This is to certify that the

thesis entitled

A COMPARISON OF SHADING ALGORITHMS

FOR REAL-TIME RASTER GRAPHICS SYSTEMS

presented by

KATHLEEN A. CIESLAK

has been accepted towards fulfillment
of the requirements for

MASTER OF SCIENCE degree in ELECTRICAL ENGINEERING

David Fisher

Major professor

Date October 17, 1985

A COMPARISON OF SHADING ALGORITHMS
FOR REAL-TIME RASTER GRAPHICS SYSTEMS

By

Kathleen A. Cieslak

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical Engineering
and
Systems Science

1985

ABSTRACT

A COMPARISON OF SHADING ALGORITHMS
FOR REAL-TIME RASTER GRAPHICS SYSTEMS

By

Kathleen A. Cieslak

Shading is an important part of computer graphics. It transforms flat, confusing line drawings to solid images that appear three-dimensional. Shaded images of real-time raster graphics systems must conform to strict timing constraints or the images degrade.

Shading is enhanced by improving the shading algorithm as well as the object model and the method of processing the data during the hidden-surface removal algorithm. This report presents eight algorithms from the current literature. They are transformed into functional-block architecture specifications then compared primarily on the basis of speed of execution.

However, the fastest algorithm is not necessarily the most suitable. The quality of the images generated is an important criterion for judging the algorithms. If there are no limitations on hardware, the best solution is a combination of algorithms since some are more suitable than others for simulating certain effects, such as specular reflections and transparency.

To my Parents and Grandparents

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to Dr. P. D. Fisher for his constant guidance and encouragement, valuable suggestions, and interminable patience throughout this study.

Also, I wish to thank my committee members, Dr. M. Shanblatt and Dr. C. Wey, for their helpful suggestions concerning this research.

Above all, I especially wish to thank Richard Baids for his endless encouragement, patience and understanding throughout the course of this study.

This research was supported in part by Lear Siegler, Inc. / Instrument Division.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure		Page

Figure **Page**

CHAPTER I

INTRODUCTION

Since the introduction of the cathode-ray tube, (CRT), as a display device for computers, one goal has been to achieve realistic, dynamic images. As opposed to printers and other such display devices, CRTs allow real-time graphical interaction with the computer. Images are drawn on the screen before the user's eyes almost immediately after the commands are entered. Thus, no more waiting for printouts to see results.

The next logical step is to improve the images themselves. For real-time dynamic applications, the image must be updated and displayed on the screen at fast rates according to strict timing constraints to create the illusion of movement. Several graphics systems were developed evolving to raster graphics which seems the most promising. Raster graphics systems are capable of creating solid, colored, realistic, dynamic images.

Realism is achieved through various techniques, one of which is shading. Shading enhances realism by providing depth cues for a three-dimensional appearance. The effect is similar to a drawing of a circle, which appears as a flat disk until the artist shades it creating the illusion of a sphere.

Shading provides for surface properties adding to the realism of the image. If the object being displayed has a smooth surface it will reflect light differently than another with a rough surface. By utilizing properties of reflection from various surfaces according to different types of available lighting shading can convey textures of objects. Patterns may also be imposed onto objects through the use of shading.

However, shading has proven a formidable task for designers since it is dependent on many factors. The type of lighting present creates different properties of reflection. The location of the lighting may affect the reflections and form shadows from the object. And as mentioned, the object's surface properties also affect reflections of the light. In addition the manner in which the human visual system operates comes into play as does the location of the viewer.

This research presents and investigates several shading algorithms as well as a new reflective model and an object modeling technique, which can be applied to the algorithms to enhance realism. All of these are available in the current literature. This investigation compares the attributes of each algorithm and determines the time complexity using functional-block transformations.

Subsequent chapters explain these concepts in more detail. Chapter II provides background information by defining general graphics terminology and explaining how a raster graphics system operates. It presents some of the problems of high performance graphics systems and explains the problem of shading more explicitly.

Chapter III presents shading algorithms available in the current literature. These encompass two general methods for representing solid objects. For the polygon-mesh method, several implementations for shading are presented since this is a popular technique for modeling solids.

The fourth chapter compares the algorithms presented mainly on the basis of execution time. There are different criteria which may be used to evaluate shading algorithms, depending upon a system's requirements and applications. For example, realism or cost may be important considerations. But for three-dimensional, real-time graphics systems the most important criterion is execution time so that the image update performance is not degraded. Chapter IV presents a processor model to use as a standard for comparison. The algorithms are transformed into functional-block diagrams then are compared for speed of execution using various architectures to efficiently implement the diagrams.

By changing the algorithms to a functional-block representation, they are transformed from an behavioral representation to an architectural representation. This can be depicted in what are known as Y-charts [13]. Y-charts are three-dimensional characterizations of a transformational system. This three-dimensionality can be depicted as a 'Y'. Each axis is associated with a different representation of the system with different levels of the particular representation located along the length of its axis. One axis is for architectural representations meaning various hardware models of a system. The second axis is for behavioral representations which are different forms in which the behavior of an algorithm may be represented. The third axis is for physical representations or the different stages of the actual implementation.

In such a system transformations may occur along the same axis at the same or different levels, or between axes. Arcs are drawn on the Y-chart to show these transformations. In this investigation, transformations occur from the Algorithmic level of behavioral representations to the functional-block architectural representations enabling an accurate assessment of the time required for each algorithm's execution. The Y-chart for these transformations is shown in Figure 1.1.

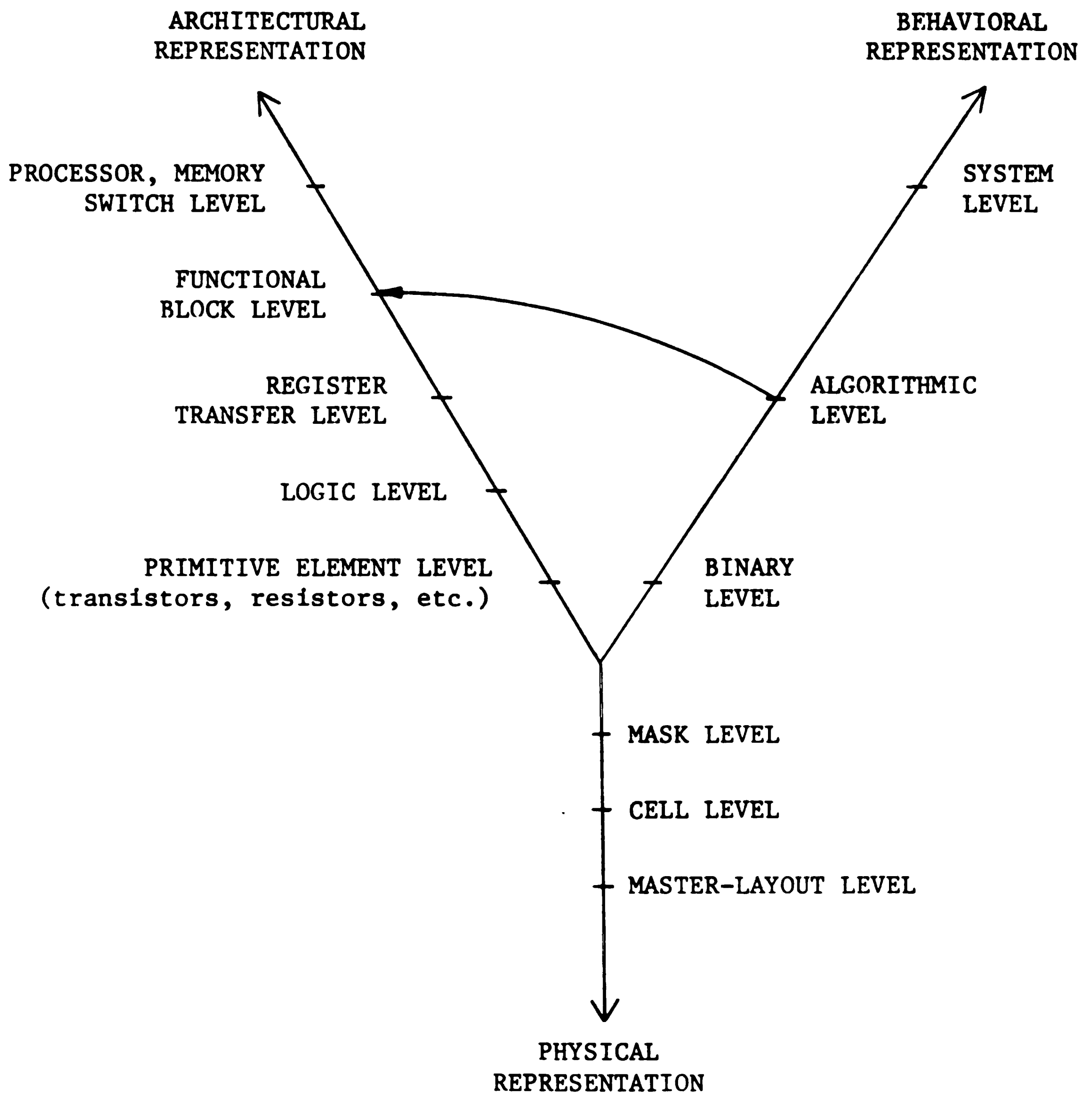

Figure 1.1: Y-chart depicting algorithm transformations

The fifth and final chapter summarizes the advantages and drawbacks as well as any conclusions derived from the comparison of the shading algorithms. Possible extensions of this work for future research are also included.

CHAPTER II

BACKGROUND

This chapter provides background information to aid in the understanding of the problem of shading computer-generated images. Some general graphics terminology is presented leading into the specific definitions associated with raster graphics systems. The second section of this chapter discusses some of the problems of graphics systems, specifically those of high-performance graphics. The chapter ends with a more detailed explanation of the shading problem.

2.1 Graphics Terminology and the Raster System

The field of computer graphics has been defined as the "creation, storage and manipulation of models of objects and their pictures via computer" [12]. It is concerned with the synthesis of pictures of objects, either real or imaginary. Computer graphics takes the three-dimensional objects and attempts to portray them on a two-dimensional viewing display. For this reason it encompasses many of the basic concepts and techniques of geometry and drafting in addition to phenomena of the real, three-dimensional world.

Essentially, all computer graphics systems are comprised of the same types of components. Each system has a host processor, a display controller and a display device. The host processor typically performs other functions as well as graphics processing. The display controller executes instructions to display the images and may have some capability to manipulate them. The distribution of the processing between these components and the implementation of the display procedure are dependent on the type of graphics system, defining the technologies it employs and its complexity. However, all systems perform two basic functions. The first is the construction and manipulation of the object's image, while the second is displaying that image.

This division of labor led to the concept of two coordinate systems. The first is the world coordinate system, or object coordinate system as it is sometimes called. This refers to the three-dimensional space, where the object to be modeled normally resides. The second is the device coordinate system, which is the planar space of the display. Here, models of the object are manipulated then projected onto the viewplaneof the screen. It is necessary to use the world coordinates during most manipulations of the object's orientation with respect to the viewer. The object is then transformed to device coordinates for displaying its image.

Simple geometric definitions of points and lines are applied during the modeling and manipulations of objects, including the notions of slope and intersections. However, even simple calculations of intersections may become time consuming when working with curved surfaces and solids. There are generally two methods for representing curved surfaces. The first is to divide the object into planar polygons and form a skeletal polygon-mesh. Two types of polygons arise during such a representation: convex and concave. Convex polygons are those satisfying the condition that for any two points contained by the polygon, all points on the line segment connecting them are also contained by the polygon. Concave polygons are those which are not convex. The polygons are easily represented by listing their vertex coordinates. Because the polygons are planar, second-order equations may be used to describe the surface, but this method tends to produce noticeable contour edges during solid modeling; more so than the next method.

The second method for representing curved objects is to define the coordinates of points on the surfaces by three-dimensional equations. Thus patches, which are not necessarily planar, are defined by three parameterized equations; one equation for each of the cartesean coordinate axes. These equations provide exact information at any point for the surface. Although this may be an advantage, the calculations can become extensive. Though the patches are typically larger than the polygons of the first method, meaning there are fewer patches to calculate, their equations are generally more complex and difficult to handle during manipulations of the object. Therefore, the class of surfaces that may be modeled is limited due to large execution times, but also because curved patches do not provide enough degrees of freedom to satisfy slope continuity between patches so they are not suitable to model arbitrary forms. Both polygon-mesh and curved-patch representations are often called wire-frame pictures or three-dimensional drawings because their appearance is like a collection of lines and arcs.

Once a model of the object has been generated a particular view must be bounded since the display screen has a limited area and/or a picture of the entire object may not be desirable. The rectangular space in the world coordinate system that frames this view is called a window. The window and its contents are then mapped to device coordinates. A viewport is a rectangular portion of the display screen in device coordinates where the window's contents are mapped and thus displayed. A viewport may include all or part of the screen. (Sometimes a viewport is also called a window by some systems, but this ambiguity can be confusing.) This mapping procedure must be repeated whenever the viewpoint of the object is changed.

To map the object enclosed by the window to the viewport, the object is projected to the plane of the screen. There are two classes of planar projections. The first is parallel projections which assume that the center of projection, (the point through which all projection rays pass), is infinitely distant from the projection plane, which in this case is the viewplane of the screen. Because the projection point is so far away, the projection rays appear parallel. Parallel lines are extended from each vertex of the object to the viewplane. The points of intersection of these lines with the viewplane are the projections of the object's vertices. These vertex projections are then connected according to lines corresponding to edges of the object. Different types of parallel projections exist depending on the number of faces and edges of the object parallel to the projection plane. In general, however, this class of projections is not very realistic because it lacks perspective foreshortening. Perspective foreshortening, performed naturally by the human visual system, is where the size of objects, or their projections, vary inversely with the distance from the center of projection. An advantage of parallel projections is that they scale measurements accurately; therefore, this method is usually used for drafting.

The second class is perspective projections which have the effect of perspective foreshortening. The center of projection is explicitly specified and is thus a finite distance from the object. Projection rays, which converge at this point and consequently are not parallel, are extended to the vertices of the object and on through to the viewplane. Again, the intersections of the lines with the viewplane define the projected image. For both classes of projections, angles are preserved only when the object's face containing the angle is parallel to the viewplane.

The above projections describe monographic images, where only one image is formed. The human visual system actually produces two images, one from each eye since the eyes are separated by a small distance. The brain fuses the two images into one forming a stereographic image. This provides a powerful depth cue known as stereopsis. For this to be effective each eye must see only one of the images formed. Some systems have been designed to take advantage of this effect [8,9].

It should also be noted that most computer displays are based on a left-handed, three-dimensional, cartesean coordinate system where the z-axis points into the screen. Using a three-dimensional projection format along with these reference axes, transformations can be performed on the images to give the impression of motion. Basically, there are two kinds of image dynamics. Update dynamics is when a change occurs in the shape, color or other physical properties of the image being viewed. Motion dynamics is when the object appears to move with respect to a stationary viewer; conversely, it could also be when the objects appear stationary and the observer appears to move, as is the case with flight simulators. This second type of dynamics is possible through transformations.

Transformations may change the actual view of the object to make other sides visible or just change the present view's orientation to the observer.

The basic types of transformations include translations, rotations and scaling. These may be implemented using matrix multiplications. They create effects such as panning, zooming and dragging. Panning uses translations to move the viewpoint parallel to the viewplane causing the entire scene to move as if the viewers turned their heads. Dragging, which also uses translations, moves one or more objects within the scene being viewed. This is often used in CAD systems while trying to find the optimal placement of objects in a scene. Zooming uses scaling to make the image larger or smaller giving the illusion of the viewer moving closer to or further from the image.

The method used to display the images on the screen is one of the main criteria used to catagorize graphics systems. There are three basic systems: vector-scan graphics also known as calligraphic-scan systems, direct-view storage tubes (DVST), and raster-scan graphics systems.

Vector-scan graphics systems were the first type of system to be developed. They are called random-stroke display systems. Pictures are drawn using plotting instructions specifying to draw a line from point A to point B. This is why they are called random-stroke systems; the starting and ending points for their instructions may be anywhere on the display screen. These instructions are stored in a display file where they are accessed by the display controller which executes them on the display device, a CRT. The inside of a CRT screen is coated with a phosphorous material which emits light when excited by an electron beam. The electron beam draws the image by moving across the surface of the screen, but the emitted light fades shortly after the beam moves away. Hence, the image must be retraced 30 times or more each second for the image to remain stable on the screen, requiring a display file to store the instructions so they can be re-executed during the refresh cycle. This refresh rate creates high bandwidth requirements of both the memory and the processor to meet the necessary timing. If the image is not refreshed soon enough it fades then again becomes bright after it is finally retraced. This noticeable flashing of the image is known as flicker and can be annoying to the system's users.

In order to be able to retrace the entire image within the allotted time, vector graphics' images had to be fairly simple. The use of a structured display file helped solve this problem. It is an extension of the regular display file because it stores hierarchies which compose the image. These hierarchies are like subroutines to a software program and can be executed faster. A structured display file allows for transformations by merely specifying parameters for the size, location and orientation of the image thereby improving dynamics. Vector systems use less memory than raster systems, but the deflection circuitry used for the electron beam is complex since it must accommodate random strokes.

The DVSTs operate on a random-stroke basis but the display screen is different. Inside the screen is a dielectric mesh which retains the image until it is deliberately erased. This eliminates the need for the display file which was a significant advantage before VLSI reduced the cost of memory chips. Unfortunately the image cannot be selectively erased; the entire image must be wiped out all at once. To erase the screen a charge is applied to the dielectric mesh which appears as a flash. This can be annoying to the user. However, it is possible to reduce the energy of the electron beam while drawing the image so that the image will not be volatile, but then a display file is needed to refresh it. The advantage of this is the capability of displaying a frame composed of both volatile and non-volatile images. This would decrease the demands of each refresh cycle to only retrace some of the image; the parts anticipated to move. This is similar to the technique used in animated cartoons. But neither of these systems can display solid, dynamic images. Raster-scan graphics systems were developed using television monitors giving them this capability as well as taking advantage of the established assembly lines of the monitors. The monitor screen is broken into an array of picture elements called pixels. The electron beam scans the pixels row by row from top to bottom in a fixed pattern known as a raster. This eliminates the need for the complicated deflection mechanism of the random-stroke display systems.

Actually, there are two types of rasters. The first is a non-interlaced pattern where the electron beam traces each horizontal line, called a raster line or scan line, in sequence from top to bottom of the display. The second is an interlaced display which alternates the lines dividing them into an even field, the even-numbered lines, and an odd field, the odd-numbered ones. To display a complete image or frame, both fields are required so two passes through the raster lines are necessary. In both rasters, as the electron beam moves along a scan line it is called active because it is displaying data. Retraces are when the beam reaches the right edge of the line and returns to the left edge of the next raster line, (horizontal retraces); or similarily, when the beam returns from the bottom right corner to the top left corner, (vertical retraces). During these retraces, (not to be confused with redrawing an image though the terms are identical), the electron beam is blanked or turned off.

A variation of either raster is called horizontal underscanning. By decreasing the horizontal deflection of the electron beam the raster pattern is altered to change the shape of the pixels, usually making them square. Whelan [34] presents what he calls a subclass of raster-scan systems. The pixels of his display are rectangular areas which are larger than typical pixels enabling his system to be faster than general raster systems while still retaining all of their attributes, though he loses some spatial resolution of the display.

The time required to completely scan the raster is called the refresh rate. The reciprocal of the refresh rate is the frame time which measures the duration of each frame before the next refresh. These two rates help determine the level of interactivity of a raster display, or how fast a new image can be generated in response to a user's input. This is usually the most critical performance measurement of a raster graphics system. Typically, raster systems are slower than random-stroke systems because they indirectly draw lines and arcs by scanning the entire screen activating only the pixels composing the lines and arcs, whereas random-stroke systems only pass over the points of the lines and arcs. But, with faster processors and shorter memory access times, raster systems are approaching the speed of the random-stroke systems.

One disadvantage of raster systems is that they require large memories. The number of distinguishable pixels in the raster determines the display's resolution. Some of the highest resolutions may be larger than an array of 4096 x 4096. The image memory; also known as the frame buffer, display or bit-map memory; has one or more bits of memory that correspond to the pixels of the display. A bit plane is an array of memory of one bit per pixel. The pixel depth of a system is its number of bit planes. This determines the number of gray-scale intensities available for shading on an achromatic display, (a monochromatic display has one bit plane and thus displays only black and white), or the number of colors available for color displays, called color resolution. An image plane is a set of bit planes, usually eight. For full color, three image planes are needed; one each for red, green and blue which are the additive primary colors necessary to combine and produce most other colors. (They are called additive primaries because individual contributions of each are added to form other colors.)

Consequently, the number of bits in the image memory must be one or more times the number of pixels of the display screen. Because of such large memories raster systems need an extremely high bandwidth for both the processor and memory to satisfy image update and display timing constraints. But this pixel-by-pixel addressing capability is the key which allows different kinds of images on a single screen; such as alphanumeric characters combined with photographic images.

Baldauf [1] made an important observation concerning bit-mapped memories and raster-scan displays; namely, to increase the quality of the display does not necessitate increasing the number of pixels of the display. Using a system with four megabits of memory, Baldauf compares the configuration of four million pixels with one bit of memory per pixel to that of one million pixels with four bits per pixel thus providing sixteen gray scales. The quality of both is very similar and the cost advantages favor the latter.

Concerning display devices in general, achromatic monitors are superior to color monitors in terms of brightness, resolution and size. Color displays are more complex and impose significantly higher bandwidth requirements on the memory and processor. Therefore, color monitors cannot easily match the quality of their achromatic counterparts. However, color adds another dimension to the information displayed. It also adds considerably to the price of the system. In 1979 they were considered too costly for widespread use [15]. But in 1984 according to Machover and Meyers [22], sales of color raster displays were growing at a rate of two to three times faster than the rate of growth of DVSTs and vector-scan displays, meaning color displays are gaining popularity.

A variation of achromatic displays which is also gaining popularity, particularly among users who enter data from printed papers, is the positive image display. Positive images are black characters on a white background. This is a bit different to implement because instead of turning pixels on to show an image, the background is always on and the pixels making up the image are turned off. But this is less tedious for the user since the screen has the same sort of contrast as printed material.

The hardware required by a raster system consists of the components of a basic graphics system mentioned earlier: a host processor, display controller, and display device. The display device's operation is as already discussed. Sometimes, in smaller systems, the host processor is called the display processor, graphics processor, video display processor or display generator. Often, there is still another processor which acts as a host performing other functions but which has the capability to manipulate the image memory, too. The display processor manipulates the data in the image memory by writing new data to update the image for the next frame. It also has the ability to read the memory to determine previous pixel values so as to establish new values. Display processors once were merely buffers, but graphics functions were eventually moved there to relieve the host processor's burden so it could perform other duties.

The raster system equivalent of the display controller, the final component of a basic graphics system, is called the video refresh controller, refresh controller, display controller or frame buffer controller. It reads from the image memory and feeds data to the output portion of the system. It does not have the ability to alter pixel data; in other words it can only read from the image memory, not write to it as well. The display controller's main function is to obtain the pixel data in parallel form to utilize higher bandwidths then convert it to a serial bit stream to store in the video buffer, frame buffer, or refresh buffer. The video output hardware converts the data to intensity levels or colors helping the display controller create the serial data. The video buffer stores the pixels' intensities or colors for the display.

In summary, there are vector-scan displays, which are random-stroke, refresh systems; raster-scan displays, which are non-random-stroke, refresh systems; and direct view storage tubes, which are random-stroke, refresh and/or nonvolatile-image systems. Of these three, raster graphics is the only one able to display solid, dynamic images but it is not entirely free of problems. These problems and some of general graphics systems are presented in the next section.

2.2 Problems of High-performance Graphics

This section begins with some general problems common to all graphics systems. One is that CRTs are analog devices and computers and processors are digital devices. A digital-to-analog converter, (DAC), is essential to graphically display information. Since the DAC converts the digital signals from the display controller to analog voltages meaningful to the CRT, the DAC's resolution directly determines either the number of gray scales or colors resolvable in the monitor. A color monitor needs three DACs, one each for red, blue and green. Castleberry [4,5] discusses this problem and presents a video DAC that meets the high-performance requirements of a graphics system, yet is low cost.

Another problem, which may be apparent from the redundant terminology, is a need for standardization. Each unique graphics configuration developed requires a new software system to support it. According to Machover and Myers [22], "the principal idea underlying the quest for standards was that the main body of software should be device independent. It should interface to any input device through a device handler, which would, of course, be device independent. Similarily, it should interface to any type of display through a display driver". One advantage to software which is adaptable to any hardware system is that once programmers have learned to use it they become portable, too, and can move about freely among systems.

Several graphics standards exist now with some moving towards national or international acceptance. In 1977 the Core Graphics System standard was developed by an ACM Siggraph Committee. It is a three-dimensional standard defining the boundary between applications and the graphics support package as well as specifying the content of that package. Although it was not officially accepted by the International Standards Organization, (ISO), it has been influential in the development of other standards, particularly the Graphical Kernel System, (GKS). GKS has been officially accepted by ISO.

It has been called a "Small Core" because it, too, defines the boundary between the applications and the graphics support package. However, it does not specify the content of the graphics support package, and it is a two-dimensional standard although three-dimensional extensions have been discussed.

Machover and Myers [22] list five other standards and their status toward national or international approval. They are as follows:

1) Initial Graphics Exchange Specification, IGES. It provides for the exchange of graphics databases among CAD/CAM systems.

2) North American Presentation Level Protocol Syntax, NAPLPS. It uses sequences of bytes of ASCII code and code extensions to describe graphics and text in separate frames. As the name suggests, it functions at the sixth level of the ISO's Open Systems Interconnection model and can transmit graphics and text over a low-capacity data communication link.

3) Programmer's Hierarchical Interface to Graphics Standard, PHIGS. It is an updated, expanded, dynamic, three-dimensional version of Core.

4) Virtual Device Metafile, VDM. It is a two-dimensional, device-independent standard conceived to satisfy both the Core concept and GKS for metafiles. It functions at the level just above device drivers and is concerned with the transfer of picture information between different graphics devices.

5) Virtual Device Interface, VDI. It also operates at the level just

above device drivers but is a two-way communications protocol. It interfaces between device-independent software and device-dependent code.

Among the problems peculiar to raster systems, the memory contention problem has received significant exposure in the literature. To avoid flicker in the displayed image, the display controller must refresh the display 30 or more times per second. For the illusion of motion the display must be refreshed even more often to avoid jerky movements, so the display controller must access the image memory for the display data frequently. Meanwhile, the display processor is calculating the data for the next frame to be displayed and must update the image memory before the display controller needs the new data. As a result both have high demands for accessing the memory creating a memory contention problem. Adding to this matter, many of the problems to be discussed later, (including shading), increase the processing time to generate the new image thus affecting the display processor's attempts to meet its timing constraints.

With most systems a stable display is the foremost requirement so the needs of the display controller are met first then tradeoffs are made to accommodate the display processor. Some specific timings that are critical to the operation of the display controller are the refresh rate, retrace times, total line time, active line time, total frame time and pixel time. All of these pertain to the display and though most build on each other, they are particularly dependent on the refresh rate. As previously stated, the refresh rate is the number of times the entire screen must be scanned per second. The retrace times are the amount of time needed by the electron beam to reposition itself after displaying a raster line. Since the beam is blanked, the display controller may not need data from the memory, depending on whether it has enough for the next active session. Often the display processor will access the memory during these retrace times.

The total line time is the average time required by the electron beam to scan a visible raster line including the horizontal retrace time. Active line time does not include the retrace time since the beam is blanked. The total frame time is the total line time multiplied by the number of visible lines in the frame plus the vertical retrace time; its reciprocal is the refresh rate. Pixel time is the average time necessary for the beam to scan a single pixel; it is the active line time divided by the number of pixels per line. Essentially, it is the rate that new pixel data must be supplied to the video output hardware to support display refresh. In other words, if data for only one pixel is obtained during each memory cycle, pixel time determines the rate in which the memory must be accessed by the display controller; therefore, the pixel time determines the necessary bandwidth of the memory and processors. According to Righter [28], the required, instantaneous, memory bandwidth is equal to the reciprocal of the pixel time. He presents a table of required memory bandwidths for a 60-Hz display versus pixel times for different display dimensions, shown in Table 2.1.

To achieve such high bandwidths, many solutions have been presented. One device designed to allow better access to the image memory is dual-ported memory chips. These devices provide two access paths with the actual memory time-shared between these ports; it does not imply that the same memory location can be accessed simultaneously from both ports but this method does provide some time savings [19,27]. Some of these chips include extra logic to help decrease access times. On-board shift registers

Table 2.1: Required memory bandwidths for a 60-Hz display as inverse of pixel time [28]

Display Size	Pixel Time	Bandwidth Required
512 x 384	66.5 ns	15 MHz
640 x 512	38.3 ns	26.1 MHz
1024 x 768	14.4 ns	69.4 MHz
1024 x 1024	10.0 ns	100 MHz
1280 x 1024	8.0 ns	125 MHz

transform a large block of parallel data to a serial bit stream leaving the device free for the next access [38]. Drumm, Harris and Ebertin [11] include on their device logic to handle memory contention so it is possible to access the same location simultaneously from both ports. Williams [37] compares the different types of general purpose memory chips available, such as SRAMs, DRAMs, EPROMs and EEPROMS.

Other solutions to the problem of memory contention include architectures for the memory itself to increase its bandwidth. Depending on the arrangement of the memory chips, different access modes can be implemented to access rows, columns, pages or nibbles. Whitton [36] summarizes the different access modes as she thoroughly discusses the memory contention problem. She claims that displays of moving, smooth-shaded solids and vector objects almost always require a technique known as double buffering. Double buffering is when two complete images are stored in the video buffer. While the display controller is accessing one copy to update the display, the display processor is writing the next frame's data into the second copy. At the end of their cycles the two processors switch copies. Double buffering nearly eliminates the memory contention problem but is costly to implement. Overall, Whitton concludes that bigger memory devices do not necessarily mean better access and stresses using different architectures to achieve higher bandwidths.

Though the display controller needs the image memory to supply data to the display, high-performance graphics dictates that a large number of pixels be written into the memory every frame time. For this reason van Dam [39] claims graphics cannot be done remotely; both the display processor and the display controller need to access the image memory. This is especially true of high-performance graphics systems because they need a front-end mini- or mainframe computer for the data manipulations. In general, as interactivity decreases, frustration increases contributing to premature user fatigue. So the display processor must be able to quickly process the new image and store it into memory for the display controller to display. As designers strive for realism, images are becoming more complex requiring excessive time to generate updated images corresponding to new viewpoints derived from user inputs. Again, architectures can be implemented to speed up processing, such as pipeline and/or parallel processing, and petri nets which are data-driven systems. Many papers exist discussing multiprocessing techniques both for their own merit and as applied to specific graphics systems. Such architectures provide increased throughput and, if modularly designed with well-defined interfaces between the processors, they may provide modification flexibility. In addition specialized hardware to perform some image generation functions greatly decreases processing time though at an increase in cost.

Most of the other problems of high-performance graphics systems affect the image-processing time. The first step to producing an image is to model the object. Two methods for modeling objects have already been discussed; polygon-mesh and curved-patch equations. Generally, objects do not follow surface models very well. Particularly with natural phenomena such as clouds or smoke, these models become extremely complex trying to imitate these objects. Also the diversity of a design within a given framework is limited. A model may be able to produce an image of a tree but it may not be able to distinguish a poplar from a maple tree. Then again, such fine detail may not be necessary, depending on the application. The model must also provide a three-dimensional projection format. This not only keeps track of faces and edges seen in the chosen view but those not seen, too, as well as the depth or z-coordinate of each. This type of format allows for hidden surfaces to be appropriately shown when the view changes and aids in most of the processes to be discussed such as clipping, hidden surface removal, shading and shadow generation.

Actually, all four of the processes mentioned are related to each other and are especially dependent on how the object was modeled. Clipping is a technique for not showing portions of the object outside of the window. It is executed before hidden surfaces are removed because it determines the depth coordinates of each of the surfaces of the image. It also decreases the number of surfaces so these are removed before any more calculations are performed on them to avoid wasting time.

Hidden-surface removal is the next process performed on the image because it, too, removes surfaces from the image so they are eliminated before they are processed any further. Also known as the visibility problem, hidden-surface removal is used on both solid models and wireframe drawings alike. It removes any surface or line that is obscured by surfaces that are closer to the viewer, thus generating a realistic, solid picture or a less confusing wireframe drawing. Unfortunately, the identification and removal of hidden surfaces is very time consuming, so many different algorithms have been developed to solve this problem. One notable contribution was from Sutherland, Spoull and Schumacker [31]. They compared ten such algorithms trying to find some fundamental insights into the problem itself. They concluded that all of the algorithms performed some sorting of the surfaces to determine which were visible; if the sorting process could be made more efficient the execution time would decrease. They presented two methods of sorting different from those used by the algorithms that they compared. A second conclusion was the detection of a principle called coherence; that objects have a local constancy about them. "Scan-line coherence" is the fact that a scan line changes very little between successive frames. "Frame coherence" is that an entire picture is nearly the same from one frame to the next. In fact the world model of an object changes less frequently than the viewing position. These principles may be used to reduce the amount of computations necessary for generating successive frames.

