
INTENSITY
1.000000
1.000000
1.000000
LT_POSIT
20.000000
20.000000
10.000000
LT_DIM
2.000000
8.000000
NUM_OBJECTS
2
OPCODE
11
B_SPHERE
2.000000
2.000000
0.000000
1.369306
```

```
NUM_SUB_OBJS
1
B_SPHERE
2.000000
2.000000
0.000000
1.369306
NUM_COMM_PARTS
6
kd
0.000000
0.000000
0.750000
ks
0.000000
0.000000
0.200000
kt
0.000000
0.000000
0.000000
REFRAC
0.000000
SPEC_EXP
50
NUM_POLYGONS
6
SURF_NORMAL
0.000000
0.000000
-1.000000
NUM_VERTICES
4
VERTEX 0
1.375000
2.625000
-0.625000
VERTEX 1
2.625000
2.625000
-0.625000
```

```
VERTEX 2
2.625000
1.375000
-0.625000
VERTEX 3
1.375000
1.375000
-0.625000
VERT_NORMAL 0
0.000000
0.000000
-1.000000
VERT_NORMAL 1
0.000000
0.000000
-1.000000
VERT_NORMAL 2
0.000000
0.000000
-1.000000
VERT_NORMAL 3
0.000000
0.000000
-1.000000
SURF_NORMAL
0.000000
0.000000
1.000000
NUM_VERTICES
4
VERTEX 0
2.625000
2.625000
0.625000
VERTEX 1
1.375000
2.625000
0.625000
VERTEX 2
1.375000
1.375000
0.625000
VERTEX 3
```

```
2.625000
1.375000
0.625000
VERT_NORMAL 0
0.000000
0.000000
1.000000
VERT_NORMAL 1
0.000000
0.000000
1.000000
VERT_NORMAL 2
0.000000
0.000000
1.000000
VERT_NORMAL 3
0.000000
0.000000
1.000000
SURF_NORMAL
0.000000
1.000000
0.000000
NUM_VERTICES
4
VERTEX 0
1.375000
2.625000
-0.625000
VERTEX 1
1.375000
2.625000
0.625000
VERTEX 2
2.625000
2.625000
0.625000
VERTEX 3
2.625000
2.625000
-0.625000
```

```
VERT_NORMAL 0
0.000000
1.000000
0.000000
VERT_NORMAL 1
0.000000
1.000000
0.000000
VERT_NORMAL 2
0.000000
1.000000
0.000000
VERT_NORMAL 3
0.000000
1.000000
0.000000
SURF_NORMAL
0.000000
-1.000000
0.000000
NUM_VERTICES
4
VERTEX 0
1.375000
1.375000
-0.625000
VERTEX 1
2.625000
1.375000
-0.625000
VERTEX 2
2.625000
1.375000
0.625000
VERTEX 3
1.375000
1.375000
0.625000
VERT_NORMAL 0
0.000000
-1.000000
0.000000
```

```
VERT_NORMAL 1
0.000000
-1.000000
0.000000
VERT_NORMAL 2
0.000000
-1.000000
0.000000
VERT_NORMAL 3
0.000000
-1.000000
0.000000
SURF_NORMAL
-1.000000
0.000000
0.000000
NUM_VERTICES
4
VERTEX 0
1.375000
1.375000
-0.625000
VERTEX 1
1.375000
1.375000
0.625000
VERTEX 2
1.375000
2.625000
0.625000
VERTEX 3
1.375000
2.625000
-0.625000
VERT_NORMAL 0
-1.000000
0.000000
0.000000
VERT_NORMAL 1
-1.000000
0.000000
0.000000
```

```
VERT_NORMAL 2
-1.000000
0.000000
0.000000
VERT_NORMAL 3
-1.000000
0.000000
0.000000
SURF_NORMAL
1.000000
0.000000
0.000000
NUM_VERTICES
4
VERTEX 0
2.625000
1.375000
-0.625000
VERTEX 1
2.625000
2.625000
-0.625000
VERTEX 2
2.625000
2.625000
0.625000
VERTEX 3
2.625000
1.375000
0.625000
VERT_NORMAL 0
1.000000
0.000000
0.000000
VERT_NORMAL 1
1.000000
0.000000
0.000000
VERT_NORMAL 2
1.000000
0.000000
0.000000
```

```
VERT_NORMAL 3
1.000000
0.000000
0.000000
OPCODE
11
B_SPHERE
1.000000
1.000000
2.000000
1.732051
NUM_SUB_OBJS
2
B_SPHERE
1.000000
1.000000
2.000000
1.732051
NUM_COMM_PARTS
1
kd
0.100000
0.200000
0.100000
ks
0.900000
0.700000
0.900000
kt
0.100000
0.000000
0.100000
REFRAC
0.050000
SPEC_EXP
200
NUM_POLYGONS
3
SURF_NORMAL
0.000000
0.000000
-1.000000
```

```
NUM_VERTICES
4
VERTEX 0
0.000000
2.000000
1.000000
VERTEX 1
2.000000
2.000000
1.000000
VERTEX 2
2.000000
0.000000
1.000000
VERTEX 3
0.000000
0.000000
1.000000
VERT_NORMAL 0
0.000000
0.000000
-1.000000
VERT_NORMAL 1
0.000000
0.000000
-1.000000
VERT_NORMAL 2
0.000000
0.000000
-1.000000
VERT_NORMAL 3
0.000000
0.000000
-1.000000
SURF_NORMAL
0.000000
0.000000
1.000000
NUM_VERTICES
4
```