Other features which enhance realism are filling adjoining surfaces with contrasting shades to show intersections, adding highlights and textures, shading surfaces to show form and three-dimensional qualities, and simulating transparent surfaces. These will all be treated in the next section which explains the shading problem more completely.

The final process covered in this section is shadow generation. Shadows enhance realism by providing depth cues reinforcing the three-dimensional effect. They are not dependent on the viewpoint but on the type of light sources available and their locations. If, however, the light source is behind or at the viewpoint no shadows will be visible; they will be cast behind the object, hidden from view. This is often the approach taken to avoid the extra calculations to generate shadows. The same algorithms used for hidden-surface removal can be used to create shadows except the light source's location is used as the viewpoint. Instead of removing the surfaces hidden by other objects, their intensity or color is changed to produce the shadow. Since the processing is so similar, it is possible to compute hidden surfaces for removal and shadows simultaneously.

Nevertheless, even after all of the above processes have been performed on the image there are still problems merely drawing it onto the display. A line or arc cannot simply be drawn from point A to point B on a raster display because of the scan pattern; it must be transformed or mapped to a pixel representation which is a process called scan conversion. This is usually performed by the raster display itself so the user does not need to be concerned about it. Once the lines are drawn, they must be smoothed. Any nonvertical or nonhorizontal line may appear as a staircase because of the arrangement of the pixels in a matrix which form the line. There are two approaches to this problem, known as aliasing. The first approach is to increase the resolution of the display. By increasing the pixel density the jaggedness of the edges and lines will be smaller and less perceptable but there will be more points to compute overall. The second approach is to use multiple bits of memory to represent each pixel's intensity or color. Varying the intensity along jagged edges will create the impression of straight lines because the edges will be fuzzed, or faded into the background. Once the lines have been smoothed, intersections of surfaces must be computed particularly for objects modeled by equations. Phillips and Odell [25] discuss this problem and present an algorithm to attempt to solve it. They stress that it is often more difficult to find intersections than to just display them. Fortunately, for many applications this is sufficient.

The next section provides more detail concerning the shading problem and many of its related processes mentioned above.

2.3 The Shading Problem

Essentially, the problem of shading is to generate solid images by filling the polygons or patches formed during the modeling stage in such a way as to enhance realism. Shading is affected not only by the method of modeling the object but also by the hidden-surface removal algorithm used; the order in which the hidden-surface removal algorithm computes visible information influences the shading algorithm. The best attribute a hidden-surface removal algorithm can possess, with respect to shading, is to generate information by scan line rather than arbitrarily ordered patches or polygons. Because these two algorithms are so closely related they are often treated together in published studies.

Shading algorithms vary in complexity ranging from those that arbitrarily fill the surfaces of an image to those that record minute details of surface properties, texture and patterns. All but the simplest involve the behavior of the human visual system and principles of optics. Peculiarities of the behavior of the human visual system often force algorithms to compensate, or even deviate from, otherwise uniform shading rules. The effects will be discussed as they pertain to the method or problem being presented.

Briefly, the optics principles involve the angles of incident light to the surfaces being shaded. The resulting reflection, absorption or transmission from the surface is dependent on many factors. One such factor is the surface properties of the object being modeled; a smooth, glossy surface will reflect more light than a dull, matte surface. Another factor is the light source itself including both the type of light and its location. The type of light as well as its luminance, or intensity, is influencial in determining how an object will be lit up. Diffuse background light, or ambient light, produces constant illumination of objects regardless of their orientation to the light. This uniform brightness makes objects appear flat and does not usually produce realistic images when used alone. Point sources of light can produce specular reflections, or highlights, which are dependent not only on the object's orientation to the light but also the viewer's orientation to the object with respect to the light's location. This creates much more interesting images.

The absorption of light determines the object's color or intensity. The transmission of light through objects is related to absorption by the object's degree of transparency. It is similar to the shadow-generation and hidden-surface removal processes because objects behind transparent objects must be identified. The illumination of these "hidden" objects is altered according to the amount of light actually allowed through the object in the foreground and whether or not that light is refracted. Moreover, transparent objects often yield specular reflections, which sometimes help reinforce the presence of clear objects. One method effectively used to generate transparent objects, in addition to opaque objects which have a zero percentage of transparency, is called ray-tracing. Using the principles of optics, light rays are followed from the viewer to the first surface where the ray will branch into its components of reflected and refracted light. Each of these components is followed forming a tree that can be used to determine the shading intensities of the surfaces viewed. One such algorithm is presented in the next chapter.

Surface detail can be conveyed through shading; textures and patterns enhance realism and can be exhibited through proper shading. One method is to map a digitized photo or model of the pattern or texture to the surface of the image. This mapping determines the pixels' intensities or colors and is probably the best way to generate patterns. Textures can be shown in this manner or they can be modeled. One approach is not to actually model the texture but to perturb the surface normal to indicate the texture. The surface normal is critical to measuring angles of incidence, reflection and refraction from the surface while computing intensity values. So, by altering the normal's true direction a smooth surface will appear as a rough surface, but the overall effect is not very realistic. Another approach is to model the texture using fractal mathematics which were developed by Benoit Mandelbrot. This method uses stochastic processes to model the randomness of natural phenomena. This method, too, will be discussed further in Chapter III.

Another method for generating shaded images is known as half-tones; this is the method used in most printed matter such as newspapers, books and magazines. The human visual system performs spatial integration where a small area, when viewed from a distance, appears as a single intensity despite the fact that a close-up examination reveals fine detail of varying intensities. This effect is used when producing half-tone images. The screen is divided into small resolution units, usually a small, square matrix of pixels. Each resolution unit is imprinted with a black dot whose area is proportional to the amount of blackness of the corresponding area of the object being displayed. (Blackness equals one minus the intensity.) This method provides more intensity levels without increasing the number of bit planes but spatial resolution of the display is sacrificed.

Color is another type of surface detail. Achromatic color includes black, white and shades of gray with its only attribute being the intensity of the colors. Chromatic displays have many attributes. Hues distinguish between different colors. Saturation, also called chroma, refers to the purity of a color or the amount of dilution by white light. (White light is 0% saturated.) Brightness, or value, is similar to intensity for achromatic color. A tint is the result of adding white light to a color thereby decreasing its saturation. Shades result when black is added to a pure color, decreasing its brightness. Tones are the result of the addition of both black and white light to a color.

Several theories have been developed concerning the eye's reaction to color. One of specific interest for raster systems is called the tri-stimulus theory. This theory states that there are three different types of cones on the retina of the eye, each having peak sensitivities to light of either red, blue or green hues. This is aligned with the concept of using these same hues in combinations to produce most colors on color television or raster-scan monitors. A comprehensive discussion of color which covers topics like different color models for monitors and printers, and objective descriptions of colors using electromagnetic energy densities and standard chromaticity diagrams is beyond the scope of this paper but is included in Foley and van Dam's book [12]. One color model of pertinence to raster graphics is the RGB color model. Using a right-handed cartesean coordinate system, a unit cube is formed with black located at the origin, white at the point (1,1,1), and red, blue and green are located on the axes where z=1, y=1 and x=1, respectively. The main diagonal connecting black and white contains equal amounts of each primary and represents the gray levels. However, this model is hardware-oriented so it is not easily controlled by the user because it does not directly relate to intuitive notions of hue, saturation and brightness. Nevertheless, to implement color on a raster display three sets of equations are necessary, one for each primary hue. This is why color systems generally require higher bandwidths of the memory and processors and are more difficult to implement.

Overall, the procedure for computing shades or RGB values for a particular pixel involves determining which polygons or patches are mapped there, finding details about the surfaces assigned color or intensity, calculating the pixel's angle and distance from the light source and from the viewer, then computing the shading value for the pixel in question. Details about the surface include taking into consideration most of the processes already discussed to create a realistic image. After everything is considered, shading is a complicated process and many algorithms have been developed to provide a solution. Several of these algorithms are presented in the next chapter.

CHAPTER III

PRESENTATION OF SHADING ALGORITHMS

Some of the first shading procedures are known as constant-shading algorithms. These are usually applied to objects modeled using the polygon-mesh technique where each planar, polygonal facet is filled with a single intensity value or color. One problem with this method is that adjacent polygons may exhibit an obvious difference in intensity; in reality, objects are composed of continuous curves and their intensities vary continuously. By shading images with this method the polygons used to model the object are apparent to the viewer.[22] This conspicous transition from one intensity to the next is called contouring and contributes to the unrealistic appearance caused by this method. Another problem with this method is known as the Mach-band effect, which is caused by the human visual system. If the light-intensity curve from illuminated surfaces has a discontinuity in magnitude or slope, the eye accentuates the change. Thus, the difference in shading of adjacent polygons is exaggerated.

More complex algorithms have been developed to overcome these problems to provide continuous shading of curved surfaces as well as simulate texture, transparency and other attributes discussed in the previous chapter. Four of the next five sections present one such algorithm. The third and seventh sections present new shading models which could be applied to some of the other algorithms. The sixth and eighth sections introduce different methods for enhancing the object models. The sixth section uses the curved-patch modeling technique discussed in Chapter II. The eighth and final section presents a new method based on fractal geometry to alter models of terrains and other natural phenomena. Both may be applied along with any of the other shading models to generate realistic pictures. Included in this text are many equations and figures that are modified versions of those that appeared in the papers which originally presented the algorithms. Some adjustments were necessary to unify the notation used throughout this text as well as to eliminate or add portions of figures to better represent the discussions contained here.

3.1 Gouraud Intensity Interpolation Shading

This algorithm [18], published in 1971, is applied to a curved-patch object-modeling technique called rational Coons patches but could easily be modified for the polygon-mesh technique. In 1964 S. A. Coons introduced a modeling technique to extend the class of objects that may be modeled; this technique allows for the definition and representation of curved surfaces. An extension was developed by T. M. P. Lee in 1969 called the rational Coons patch. One of its properties is that patches can be reparameterized without modifying their geometric shapes. Gouraud's algorithm is based on a hidden-surface removal algorithm developed by G. S. Watkins which accepts nonplanar polygons; so, Gouraud extended it to rational Coons patches.

Watkins' algorithm computes information about the image scan line by scan line, which facilitates the shading process. The actual shading rule implemented utilizes basic principles of optics which take into consideration the object's orientation and its distance from the viewer. The light source is assumed to be at the same location as the observer to avoid the need to generate shadows. The object's orientation is measured as the cosine of the angle, θ, between the surface normal, $\overline{N}$, and the direction of the light, $\overline{L}$; or as in this case, the viewer, $\overline{V}$; shown in Figure 3.1. The distance from the viewer is introduced to distinguish between overlapping, parallel planes which would otherwise be shaded with the same intensity since both have the same orientation. This also simulates the manner in which the eye perceives illuminated objects from a distance; because light energy decreases as the inverse square of the distance, parallel faces at different distances from the viewer would appear to be different intensities in reality. According to Gouraud, the method used to compute the distance is not important as long as the relative ordering of the faces is preserved. Using the perspective transformation, the (x,y,z) coordinates of a point become the projection coordinates (x/z, y/z, 1/z) if the observer is located at the origin of the coordinate system and is facing in the positive z direction. Using the 1/z coordinate as an approximation of the distance, the shading equation becomes

$$S = \frac{1}{z} \cos^2 \theta \tag{1}$$

Since the distance values, 1/z, are only known at the vertices of the polygons, it is necessary to perform a linear interpolation for points between the vertices to obtain the distance values. After the distance values have been computed, the shading for point P located on the scan line between points E and F shown in Figure 3.2 is approximated using the equation

$$S_P = (1 - \alpha) \frac{1}{z_E} \cos^2 \theta + \alpha \frac{1}{z_F} \cos^2 \theta \tag{2}$$

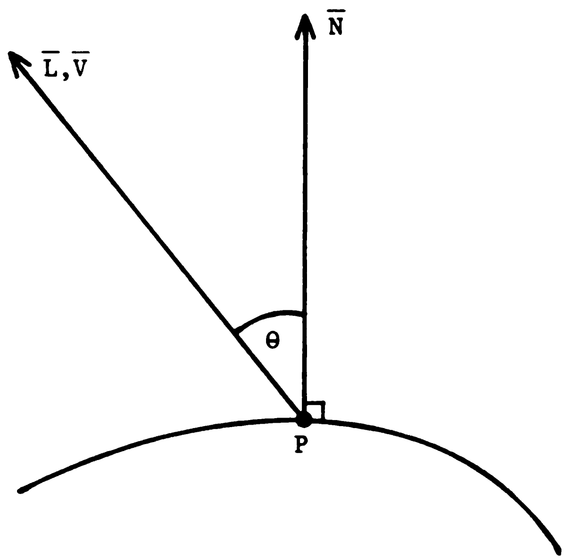

Figure 3.1: Geometry of reflection model for Gouraud's algorithm

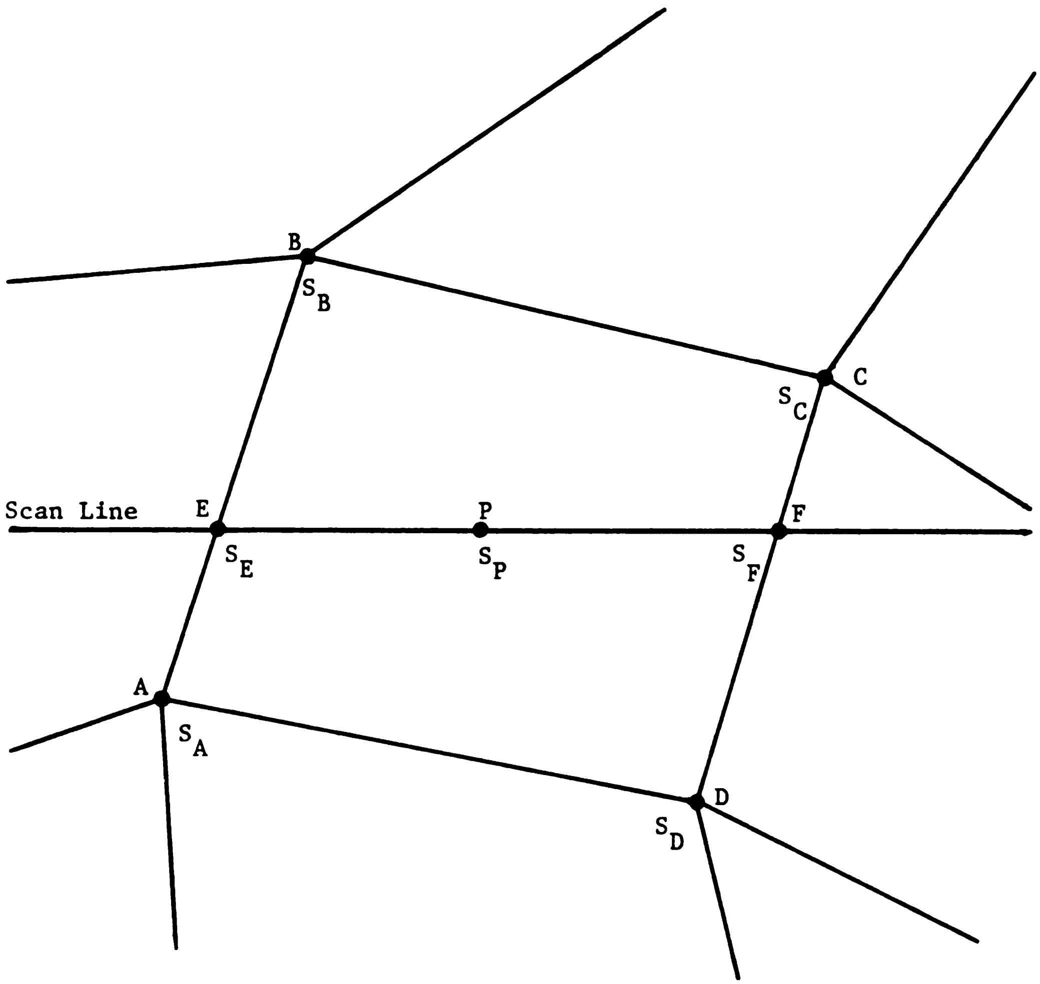

Figure 3.2: Projection of one polygon intersected by the scan line [18]

The coefficient α ranges as $0 \lesssim \alpha \lesssim 1$ and denotes the position of the point P on the scan line between the end points E and F: if P is located at E, then $\alpha = 0$; if P is at point F, then $\alpha = 1$. Gouraud claims there is no noticeable degradation in the shading from using this approximate equation.

To smoothly shade curved surfaces Gouraud modifies the shading computation of each patch so that continuity exists across each boundary. Since each vertex of the patch will be oriented differently, thus requiring different shading, interior points have to be shaded as a continuous function of the vertex shading. Generally, these modifications attempt to alleviate the effects of contouring and the Mach-band effect. To help achieve this shading continuity a normal for each vertex is computed by either averaging all of the patch vertex-normals associated with the vertex or using an analytical description of the surface to compute the exact normal.

Two successive linear interpolations are performed to compute the shading of interior points for each patch. Referring again to Figure 3.2, the surface normals are assumed to be known at the vertices A, B, C and D. The scan line intersects edge AB at point E and edge CD at point F. The point P is any point inside the patch ABCD that is on the scan line. The shading at points E and F is interpolated using the shading values calculated at the vertices. The shading at point E is calculated using the shading values from point A, S_A, and from point B, S_B, in the equation

$$S_E = (1 - \alpha_E)\ S_A + \alpha_E\ S_B \quad (0 \lesssim \alpha_E \lesssim 1) \qquad (3)$$

where α_E is defined similarily to α in equation (2). Likewise, S_F and S_P can be calculated using the equations

$$S_F = (1 - \alpha_F)\ S_D + \alpha_F\ S_C \quad (0 \lesssim \alpha_F \lesssim 1) \tag{4}$$

$$S_P = (1 - \alpha_P)\ S_E + \alpha_P\ S_F \quad (0 \lesssim \alpha_P \lesssim 1) \tag{5}$$

Using these equations, it can be verified that if

$P \equiv A$, then $S_P \equiv S_A$

$P \equiv B$, then $S_P \equiv S_B$

$P \equiv C$, then $S_P \equiv S_C$

$P \equiv D$, then $S_P \equiv S_D$

Since Watkins' technique for computing hidden surfaces efficiently calculates and tabulates data for the image, this was extended to include the shading calculations to help minimize the computation of a new shade for each point. Watkins scans the picture from top to bottom by scan line, computing the following information for each polygon edge:

1) The number of the first scan line that intersects the edge.
2) The total number of scan lines that intersect the edge.
3) The x and z coordinates of the highest point of the edge.
4) The slope in x and z for the edge.

The necessary shading information is easily added to this list:

5) The shading, S, of the surface at the highest point of the edge.
6) The "slope" of the shading along the edge.

The shading "slope" is calculated as

$$\Delta S = \frac{S_2 - S_1}{n} \tag{6}$$

where S_1 and S_2 are the shading of the two endpoints of the edge and n is the total number of scan lines that intersect the edge.

With the above information the shading may be computed for a given scan line. An edge will become "active" when its first point is reached by a scan line which is being used in the shading computation. The xyz coordinates are known for this point along with the value of the shading. A segment is created when an edge becomes active by pairing edges which belong to the same patch; the segment is the portion of the scan line between the paired edges and contains information about the coordinates of the endpoints, the values of the shading at the endpoints, and both the coordinate slope and shading "slope" necessary to update shading information from scan line to scan line. From the present scan line the slopes are added to the coordinate information of the point on the edge and to the point's shading value to find the coordinates and shading of the next point where the next scan line intersects the same edge. After the hidden-lines computation is performed, many of the segments are totally or partially visible. The shading is calculated for each visible point of a segment along a scan line by computing a coefficient as

$$\alpha = \frac{X_P - X_E}{X_F - X_E} \tag{7}$$

where the X's represent the displacement along the scan line. Using this value for the coefficient α, the shading can be calculated for the point P of Figure 3.2 using equation (5).

According to Gouraud, this linear interpolation for the shading intensities produces shading across patch boundaries which is continuous in value but not in derivative. This eliminates most of the contouring effects but some Mach-band effects may be seen in the vicinity of silhouette curves and where the surface curves sharply. He stresses that this algorithm may be implemented in hardware since it is only a linear interpolation being performed, and he compares the execution times required for Watkins' algorithm and the extended algorithm proposed by Gouraud. If Gouraud's algorithm is totally implemented in hardware, no extra time will be required for execution though extra demands will be made of the memory. If the algorithm is implemented in software, the total time required by Watkins' algorithm would be multiplied by less than 1.2 for Gouraud's algorithm's execution time. Two systems which have implemented Gouraud's algorithm are described in papers published by Fujimoto, et al. [17], and Fuchs, et al. [16].

The next section presents an algorithm which tried to improve upon Gouraud's by reducing the Mach-band effect.

3.2 Phong Normal-vector Interpolation Shading

This algorithm [26], published in 1975, was developed by Bui Tuong Phong. It expands upon Gouraud's algorithm because instead of linearly interpolating the intensity value of the shading, Phong's algorithm interpolates the surface normal and then calculates the new shading values using these normal vectors. Phong also uses a more complex shading equation which allows the viewer to be in a different location than the light source in addition to taking into account reflectivity of the object and specular reflections. Phong presents the more complex shading rule with his algorithm as an attempt to achieve more realistic shading. Another difference between the two algorithms is that Phong's algorithm is presented for objects modeled using the polygon-mesh technique.

Phong's shading equation is based on the physical principles of optics with a few empirical adjustments. The direction of the incident light is always measured as an angle with respect to the surface normal, θ. The angle of incidence equals the angle of reflection, so the direction of the reflected light is also measured as θ with respect to the normal. Different types of lighting affect the object's illumination in different ways as do different types of surfaces. Rough or dull surfaces scatter the reflected light in all directions equally, an effect called diffuse reflection. This type of reflection follows Lambert's cosine law which relates the amount of light reflected and the direction of the light source to the surface as shown in the equation

$$S_{P,d} = C_p \cos \theta \tag{8}$$

where C_p is the coefficient of reflectivity of the surface; C_p is a ratio of the light reflected from the surface to the total amount of incoming light at the point P. This type of reflection is not dependent on the viewer's location because the object appears a constant intensity from all directions.

Diffuse background light, or ambient light, produces constant illumination of objects regardless of the object's orientation; reflection is dependent only on the object's coefficient of reflectivity and the intensity or brightness of the light, for which Phong uses an environmental diffuse reflection coefficient, C_d. Normally, an object subjected to only ambient light would be illuminated according to the equation

$$S_{P,a} = C_P \, C_d \tag{9}$$

but Phong combines the equations (8) and (9) as

$$S_{P,(d + a)} = C_P \, (\cos \Theta \, (1 - C_d) + C_d). \tag{10}$$

Diffuse reflection from colored surfaces requires three equations, one for each primary color. Most of the light is absorbed but the color of the light reflected is the perceived color of the object.

Specular reflections depend on the location of the viewer because the light is reflected unequally in different directions. Such highlights are emitted from shiny surfaces and appear white, or the color of the incident light because most of the light is reflected. For a perfect mirror, light is reflected only in the direction of perfect reflection; that is when the angle of incidence equals the angle of reflection. This is the reason for the concentration of reflected light in the highlight. For nonperfect reflectors the reflected light is not quite as concentrated but falls off rapidly as the direction moves from that of perfect reflection. Therefore, the viewer's location is critical; as the viewer moves from the direction of reflection, less light is available to be seen as a highlight. The direction of the line of sight is measured as the angle σ from the reflection vector, $\overline{R}$, as shown in Figure 3.3. Phong approximates the specular reflection as the cosine of σ raised to the power c_1, where c_1 usually ranges from one to ten. When the surface is a perfect reflector, c_1 is large so the value will rapidly go to zero as σ deviates from Θ. This cosine term is multiplied by a function $W(\Theta)$, which is a function of the ratio of the specularly reflected light and the incident light as a function of the incident angle; $W(\Theta)$ ranges between 10 and 80 percent. Both c_1 and

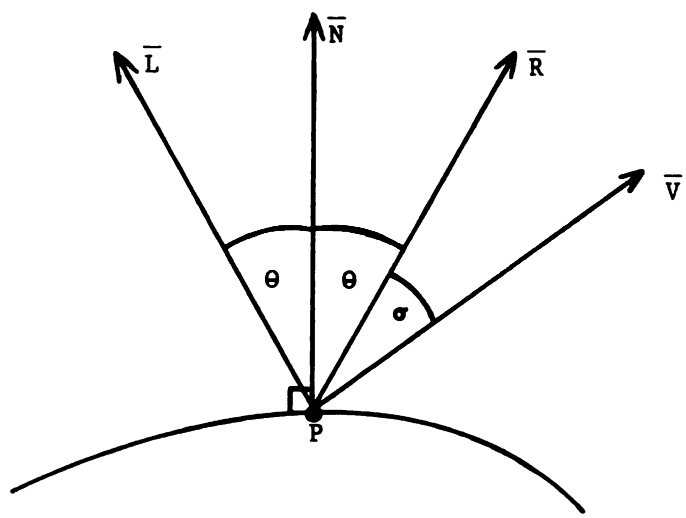

Figure 3.3: Geometry of reflection model for Phong's algorithm

$W(\theta)$ are empirically adjusted for the picture and no physical justifications are made by Phong. Hence, the complete shading equation becomes

$$S_P = C_P (\cos\theta (1 - C_d) + C_d) + W(\theta) \cos^{C_1} \sigma. \qquad (11)$$

For each point on the surface Phong calculates the normal, which is used in the above shading equation, using a linear interpolation technique similar to that used by Gouraud to interpolate the shading intensities. Initially, the normals are only known for the vertices of the polygons. Normals along polygon edges and interior to the polygon are computed using

$$\overline{N}_P = (1 - \alpha)\, \overline{N}_1 + \alpha\, \overline{N}_2 \qquad (12)$$

where α is defined in the same manner as in equation (7). The normal to a visible point inside the polygon is determined from a linear interpolation of the normals at the intersections of the two edges of the polygon with the scan plane passing through the point under consideration. Thus, the general surface normals are quadratically related to the vertex normals.

Using these normals, the shading values can be determined. Some assumptions are made to simplify the cosine terms; both the light source and viewer are assumed infinitly far away. The cosine terms may then be rewritten as

$$\cos\theta = \overline{L} \cdot \frac{\overline{N}_P}{|\overline{N}_P|} \qquad (13)$$

and

$$\cos\theta = \overline{V} \cdot \frac{\overline{R}_P}{|\overline{R}_P|} \qquad (14)$$

where $\overline{L}$ and $\overline{V}$ are unit vectors in the direction of the light and viewer, respectively, $\overline{N}_p$ is the surface normal at the point P, and $\overline{R}_p$ is the reflected light vector at P. The quantity represented in equation (13) is the projection of a normalized vector, $\overline{N}_p$, on an axis parallel to the direction of the light source. If the magnitude of $\overline{N}_p$ is unity, then equation (13) is one component of $\overline{N}_p$ in a coordinate system where one axis is in the direction of the light. In this case the quantity in equation (14) can be obtained directly from $\overline{N}_p$.

To find the value of equation (14) from $\overline{N}_p$ a Cartesean coordinate system with the origin located at the point P and the z-axis parallel to the light but pointing in the opposite direction, as shown in Figure 3.4, must be used. Four assumptions must be made about the model:

1) The normalized vector $\overline{N}_p$ makes an angle θ with the z-axis; therefore, $\overline{R}_p$ makes an angle 2θ with the z-axis.
2) $\theta \leq 90°$. If $\theta > 90°$, then the light is behind the surface being considered. In the case where a view of the back surface is desired when it is visible, the normal is assumed to always point toward the light source.
3) If $\overline{L}$ is the unit vector along the Pz-axis, then vectors $\overline{L}$, $\overline{N}_p$, and $\overline{R}_p$ are coplanar.
4) The two vectors $\overline{N}_p$ and $\overline{R}_p$ are of unit length.

Using assumption (3), the projections of the vectors $\overline{N}_p$ and $\overline{R}_p$ onto the plane defined by (Px, Py) are merged into a line segment as shown in Figure 3.5. Therefore,

$$\frac{\overline{X}_r}{\overline{Y}_r} = \frac{\overline{X}_n}{\overline{Y}_n} \qquad (15)$$

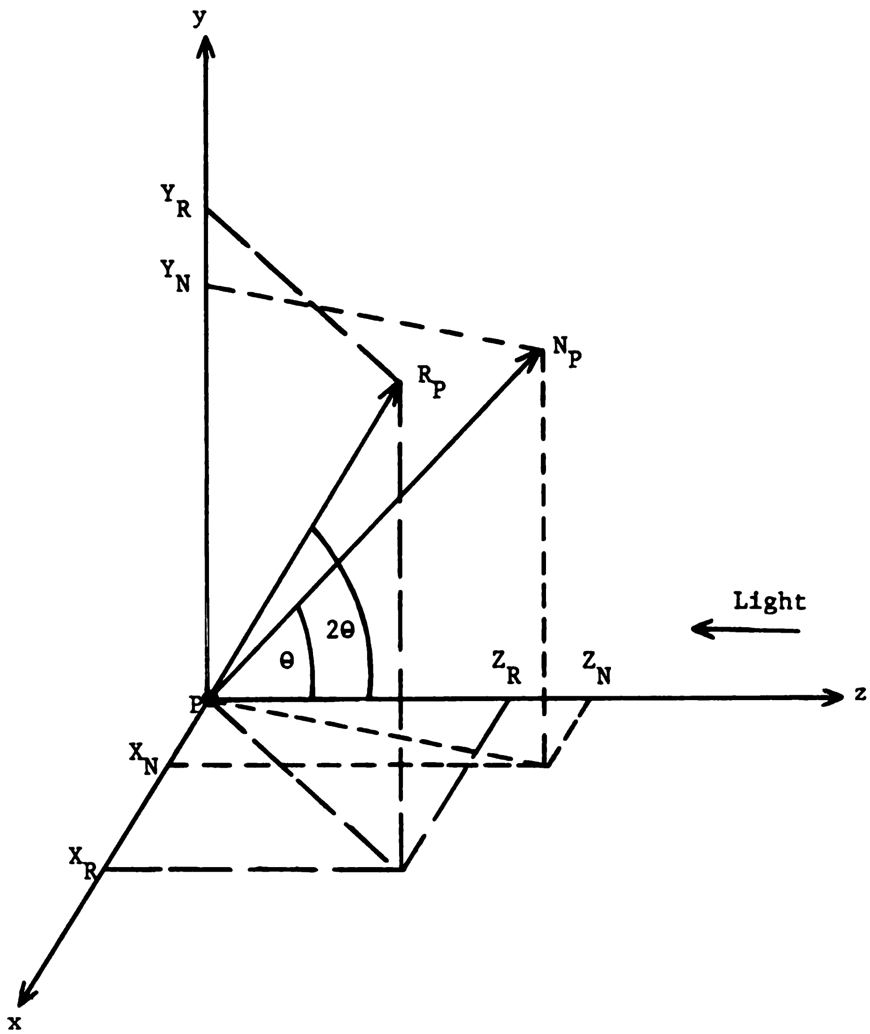

Figure 3.4: Determination of the reflected light [26]

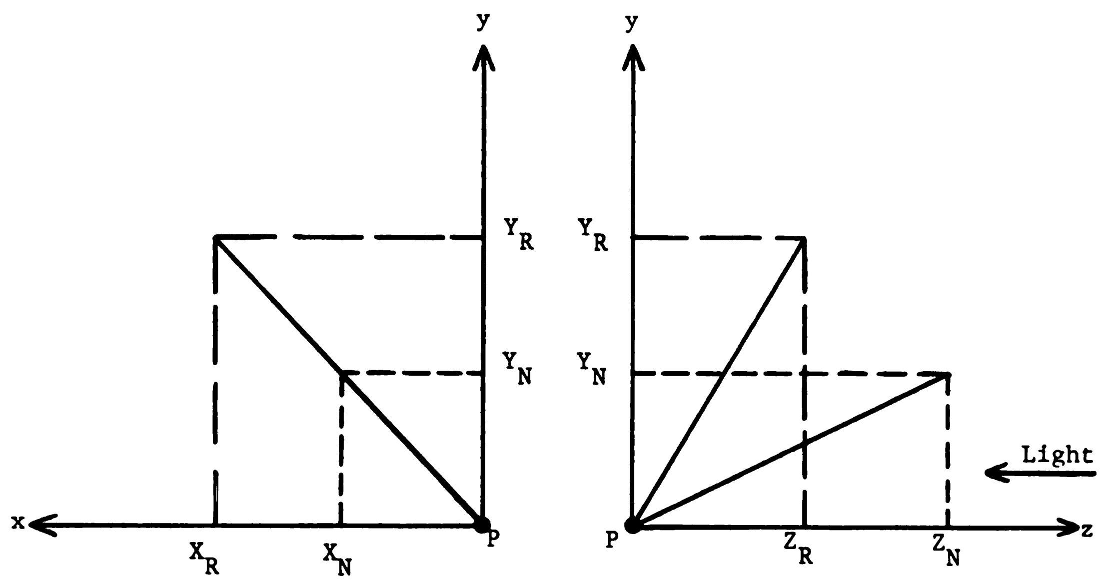

Figure 3.5: Projections of the reflected light [26]

where $\overline{X}_r$, $\overline{X}_n$, $\overline{Y}_r$, and $\overline{Y}_n$ are components of $\overline{R}_P$ and $\overline{N}_P$ in the x and y directions, respectively.

Using assumptions (1) and (2), the component $\overline{Z}_n$ of $\overline{N}_P$ is

$$\overline{Z}_n = \cos \theta, \tag{16}$$

meaning $0 \leq \overline{Z}_n \leq 1$. The following relations are obtained from trigonometric identities

$$\overline{Z}_r = 2\overline{Z}_n^2 - 1 \tag{17}$$

and

$$\overline{X}_r^2 + \overline{Y}_r^2 = 1 - \cos^2 2\theta. \tag{18}$$

Using equations (15) and (18), we obtain

$$\overline{X}_r = 2\overline{Z}_n \overline{X}_n \tag{19}$$

and

$$\overline{Y}_r = 2\overline{Z}_n \overline{Y}_n \tag{20}$$

Thus, the three components of $\overline{R}_P$ are obtained from $\overline{N}_P$ and are known in the light source coordinate system. The projection of $\overline{R}_P$ onto the z-axis of the viewer coordinate system requires finding the dot product of $\overline{R}_P$ with this z-axis. The component of $\overline{R}_P$ on an axis parallel to the viewing direction is then evaluated as the cosine of σ, which is used to simulate specular reflections.

Phong states that interpolating $\overline{R}_P$, as is done for the normal vectors, would be a more time-consuming process, calculating these vectors directly requires less storage space as well. By calculating the shading values from interpolated normals, a better approximation of the curvature of the surface is obtained and highlights are more accurately simulated. Unfortunately, the Mach-band effect is not completely eliminated because a continuous derivative of the shading function across polygon edges is not guaranteed; subjective brightness along abrupt changes in orientation of adjacent polygons will be visible. This is inevitable because, according to the Mach-band effect, it will be visible at abrupt changes in the slope of the intensity distribution curve regardless of whether or not the first derivative of the curve is continuous. Phong tried using higher-degree interpolation schemes and the effect was still visible. Furthermore, the images produced differed very little from those produced by the method presented here so the latter was determined a better technique because it uses less time and may be implemented in hardware. However, this method did produce a marked improvement over Gouraud's algorithm for simulating smooth shading, although it requires more than three times the hardware to implement and a slight increase in execution time; but Phong feels the improved quality is worth it.

The next shading algorithm improves on Phong's method by not using any empirical adjustments in the shading model.

3.3 Blinn Normal-vector Interpolation Shading

In this algorithm [3], published in 1977, James F. Blinn uses a theoretical shading model derived by K. E. Torrance and E. M. Sparrow. Blinn does not actually present an algorithm to implement the shading model but applies his model to existing algorithms then compares his images generated to those from Phong's algorithm. Blinn's experimental results generally match those from Phong's algorithm but some differences arise.

Blinn's shading model simulates highlights more accuractely because it uses a theoretical model whereas Phong's shading model includes some empirical adjustments. One of the two main differences is that the amount of specular reflection varies with the direction of the light source. The second main difference is that the direction of the peak specular reflection does not always coincide with the direction of reflection, $\overline{R}$, where the angle of the reflected light with the surface normal equals the angle of the incident light.

Blinn's algorithm assumes that the surface is composed of a collection of mirror-like microfacets that are oriented in random directions. The specular component of the reflected light is assumed to come from facets that are oriented in the direction of maximum highlights, $\overline{H}$. If the surface was a perfect mirror, light would only reach the viewer if the surface normal bisected the angle between the directions of the viewer and of the light source. This required direction of the normal is $\overline{H}$ and can be defined as

$$\overline{H} = \frac{\overline{L} + \overline{V}}{|\overline{L} + \overline{V}|} \tag{21}$$

The diffuse component of reflected light results from multiple reflections between the facets, and from internal scattering. The Torrance-Sparrow shading model implemented combines four factors to generate the shading intensity:

$$S = \frac{D\ G\ F}{(\overline{N} \cdot \overline{V})} \tag{22}$$

where D is the distribution function of the facets' orientations, G is the amount by which the facets shadow and mask each other, F is the Fresnel reflection law, and $(\overline{N}\cdot\overline{V})$ is the cosine of the angle between the surface normal and the viewer. All vectors are assumed normalized. Each of these terms will be discussed more fully in turn.

Light will be specularly reflected only by facets posessing a local normal vector which points in the direction of $\overline{H}$. The distribution function, D, evaluates the number of facets pointing in this direction. Several different distribution functions have been proposed. Phong uses a cosine function raised to a power as presented previously except instead of measuring the angle between the directions of the viewer and the reflected light, the angle β is measured between the average surface normal and $\overline{H}$ of each facet to conform to Blinn's representation of the surface using microfacets. This angle may be defined as

$$\beta = \cos^{-1} (\overline{N}\cdot\overline{H}). \tag{23}$$

Blinn's version of Phong's distribution function becomes

$$D_1 = \cos^{c_1} \beta \tag{24}$$

The distribution function used in the Torrance-Sparrow model is a standard Gaussian distribution:

$$D_2 = e^{-(\beta\ c_2)^2} \tag{25}$$

where D_2 is the proportionate number of facets whose local normals form an angle β from the average surface normal. The factor c_2 is the standard deviation for the distribution which is a property of the particular surface being modeled; c_2 is large for dull surfaces and small for shiny surfaces.

A third distribution function has been proposed by T. S. Trowbridge and

K. P. Reitz which generates a very general class of surface properties by modeling the facets as ellipsoids of revolution:

$$D_3 = \left[\frac{c_3^2}{\cos^2 \beta \, (c_3^2 - 1) + 1} \right]^2 \tag{26}$$

where c_3 is the eccentricity of the ellipsoids. c_3 is 0 for very shiny surfaces and 1 for very diffuse surfaces.

Each of these distribution functions peaks when the value of the cosine term is 1, which is when facets point along the average surface normal so that β is 0. As β increases or decreases the values of the functions decrease at rates that are controlled by the values of c_1, c_2 and c_3. Blinn used a uniform angle, ω, at which the distribution falls to one half to compare the functions. In terms of ω the three controls become

$$c_1 = - \frac{\ln 2}{\ln \cos \omega} \tag{27}$$

$$c_2 = \frac{(\ln 2)^{0.5}}{\omega} \tag{28}$$

$$c_3 = \left[\frac{\cos^2 \omega - 1}{\cos^2 \omega - (2)^{0.5}} \right]^{0.5} . \tag{29}$$

Although similar plots are obtained of the functions for equal values of ω, Blinn uses D_3 for his shading model because it has experimental as well as theoretical justifications, and it is the easiest to compute. If ω does not change within a frame, D_3 can be calculated using intermediate values calculated once per frame:

$$k_1 = 1 \;/\; (c_3^2 - 1) \tag{30}$$

$$k_2 = k_1 + 1 \tag{31}$$

Using equations (30) and (31), D_s becomes

$$D_s = \left[\frac{k_2}{\cos^2 \beta + k_1} \right]^2 \qquad (32)$$

This speeds up the computation of the distribution function.

To simulate surfaces of varying shininess, c_s changes from place to place on the surface and D_s must be normalized. In equation (26) D_s is normalized so that $D_s(0) = 1$. If c_s varies across the surface, a constant normalizing factor must be used that is based on the minimum value of c_s over the surface:

$$c_s = c_{min} + (1 - c_{min})\ t(u,v) \qquad (33)$$

where t(u,v) is the texture value. The texture-modulated distribution function is:

$$D_s = \left[\frac{c_{min}\ c_s}{\cos^2 \beta\ (c_s - 1) + 1} \right]^2 \qquad (34)$$

The second factor in the specular reflection model measures the degree to which the facets shadow each other and is called the "geometric attenuation factor", G. G ranges in value from 0 to 1 and represents the proportion of light from the source that reaches the viewer after the shadowing takes place. An assumption is made that the microfacets are V-shaped grooves with the sides at equal but opposite angles from the average surface normal. Only grooves where one of the sides has a local normal in the direction of $\overline{H}$ contribute to the highlight. Three cases may arise for different positions of the light source and viewer; these are illustrated in Figure 3.6. Note that $\overline{L}$ and $\overline{V}$ do not necessarily lie in the plane of the figure which contains $\bar{H}$ and $\bar{N}$. For case (a) of Figure 3.6, G is 1 since the light rays are not blocked by other surface facets. To compute G for cases (b) and (c) the proportion of the facet contributing to the reflection must be calculated. This is the ratio 1 - (m/k) as shown in Figure 3.7. By projecting the vector $\bar{V}$ or the vector $\bar{L}$ onto the plane containing $\bar{N}$ and $\bar{H}$, the problem is reduced to two dimensions. Applying the law of sines and several trigonometric identities, the ratio is determined for cases (b) and (c) in terms of the vectors $\bar{N}$, $\bar{H}$, $\bar{V}$, and $\bar{L}$:

$$G_b = 1 - m/k = \frac{2(\bar{N}\cdot\bar{H})\ (\bar{N}\cdot\bar{V})}{(\bar{V}\cdot\bar{H})} \tag{35}$$

$$G_c = 1 - m/k = \frac{2(\bar{N}\cdot\bar{H})\ (\bar{N}\cdot\bar{L})}{(\bar{L}\cdot\bar{H})} = \frac{2(\bar{N}\cdot\bar{H})\ (\bar{N}\cdot\bar{L})}{(\bar{V}\cdot\bar{H})} \tag{36}$$

The value of G will be the minimum of G_a, G_b and G_c.

The next factor in the shading model is the Fresnel reflection, F, which determines the actual amount of incident light reflected from a facet as opposed to being absorbed. F is a function of the index of refraction, r, of the substance and the angle of incidence, θ, which is defined in this case as

$$\theta = \cos^{-1} (\bar{L}\cdot\bar{H}) = \cos^{-1} (\bar{V}\cdot\bar{H}). \tag{37}$$

Thus the Fresnel function is given by

$$F = 0.5 \left[\frac{\sin^2 (\theta - \gamma)}{\sin^2 (\theta + \gamma)} + \frac{\tan^2 (\theta - \gamma)}{\tan^2 (\theta + \gamma)} \right] \tag{38}$$

where

$$\sin \theta = \frac{\sin \gamma}{r} .$$

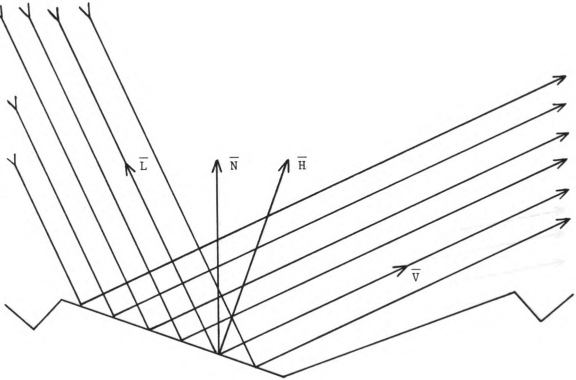

Figure 3.6a: Positions of light source and viewer with no interference [3]

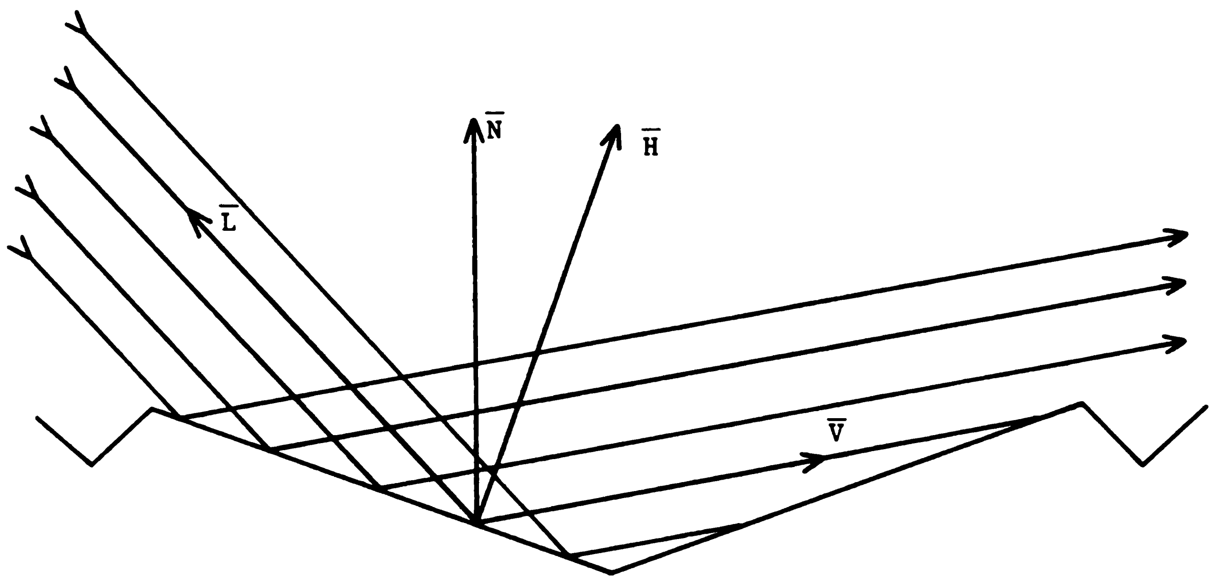

Figure 3.6b: Positions of light source and viewer where some of the reflected light is intercepted [3]

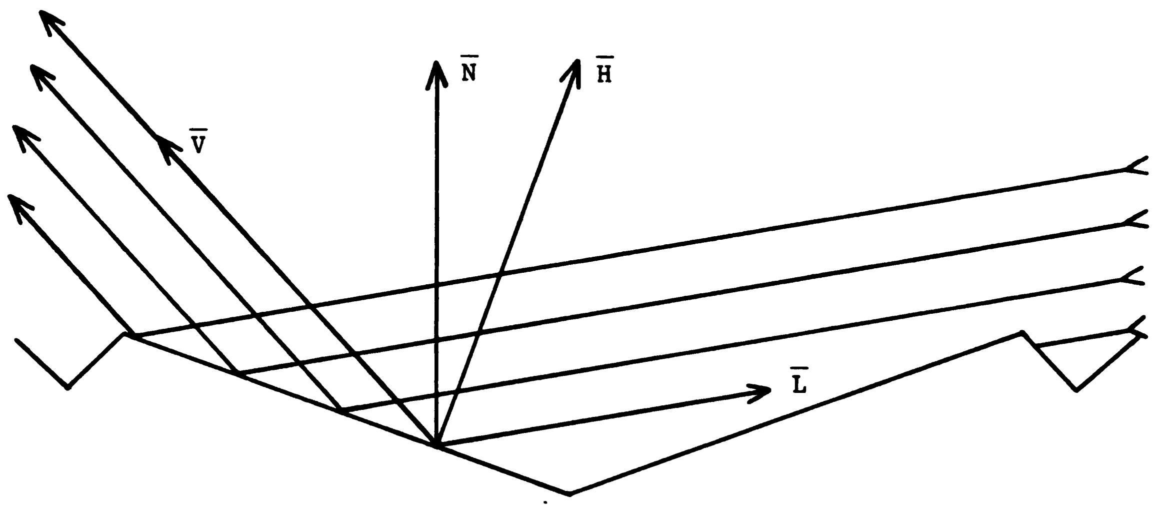

Figure 3.6c: Positions of light source and viewer where some of the incident light is masked off [3]

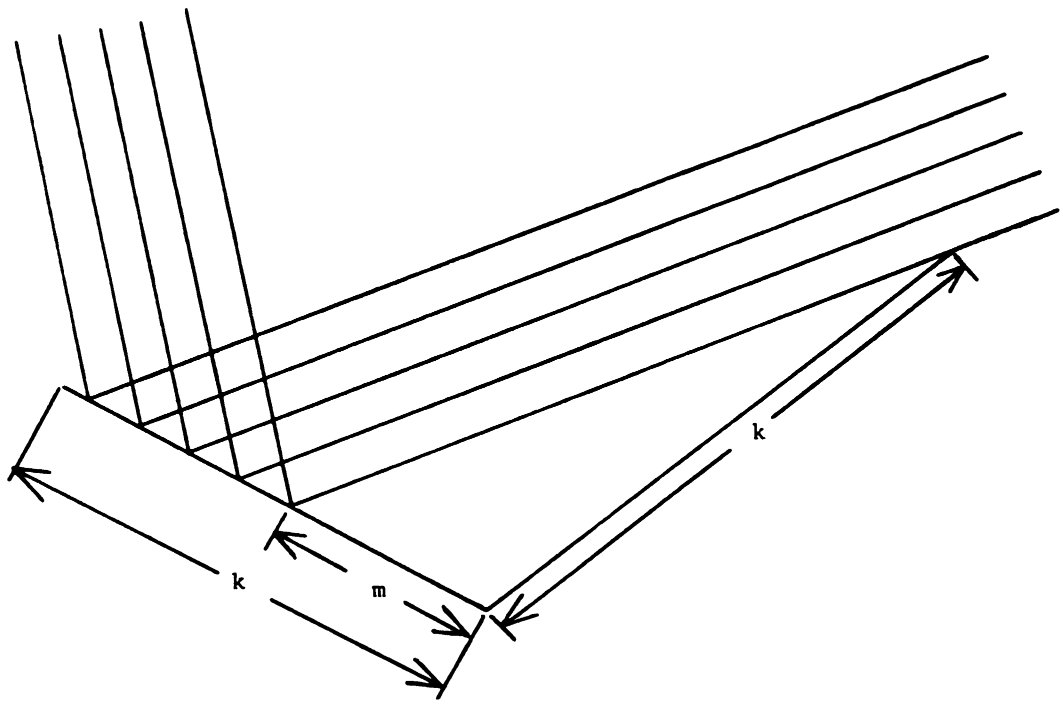

Figure 3.7: Proportion of facet contributing to the reflected light is 1 - (m/k) [3]

For metallic substances r will be large in value and $F(\theta,r)$ is nearly constant at 1; for nonmetallic substances r is small and F has an exponential appearance beginning at 0 for $\theta = 0$, and reaching 1 at $\theta = 90°$. If the light source and the viewer are assumed infinitely far away, the light rays reaching the viewer will be parallel and $\bar{L}$ and $\bar{V}$ will be constant vectors. This means that the calculations of the directions of $\bar{L}$, $\bar{V}$ and $\bar{H}$ and of $(\bar{V}\cdot\bar{H})$ need to be performed only once per change in light source direction. Using some trigonometric identities, F, too, only needs to be calculated once using the equation

$$F = \frac{(g - j)^2}{(g + j)^2} \left[1 + \frac{(j\ (g + j) - 1)^2}{(j\ (g - j) + 1)^2} \right] \tag{39}$$

where

$$j = (\bar{V}\cdot\bar{H}) \quad \text{and} \quad g = (r^2 + j^2 - 1)^{0.5}.$$

This helps reduce the computation time.

The final factor in the shading model is the division by $(\bar{N}\cdot\bar{V})$. Since the viewer sees more of the surface when it is tilted, more facets with local normals in the $\bar{H}$ direction will contribute to the intensity of the specular reflection. The increase in area seen is proportional to the cosine of the angle between the average surface normal and the line of sight, thus explaining the presence of this term. Combining this term with the computation of G, it is possible to avoid a division by zero by making some comparisons to find the minimum of G_a, G_b and G_c before doing the divisions:

If $(\bar{N}\cdot\bar{V}) < (\bar{N}\cdot\bar{L})$ then

If $2(\bar{N}\cdot\bar{V})\ (\bar{N}\cdot\bar{H}) < (\bar{V}\cdot\bar{H})$ then

$G := 2(\overline{N}\cdot\overline{H}) \;/\; (\overline{V}\cdot\overline{H})$

else $G := 1 \;/\; (\overline{N}\cdot\overline{V})$

else

If $2(\overline{N}\cdot\overline{L})\;(\overline{N}\cdot\overline{H}) < (\overline{V}\cdot\overline{H})$ then

$G := 2(\overline{N}\cdot\overline{H})\;(\overline{N}\cdot\overline{L}) \;/ (\overline{V}\cdot\overline{H})\;(\overline{N}\cdot\overline{V})$

else $G := 1 \;/\; (\overline{N}\cdot\overline{V})$

This also helps speed up the computation of the highlight function.

Comparing this highlight function to the one Phong used, Blinn notes that for small angles of incidence, the two are very similar. However, the intensity of the highlight and its direction differ for large Θ, thus the differences are most noticeable for edge-lit objects. Also, Phong's model does not simulate nonmetallic objects as well as Blinn's does. Because D_3 is easier to compute than D_1, the savings in computation time offsets the extra time required to generate G and F so Blinn claims there is no overall increase in computation time yet the degree of realism is increased.

The next section presents an algorithm which uses the method of half-tones described in Chapter II to generate shaded images.

3.4 Newell-Sancha Half-tone Shading

Although the authors claim this is a half-tone algorithm [23,24], M. E. Newell, R. G. Newell and T. L. Sancha do not disclose the specifics on how the half-tones are implemented. However, the algorithm, published in 1972, is significant because it is one of the early attempts at simulating transparent objects. The algorithm also uses a different method to generate the information about the objects being modeled; images are created by calculating the shading values per polygon in order of decreasing distance from the viewer instead of on a scan line basis as with the previous three algorithms.

The basic idea behind the Newell-Sancha hidden-surface algorithm is to order the polygons or patches in order of decreasing distance from the viewplane then to paint the object face by face, overlapping any existing faces thereby covering hidden surfaces. If conflicts arise where the faces cannot be properly placed in order, perhaps due to cyclical obscurings or intersections of faces, faces are split to attempt to resolve the problems, thereby increasing the total number of faces composing the object. The faces are painted into the image memory, which they call a screen map, then the information is processed again according to scan lines before being displayed.

The shading function is performed during the painting of the faces to the image memory. The model used has a diffuse, an ambient and a specular component. The diffuse component is the cosine of the incident angle, Θ, raised to the power s and multiplied by a coefficient, C_d, which is the intensity range. s is an arbitrary power; when $s = 1$, the function simulates diffuse reflection. As s increases, the object appears darker except for a few faces which appear at the brightest intensity; this gives the effect of a shiny black surface. The ambient component, S_a, is a constant representing the ambient level of lighting. The specular component is used in particular to simulate longitudinal reflection patterns of bottles or other objects of revolution and has the form of the sine of the incident angle raised to a high power, i, and multiplied by the specular intensity range, C_s. Thus, the shading equation becomes

$$S = C_d \cos^s \theta + S_a + C_s \sin^i \theta. \tag{40}$$

Transparent materials can be simulated by slightly altering the painting routine. When the new surface being painted is transparent, instead of simply replacing the previous shading value of the old face covered by this new surface with the new face's value, the shading value stored is a combination of both the old, S_o, and new, S_n, shading values. A comparison is made between the old and new values and depending on the outcome, the resulting shading value is a weighted sum of these two as follows:

$$\text{If } S_n < S_o;\ S = w\, S_n + (1 - w)\, S_o \tag{41}$$

$$\text{If } S_n > S_o;\ S = S_n \tag{42}$$

where w is a weighting factor.

Newell, et al. claim that these functions are not an attempt to simulate the real world but can considerably enhance the appearance of the images produced. Although the effects of transparency are simulated, no provision is made for the effects of refraction which are apparent through many transparent objects. Also, they feel that "the time taken to produce an image precludes the possibility of using shaded pictures in a truly interactive way". Their second paper, [23], presents ten figures of images produced with their algorithm. They compare the complexity of each figure and the time required to produce the image. The entire algorithm is broken into four parts with the third being the writing of the fragments to the image memory, which includes the performance of the shading function. They list times for this part ranging from 1.4 to 33.4 seconds. However, it is not known exactly how much of these times was spent on the shading alone. Regardless, pictures of images generated using this method which were included with the paper greatly exhibited contouring effects.

The next section presents an algorithm to generate images using the method of ray tracing described in the last chapter. It simulates the effects of both transparency and refraction as well as shadows and light reflected from object to object within a scene.

3.5 Whitted Ray-tracing Shading

In this algorithm [35], published in 1980, Whitted incorporates a technique known as ray tracing. The algorithm is based on a hidden-surface algorithm that produces "trees" of global information for each pixel of the display. The trees are formed by tracing light rays from the viewer to the first surface encountered, then tracing the components of reflection and refraction from the first surface to the next until reaching a light source. Shading is then performed by traversing the tree to determine the light intensity received by the viewer.

The hidden-surface algorithm does not perform the usual functions of clipping and removal of faces hidden to the viewer; these may be visible as reflections on other objects within view of the observer. Rays are traced from the viewer to the first surface to the next surface and onto the last surface before reaching the light source; therefore, objects not included in the view may affect the lighting of visible objects.

The ray tracing is performed by calculating the intersection of an incident ray of light with a reflecting surface. Since the rays are traced from the viewer, the direction of the incident ray, $\overline{L}$, coincides with the direction of the viewer, $\overline{V}$, for the first surface. The incident ray is broken into two components: the reflected light in the direction of $\overline{R}$ and the light transmitted through the surface in the direction of $\overline{T}$. The $\overline{R}$ direction follows the rule that the angle of incidence equals the angle of reflection as established in previous sections. The direction of the transmitted light, $\overline{T}$, obeys Snell's law of refraction. Thus, $\overline{R}$ and $\overline{T}$ are functions of $\overline{N}$ and $\overline{V}$ given by

$$\overline{V}' = \frac{\overline{V}}{|\overline{V}\cdot\overline{N}|} \tag{43}$$

$$\overline{R} = \overline{V}' + 2\overline{N} \tag{44}$$

$$\overline{T} = k_f\ (\overline{N} + \overline{V}') - \overline{N} \tag{45}$$

where

$$k_f = (k_n^2\ |\overline{V}'|^2 - |\overline{V}' + \overline{N}|^2)^{-0.5}$$

and k_n = the index of refraction. These equations assume that $(\overline{V}\cdot\overline{N})$ is less than zero so $\overline{N}$ must point to the side of the surface that from which the ray is incident. Likewise, k_n must be adjusted to account for the change. If the denominator of k_f is imaginary, $\overline{T}$ is assumed to be zero because of total internal reflection. The intersection process is performed recursively until all branches of the tree are terminated. These relationships are pictured in Figure 3.8. Figure 3.9 shows how the rays are traced from surface to surface with Figure 3.10 showing the tree formed from the components of light reaching the viewer from point P in Figure 3.9.

The shading model used by Whitted is dependent on the vectors generated by the hidden-surface removal algorithm, namely, $\overline{N}$, $\overline{R}$ and $\overline{T}$. It includes a

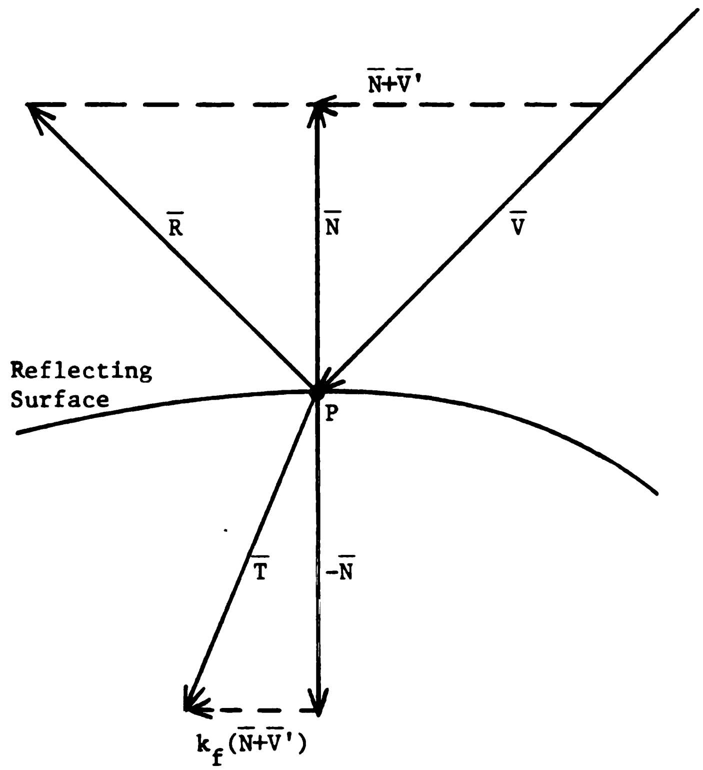

Figure 3.8: Geometry of reflection model for Whitted's algorithm [35]

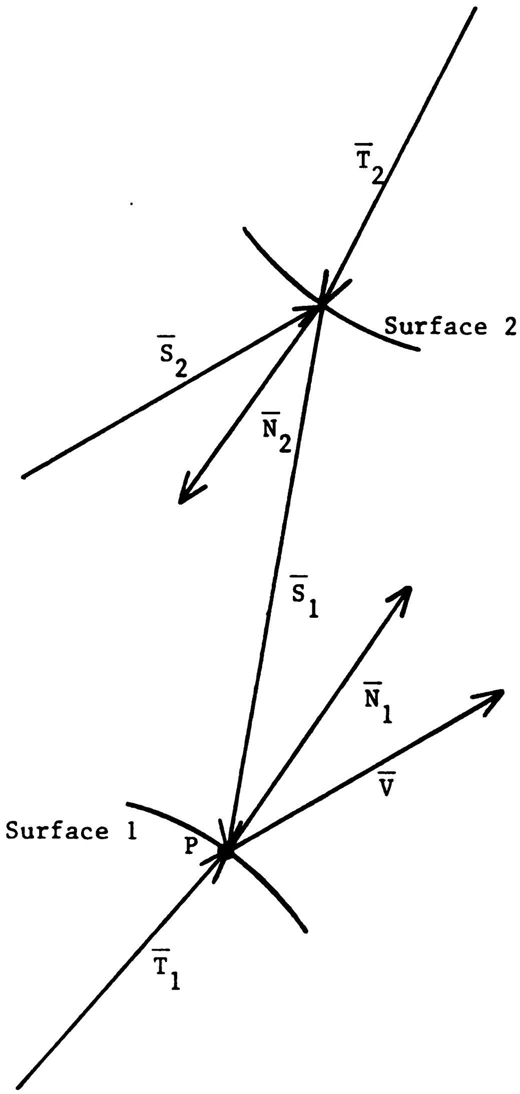

Figure 3.9: Paths of reflected light that reach the viewer [35]

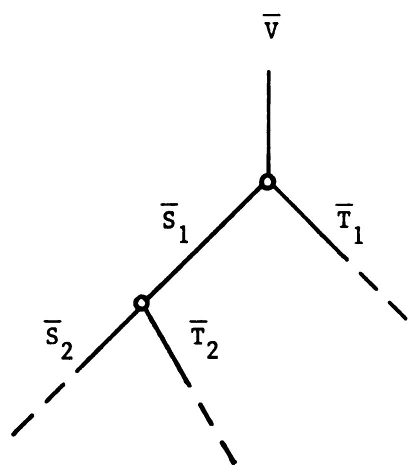

Figure 3.10: Tree formed from components of light reaching the viewer from point P of Figure 3.9 [35]

constant representing the ambient reflection, S_a, and terms for the diffuse and specular reflections and the transmitted intensity. The diffuse component ideally would include contributions reflected from nearby objects as well as light from all of the sources. These contributions would simply add together to form the total diffuse reflection. However, the computation required to sum components from other objects in the scene would be too extensive so a diffuse component similar to that used by Phong is implemented which only accounts for the sources. Assuming that $\overline{N}$ and $\overline{L}$ are normalized, the diffuse component becomes their dot product multiplied by the diffuse reflection coefficient, C_d. The complete equation for the shading model is

$$S = S_a + C_d\,(\overline{N}\cdot\overline{L}) + C_s\,R + C_t\,T \qquad (46)$$

where C_s and C_t are the specular reflection and transmission coefficients, respectively, R is the intensity of light incident from the $\overline{R}$ direction and T is the intensity of light from the $\overline{T}$ direction. Although the coefficients C_s and C_t were held constant to generate the pictures included in Whitted's paper, more accuracy is obtained by making them functions incorporating the Fresnel reflection law; the coefficients would then vary as a function of the incident angle in a manner depending on the properties of the surface being displayed. Instead, they must be chosen to correspond to physically reasonable values to generate realistic pictures. As the tree from the hidden-surface algorithm is traversed, shading intensities are calculated at each node using this model. The intensities are then linearly attenuated as a function of the distance between the nodes. The linear function is used because it provides a good approximation of the effects of distance; for non-planar surfaces the square-law approximation does not apply.