```
VERTEX 0
2.000000
2.000000
3.000000
VERTEX 1
0.000000
2.000000
3.000000
VERTEX 2
0.000000
0.000000
3.000000
VERTEX 3
2.000000
0.000000
3.000000
VERT_NORMAL 0
0.000000
0.000000
1.000000
VERT_NORMAL 1
0.000000
0.000000
1.000000
VERT_NORMAL 2
0.000000
0.000000
1.000000
VERT_NORMAL 3
0.000000
0.000000
1.000000
SURF_NORMAL
0.000000
1.000000
0.000000
NUM_VERTICES
4
VERTEX 0
0.000000
2.000000
1.000000
```

```
VERTEX 1
0.000000
2.000000
3.000000
VERTEX 2
2.000000
2.000000
3.000000
VERTEX 3
2.000000
2.000000
1.000000
VERT_NORMAL 0
0.000000
1.000000
0.000000
VERT_NORMAL 1
0.000000
1.000000
0.000000
VERT_NORMAL 2
0.000000
1.000000
0.000000
VERT_NORMAL 3
0.000000
1.000000
0.000000
B_SPHERE
1.000000
0.000000
2.000000
1.414000
NUM_COMM_PARTS
1
kd
0.100000
0.200000
0.100000
ks
0.700000
0.700000
0.800000
```

```
kt
0.400000
0.400000
0.400000
REFRAC
0.050000
SPEC_EXP
200
NUM_POLYGONS
1
SURF_NORMAL
0.000000
-1.000000
0.000000
NUM_VERTICES
4
VERTEX 0
0.000000
0.000000
1.000000
VERTEX 1
2.000000
0.000000
1.000000
VERTEX 2
2.000000
0.000000
3.000000
VERTEX 3
0.000000
0.000000
3.000000
VERT_NORMAL 0
0.000000
-1.000000
0.000000
VERT_NORMAL 1
0.000000
-1.000000
0.000000
```

```
VERT_NORMAL 2
0.000000
-1.000000
0.000000
VERT_NORMAL 3
0.000000
-1.000000
0.000000
B_SPHERE
1.000000
1.000000
2.000000
1.732051
NUM_COMM_PARTS
2
kd
0.900000
0.100000
0.000000
ks
0.200000
0.700000
0.900000
kt
0.000000
0.000000
0.000000
REFRAC
0.000000
SPEC_EXP
100
NUM_POLYGONS
1
SURF_NORMAL
-1.000000
0.000000
0.000000
NUM_VERTICES
4
VERTEX 0
0.000000
0.000000
1.000000
```

```
VERTEX 1
0.000000
0.000000
3.000000
VERTEX 2
0.000000
2.000000
3.000000
VERTEX 3
0.000000
2.000000
1.000000
VERT_NORMAL 0
-1.000000
0.000000
0.000000
VERT_NORMAL 1
-1.000000
0.000000
0.000000
VERT_NORMAL 2
-1.000000
0.000000
0.000000
VERT_NORMAL 3
-1.000000
0.000000
0.000000
kd
0.000000
0.900000
0.000000
ks
0.000000
0.200000
0.000000
kt
0.000000
0.000000
0.000000
REFRAC
0.000000
```

```
SPEC_EXP
50
NUM_POLYGONS
1
SURF_NORMAL
1.000000
0.000000
0.000000
NUM_VERTICES
4
VERTEX 0
2.000000
0.000000
1.000000
VERTEX 1
2.000000
2.000000
1.000000
VERTEX 2
2.000000
2.000000
3.000000
VERTEX 3
2.000000
0.000000
3.000000
VERT_NORMAL 0
1.000000
0.000000
0.000000
VERT_NORMAL 1
1.000000
0.000000
0.000000
VERT_NORMAL 2
1.000000
0.000000
0.000000
VERT_NORMAL 3
1.000000
0.000000
0.000000
```

LIST OF REFERENCES

1. Cook, Robert L., "Stochastic Sampling in Computer Graphics," Special Interest Group on Computer Graphics of the Association for Computing Machinery (SIGGRAPH), in **State of the Art in Image Synthesis**, SIGGRAPH 86 Course No. 15 Notes, August 18, 1986.

2. Cook, Robert L., Porter, Thomas, and Carpenter, Loren, "Distributed Ray Tracing," **Computer Graphics**, v. 18, no. 3, pp. 137-145, July 1984.

3. Foley, James D. and Van Dam, Andries, **Fundamentals of Interactive Computer Graphics**, Addison-Wesley, 1982.

4. Gouraud, H., "Computer Display of Curved Surfaces," Doctoral Thesis, University of Utah, 1971.

5. Greenberg, Donald P., "Global Illumination, Ray Tracing and Radiosity," Special Interest Group on Computer Graphics of the Association for Computing Machinery (SIGGRAPH), in **State of the Art in Image Synthesis**, SIGGRAPH 86 Course No. 15 Notes, August 18, 1986.

6. Hearn, Donald and Baker, M. Pauline, **Computer Graphics**, Prentice-Hall, 1986.

7. Phong, B. T., "Illumination for Computer-Generated Images," **Communications of the ACM**, v. 18, no. 6, pp. 311-317, June 1975.

8. Rogers, David F., **Procedural Elements for Computer Graphics**, McGraw-Hill, 1985.

9. Whitted, Turner, "An Improved Illumination Model for Shaded Display," **Communications of the ACM**, v. 23, no. 6, pp. 343-349, June 1980.

10. Taylor, Gary W., **Parametric Representation and Polygonal Decomposition of Curved Surfaces**, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1986.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Computer Technology Curricular Officer, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Dr. Michael J. Zyda, Code 52Zk Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
6.	Dr. Thomas C. Wu, Code 52Wq Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
7.	Lieutenant Commander John S. Falby 119 Forest Street Hillsboro, Oregon 97123	1

Thesis
F2145 Falby
c.1 A data structure for a multi-illumination model renderer.