When modeling surface properties, if the value of C_s is decreased and that of C_d increased, the surface will appear less glossy but the highlight will not spread out realisticly as it did for Phong's model whenever the specular exponent was reduced to simulate less smooth surfaces. To improve the highlights generated, a random perturbation is added to the surface normal to simulate the randomly oriented microfacets of a rough surface, thereby assuming a surface modeled in the manner described by Blinn in section 3.3. If the surface is smooth and shiny, the perturbation has a small variance; rough surfaces necessitate using larger variances. This method will also give transparent objects a frosted appearance by using larger variances. But because this method requires a great amount of extra computation, it is avoided whenever possible. One such case is when specular reflections are caused directly by a point light source, where Phong's model of specular reflections can effectively be used at the point of reflection.

Shadows may be simulated using this algorithm by extending the trees from the hidden-surface algorithm to include rays associated with light sources at each node. If one of these rays intersects a surface before it reaches the source, the point of intersection represented by the node lies in shadow with respect to that light source. This source will not contribute to that point's diffuse reflection, thus creating a shadow.

The pictures included in Whitted's paper were created using a VAX-11/780 and are probably the best generated by any of the algorithms thus far. However, they required between 44 and 122 minutes to be processed. For simple pictures 12 percent of the processing time is attributed to just the shading with the majority of the time needed to compute the intersections in the hidden-surface removal algorithm. Some shortcomings of the algorithm is that it does not provide for diffuse reflections from distributed light sources, nor do specular reflections degrade gracefully as surfaces get less glossy.

While this algorithm was able to shade objects modeled by both polygon-mesh and curved-patch representations, the next section presents an algorithm designed specifically for curved patches, particularily a class known as the bicubic patch.

3.6 Catmull Bivariate Surface-patch Shading

This algorithm [6,7] was developed in 1974 by Edwin Catmull. The method was primarily developed for objects modeled using a class of curved patches called bicubic patches but is not limited to them alone and can be applied to other kinds of surfaces as well. The algorithm is based upon a subdivision procedure which divides the patches into subpatches. After the subdivisions are accomplished, hidden-surface removal and shading functions are performed. Four different methods to determine the shading values are discussed.

Catmull feels that bicubic patches are better for modeling objects to be displayed. The polygon-mesh technique produces unrealistic effects such as a faceted appearance caused by contouring and jagged silhouettes formed by straight-line segments. Also, quadric curved patches do not provide enough degrees of freedom and are therefore unsuitable for modeling many objects. Bicubic patches are easily joined with slope continuity across the boundaries so they produce continuous shading and smooth silhouettes. And as mentioned, the basis of Catmull's algorithm is the subdivision algorithm and this process can be performed quickly using bicubic patches.

The subdivision algorithm divides all patches into subpatches until each subpatch's projection represents only a single pixel on the raster display. This is done by joining the midpoints of opposite sides of the patch, thus dividing it into four subpatches. Eventually the patch will appear as in Figure 3.11. Subpatches which do not cover any pixels are associated with the nearest subpatch covering a pixel or sample point. Clipping is performed during this process to determine if a subpatch will be on the screen before dividing it any further. Overall, the number of subdivisions required is slightly greater than one third of the number of pixels covered by the original complete patch. According to Catmull, the subdivision of each bicubic component requires thirty additions with values passing through four adders; it is best to have a subdivider for each of the three bicubic comoponents to work simultaneously and reduce the total execution time required.

When the subdivisions are completed, hidden surfaces are removed then a shading value is assigned to each pixel and accordingly, to each subpatch. Catmull sites four methods to determine the shading value for each pixel.

The first shading method can be any of those mentioned in this text which uses the surface normal to calculate the shading intensity. However, since the equation of the normal to a bicubic patch is a fifth degree polynomial, it is difficult to find. A fifth degree subdivision equation could be used to solve the normal equation but this is impractical so Catmull used the following method to generate the pictures in his paper.

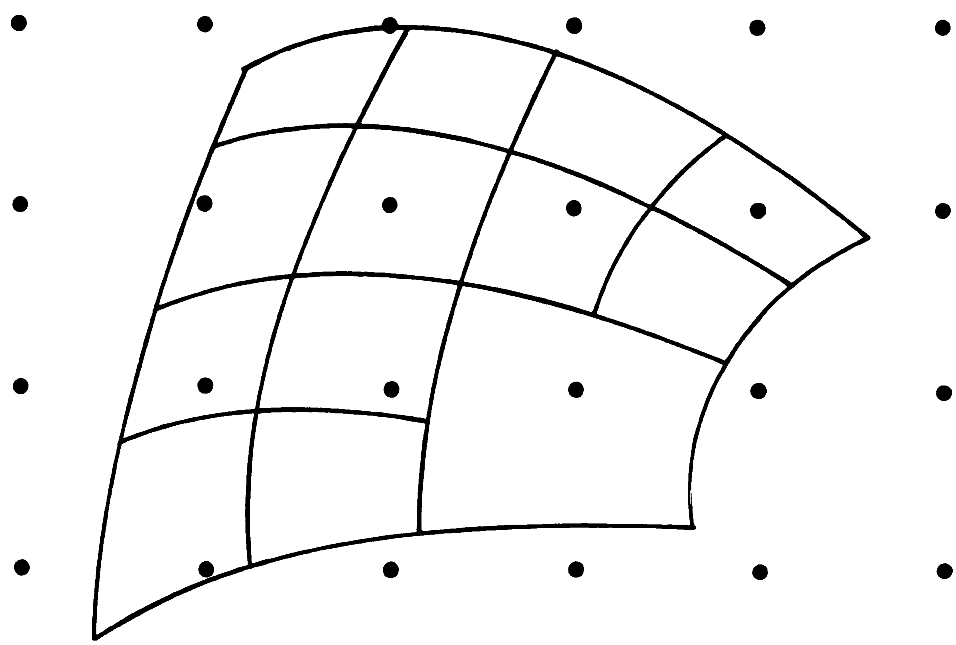

Figure 3.11: Patch subdivided so that no subpatch covers more than one sample point [7]

The normal equation is approximated with a cubic equation. Its components are then subdivided along with the components of the patch equation to obtain approximate normal equations for the subpatches. The patch and normal equations are functions of two variables, u and v. The notation for the x-component is

$$x(u,v)=U\ M_x\ V \tag{47}$$

where U and V are matrices defined as

$$U = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \tag{48}$$

$$V = \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}^T \tag{49}$$

and M_x is a four-by-four matrix of coefficients defined as

$$M_x = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{50}$$

The derivative of the x-component in the u-direction is given by

$$x_u = U'\ M_x\ V \tag{51}$$

and the derivative in the v-direction is defined as

$$x_v = U\ M_x\ V' \ . \tag{52}$$

The normal vector, $[x_n\ y_n\ z_n]$, is found by forming the cross product of the tangent to the surface in the u-direction, $[x_u\ y_u\ z_u]$, and the tangent in the v-direction, $[x_v\ y_v\ z_v]$. Thus, the normal components are defined as

$$x_n(u,v) = U' \, M_y \, V \, U \, M_z \, V' - U \, M_y \, V' \, U' \, M_z \, V \, , \tag{53}$$

$$y_n(u,v) = U' \, M_z \, V \, U \, M_x \, V' - U \, M_z \, V' \, U' \, M_x \, V \tag{54}$$

and

$$z_n(u,v) = U' \, M_x \, V \, U \, M_y \, V' - U \, M_x \, V' \, U' \, M_y \, V \, . \tag{55}$$

To find the approximate components of the normal requires a similar matrix multiplication for each component as shown for only the x-component:

$$x = C \, P_x \, C^T \tag{56}$$

where C is the Coons matrix defined as

$$C = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{57}$$

and

$$P_x = \begin{bmatrix} x_n(0,0) & x_n(0,1) & \frac{dx_n(0,0)}{dv} & \frac{dx_n(0,1)}{dv} \\ x_n(1,0) & x_n(1,1) & \frac{dx_n(1,0)}{dv} & \frac{dx_n(1,1)}{dv} \\ \frac{dx_n(0,0)}{du} & \frac{dx_n(0,1)}{du} & \frac{dx_n^2(0,0)}{dudv} & \frac{dx_n^2(0,1)}{dvdv} \\ \frac{dx_n(1,0)}{du} & \frac{dx_n(1,1)}{du} & \frac{dx_n^2(1,0)}{dudv} & \frac{dx_n^2(1,1)}{dvdv} \end{bmatrix} \tag{58}$$

where

$$dx_n(u,v)/du = U'' M_y\ V\ U\ M_z\ V' + U'' M_y\ V\ U'\ M_z\ V'$$
$$- U'\ M_y\ V'\ U'\ M_z\ V - U\ M_y\ V'\ U''\ M_z\ V \qquad (59)$$

$$dx_n(u,v)/dv = U'\ M_y\ V'\ U\ M_z\ V' + U'\ M_y\ V\ U\ M_z\ V''$$
$$- U\ M_y\ V''\ U'\ M_z\ V - U\ M_y\ V'\ U'\ M_z\ V' \qquad (60)$$

$$d^2x_n(u,v)/dudv = U''\ M_y\ V'\ U\ M_z\ V' + U''\ M_y\ V\ U\ M_z\ V''$$
$$+ U'\ M_y\ V'\ U'\ M_z\ V' + U'\ M_y\ V\ U'\ M_z\ V''$$
$$- U'\ M_y\ V''\ U'\ M_z\ V - U'\ M_y\ V'\ U'\ M_z\ V'$$
$$- U\ M_y\ V''\ U''\ M_z\ V - U\ M_y\ V'\ U''\ M_z\ V'. \qquad (61)$$

The equations for the y- and z-components are similar and can be found by comparing equation (52) with equations (53) and (54) to derive similar equations as (55) and (57) through (60) for the other components of the approximated normal vector.

A second method to find the shading values is to use an intensity function. This associates numbers with the pixels as derived by a function. The function could be based on anything such as pressure, strain, height, density, artistic whim, etc. Some checks must be used to stay within the bounds of the display as described in [7].

The third method is to map a picture to the surface. This is done by forming a one-to-one correspondence between either the pixels covered by the patches and the intensities of the picture or between areas of the picture and entire patches. The intensity of the picture could be a function of u and v as is the patch. The function or area is divided as the patch is to maintain the correspondences. One problem that may occur is a sampling problem. If the the picture contains more intensity values than the patch has pixels, all of the information in the picture will not be displayed on the patch. In the area-mapping method the sampling problem is less noticable because the average intensity of the area is mapped to the subpatch, however, the problem is not eliminated.

The forth and final method to calculate the shading values is to modify the intensity for effects such as shadows or transparency. Another method must be used to find the initial intensities then this method adjusts the values. A method similar to that of Newell, et al. is used for displaying transparent objects. Shadows are created by using "shadow-patches" formed from the silhouette of an object from the point of view of the light. After determining which portions of the object are behind these patches, the shading values of the shadowed portions are attenuated. One problem is that this method merely diminishes highlights rather than eliminating them.

Several pictures generated by this method were included in both of Catmull's papers. Times to produce the pictures ranged from 115 seconds to 15 minutes. It was not disclosed exactly what portion of the time was required to generate only the shading values. Despite the complexity of many of the pictures, they were very realistic.

The next section presents a shading model that utilizes the different characteristics of various materials to display how they reflect light more accurately to achieve more realistic images.

3.7 Cook-Torrance Reflective Model for Shading

This model [10], published by Robert L. Cook and Kenneth E. Torrance in 1981, is a reflectance model for shading computer images with emphasis on generating color images. It is based on geometrical optics like most of the previous algorithms but is applicable to a broader range of materials, surfaces and lighting situations. Here, the intensity of the reflected light is determined by the intensity and size of the source, and by the object's reflecting ability and surface properties. The spectral highlights are determined by the spectral composition of the source and the wavelength-selective reflection property of the surface.

Like the previous algorithms, this shading model uses the vectors $\overline{N}$, $\overline{V}$, $\overline{L}$, and $\overline{H}$, where all are normalized and defined as in previous sections and shown again in Figure 3.12. Note that the angles between $\overline{H}$ and the two vectors $\overline{V}$ and $\overline{L}$ are defined as ξ, and that these three vectors lie in the same plane; $\overline{N}$ is not necessarily contained in that same plane. This model also derives its specular component in a similar manner as Blinn's model and depends on the description of a surface as being composed of randomly oriented microfacets.

Unlike any previous algorithm, this model determines the energy of the incident light as expressed as energy per unit time and per unit area of the surface. Most non-mirror surfaces reflect the incident beam over a wide range of angles, thus, the reflected intensity in any given direction depends on the incident energy and the incident intensity. The intensity of the incident light is expressed in a similar manner to the energy but is per unit projected area and per unit solid angle. The energy of an incident beam of light, E_i, is given by

$$E_i = I_i\ (\overline{N}\cdot\overline{L})\ \omega_i \qquad (62)$$

where I_i is the average intensity of the incident beam and ω_i is the solid

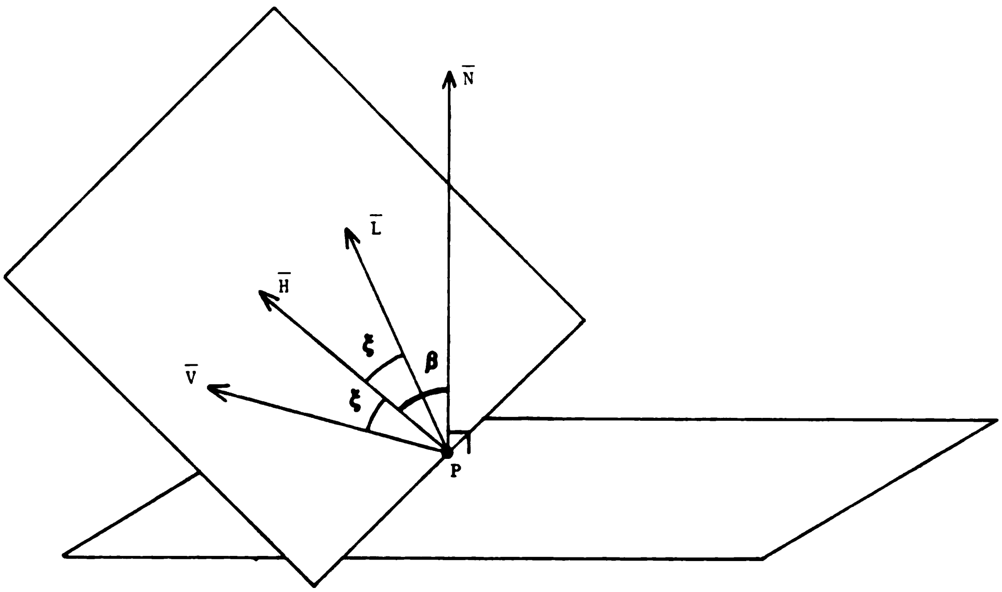

Figure 3.12: Geometry of reflection model for Cook and Torrance's algorithm [10]

angle of the beam.

Unless the surface is a perfect mirror, the incident light will be reflected over a wide range of angles. Each light source is associated with a bidirectional reflectance, R_b, which is the ratio of the reflected intensity in a specified direction to the incident energy from another direction both within a small solid angle. R_b is given as

$$R_b = \frac{S}{E_i} \tag{63}$$

where S is the reflected intensity or shading value that the viewer sees from each light source and is given by

$$S = R_b\ E_i = R_b\ I_i\ (\bar{N}\cdot\bar{L})\ \omega_i \tag{64}$$

R_b is a linear combination of two components. The diffuse component, R_d, is from either internal scattering where the incident light penetrates the surface or from multiple surface reflections such as from a rough surface. The specular component, R_s, is from light that is reflected at the surface of the object. If the object being modeled is not composed of a homogeneous material, these two components may have different colors. If d is the fraction of reflectance that is diffuse, (1 − d) is the fraction of reflectance that is specular and R_b is given by

$$R_b = d\ R_d + (1 - d)\ R_s. \tag{65}$$

The light reflected toward the viewer from ambient light, when integrated over the entire hemisphere of illuminating angles, can be defined by a hemispherical-directional reflectance, R_a, which is an integral of R_b and, therefore, is a linear combination of R_d and R_s. R_a is assumed independent of the direction of $\overline{V}$ and the ambient light is assumed uniformly incident. The reflected intensity due only to the ambient light, S_a, is given by

$$S_a = R_a \, I_{i,a} \, f \tag{66}$$

where $I_{i,a}$ is the intensity of the incident ambient light and f is the fraction of the illuminating hemisphere that is not blocked by other objects, given by

$$f = \frac{1}{\pi} \int (\overline{N} \cdot \overline{L}) \, \omega_i \tag{67}$$

where the integration is performed over the unblocked portion of the illuminating hemisphere.

Hence, the total intensity of the light observed is the sum of the reflected intensities from all of the light sources plus any reflected intensity from the ambient light. With f = 1, the shading model is defined as

$$S = I_{i,a} \, R_a + \sum_j I_{i,j} \, (\overline{N} \cdot \overline{L}) \, \omega_{i,j} \, (d \, R_d + (1 - d) \, R_s). \tag{68}$$

This equation takes into account light sources with different intensities and projected areas. For instance, if two incident beams have the same intensity and incident angle but one has twice the solid angle as the other, the first will make the surface appear twice as bright as the second will. Similarily, if an incident beam has twice the intensity but the same incident angle and solid angle as a second beam, the first will make the surface appear twice as bright. However, the model depends on several variables. For example, the intensities depend on the wavelength of the light, d depends on the material composing the object and the reflectances depend on these variables in addition to the reflection geometry and the surface roughness.

Directional dependence only affects the specular component, R_s, since it relies on the location of the viewer. Similar to Blinn's model, this component can be defined as

$$R_s = \frac{F\ D\ G}{\pi\ (\overline{N}\cdot\overline{L})\ (\overline{N}\cdot\overline{V})} \quad . \tag{69}$$

The terms G, D and F have the same meaning as they did in Blinn's model but their equations are defined differently.

G is the geometrical attenuation factor which accounts for the shadowing and masking among the microfacets. It is defined as

$$G = \min\left\{1,\ \frac{2(\overline{N}\cdot\overline{H})\ (\overline{N}\cdot\overline{V})}{(\overline{V}\cdot\overline{H})},\ \frac{2(\overline{N}\cdot\overline{H})\ (\overline{N}\cdot\overline{L})}{(\overline{V}\cdot\overline{H})}\right\}. \tag{70}$$

D represents the the fraction of facets that are oriented in the direction of $\overline{H}$. Cook and Torrance consider the Gaussian model proposed by Blinn in equation (24), but also a model developed by Petr Beckmann and Andre Spizzichino for rough surfaces

$$D = \frac{1}{m^2\ \cos^4\ \beta}\ e^{-[\tan^2\ \beta\ /\ m^2\ ]} \quad . \tag{71}$$

where m is the root-mean-square slope of the facets, which controls the spread of the specular component. If m is small, the surface will appear smooth and the distribution of the specular component from the facets will be highly directional around the vector $\overline{H}$. If m is large, rough surfaces are simulated and the specular component will be more spread out. Comparing this model to Blinn's of equation (24), the differences are very slight.

The advantage of this function is that it gives the absolute magnitude of the reflection without introducing any arbitrary constants; however, it requires more computation.

When objects have two or more different surfaces of different roughness, the distributive functions will have different slopes m. The overall D can be expressed as a weighted sum of the respective distribution functions:

$$D = \sum_j w_j \, D(m_j) \tag{72}$$

where w_j is the weight of the j^{th} distribution function. The sum of all the weights must equal 1.

F is the Fresnel term which describes how light is reflected from each microfacet. It is a function of the incident angle, ξ, and the wavelength of the light, λ. F, as well as the other reflectances R_d and R_a, may be obtained from the reflectance spectra for the material. This information has been measured for many materials, usually for illumination at normal incidence, and tabulated. The measurements were made for only a few wavelengths so the values may need to be interpolated. By multiplying the reflectance spectra for the surface by the spectral energy distribution of the incident light, the spectral energy distribution of the reflected light is obtained. Since F and R_d also vary with the geometry of the reflection, R_d is taken to be the bidirectional reflectance for illumination in the direction normal to the reflecting surface, whereas F's directional dependence leads to a color shift when the directions of incident and reflected light are near grazing.

The Fresnel equation expresses the reflection in terms of the index of refraction, r, and the extinction coefficient of the surface, C_e, and the angle of illumination of the microfacets, ξ. If r and C_e are known, the Fresnel equation is used to find the spectral and angular dependence of F. If not, r is estimated by setting $C_e = 0$ using an equation similar to Blinn's equation (39) except here is has an additional factor of 1/2:

$$F = \frac{1}{2}\frac{(g - j)^2}{(g + j)^2}\left[1 + \frac{(j\ (g + j) - 1)^2}{(j\ (g - j) + 1)^2}\right] \tag{73}$$

where

$$j = (\bar{V}\cdot\bar{H}) \quad \text{and} \quad g = (r^2 + j^2 - 1)^{0.5}.$$

This dependence of reflectance on wavelength and incident angle implies that the color of the reflected light changes with the angle ξ. The computation of this color shift is excessive so it is approximated from the spectral energy distribution, thereby approximating the RGB values for the color. Since all of the other algorithms have only dealt with intensities for achromatic displays, the procedure for calculating these values is not discussed. However, some important conclusions were drawn concerning the realism of computer-generated images.

One conclusion is that nonhomogeneous materials may have specular and diffuse components of different colors. Plastics are one such material. The color of the specular component, which is reflected from the surface, is only slightly altered by the color of the incident light, depending upon the reflectance of the surface material. The diffuse component is of the color of the plastic alone. This is not the case with metallic objects. Reflections from metals occur almost completely at the surface. The specular component is still only slightly altered by the color of the light source but if the surface is smooth, there may be barely any diffuse reflection. According to Cook and Torrance, most of the other algorithms create images of objects that appear to be made of plastic. Their model increases the realism of the images by simulating other materials.

The next section presents a new method for modeling objects, particularily those that occur in nature such as mountain ranges, continent outlines and other objects with random surfaces.

3.8 Fournier-Fussell-Carpenter Fractal-surface Shading

In 1982 Alain Fournier, Don Fussell and Loren Carpenter published a detailed discussion [14] of a new method to model objects, particularly for those that occur naturally due to the randomness of their surfaces. The method was derived from fractal mathematics techniques which were largely developed by Benoit Mandelbrot in the late 1960's. One of the methods presented by Fournier, et al. was implemented by Stephen L. Stepoway, David L. Wells and Gerald R. Kane in a multiprocessor architecture in a paper published in 1984 [30].

Fractal mathematics techniques are very useful to model non-deterministic phenomena such as terrains, smoke and clouds. Traditional methods for modeling such objects; i.e., polygon-mesh or curved-patch techniques, do not generate realistic images of these types of objects. Even so, extremely large numbers of polygons or patches are necessary to reproduce the natural features. The use of texture-mapping, a technique which was discussed in both Chapter II and Section 3.6, is more effective for these types of objects but even this tends to have a repetitive regularity not characteristic of the real objects. Texture-mapping also has limitations to the detail that can be conveyed as the viewer is brought closer to the surface. Conversely, fractal mathematics can create the resolution required of a particular scene by continuing the texturing process to the extent necessary. The nature of the process is random enough to effectively model smoke, trees, rocks and other such objects yet can be controlled enough to follow a basic outline, such as the coast of a continent.

The algorithm is similar to that presented by Catmull in Section 3.6. Both relied on a subdivision process to break the model into pieces which covered only a single pixel on the raster display then calculated surface normals to be used in the shading computation. In this algorithm the object is modeled at the start using a polygon-mesh technique where the polygons are all triangles. The description is coarse requiring just a few dozen triangles to give a general outline of the overall shape of the object; the algorithm provides the texture definition. The more triangles used at this phase, the more specific and controlled will be the object's definition.

As stated the subdivision algorithm breaks the triangles into smaller triangles continuously until each triangle corresponds to only one pixel. The new triangles are somewhat noncoplanar. The midpoints of the edges forming a triangle under consideration are moved a random distance from the edges in a direction related to the normal of the triangle. These new midpoints are joined to form a new triangle. Each edge of the new triangle is connected to a corresponding vertex of the original triangle to form three more triangles. In this manner each triangle is divided into four smaller triangles as illustrated in Figure 3.13. Some problems may arise if

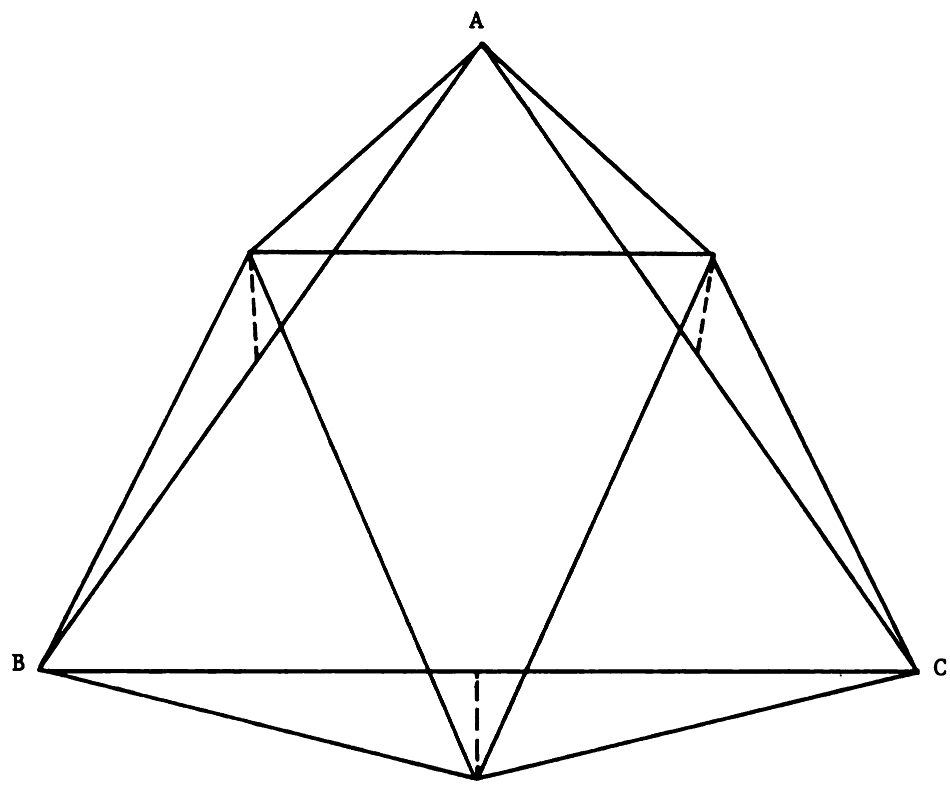

Figure 3.13: Subdivision of a triangle [30]

adjacent triangles are processed simultaneously. If their shared midpoint is not moved to the same position, the surfaces will not meet at the common edge. This can be corrected by sharing information between the processors about the midpoint's new position so both will use the same placement. Another solution is to move the midpoint in the direction of the average normal of the two triangles sharing the edge; still the processors must share information about the midpoint's exact displacement along the average normal.

Once the subdivision process is completed the shading values are determined from the shade of the original triangle and the orientation of the normals of the new triangles. The shading values may be obtained using any of the shading models mentioned in this chapter which uses the surface normal to calculate the value.

Several pictures were included in the paper of Fournier, et al., mostly depicting terrains. The effect was very realistic. No execution times were provided for the time required to generate the images; however, Stepoway, et al., claim that fractal surfaces cannot be used in real-time applications because of the complexity of generating images. Nevertheless, the realism is impressive and fractal techniques have been used in movies such as "Star Trek II", "Vol Libre" by Carpenter and "Peak" by Mark Snilily.

Overall, this chapter has presented a wide range of shading models and techniques to achieve realistic, computer-generated images. The next chapter will examine the time-space complexity of the algorithms and discuss the advantages and disadvantages of each.

CHAPTER IV

COMPARISON OF THE ALGORITHMS

There are different criteria by which to judge the shading algorithms presented in Chapter III. One such standard is the realism of the images generated by the algorithm. Many of the algorithms build upon previous work in an attempt to enhance the realism of the final images. The shading process is closely related to the modeling technique and the hidden-surface removal algorithm implemented. The modeling technique provides the basic data about the surfaces of the object. The manner in which the hidden-surface removal algorithm processes this information determines how it will be available for the shading algorithm and its reflectance model; most shading algorithms prefer a hidden-surface removal algorithm that processes the information scan-line by scan-line. Increased realism of the shaded images results from improvements in any of these three processes.

It should be noted that because the images are produced from a numerical description of the objects, only approximate images of the subject matter will be attained. There are many factors which affect the realism of the images generated by the computer; such as the resolution of the screen, the number of intensity levels obtainable, the processing power available, and the lack of a visual feedback system, to name a few. Therefore, precise duplicates of the objects are not possible and a degree of desired realism must be defined.

Generally, the algorithms presented in Chapter III are based on two modeling techniques, each with a modified version as well, and various reflectance models. Gouraud's algorithm was presented for the polygon-mesh technique and relies on the data being processed in scan-line order. The polygon-mesh technique uses low-order equations which are easy to solve, and it does not restrict the class of objects that can be modeled. Though Gouraud's algorithm improves upon the constant shading algorithm by reducing contouring, it still exhibits Mach band effects. According to Catmull [7], Gouraud's algorithm is difficult to use to generate highlights and the shading is affected by the orientation of the polygons in the picture. This last problem is due to the viewer and light source being at the same location and causes frame discontinuities for motion pictures. Despite these problems, Gouraud's images are acceptable and the algorithm has been implemented in systems as mentioned in Chapter III.

Phong improved upon Gouraud's algorithm by maintaining shading continuity across the boundaries of the polygons. It still exhibits Mach band effects though not as noticably. However, Phong's reflectance model was based on some empirical adjustments so Blinn applied a theoretical model which portrays highlights more accurately. Though Blinn's algorithm was applied to Phong's algorithm, it is a reflectance model so it alone is not dependent on the polygon-mesh modeling technique or scan-line ordering of processing data as is Phong's algorithm. Both of these algorithms produce images of better quality than Gouraud's.

The Newell-Sancha algorithm also uses the polygon-mesh modeling technique but it requires the data to be processed by area rather than scan-line because it shades entire polygons at the same time. Since no effort was made to shade continuously across the polygons' edges, this algorithm exhibits considerable contouring and the Mach band effect; and thus, the images are not as realistic as those from the previous algorithms. Although this algorithm was an early attempt to display transparent objects, no effects of refraction are modeled.

Whitted's algorithm could be applied to either polygon-mesh or curved-patch modeling techniques. It requires that the information about the objects be processed in the form of ray-tracing trees. The algorithm accounts for light sources within the scene being displayed and reflections between objects, neither of which were modeled in the previous algorithms. The images are very realistic but require large amounts of computations, and do not account for diffuse reflections from distributed sources; also highlights do not degrade gracefully as surfaces become less glossy.

Catmull's algorithm created realistic images using a specific class of curved patches called bicubic patches. He claims polygons create silhouettes that are not smooth and quadric patches cannot model any arbitrary object. The algorithm divides the patches into subpatches no larger than the size of one pixel before performing the shading or hidden-surface removal processes; thus, the information is processed by pixels. Overall, the bicubics are high-order equations and are difficult to deal with. The subdivisions require large amounts of computations. Working at the pixel level creates aliasing problems not easily solved but eliminates the Mach band effect.

Cook and Torrance presented a reflectance model independent of both the modeling technique and the manner in which the information is processed during the hidden-surface removal procedure. It relates the brightness of the object, as well as what it is composed of, to the intensity and size of each light source. Cook and Torrance feel that all of the previous algorithms display all objects as if they were made of plastic. They use the information about the object's composition to determine exactly how light will be reflected to display more realistic images true to the material that composes the objects. They claim to do all this without increasing the overall execution time.

The final algorithm presented in Chapter III explains a new modeling technique based on fractal mathematics. It is specifically developed for natural objects because of their non-deterministic nature. Using the previous modeling techniques on objects such as smoke, clouds or mountains, which do not have regular features or simple macroscopic structures, requires excessively large numbers of primitives, either polygons or patches. Fractals provide random texture and structure to objects modeled coarsely with planar triangles. Texture-mapping attempts this effect but is too regular because it repeats the pattern and therefore, appears unnatural. Another advantage is that fractals are not defined at a predetermined level of resolution so distant and very close scenes are still quite realistic. Also, the computational effort is proportional to the complexity of the images. Fractals can model deterministic objects, too, but the computations are more demanding than any of the previous algorithms. This concludes the comparison of the algorithms presented in Chapter III based on realism and other problems, such as implementation. Unfortunately, despite its importance, realism is a very subjective criterion.

The most important criterion is the speed of execution so that the images are able to respond to inputs on a real-time basis. If the time required to generate successive images is too long, the image update rate will degrade causing flicker. The time complexity of the algorithms will be analyzed in this chapter. The first section presents routines for standard functions, such as addition and multiplication, then analyzes them for their speed of execution. The hardware requirements of the processor model of the system are discussed based on these functions. These routines serve as subroutines for encoding the algorithms into functional-block representations in the second section and different architectures are applied to make the algorithms run more efficiently. The final section compares the algorithms based on the criterion of speed of execution.

4.1 Basic Routines and Processor Model as Standards for Comparison

The purpose of this chapter is to analyze the execution times required for the shading algorithms. Some assumptions have been made about the processor model to isolate the amount of time required for the shading algorithms. First, the size of the memory is as large as necessary. Since the mapping of the memory is beyond the scope of this paper, no contention problems exist so the time required to access memory locations is assumed to be negligible. Also, the time to actually display the image on the screen is not considered. Lastly, there is no limit to the hardware available to implement the system. If four multiplications are performed simultaneously, four multipliers are available. This increases the cost of the system but reduces the speed of execution. Of course, other methods besides simply adding hardware are investigated. This is particularily true when computing an undetermined number of values; since the total hardware requirements are unknown, other architectures must be utilized to minimize the execution time.

The processor model contains the necessary hardware as described in the procedures that will follow. Nevertheless, some information is vital to the shading computations and the time necessary to calculate this information is discussed. Examples of such information include the determination of normal vectors for polygons and patches, extra processing of object models, and correlating data within the scene as with ray-tracing and some hidden-surface removal algorithms. The processor model is discussed in this section after the analysis of execution times for addition; subtraction; multiplication; division; square roots; exponentials; cosines; sines; inverse cosines; tangents; exponentiation; dot products; matrix products; vector addition, magnitudes and normalization; and summations as well as the determination of normals and midpoints. All numbers are assumed to be signed, two's-complement, fixed-point numbers that are scaled to values less than one. Although the system is a 32-bit machine, most of the values are only 16-bit with the higher precision provided for multiplication. Here n will be considered 16 and the system will be a 2n-bit system. Therefore, two successive multiplications may be performed before an overflow is likely to occur and truncation becomes necessary.

As stated, the execution times of several functions and processes are analyzed in this section. These times will be used in the next section when the shading algorithms are transformed into functional-block representations. Many of the execution times for the procedures were calculated using the material from Hwang [20] and Shanblatt [29]. Most of these times are measured in increments known as Δ_g which is the delay of a single NAND, NOR or INVERTER gate. This helps to keep the comparison independent of the technology used to implement the system.

The first procedure is addition. An n-bit, (up to 64 bits), Carry-Look-Ahead Adder, CLA, is used which produces an (n + 1)-bit sum. It is a two-level design based on 4-bit Block CLAs. The execution time required, Δ_T, is 12 Δ_g. For subtraction the subtrahend is complemented then is added to the minuend to form the desired difference. Complementation requires 3 Δ_g so the total time needed for subtraction is 15 Δ_g.

A Baugh-Wooley array multiplier is used to implement multiplication [2]. An n-bit multiplicand and multiplier will produce a 2n-bit product. The execution time required is based on the number of bits; $\Delta_T = (4n + 3)\ \Delta_g$.

The next eight procedures are implemented with the CORDIC algorithm first developed by Volder in 1959 [32] then later discussed by Walther [33] and Lawitzke [21]; Walther provides a comprehensive discussion of how to implement the CORDIC algorithm for circular, linear and hyperbolic functions using either the rotation or vectoring modes which he calls a unified algorithm. The unit of CORDIC delay, Δ_C, is the result of n shifts and adds requiring 2n clock cycles where n is the number of bits in the result and each clock cycle is at least as long as Δ_T for addition. Based on the CLA's execution time, Δ_C is approximately 24n Δ_g. One function implemented is division. Generally, the dividend is at least n bits but no more than 2n bits in length and the divisor and quotient have n bits. Division is a linear function and is implemented using the vectoring mode. It requires one Δ_C or 24n Δ_g where n refers to the number of bits in the quotient. Most of the seven remaining functions implemented with the CORDIC algorithm require feeding back the outputs and dividing out a constant term. The square root function is a hyperbolic function using the vectoring mode and requires Δ_C and one division process or 48n Δ_g. Exponentials are also hyperbolic functions but use the rotation mode. Two CORDIC delays are again needed so the delay is 48n Δ_g. The cosine function is circular and uses the rotation mode. It also requires $2\Delta_C$ so that Δ_T is 48n Δ_g. The sine function is similar to the cosine function and requires the same delay. The inverse cosine is first evaluated by finding the inverse sine then using a trigonometric relation for the final result. The inverse sine is defined as

$$\sin^{-1} \gamma = \tan^{-1} \left[\frac{\gamma}{(1 - \gamma^2)}\right] \tag{74}$$

This function requires a multiplication, subtraction, square root and one division then the inverse tangent is evaluated using the CORDIC algorithm which uses one Δ_C. Then the arc cosine is evaluated using

$$\cos^{-1} \gamma = \frac{\pi}{2} - \sin^{-1} \gamma \tag{75}$$

which has a total time delay of (100n + 33) Δ_g. The tangent function is implemented by simultaneously calculating the sine and cosine functions and then dividing them, during which the constant terms cancel. The tangent requires $2\Delta_C$ or 48n Δ_g. Exponentiation is the last Cordic function and is evaluated using the identity for $c = a^b$:

$$c = e^{(b \ln a)} \tag{76}$$

Natural logarithms can be implemented with the CORDIC algorithm using $2\ \Delta_C$. Thus, the total delay will be ($4\ \Delta_C$ + 131) Δ_g or 3203 Δ_g when n is 32.

Since the images are three-dimensional each vector has three components. Thus, to form dot products requires three separate multiplications which may be performed in parallel and two additions so the delay is that of multiplication plus 24 Δ_g. On the other hand, because of the bicubic patches discussed in Catmull's algorithm, some matrices are as large as 4 x 4. If matrix A is defined as m x c and matrix B is c x p, then as each row of A is multiplied by each column of B, c multiplications and (c - 1) additions are necessary. However, the multiplications can be completed simultaneously. If c = 4, the number of addition delays can be reduced to two by finding the sums in parallel according to the recurrence computation array, also called tree reduction, shown in Figure 4.1. For all matrix multiplications performed in this paper c = 4 so the number of additions needed is two. But overall these operations are required mp times for the completion of the entire matrix multiplication procedure when performed separately. By using m processing elements, PEs, entire columns of the product can be calculated by each PE simultaneously. This reduces the total delay time to p (4n + 27) Δ_g but substantially increases the hardware because each PE must have c multipliers and at least (c/2) adders with a maximum of c adders.

Vector addition uses three parallel adders to sum the vector components separately and simultaneously. It requires the same delay as scalar addition. Determining the magnitude of vectors requires taking the square root of the sum of the squared vector components. The vector normalization process then divides this number into each component of the original vector. The former requires three simultaneous multiplications, two additions, and one square root while the latter additionally requires three simultaneous divisions. Thus the total delay to calculate the magnitude is (52n + 27) Δ_g and to normalize the vector is (76n + 27) Δ_g.

Summations are generated using recurrence computations as depicted in

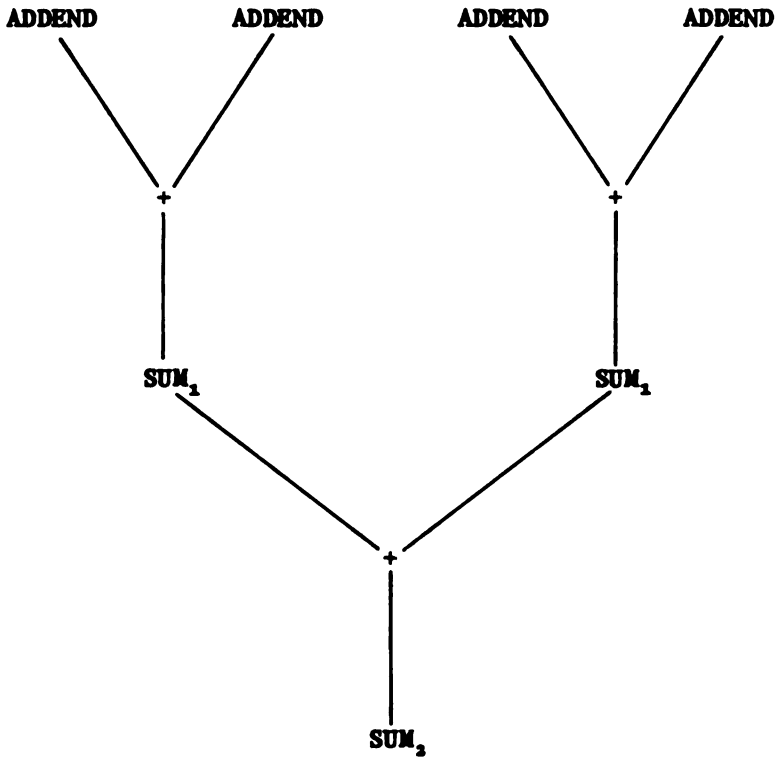

Figure 4.1: Addition array to sum four addends most expediently

Figure 4.1. This reduces the computation from (a - 1) addition delays, where a is the number of addends, to $\log_2 a$. Thus, the total delay is $12 \log_2 a \, \Delta_g$. The minimum number of adders needed to accomplish this is half the number of addends where each can send its sum directly to any other adder to expediently transfer data. The maximum would be equal to a with the adders arranged in an array with predefined data paths similar to that shown in Figure 4.1.

Finding the normal vector of a planar polygon is done by calculating the cross product of any two adjacent sides. Since the equations for the sides of the polygons are not known, vectors in the same direction can be calculated by subtracting corresponding components of the endpoints; this requires three simultaneous subtractions for each vector or $15 \, \Delta_g$ overall. Then six simultaneous multiplications and three simultaneous subtractions are necessary to perform the cross product. The total delay, including finding the vectors, is $(4n + 33) \, \Delta_g$. The final procedure is to find the midpoint of linear line segments. The corresponding components of the endpoints are added needing three simultaneous additions and then each sum is split in half using three simultaneous divisions. The total delay is $(24n + 12) \, \Delta_g$.

All of the execution times discussed above are tabulated in Table 4.1 for both the general case and when n is 32. They will be used in the next section while transforming the algorithms into functional-block representations.

The hardware requirements now presented for this system are subject to change after the shading algorithms have been transformed. These requirements are based on the parallel operations performed to calculate the

Table 4.1: Summary of execution times for basic procedures

Procedure	Execution Time, Δ_T	Δ_T When n=32 [1]
Addition	12 Δ_g	12 Δ_g
Subtraction	15 Δ_g	15 Δ_g
Multiplication	(4n + 3) Δ_g	131 Δ_g
Division	24n Δ_g	768 Δ_g
Square Root	48n Δ_g	1536 Δ_g
Exponential	48n Δ_g	1536 Δ_g
Cosine	48n Δ_g	1536 Δ_g
Sine	48n Δ_g	1536 Δ_g
Arc Cosine	(100n + 33) Δ_g	3233 Δ_g
Tangent	48n Δ_g	1536 Δ_g
Exponentiation	120n Δ_g	3203 Δ_g
Dot Product	(4n + 27) Δ_g	155 Δ_g
Matrix Multiplication	p (4n + 27) Δ_g [2]	155 p Δ_g
Vector Addition	12 Δ_g	12 Δ_g
Vector Magnitude	(52n + 27) Δ_g	1691 Δ_g
Normalization	(76n + 27) Δ_g	2459 Δ_g
Summation	12 $\log_2$ a Δ_g [3]	12 $\log_2$ a Δ_g
Normals	(4n + 33) Δ_g	161 Δ_g
Midpoints	(24n + 12) Δ_g	780 Δ_g

1- n is the number of bits in the numbers.
2- Matrix A is m x c, B is c x p. Function performed is A * B.
3- a is the number of addends to be summed.

procedures discussed above. The majority of the hardware is necessary because of the vector computations that are performed, particularily the matrix multiplication.

First, the consequences of using a 32-bit machine are discussed. As stated, the majority of the numbers are limited to only 16 bits but the system is capable of 32 to accommodate the larger products from multiplication. Adhering to these limitations means there are initially 16 bits to form values but the number of bits in the shading values is likely to increase since most of the shading algorithms require at least one multiplication. Thus, the number of bits used to calculate shading intensities provides a maximum of 2^{32} intensity levels. Even though the values of x, y and z are limited to 16 bits, this limitation provides more than adequate screen resolution. Besides, the relationship between the screen resolution and the number of intensity levels available was discussed in Chapter II with the greater number of intensities favored over the higher resolution. For this reason even if these limitations did not allow for such high resolutions or vastly numerous intensity levels, they would not adversly affect the images generated.

The most demanding procedure, in terms of hardware, is matrix multiplication. The recurrence computation was used for this procedure as well as to generate summations. Using the recurrence computation array to reduce the delay caused by the additions means needing from two to four adders per processing element because there are two possible ways to implement the recurrence array. The first is to use two adders which are capable of sending their sums to each other and themselves directly to avoid delays from transferring data. These sums are used as inputs for the next addition. If there are four addends, as is the case with matrix multiplication, each adder receives two addends and forms a sum. Then one of the adders adds the two sums just produced to form the final sum. The second method of implementation is to use four adders in an array as denoted by the plus signs in Figure 4.1. The adders of the first row compute their sums then send them to the adders in the next row as designated by the lines marking the data flow pattern. The direction of the data flow is predetermined and is static so this hardware is dedicated whereas the data flow pattern of the previous method is programmable so a single addition can easily be performed. The advantage of this second method is that if the number of addends used as inputs to the summation is greater than four, this method lends itself readily to pipelining; four addends could be applied to the first row of adders after one addition cycle while the final sum of the previous four is being calculated by the second row. However, the matrix multiplication procedure requires four of the recurrence arrays so large numbers of addends can be efficiently summed using all four arrays. Thus, using the first method requires less hardware and the advantages of the second method are still retained. Overall, the matrix multiplication procedure needs eight adders, two for each processing element. It also needs 16 multipliers, four for each processing element. The quadruplicate hardware demands cut the execution time to one quarter of that without the extra hardware.

Vector normalization and cross products require three simultaneous divisions and hence, three dividers. The process of determining vectors from polygon vertices to use for the cross products requires six simultaneous subtractions. Since there are already at least six adders because of matrix multiplication, only six complementor circuits need to be added to the hardware list. And lastly, the CORDIC algorithm requires a shift register. Since three dividers are needed, the CORDIC circuitry is tripled.

This concludes the hardware requirements of the basic routines discussed above. Besides the assumptions stated at the beginning of this section, no other demands are made of the processor model at this time. Nevertheless, if some of the procedures are performed in parallel to speed the execution of the shading algorithms, more hardware may be necessary. Methods to reduce the execution times of the algorithms are investigated in the next section of this chapter, along with their impact on hardware requirements. The algorithms are transformed into functional-block representations using the processes described in this section as basic building blocks.

4.2 Functional-block Transformations of the Algorithms

In this section the shading algorithms presented in Chapter III are transformed into functional-block representations using the routines analyzed in the previous section as the functional blocks. One problem with performing a time analysis on these algorithms is due to the relationship between the object-modeling technique, the hidden-surface algorithm and the shading algorithm implemented. Because the overall appearance of the shaded images can be improved by introducing changes to any of these three areas, one cannot simply analyze the shading algorithm alone. For this reason improvements to the object-modeling technique and the hidden-surface algorithm are also transformed to functional-block representations when deemed that they are critical to the operation of the shading algorithm. These times must be considered along with the actual shading algorithm execution time during the time analysis.

Another problem with comparing the algorithms is that each makes different assumptions. For example, most of the algorithms assume that the normal vector to the polygon is known before performing the shading but Catmull carries out an extensive approximation of the normals. Gouraud and Newell, et al., claim to know the incident angle and calculate its cosine directly whereas Blinn, Whitted and Cook, et al., know the vectors for the viewer and light source's directions and use dot products to determine the cosines of their angles with the normals. On the other hand, Phong approximates the cosine of the angle for the specular component directly from the normal but still uses a dot product for the cosine of the incident angle. Blinn approximates the direction of specular reflection and its angle with the normal but Cook, et al., assume they are known while the other algorithms do not utilize them at all. Catmull and Fournier, et al., do not implement their own shading algorithm so they make the same assumptions of the shading algorithm used. Cook, et al., and Whitted are the only algorithms that demonstrate the additive property of different light sources.

Yet, the impact of these assumptions is not always noticeable. To calculate the normal for planar polygons requires much less time than to carry out the approximations of normals for bicubic patches. Therefore, the planar-polygon normal calculation is nearly negligible in comparison. The additive property associated with multiple light sources is applicable across the board so it is ignored. For Phong to approximate the cosine is only one gate delay greater than using a dot product. However, for Newell, et al., and Gouraud to directly calculate the cosine takes almost ten times longer than performing the dot product so some advantages are gained here that will be considered during the final analysis.

After each algorithm has been transformed, it is applied to the simple image of Figure 4.2. The figure depicts a raster of pixels with an object modeled having six surfaces. Three of the surfaces are hidden from the viewer and are drawn with dashed lines. Note that point G is located behind point A and each raster point is represented by its corresponding visible and hidden surfaces. The algorithms are applied to this object while assuming only one light source as a standard for the time analysis. The last section of this chapter summarizes the time studies of the next eight subsections.

4.2.1 Gouraud Transformation

This algorithm is presented in Section 3.1. The shading value is interpolated across patches or polygons to give the appearance of smooth, curved surfaces.

Although Gouraud presented his algorithm for objects modeled using Coons patches, it is analyzed as applied to polygons. Gouraud assumes that the viewer and light source are at the same location and thus, both make the same angle, θ, with the normal to the surface. The shading value is calculated at the highest and lowest points of the edges of the polygons, (the vertices of the polygons), during the hidden-surface removal algorithm

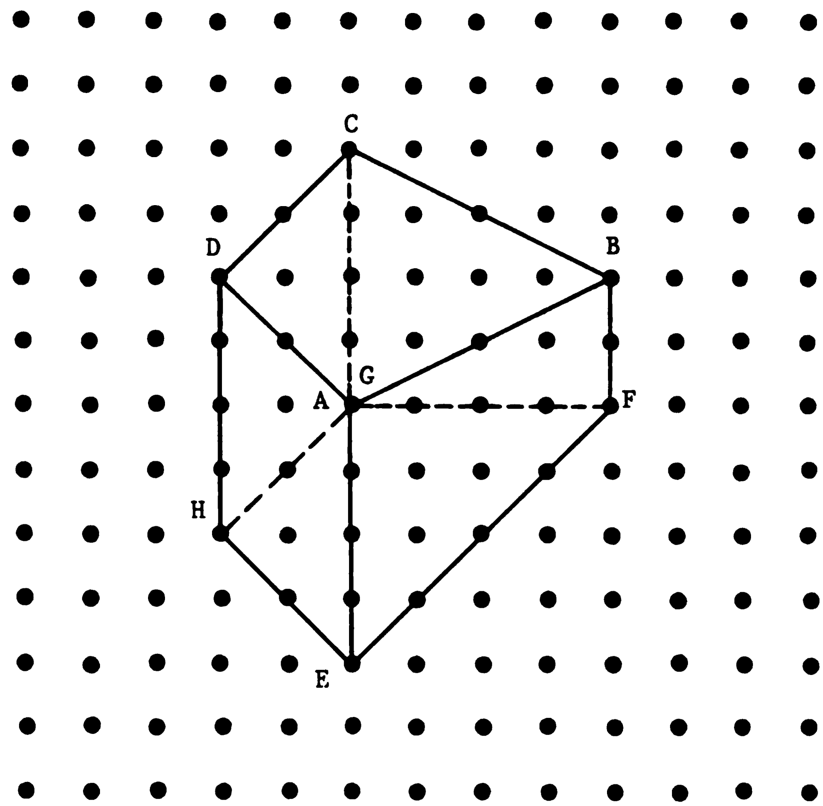

Figure 4.2: Image to be shaded with raster of pixels

using equation (1). The functional-block diagram for this equation consists of a cosine operation and two successive multiplications as shown in Figure 4.3. The total time needed for this calculation is 2798 Δ_g.

The number of scan lines intersecting each edge is determined during the hidden-surface algorithm as well. Using this and the two endpoints' shading values, a shading "slope" is calculated using equation (6). This slope is also calculated during the hidden-surface algorithm. The functional-block diagram for the "slope" is shown in Figure 4.4. Not including the time to calculate the endpoints' shading values, the "slope" requires 783 Δ_g.

Starting from the highest endpoint of a polygon edge, this "slope" is added to the shading value of the present edge-pixel to obtain the shade of the point of intersection of the next scan line with the same edge. Since the endpoint shades and edge slopes have all been calculated during the hidden-surface algorithm, this procedure requires only an addition for each interior point on the polygon edges. Using these values, the interior points of the polygon, (those not on edges), are interpolated using equations (7) and (5) as depicted in Figures 4.5 and 4.6. The complete interpolation requires 941 Δ_g. The hardware required for each of these procedures is within the limits of the existing processor model.

This is really the only algorithm that provides a method for its implementation. It uses the shading "slope" to speed the calculation of the shading values for the polygon edges and interpolates all interior points as discussed more fully in Section 3.1. It is possible to process segments of the scan lines in parallel after the segment endpoints' shades have been calculated. The image in Figure 4.2 has a total of 13 visible segments

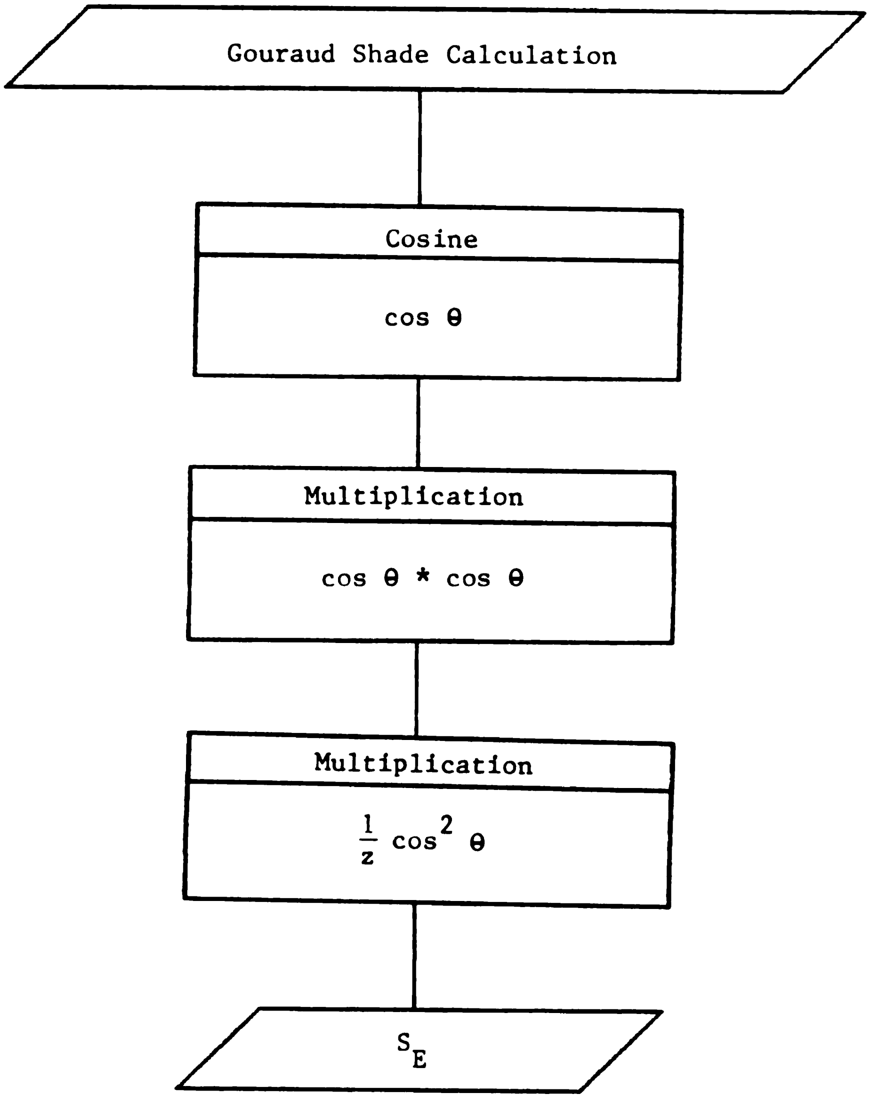

Figure 4.3: Functional-block diagram of Gouraud's shading model

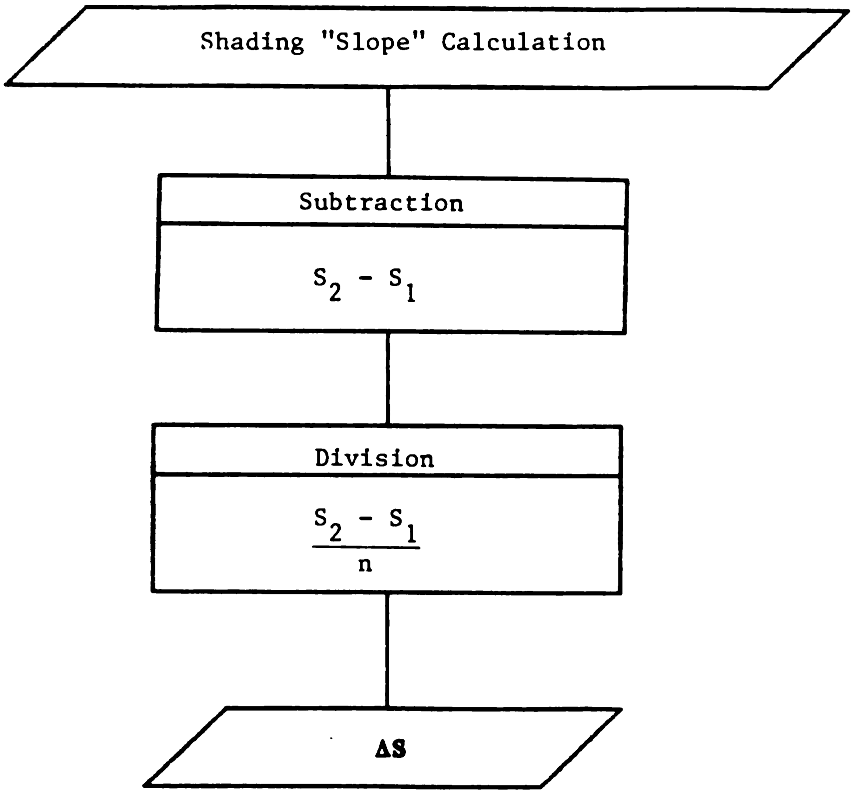

Figure 4.4: Functional-block diagram for calculating the shading "slope"

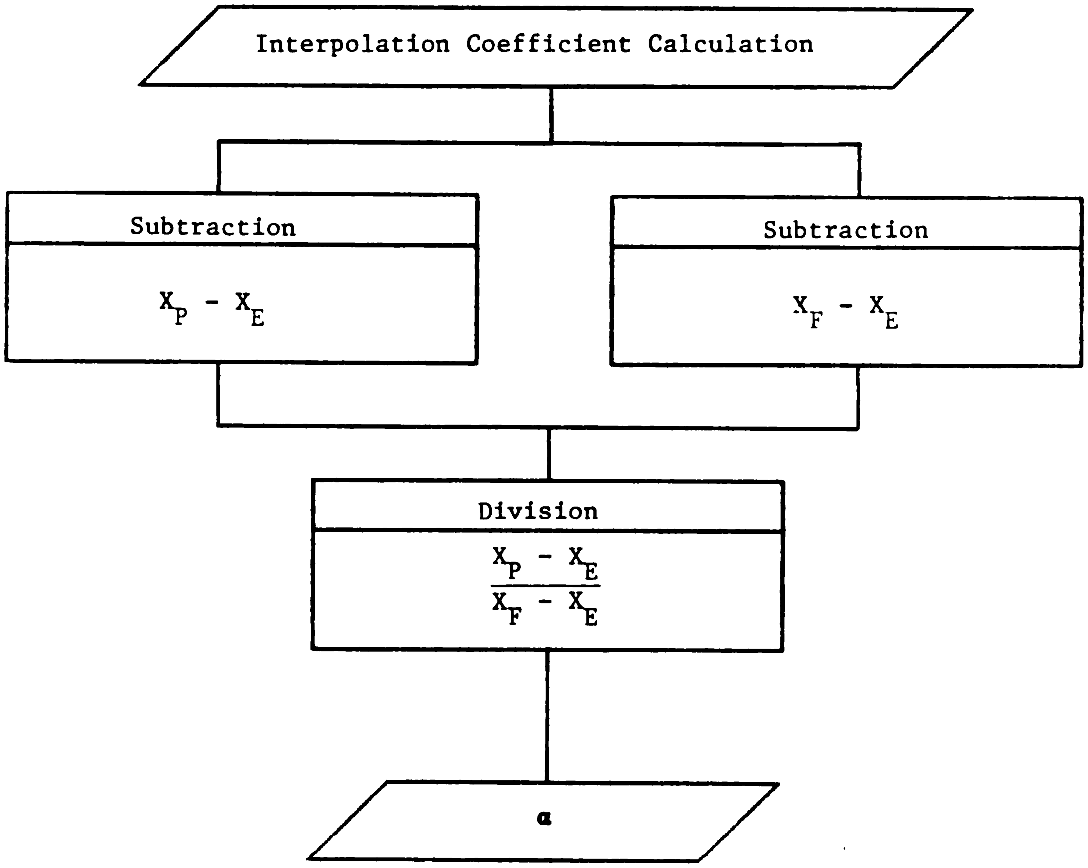

Figure 4.5: Functional-block diagram for calculating the interpolation coefficient

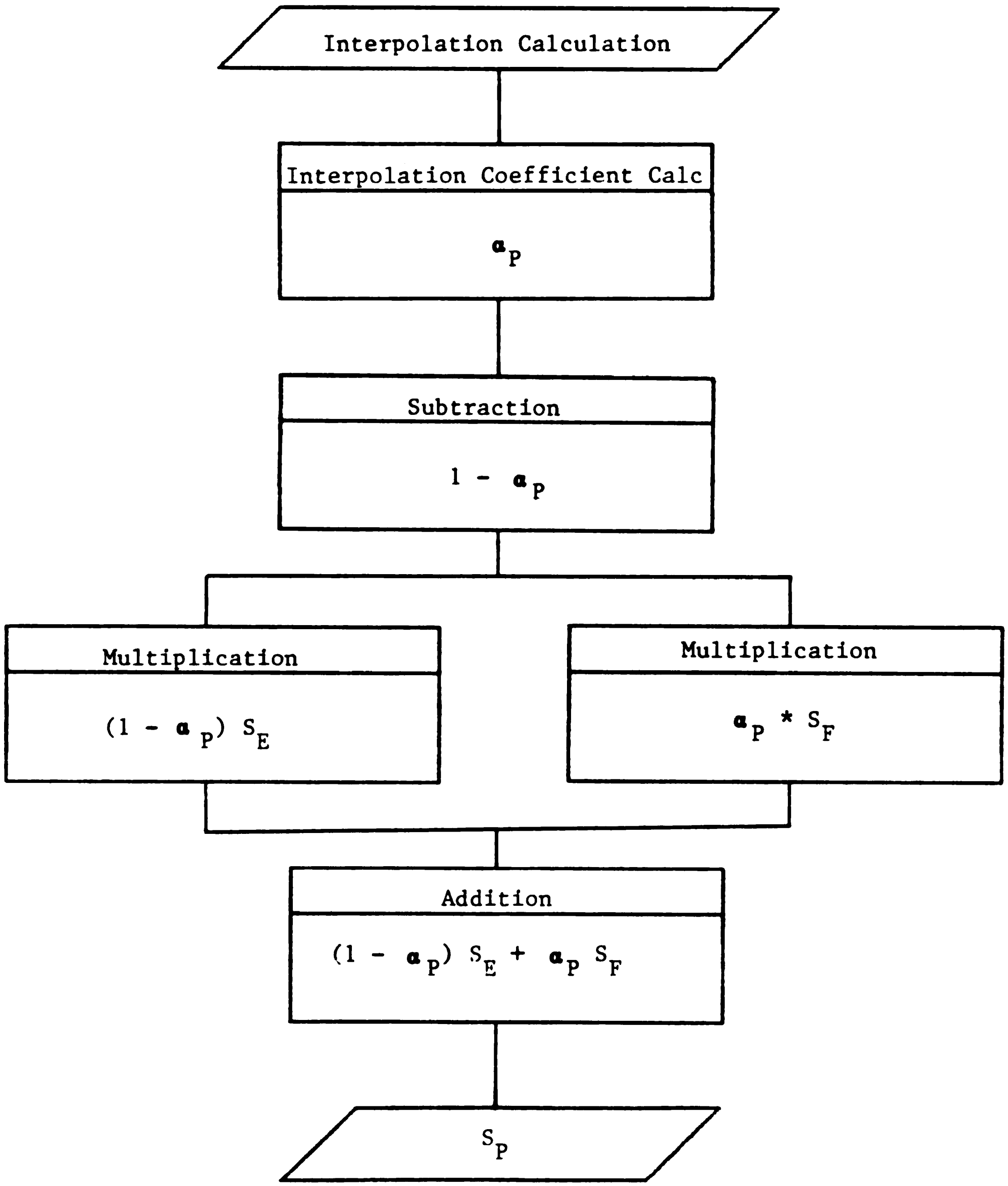

Figure 4.6: Functional-block diagram of the interpolation calculation

which could easily be done in parallel by adding enough hardware. However, the number of segments composing images is neither constant nor predetermined. Also, since the length of each segment is not the same, some segments require more total time than others.

A better method is to calculate shades for pixels in parallel. As long as the endpoint shades can be coordinated with each segments' interior points, each interior pixel's interpolation can be performed in parallel. By making each pixel's interpolation independent, the timing is more uniform and lends itself better to pipeline and parallel processing than segments do. Each interpolation requires three subtractors, two multipliers, one adder and one divider. The entire interpolation is used as a pipeline, there are several parallel pipelines to calculate data for several pixels at once. Using two parallel adders to calculate each segment's endpoints' shading values, the information can be available every 12 Δ_g. After this delay, one or more of the pipelines could begin calculating pixel shading values. If there are fewer pixels in the first segment than parallel pipelines, each remaining pipeline may have to wait one or more additional delays.

The delay of each stage of the pipeline must be 783 Δ_g plus some latch delay because the division process is the most time-consuming. After the first stage is completed, several pairs of endpoints' values will be available so the pipelines can continue processing the entire picture. The pipelines will have six stages so the delay for each is (N + 5) 768 Δ_g, where N is the number of values processed. This delay includes filling and emptying the pipeline. Since the pipelines are operated in parallel, the total delay is this plus at least one addition delay: (N + 17) 768 Δ_g.

The hardware necessary for this architecture primarily depends on the number of parallel pipelines used. Assuming there are three, the system would need nine subtractors, six multipliers, three adders and three dividers plus two more adders to calculate the edge shades and a buffer to store the edge shades until a pipeline is ready to use them as well as latches between the pipeline stages. So besides the latches and buffers, six adders and three complementors must be added to the system. A general block diagram of the architecture is presented in Figure 4.7.

When applying this algorithm to the image in Figure 4.2, there are two parts to consider. The first is the extra processing that must be done during the hidden-surface removal algorithm to find data specifically for the shading portion. The second part is the shading portion itself.

During the hidden-surface algorithm, shades are calculated for the vertices of the polygons and "slopes" are determined for each edge. The widget in Figure 4.2 has six polygons, eight vertices and 12 edges. Assuming that the shading information is only calculated for visible portions of the image, seven vertex shading values and nine "slopes" must be calculated. This requires a total of 26,633 Δ_g. This time is greatly dependent on the implementation of the hidden-surface algorithm and could possibly be reduced. However, as it stands, it does not impose any new requirements on the hardware of the system.

Overall, the image has 15 interior edge points whose shading values must be calculated. In total there are 13 segments and 19 interior polygon points. Starting from point C of edge CD, the first segment has two pixels so after 12 Δ_g two pipelines will be started. Normally, after another 12 Δ_g the third pipeline is started but in this case the endpoints of the next

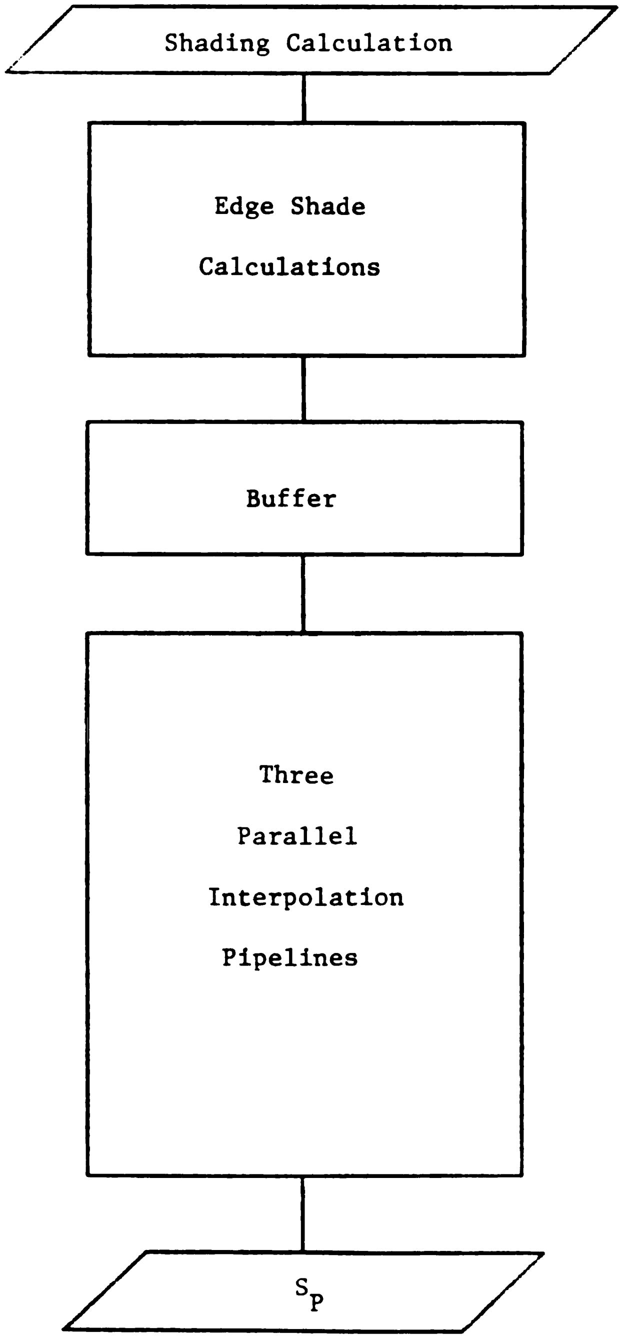

Figure 4.7: Functional-block diagram of the entire shading calculation

segment are vertices of polygons meaning their shading values are already known so no addition is necessary. Since there are 19 pixels, each pipeline calculates shading values for six. The last pixel's value is found by one of the first two pipelines since they finish before the third. (This last pipeline delay would have absorbed the extra addition delay in the beginning.) Thus, the total delay for Gouraud's algorithm when applied to the standard image for N = 7 is the pipeline delay and one addition; thus, Δ_T is 9228 Δ_g.

4.2.2 Phong Transformation

Phong's algorithm is presented in Section 3.2. It assumes that the light source and viewer are infinitely far away. This means that the rays of light will be parallel and that distances do not affect the shading values. The algorithm assumes normal vectors are known at vertices then instead of interpolating shading values across the polygons, normals are interpolated for each pixel. The most time-consuming portion of this process is that the normals must be normalized after they have been approximated. Although Phong assumes to know the directions of the viewer and light source, the cosine of the angle of the viewer is approximated directly from the normal vector. Phong claims that finding cos σ in this manner is faster than interpolating it, but the difference between this method and calculating the cosine using a dot product is actually one extra Δ_g.

In this algorithm the calculation of normals can be considered part of the information processing during the hidden-surface algorithm. Normals are approximated for each pixel of the image using the same interpolation scheme of Figures 4.5 and 4.6 that was used to approximate shading values. The functional-block diagram uses the previous figures as its blocks and is shown in Figure 4.8. The total delay is 4183 Δ_g. Because of the large number of pixels in an image, the normal approximations would greatly benefit from a similar architecture used for the shading interpolations for Gouraud's algorithm. One difference is that the circuitry to calculate the segments' endpoints' shading values is not needed. The other is the need for normalization circuitry.

The best architecture is to break up the normalization process into its five steps: three simultaneous multiplications, two sequential additions, a square root and three simultaneous divisions. These are added at the end of the interpolation pipeline making it an 11-stage pipeline. Now the stage delay must be equal to the square rooter's delay: 1536 Δ_g. Since the total delay of any two consecutive steps does not exceed the square rooter's delay, pairs of consecutive stages are joined as one stage making a six-stage pipeline with better hardware utilization. This also decreases the total pipeline delay which is now (N + 5) 1536 Δ_g. As with all pipelines, the greatest time savings are obtained when large numbers of the calculations are performed. The hardware requirements of this pipeline are those specified for the normalization process mentioned above and for the interpolation pipeline in the last section. The total requirements are five multipliers, four adders, three subtractors, four dividers and a shift register. If three of these are placed in parallel, the extra hardware units required by the system are 13 adders, three complementors and nine dividers.

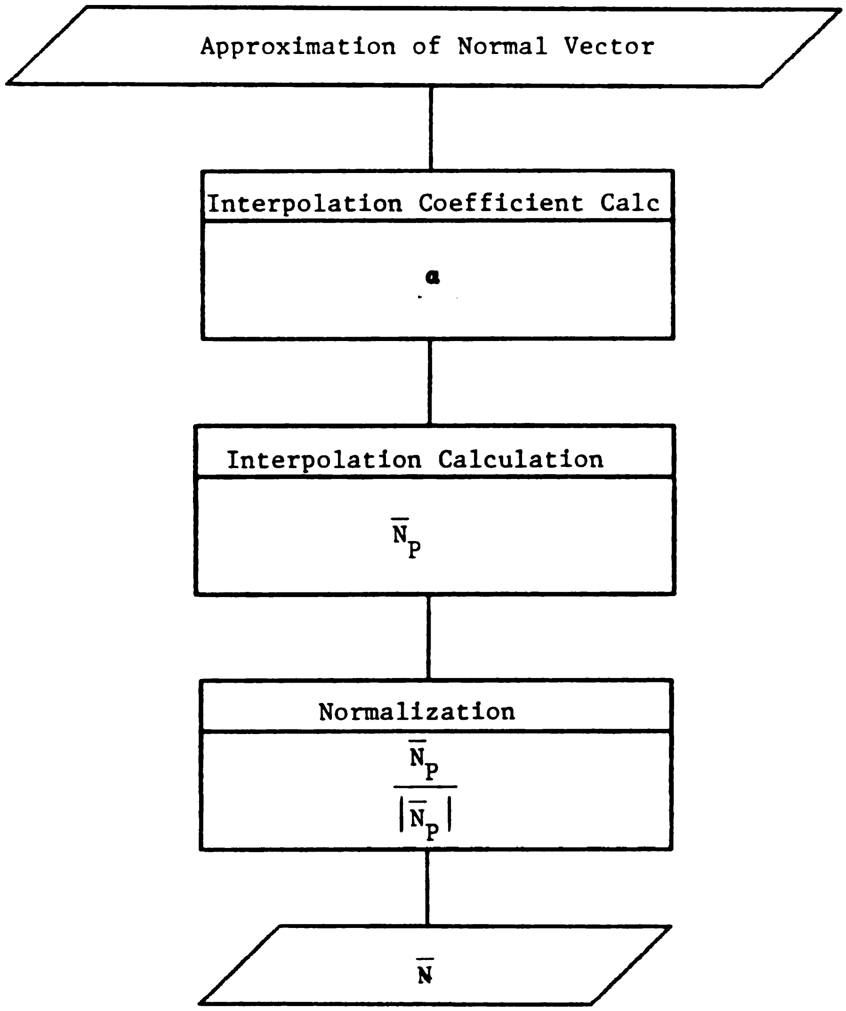

Figure 4.8: Functional-block diagram of the normal vector approximation

The remaining calculations are part of the shading portion of the algorithm. To calculate the cosine of the incident angle requires a dot product as shown in Figure 4.9 which uses 155 Δ_g. Once cos Θ is known, estimating cos σ as in equation (17) requires 156 Δ_g; this calculation is pictured in Figure 4.10. Lastly, Phong's shading calculation is equation (11). Using the cosine values, a shading value is computed according to the functional-block diagram of Figure 4.11. The entire calculation, including evaluations of the cosines, uses 4284 Δ_g. The biggest time-user is the exponentiation of the cosine value for the specular reflection. With the CORDIC implementation of this function, unless c_1 is greater than 30, it is faster to perform the function using consecutive multiplications. However, since c_1's value is not known, exponentiation is used here.

Nevertheless, this function can be broken into three steps. The first and last require 1536 Δ_g so this is the stage delay of a pipeline implementation. Both cosine operations can be combined as one stage and the operations parallel to the exponentiation form four stages parallel to those of the exponentiation. All in all, there are six stages in the pipeline that contribute to the total delay of (N + 5) 1536 Δ_g. Using three of these in parallel, the hardware must have 27 multipliers, 18 adders, six subtractors and six shift registers. Therefore, in addition to the hardware added for the normal interpolations, 11 multipliers, six adders and three shift registers must also be added to to the system.

Applying this algorithm to the image in Figure 4.2 increases the execution time of the hidden-surface algorithm. There are 34 visible points that need to have normal vectors. Splitting these in three, each pipeline calculates at least 11 normals with one doing an extra. The delay for this,

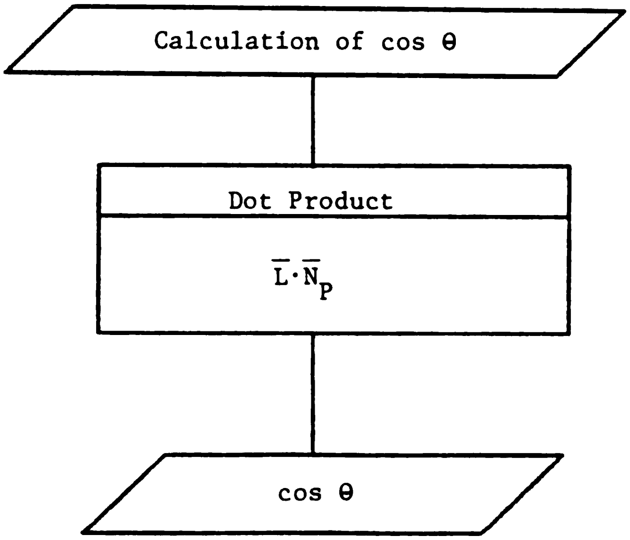

Figure 4.9: Functional-block diagram for calculating cos θ

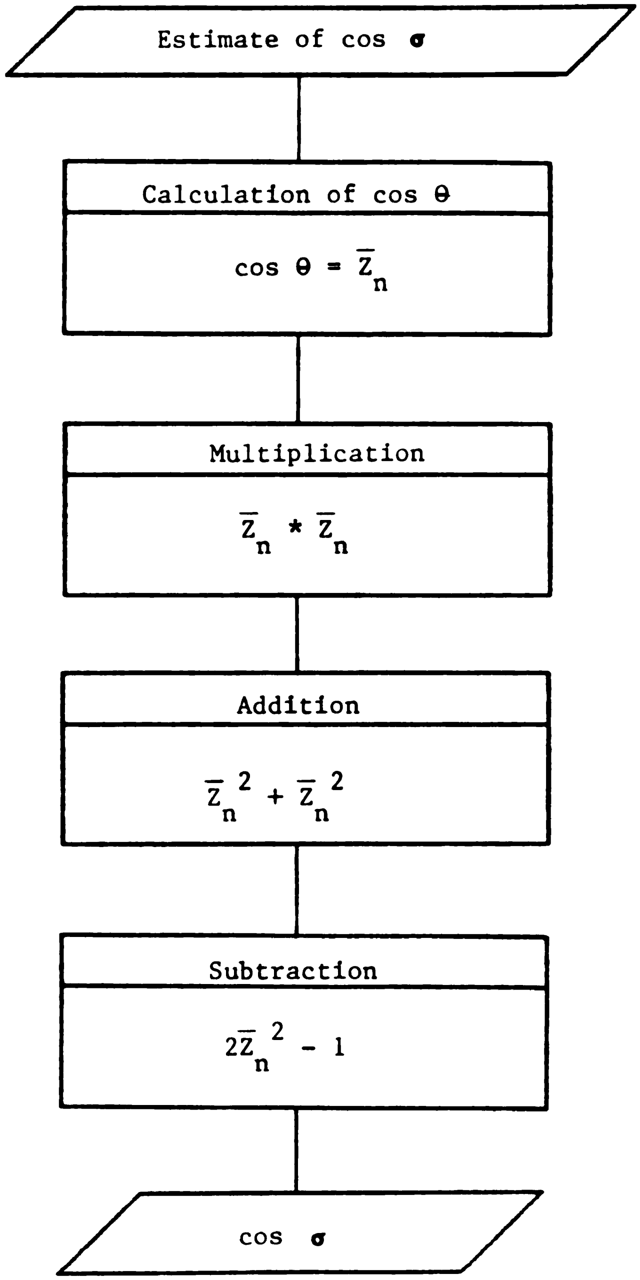

Figure 4.10: Functional-block diagram for estimating cos σ

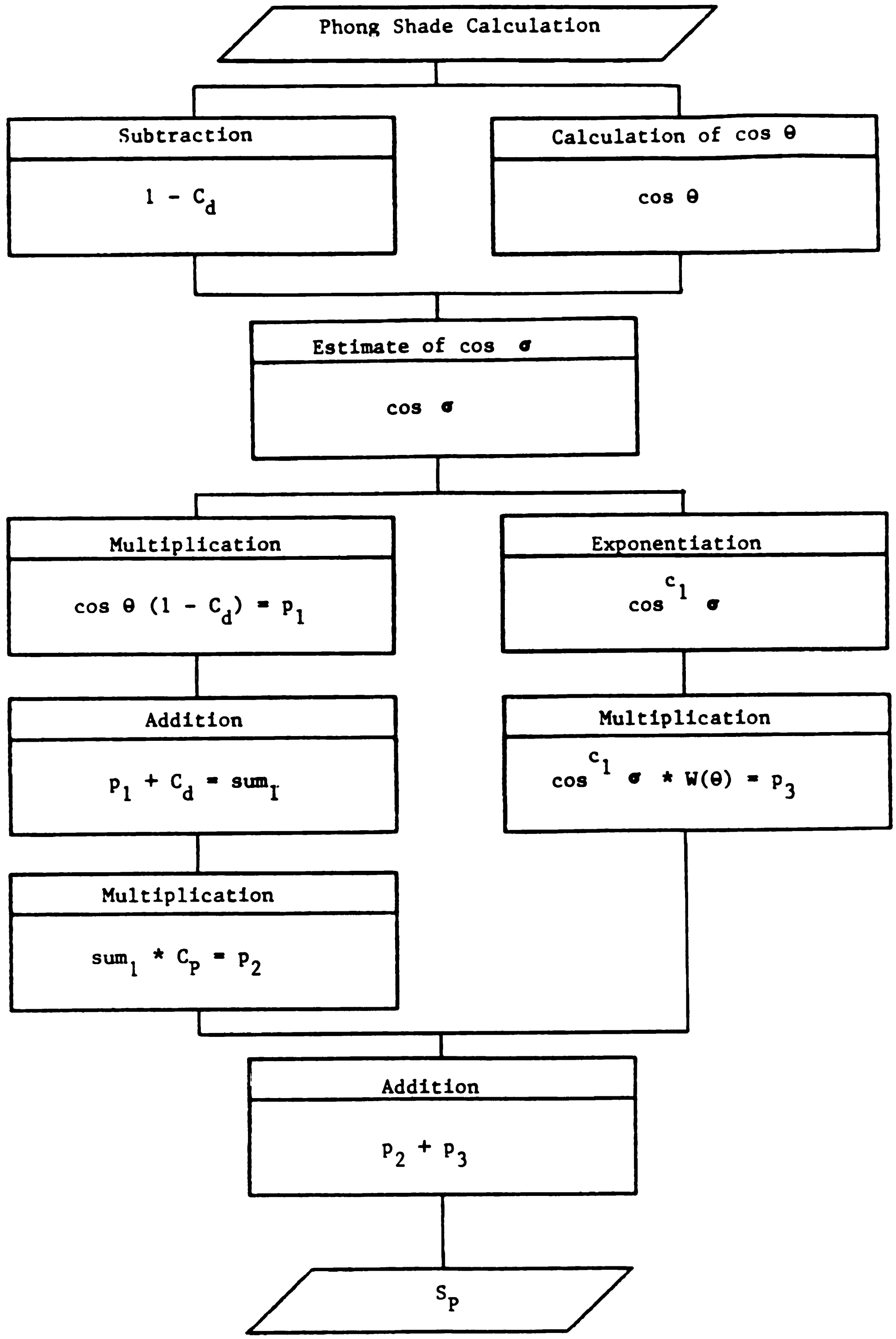

Figure 4.11: Functional-block diagram for Phong's shading model

with N = 12, is 26,112 Δ_g. In the same manner, the shades must be calculated for the same 34 points plus the seven vertices. This means two of the shading pipelines find 14 values and the last only finds 13. The total delay for the Phong shading calculation is 29,184 Δ_g.

4.2.3 Blinn Transformation

Blinn's algorithm is presented in Section 3.3. Although the polygon-mesh modeling technique is used, the surface is presumed to be composed of microfacets oriented in random directions.

Blinn claims that there is no increase in execution time over Phong's algorithm. He assumes that the directions of the viewer, light source and normal are known. However, Blinn's algorithm is an attempt to improve on Phong's algorithm. Since no attempt is made to alter the shading values across polygons to smooth the shading, it is assumed that Blinn relies on Phong's technique to interpolate the normal vectors. Thus, everything in the last section that pertains to interpolating the normals applies here as well.

Another process added to the hidden-surface algorithm is the determination of the direction of the specular reflection, $\bar{H}$, as in equation (21) which is shown in Figure 4.12. $\bar{H}$ needs to be calculated only once per change in light source direction so this is an addition of 2471 Δ_g to the hidden-surface algorithm's delay and two multipliers, 12 dividers, 17 adders, three complementors and one shift register, including the hardware for the normal interpolations.

Blinn uses $\bar{H}$ to calculate the angle β that it makes with the normal.

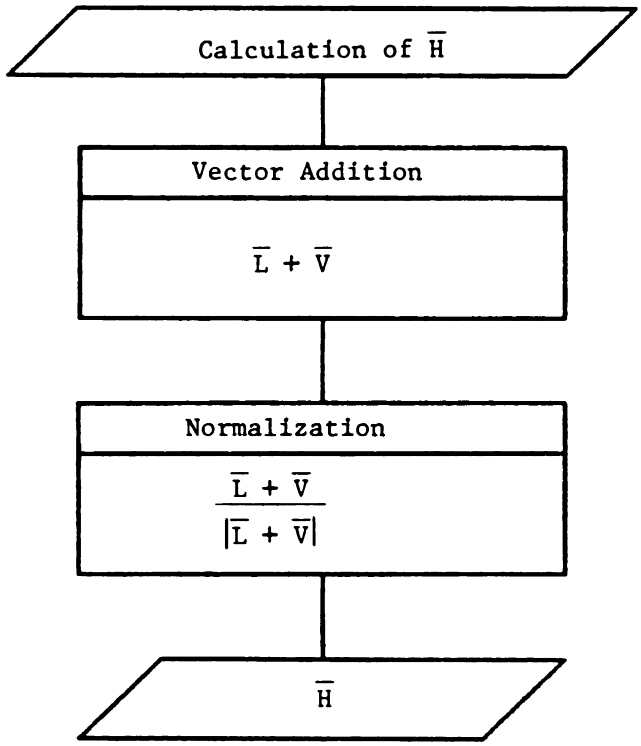

Figure 4.12: Functional-block diagram for calculating the direction of the peak specular reflection

This is unnecessary and time-consuming. The cos β is the value needed for the shading calculations and it can be found using a dot product which uses 155 Δ_g.

The calculation of Blinn's shades is more complicated because it uses the Fresnel Reflection Law, F; a distribution function for the facets' orientations, D_s; a division by $(\overline{N}\cdot\overline{V})$ and a geometric attenuation factor, G. D_s utilizes calculations described in equations (29), (30), (31) and (32) for which functional-block diagrams are shown in Figures 4.13, 4.14 and 4.15. The complete calculation of D_s takes 5954 Δ_g. This calculation needs to be performed only once per frame so it does not drastically increase the execution time of the shading equation. The Fresnel function of equation (39) is depicted in Figures 4.16 and 4.17. Despite the fact that its calculation requires 3832 Δ_g, it has to be found only once per change in light source direction. G is only calculated once per change in illumination as well. It was presented as equations (35) and (36) but is combined with the division by $(\overline{N}\cdot\overline{V})$ using the short program at the end of Section 3.3. The functional-block diagram is Figure 4.18 and its total delay can be one of four from the various paths possible; the maximum is 1227 Δ_g. This software requires that a magnitude comparitor be added to the system which has a delay of 48 Δ_g. These last three functions can be performed in parallel so the delay of D_s is added once to the shading calculation's time. Then the shading computation of equation (22) uses an additional 262 Δ_g for consecutive multiplications as shown in Figure 4.19. There is sufficient hardware after the hidden-surface algorithm requirements are fulfilled.

Applying Blinn's algorithm to the widget in Figure 4.2, the normal and

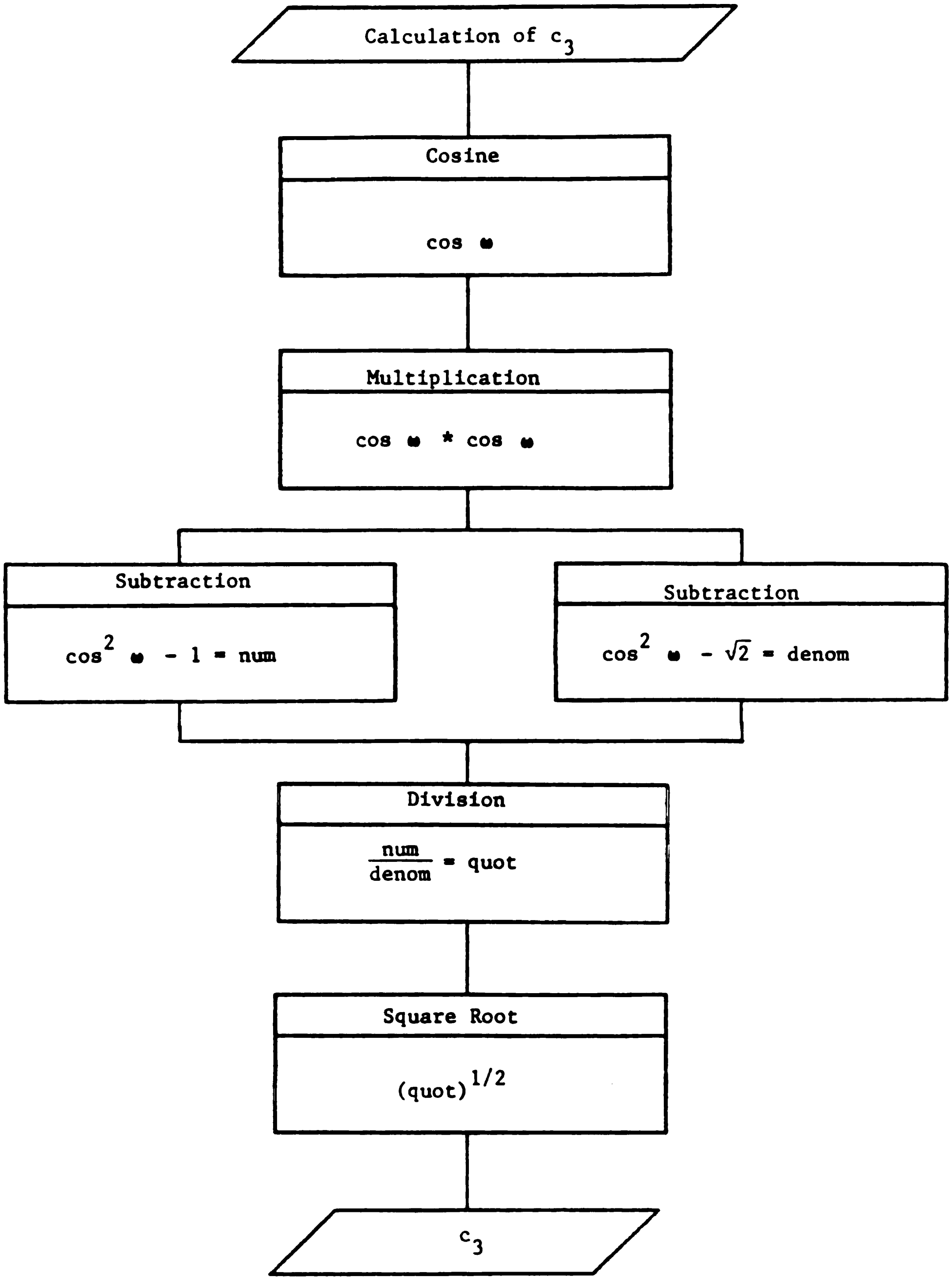

Figure 4.13: Functional-block diagram for calculating c_3

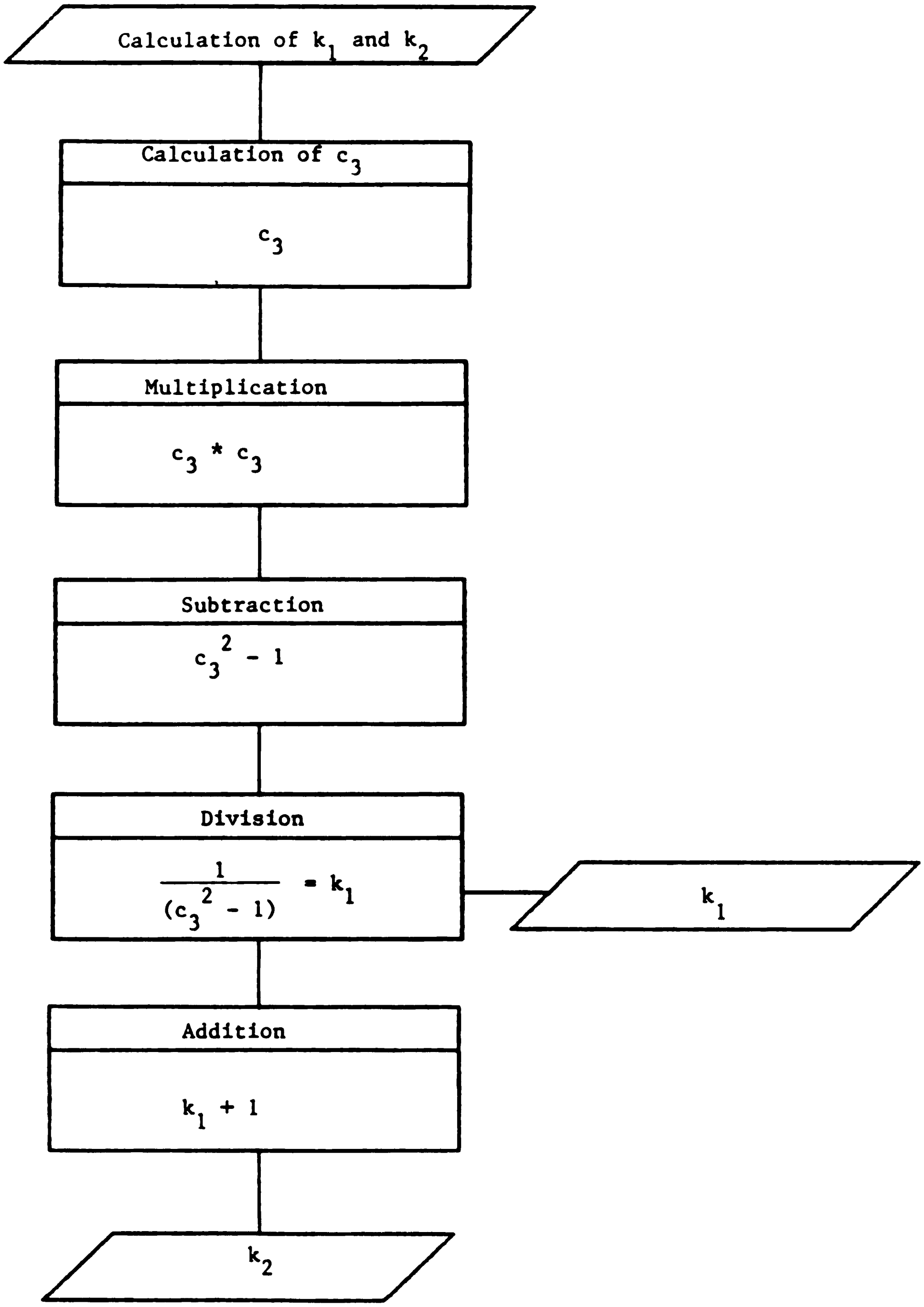

Figure 4.14: Functional-block diagram for calculating k_1 and k_2

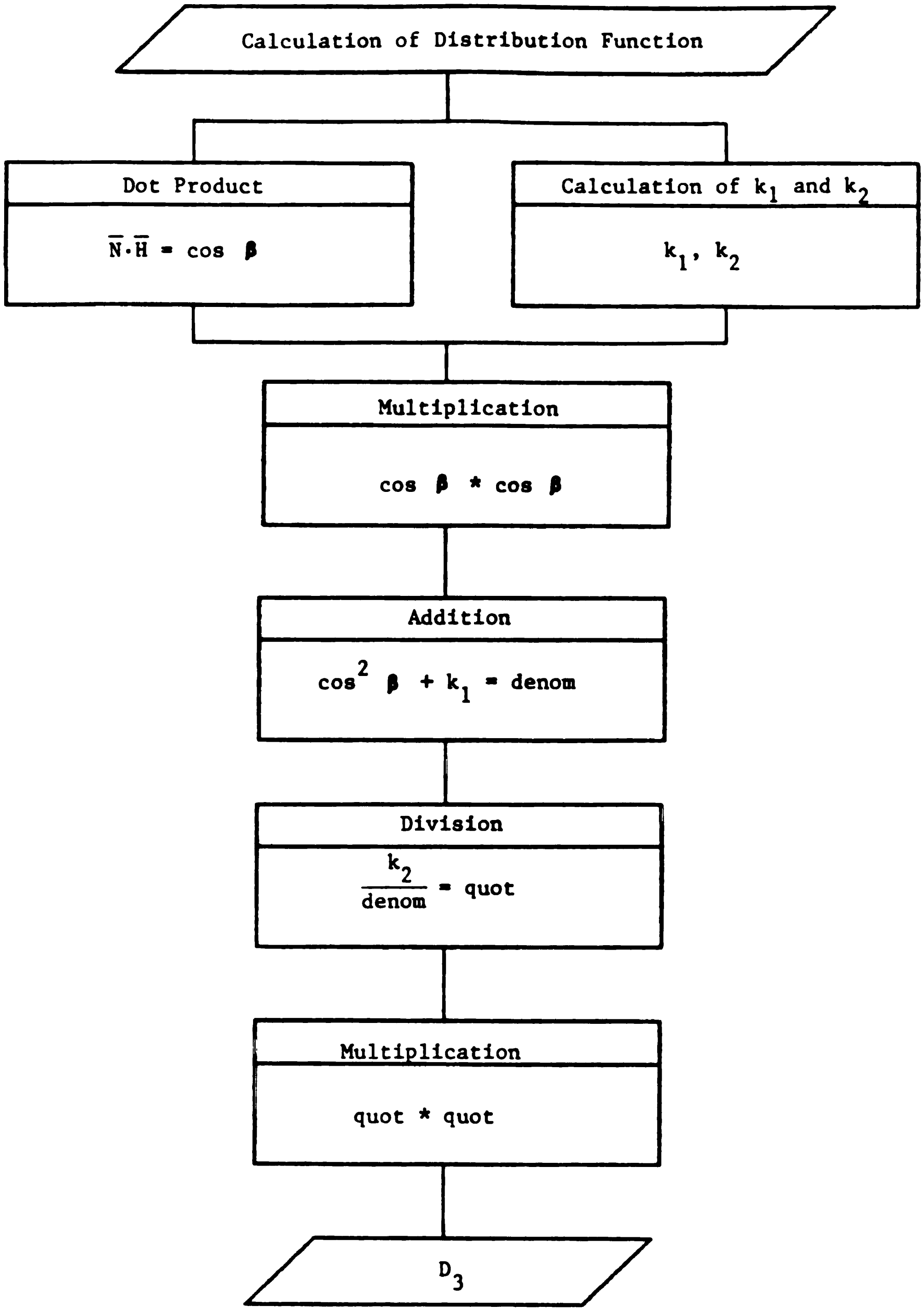

Figure 4.15: Functional-block diagram for calculating the distribution of the facets' orientations

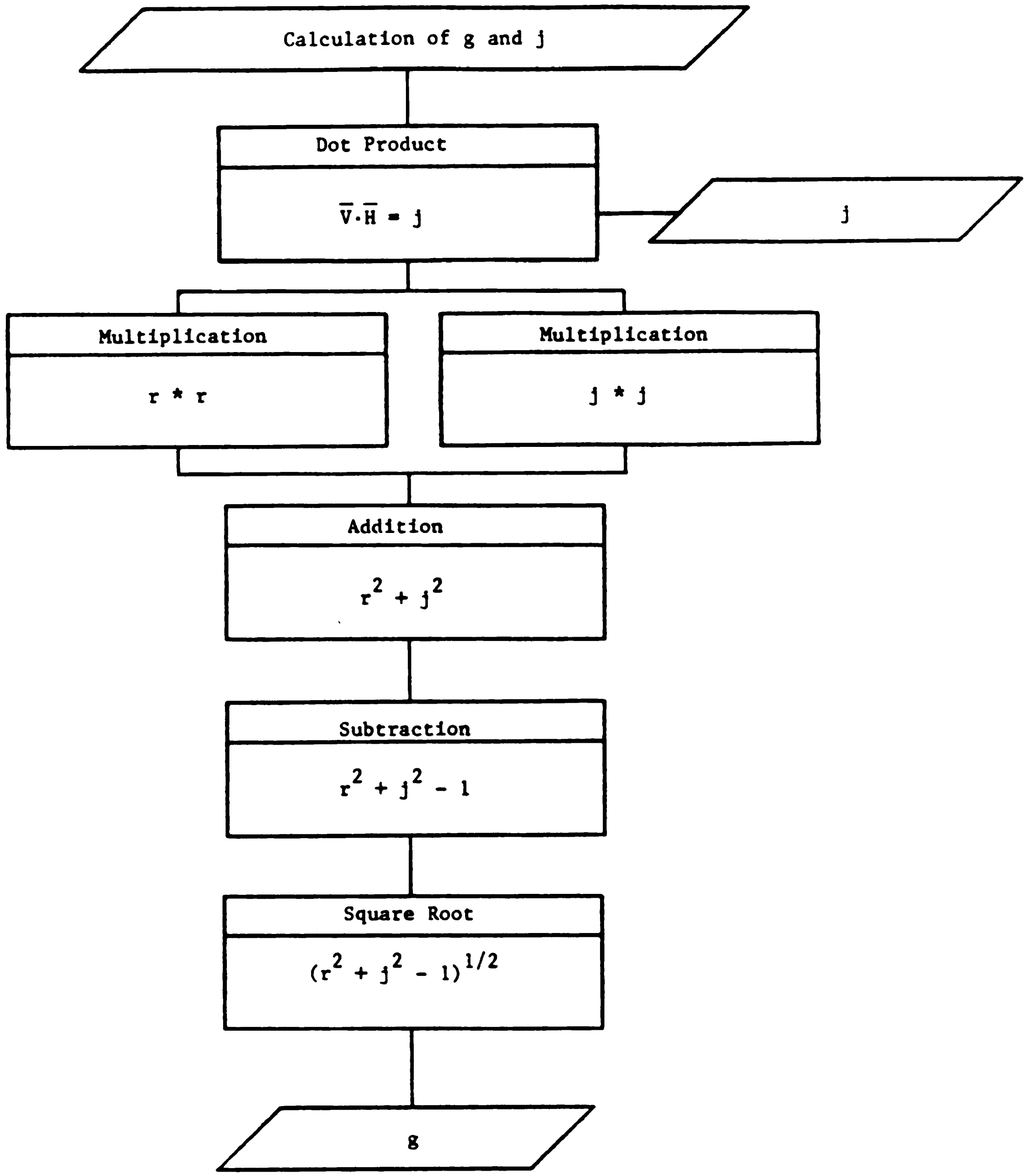

Figure 4.16: Functional-block diagram for calculating g and j

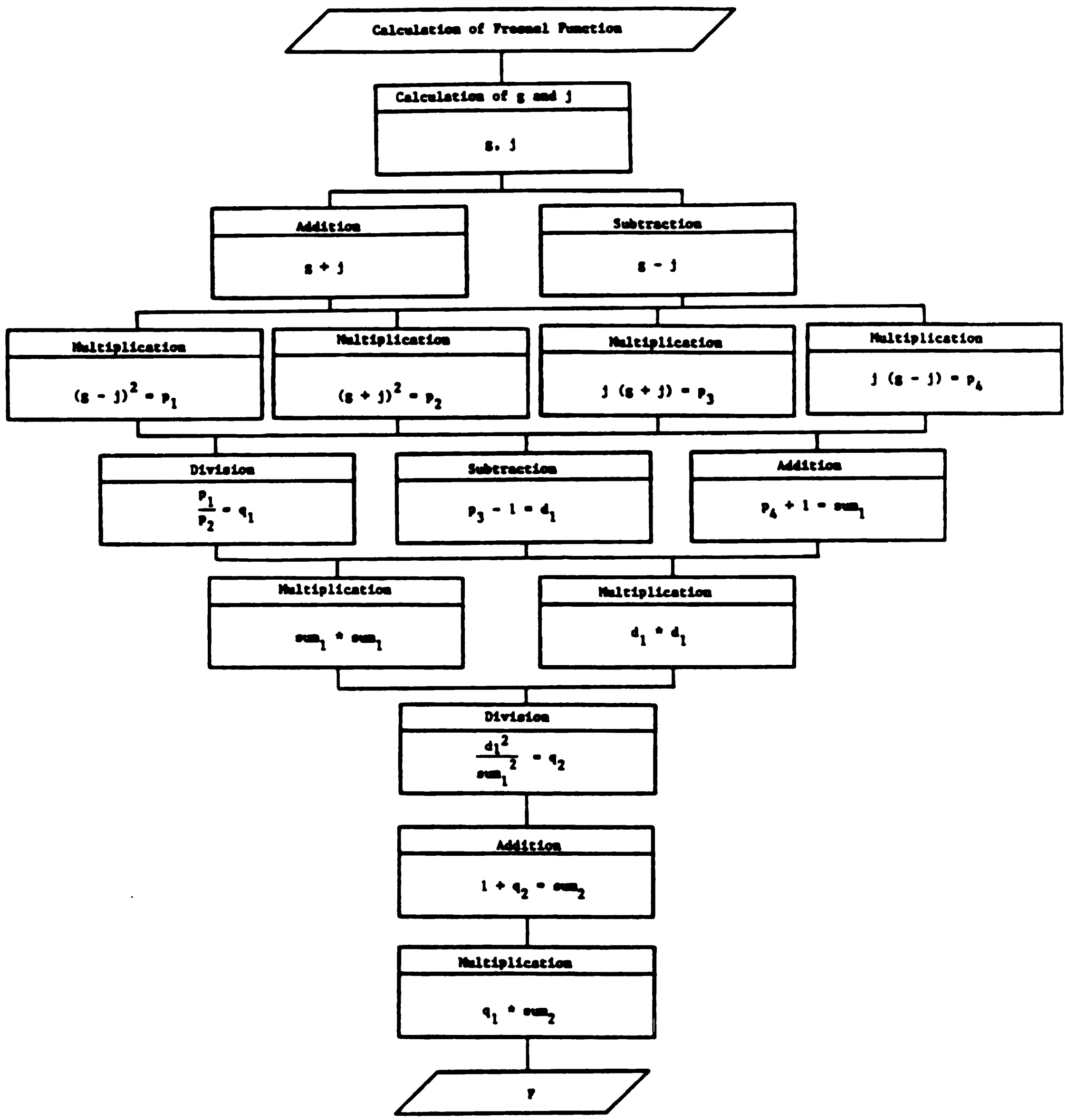

Figure 4.17: Functional-block diagram for calculating the Fresnel function

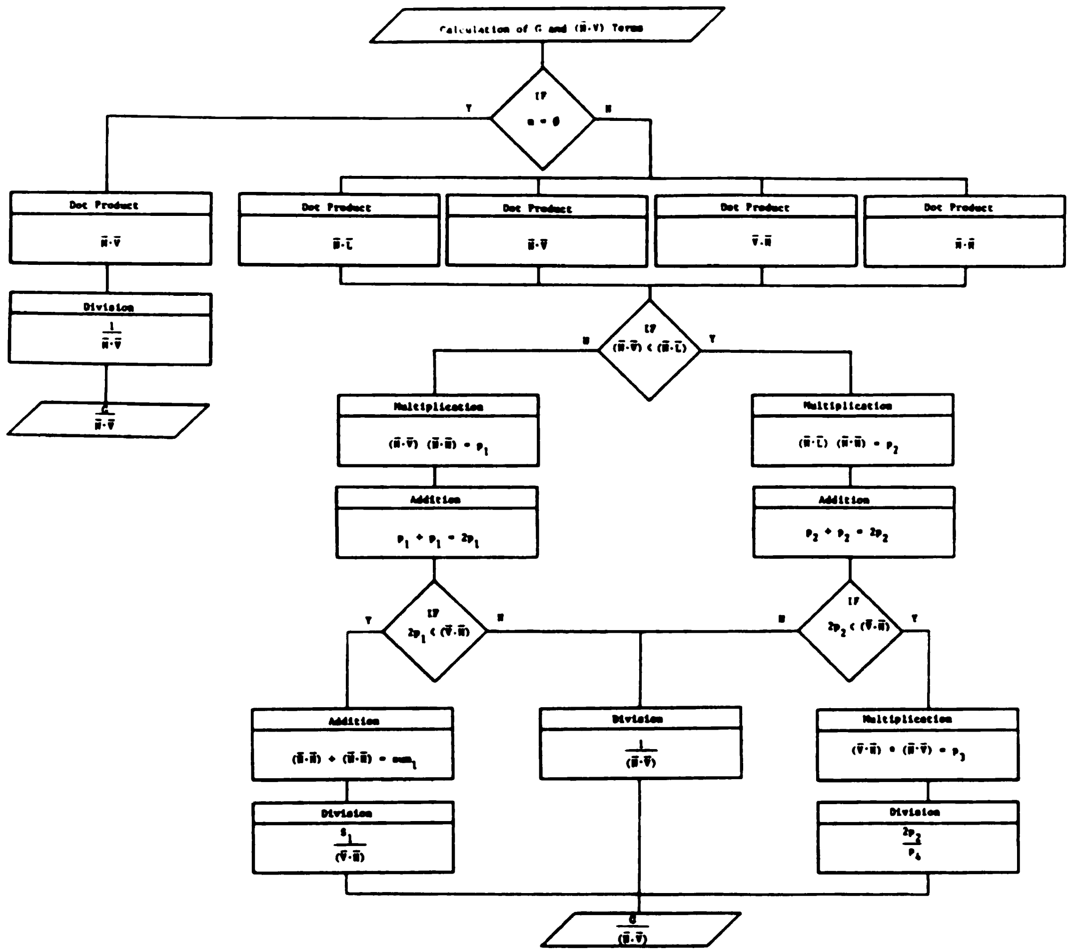

Figure 4.18: Functional-block diagram for calculating G and the $(\bar{N}\cdot\bar{V})$ terms

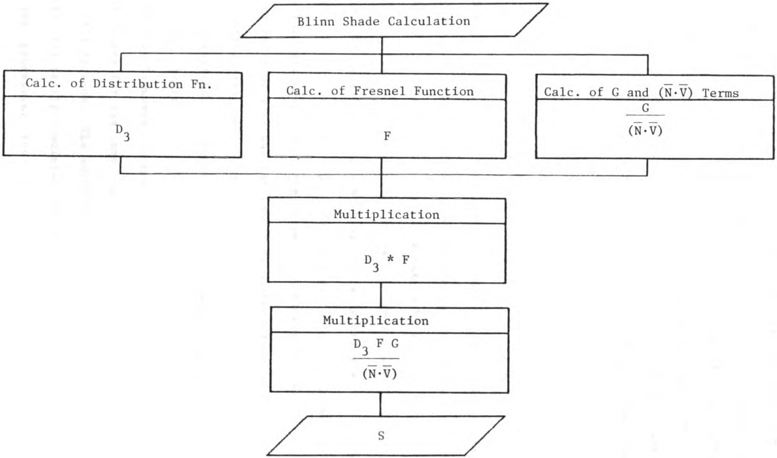

Figure 4.19: Functional-block diagram for Blinn's shading model

$\bar{H}$ calculations add 28,583 Δ_g to the hidden-surface algorithm's execution time. There is a one time delay of 5954 Δ_g to start the shading calculations, then the shading values for the 41 pixels covered by the image, each using 262 Δ_g, are calculated. The total delay for the shading portion of Blinn's algorithm is 16,696 Δ_g. Thus, Blinn's claim that his algorithm did not increase the execution time of Phong's algorithm has been verified.

4.2.4 Newell-Sancha Transformation

Newell, Sancha and Newell's algorithm, presented in Section 3.4, performs most of the hidden-surface removal and shading functions at the same time. It calculates shading information by polygon rather than scan line. Shading values must still be calculated per pixel but the values are constant throughout a polygon meaning less calculations are necessary.

The hidden-surface algorithm arranges the polygons in order of increasing distance from the viewer. Shading values are calculated for polygons starting with the furthest ones using equations (40), (41) and (42). After the polygon's shading value has been determined, the entire polygon is written into memory. In this manner polygons are placed in the memory and closer ones that obscure others simply overlap them. This automatically "removes" hidden surfaces but it is possible to calculate several values for the same pixel as will be evident from the example at the end of this section. The advantage of this method is that transparent surfaces are easily simulated though refraction effects are ignored. Also, no extra processing for the shading function is required in the hidden-surface algorithm since they are performed together in the shading calculation.

The shading calculation assumes that the incident angle is known and so the cosine and sine functions are evaluated directly. Figure 4.20 shows the functional-block diagram for the shading calculation. The total delay is 4894 Δ_g. If the closest object is transparent, when a shading value is recalculated for a pixel, the new shading value is a combination of the old and new values as found using equations (41) and (42) and illustrated in Figure 4.21. The transparency calculation is basically an interpolation but the weighting factor is already known and does not have to be calculated; however, a magnitude comparator must be added to the system hardware. The total delay of this computation is 206 Δ_g.

Overall, the equations can be arranged into a three-stage pipeline with a stage delay of 1536 Δ_g by breaking up the exponentiation function as was done for previous algorithms. The last three operations in Figure 4.20 plus the entire transparency calculation can be combined as the third stage of this pipeline. Thus, the pipeline delay is (N + 2) 1536 Δ_g. In this case parallel pipelines are not as necessary because fewer calculations need to be performed since the data is calculated per polygon instead of per pixel. Thus, the only extra hardware needed for the system is a shift register and a magnitude comparator.

When applied to the image in Figure 4.2, this algorithm calculates a shading value for each of the six polygons comprising the object. Once the shade is determined it is assigned to each pixel covered by the polygon. Using the pipeline, the total delay is 12,288 Δ_g. Though the delay is not too extensive, some of it is unwarranted because the algorithm is not

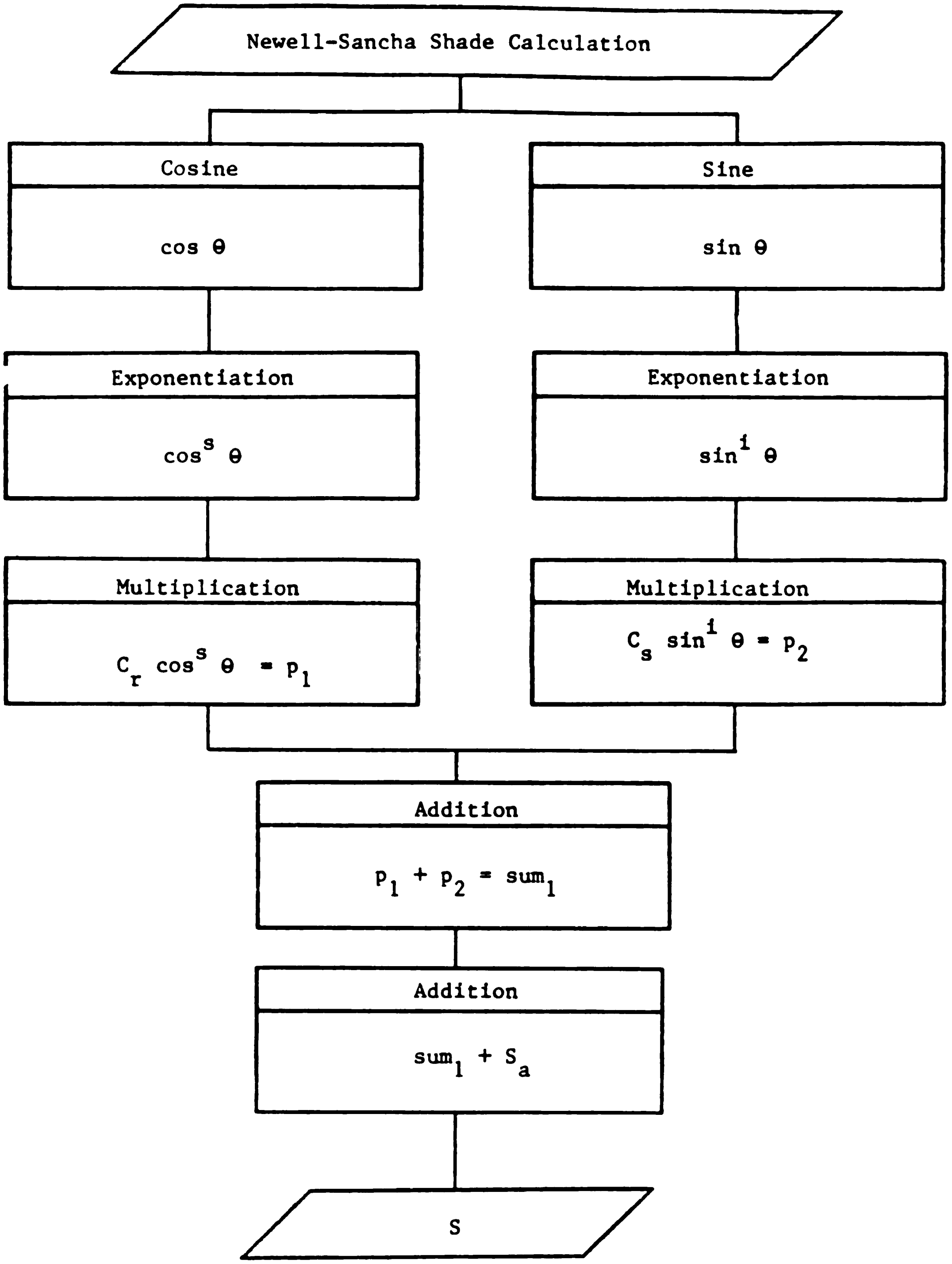

Figure 4.20: Functional-block diagram for Newell et al.'s shading model

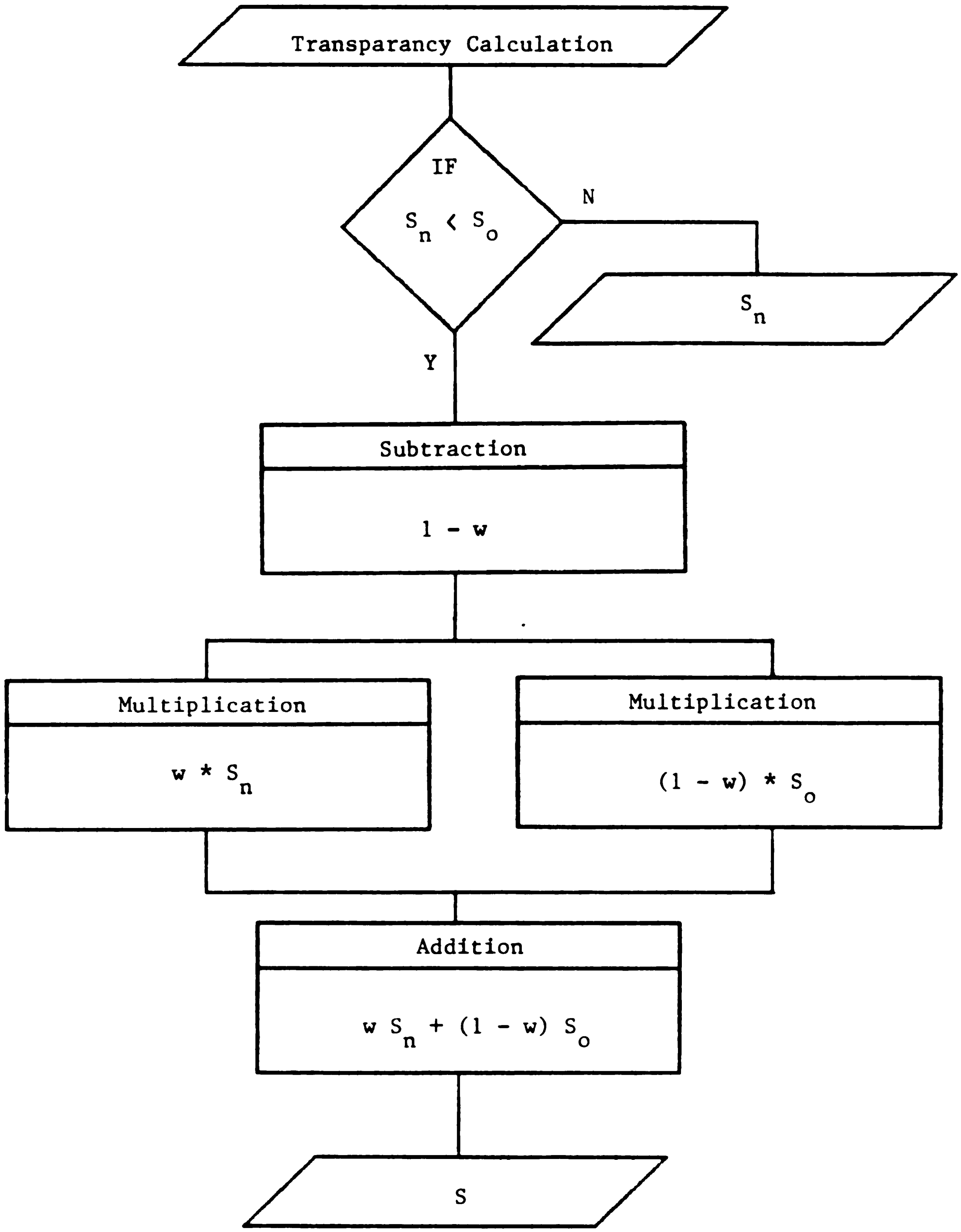

Figure 4.21: Functional-block diagram for calculating shading values for transparent objects

efficient. Assuming the image of Figure 4.2 is opaque, each pixel will be replaced by another shading value at least once. Worse yet, pixels on edges common to two or more polygons are replaced more frequently. Thus, at least half of the execution time could be eliminated by a hidden-surface algorithm in this example image with the object being opaque. Now assuming that the polygon AEFB is an opening rather than a surface, seven of the 41 pixels have only one shading value assigned to them but the seven are a small percentage of the total so time may be saved by using a hidden-surface algorithm. However, whether or not an actual time-savings can be realized by using a hidden-surface algorithm would depend on its implementation.

4.2.5 Whitted Transformation

This algorithm, presented in Section 3.5, can be applied to planar-polygons or curved-patches; the polygon implementation is discussed in this section. Whitted assumes that the normal vectors and directions of the light source and viewer are known. Nevertheless, the vectors for the reflected light, $\bar{R}$ and transmitted light, $\bar{T}$, are calculated during the hidden-surface algorithm using the method of ray-tracing described in Section 3.5.

Information about the way light illuminates the object is calculated on a global basis during the ray-tracing procedure. Ray-tracing is performed by starting at a point on the surface and calculating $\bar{R}$ and $\bar{T}$ from that point. These new vectors are then followed until they reach another surface where they become the new incident rays and the procedure is repeated. Therefore, these calculations are sometimes performed several times starting from a single pixel. Equation (43) calculates $\overline{V}'$ and the functional-block diagram is shown in Figure 4.22. The total execution time is 2614 Δ_g. A pipeline configuration could be used but is not worthwhile as will be explained.

$\overline{V}'$ is used to calculate $\overline{T}$ and $\overline{R}$. The functional block diagram for equation (44) that computes $\overline{R}$ is shown in Figure 4.23 and has a delay of only 24 Δ_g after $\overline{V}'$ is known. $\overline{T}$ is found using Snell's law to simulate refraction effects through transparent objects. The calculation of the coefficient k_f is diagrammed in Figure 4.24. Note that the vector magnitude procedures in the second step do not include the square root step of the usual procedure so their delay is only 155 Δ_g. The remaining portion of the $\overline{T}$ calculation is shown in Figure 4.25. The total delay, not including the $\overline{V}'$ calculation, is 2882 Δ_g. It is not worthwhile to pipeline this computation because of the data dependency among the various steps; none can proceed until k_f is known but the delay for k_f practically causes the entire delay of the $\overline{T}$ calculation.

However, a two-stage pipeline could be used where the first stage finds $\overline{V}'$ and the last stage finds $\overline{R}$ and $\overline{T}$. The stage delay would be 2906 Δ_g so the pipeline delay is (N + 1) 2906 Δ_g. The required hardware includes six multipliers, five adders, one complementor, two dividers and a shift register which are within the limits of the processor model. Using three of these pipelines in parralel, two multipliers, seven adders, three dividers and three shift registers need to be added to the system.

The shading calculation of equation (45) is diagrammed in Figure 4.26. The total delay is 298 Δ_g. This computation can be pipelined into three stages with a stage delay of 155 Δ_g. The first stage includes the first

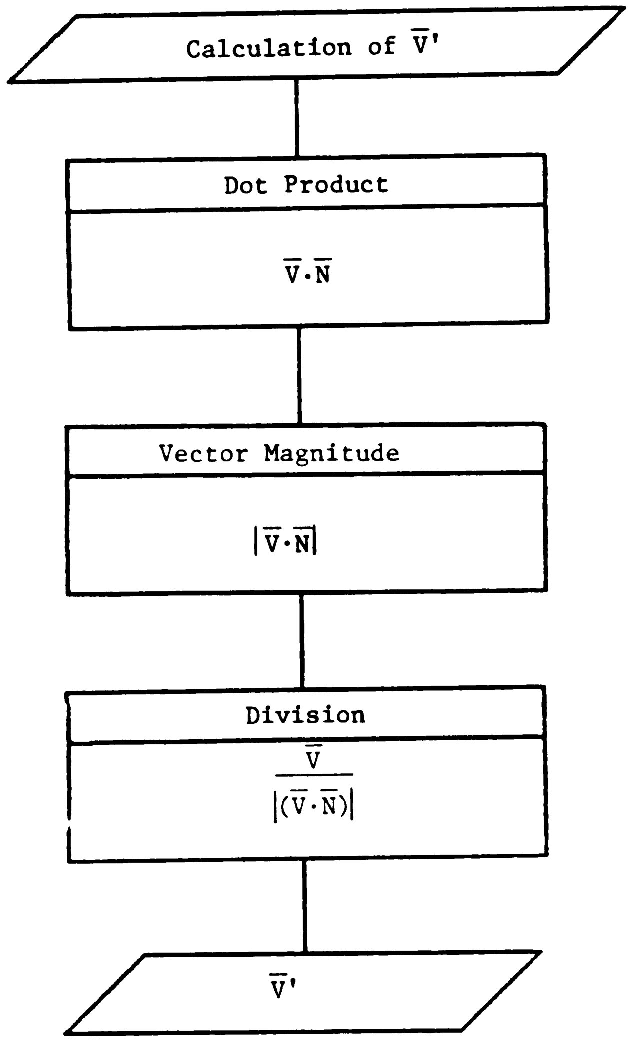

Figure 4.22: Functional-block diagram for calculating $\overline{V}'$

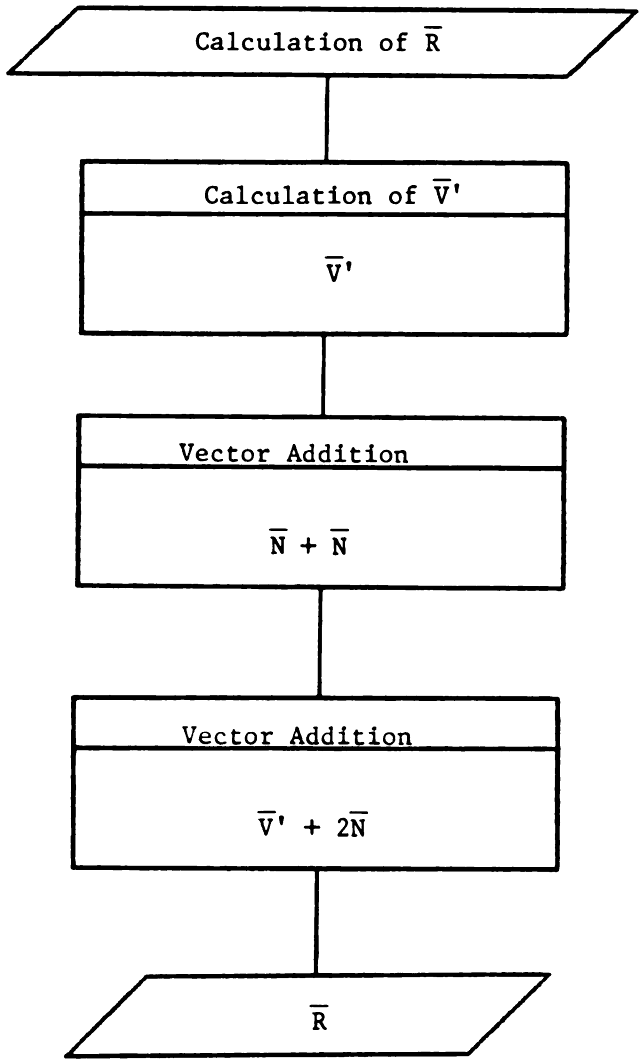

Figure 4.23: Functional-block diagram for calculating the direction of reflection

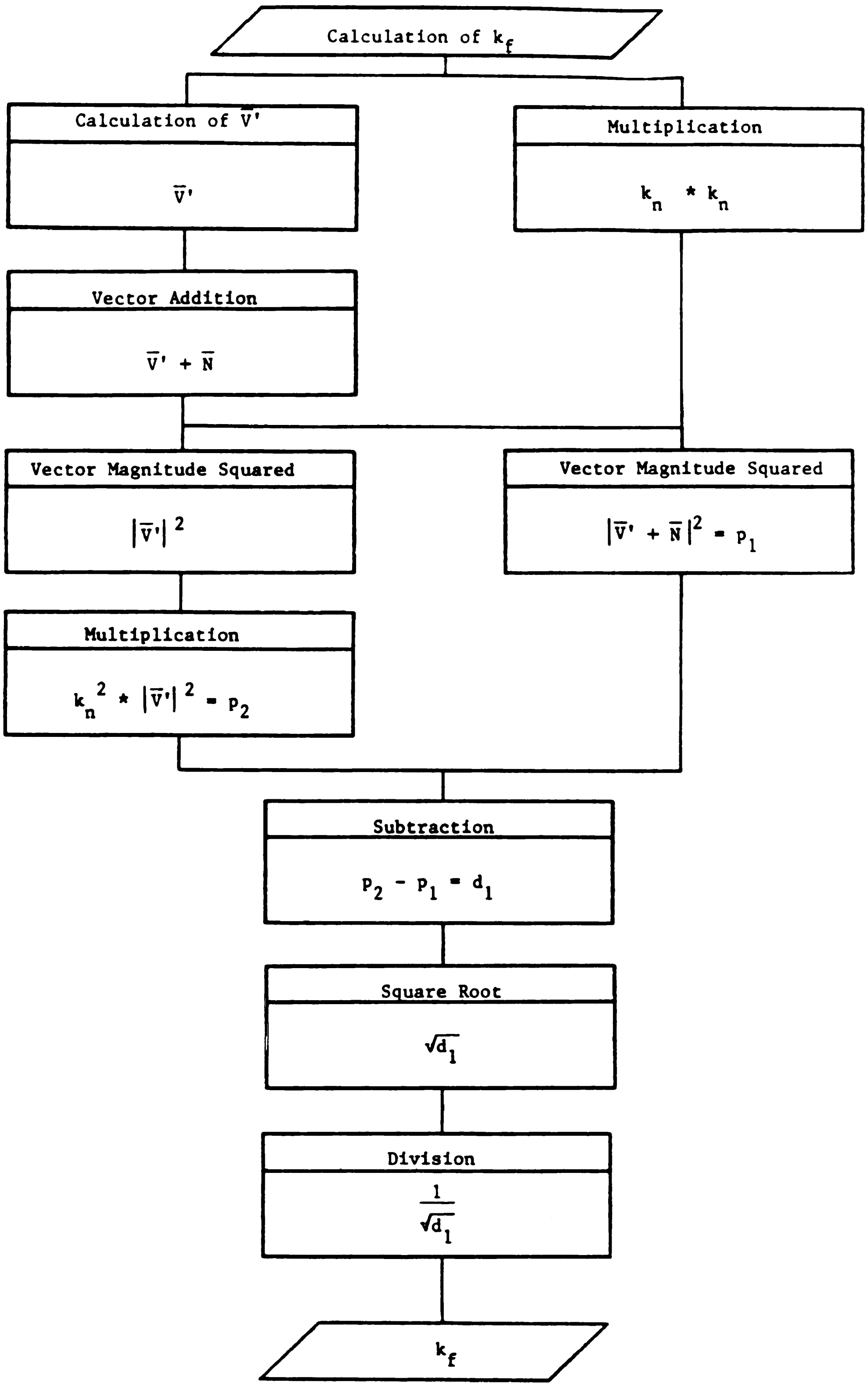

Figure 4.24: Functional-block diagram for calculating k_f

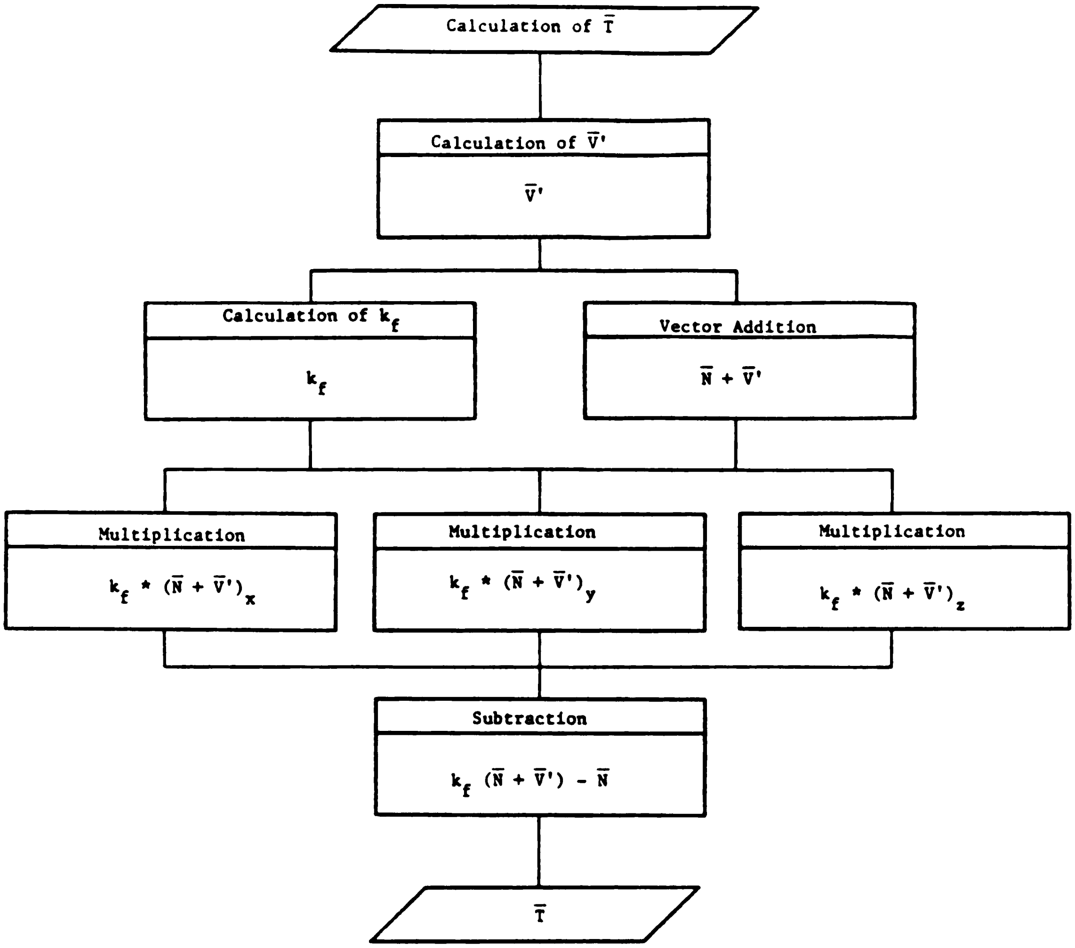

Figure 4.25: Functional-block diagram for calculating the direction of transmitted light

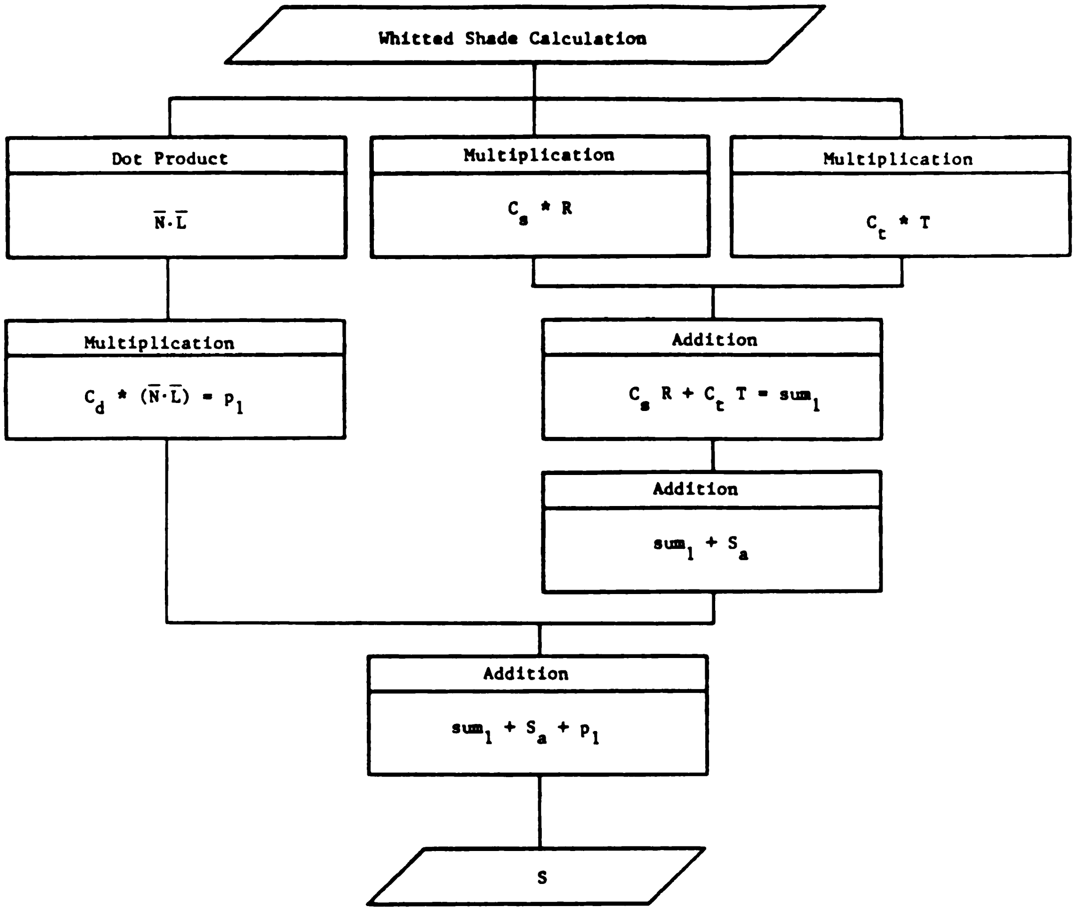

Figure 4.26: Functional-block diagram for Whitted's shading model

three parallel operations. The second stage has the multiplication performed parallel to two successive additions. The last stage is the last addition. The total pipeline delay is (N + 2) 155 Δ_g. Even with three parallel pipelines the procedure does not need any extra hardware.

Applying the ray-tracing procedure to the image in Figure 4.2 is somewhat trivial if the object is opaque because there is only one object and, therefore, no surfaces for the $\overline{R}$ and $\overline{T}$ vectors to bounce off of. There are 82 pixels in both the visible and hidden faces but since the light rays never strike the hidden-surfaces only 41 pixels require one ray-tracing calculation each. Each pipeline computes values for 13 pixels with two doing an extra. The total delay added to the hidden-surface algorithm is 43,590 Δ_g. After the hidden-surface algorithm is performed, only 41 pixels are visible so the delay for Whitted's shading procedure is 2480 Δ_g when N is 14. So the ray-tracing computations are very time-consuming but the shading is very fast. If part of the object is transparent, the number of vectors to calculate is greater and the ray-tracing execution time increases but not the shading time. Assuming polygon ABFE is transparent, seven extra sets of $\overline{R}$ and $\overline{T}$ vectors must be calculated because the initial $\overline{T}$ values will strike the previously hidden faces. Thus, the ray-tracing time increases to 49,404 Δ_g; an increase of 5814 Δ_g.

4.2.6 Catmull Transformation

Catmull's algorithm is presented in Section 3.6. It does not actually present a specific shading model but claims any can be used that are based on the surface normal. The algorithm can be applied to both polygons and curved-patches but is presented for curved-patches called bicubics. First, surface normals are approximated for each patch. This is a very time-consuming process attributed to the hidden-surface algorithm. Next, the object model is modified using a subdivision algorithm to break the patches into pixel-size subpatches. The approximated surface normals are divided with the patches so that each subpatch has its own normal.

The normals are approximated using several matrix multiplication operations. All of the operations have at least one 4 x 4 matrix; several involve the vectors U and V from equations (48) and (49) since the bicubic equations are functions of u and v. It is assumed that the first and second derivatives of these vectors are known. Values for (u,v) that are of particular interest are (0,0), (0,1), (1,0) and (1,1). If these values are plugged into the vectors and stored as constants for use during the matrix multiplications, long strings of summands can be avoided because the additions can be performed in parallel with the multiplications. Otherwise, each summand as found in Figure 4.27 would have a possible 256 terms, plus the number of multiplications would increase because polynomials are being multiplied. By using the constant vectors, the number of summands that must be generated increases but there is an overall savings in time.

The time analysis for the normal approximation is only discussed for one component of the bicubics, x. The other two are calculated in a similar procedure that requires the same amount of time and hardware. Referring to Figure 4.27, the summands are calculated for equations (53), (59), (60) and (61). Since matrix multiplication is associative, the rightmost pair are multiplied first to fully utilize the hardware and minimize the delay. The matrix multiplication of a m x c matrix and a c x p matrix is set up so that

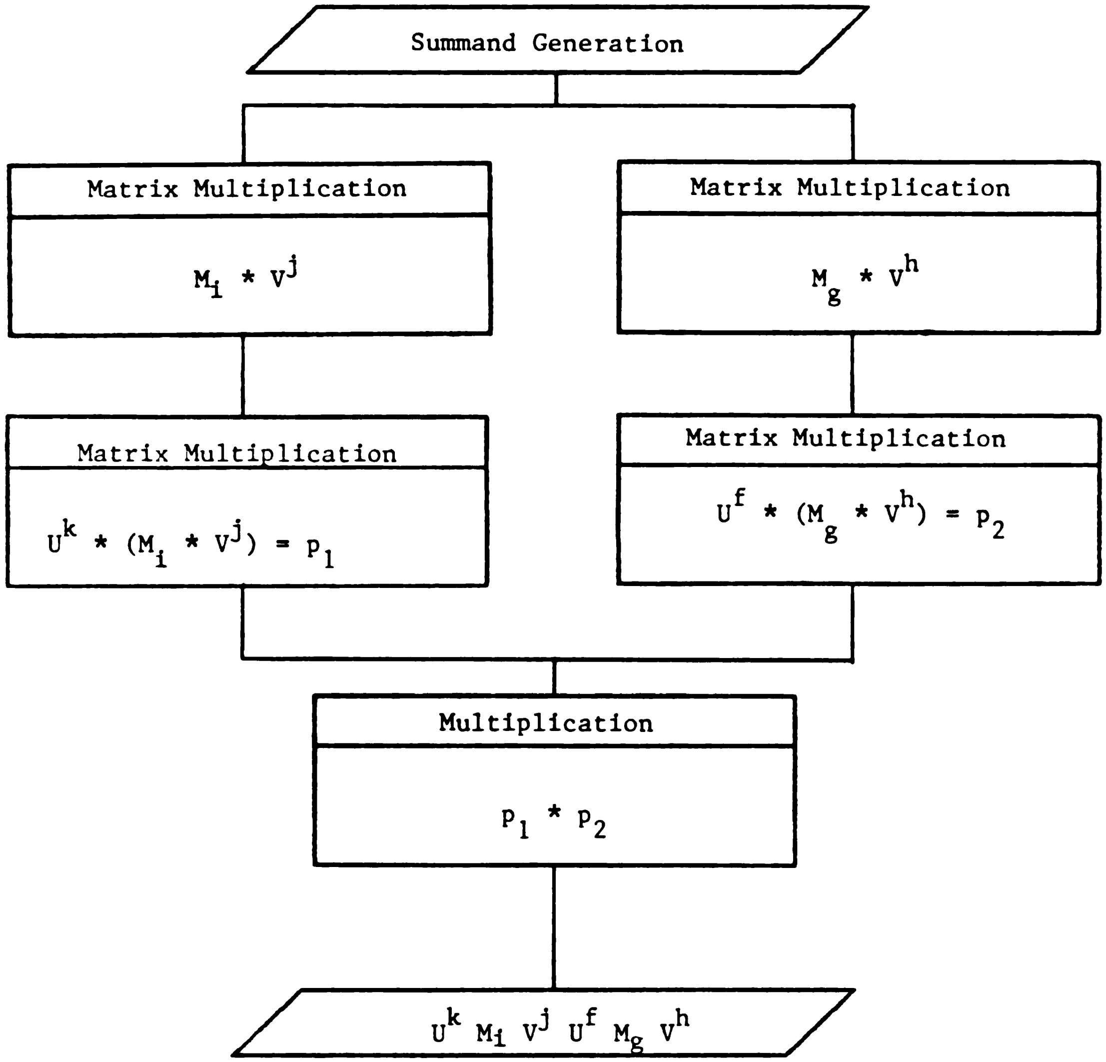

Figure 4.27: Functional-block diagram for summand generation

the delay is dependent on the value of p. For V^j the value of p is one so the delay is minimal. The product of the first step is also a 4x1 matrix so the delay of the second matrix multiplication is also minimized. The total delay to generate a summand is 441 Δ_g. This can be made into an efficient, three-stage pipeline because each of the stages has nearly the same delay. The pipeline's delay is (N + 2) 155 Δ_g.

Using these summands, equations (53), (59), (60) and (61) can be evaluated using the summation and complementing processes. In equation (53) x_n has two terms so the delay is (12 + 3) Δ_g. The number of summands in equations (59) and (60) is four. Complementations can be performed in parallel before the summation begins. The total delay is 27 Δ_g. Equation (61) has eight summands and its delay is 39 Δ_g. These summations can be performed as the summands are generated so the delays are insignificant. The extra hardware necessary for the pipeline and summations is 49 multipliers and 43 adders.

The final computation for the normal is a 4 x 4 multiplication as in equation (56) and shown in Figure 4.28. The value of p is four so each stage takes 524 Δ_g. Forming a two-stage pipeline, the total delay is (N + 1) 524 Δ_g. The necessary hardware is covered by the extra added for the extra hidden-surface calculations. However, all of the hardware must be tripled so that all three components can be calculated in parallel. The total additional hardware is 179 multipliers and 145 adders which is significantly more than that needed by any of the other algorithms.

According to Catmull, each component requires 30 additions during each subdivision in the subdivision algorithm. The components may be calculated simultaneously. Catmull estimates that the number of subdivisions needed to

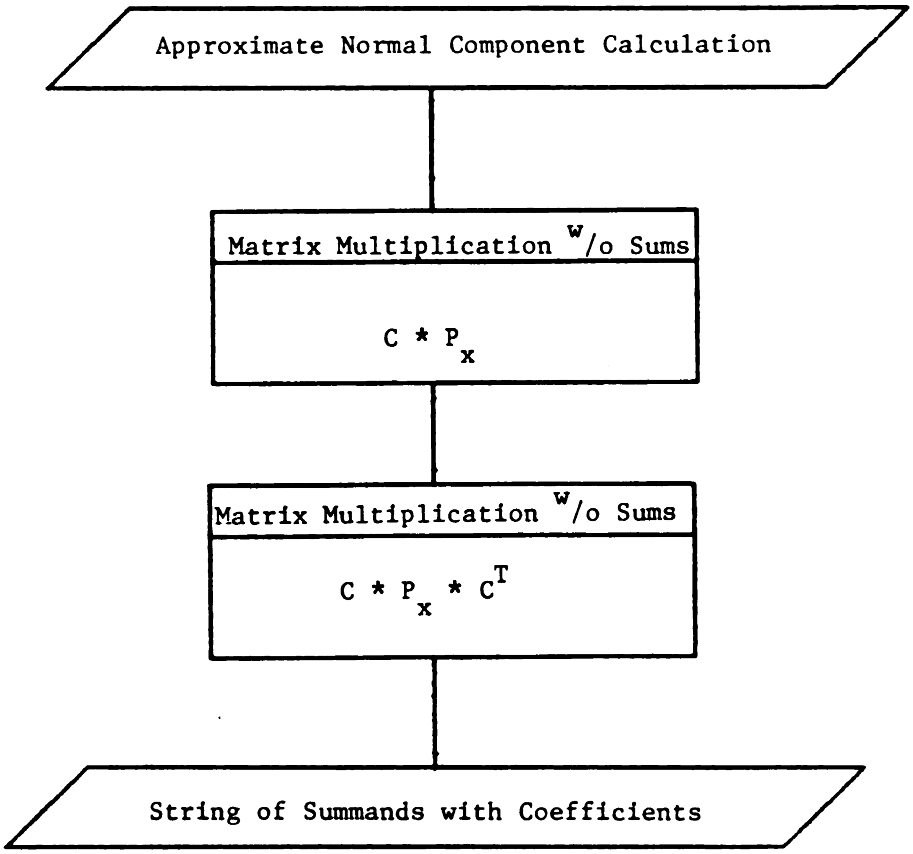

Figure 4.28: Functional-block diagram for approximating one normal component for a bicubic patch

completely divide a patch is slightly greater than one third of the number of pixels covered by the patch. The total delay added to the modeling portion of the processing is then (360 * sub) Δ_g where sub is the number of subdivisions required.

Applying this algorithm to the gizmo in Figure 4.2, it requires slightly greater than 35 subdivisions when edge pixels that are common to two patches are counted twice; once for each patch. This means an increase in the modeling procedure of 12,600 Δ_g. Normals are approximated per patch so six are needed here. This adds 3668 Δ_g to the hidden-surface algorithm's execution time. Altough the additional delays are relatively short, the extra hardware is quite extensive.

4.2.7 Cook-Torrance Transformation

Cook and Torrance developed a shading model that is presented in Section 3.7. In their original paper they discussed color shaded images extensively. One of their conclusions is that objects made of certain materials have specular reflections and diffuse reflections of different colors. Since all of the other algorithms discussed in this report do not address the issue of colored images, this part of the Cook-Torrance algorithm is largely ignored.

Overall, the algorithm is very similar to Blinn's algorithm discussed in Sections 3.3 and 4.2.3 because both assume that the surface is made up of randomly oriented microfacets. Yet Cook and Torrance assume that the direction of the specular reflection, $\bar{H}$, is known. If this is calculated during the hidden-surface algorithm as in Section 4.2.3, which is shown in

Figure 4.12, then the delay is 2471 Δ_g which occurs only once per change in the light source's direction. The hardware needed for this calculation is within the limits of the system.

The shading calculation depends on the geometric attenuation factor, a division by $(\overline{N}\cdot\overline{V})$, the Fresnel Refraction Law and a distribution function for the facets' orientations. The calculation of G and the $(\overline{N}\cdot\overline{V})$ division are performed exactly as shown in Figure 4.18 and discussed in Section 4.2.7. Therefore, their delay is a maximum of 1227 Δ_g. The Fresnel function calculation is the same as Blinn's in Figures 4.16 and 4.17 except the final value is multiplied by one half in this model. However, this multiplication can be performed in parallel with the last set of simultaneous multiplications so the execution time is unchanged at 3832 Δ_g. The distribution function used is different from Blinn's and is presented as equation (71). Its functional-block diagram is shown in Figure 4.29 and its total delay amounts to 4105 Δ_g. These four functions are used to calculate the specular component of the reflected light, R_s. R_s is found using equation (69) and its functional-block diagram is presented in Figure 4.30. The total time needed for this calculation is 4105 Δ_g for G, D, F and $(\overline{N}\cdot\overline{V})$ which are found simultaneously and only once per change in light source direction. The remaining time needed per pixel is only 262 Δ_g.

The shading model calculation is diagrammed in Figure 4.31. R_a, the hemispherical-directional reflectance, and R_b, the bidirectional reflectance, are linear combinations of the diffuse and specular components of the reflectance; R_d and R_s, so they are calculated using the interpolation calculation of Figures 4.5 and 4.6 in Section 4.2.1. If the interpolation calculation is broken into three steps, the shading

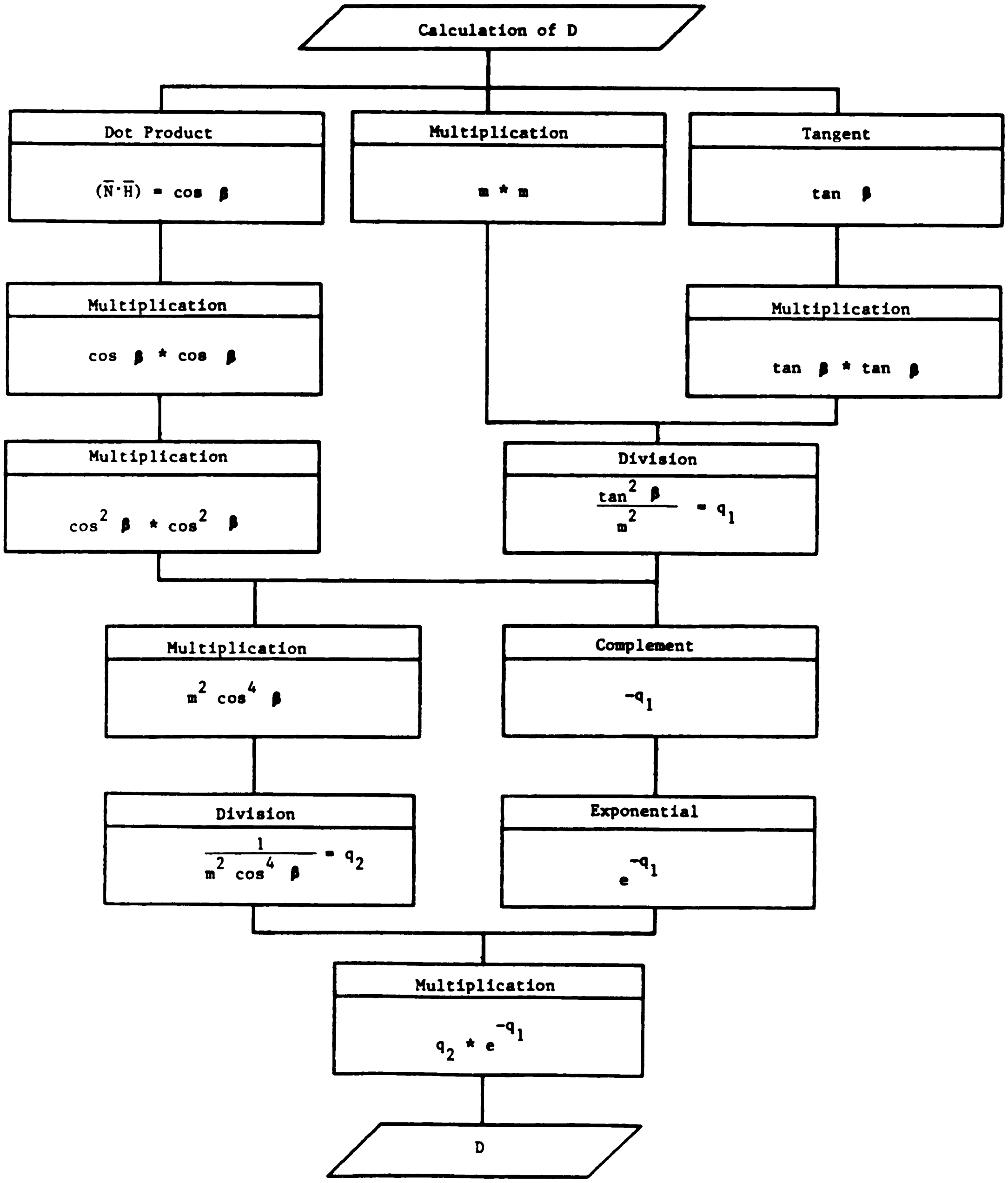

Figure 4.29: Functional-block diagram for calculating the distribution of the facets' orientations

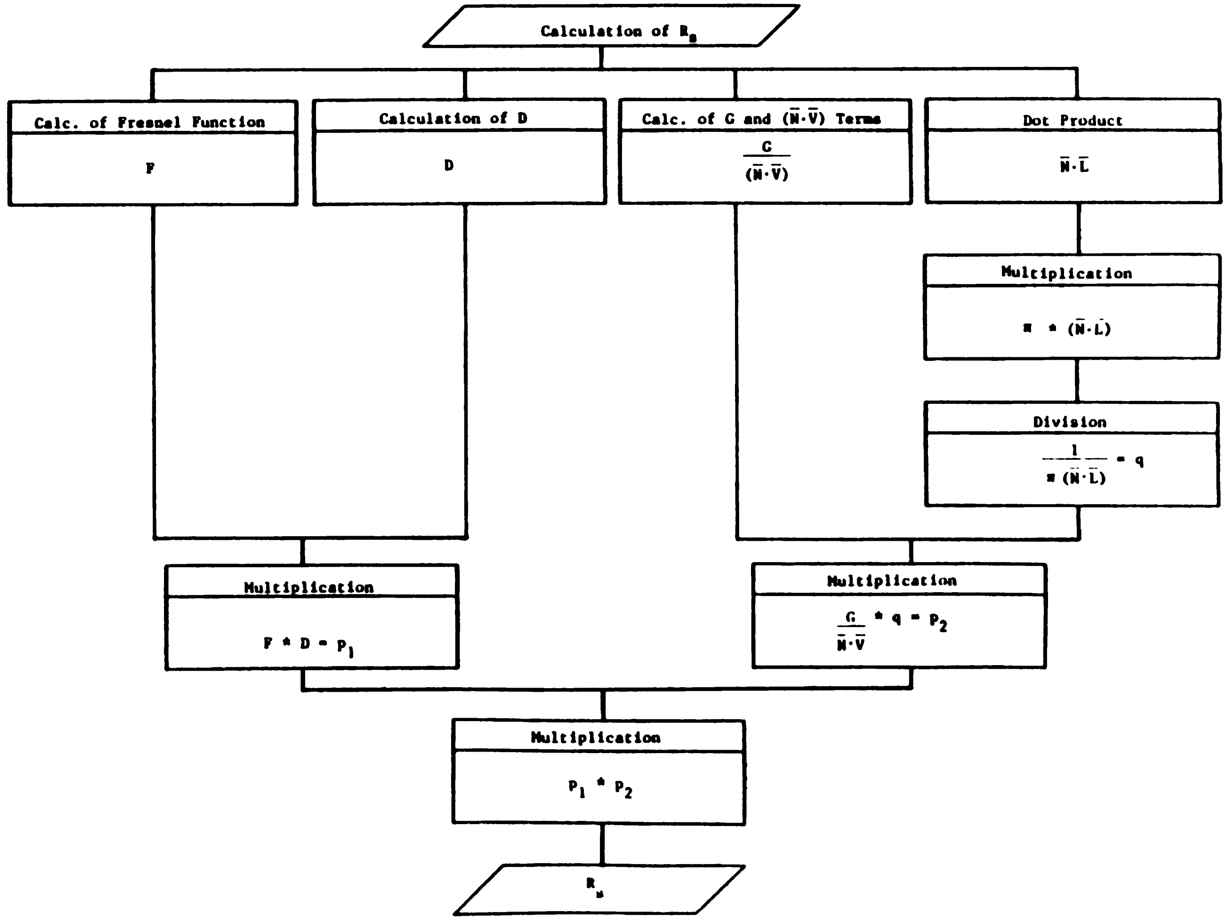

Figure 4.30: Functional-block diagram for calculating the spectral component of reflection

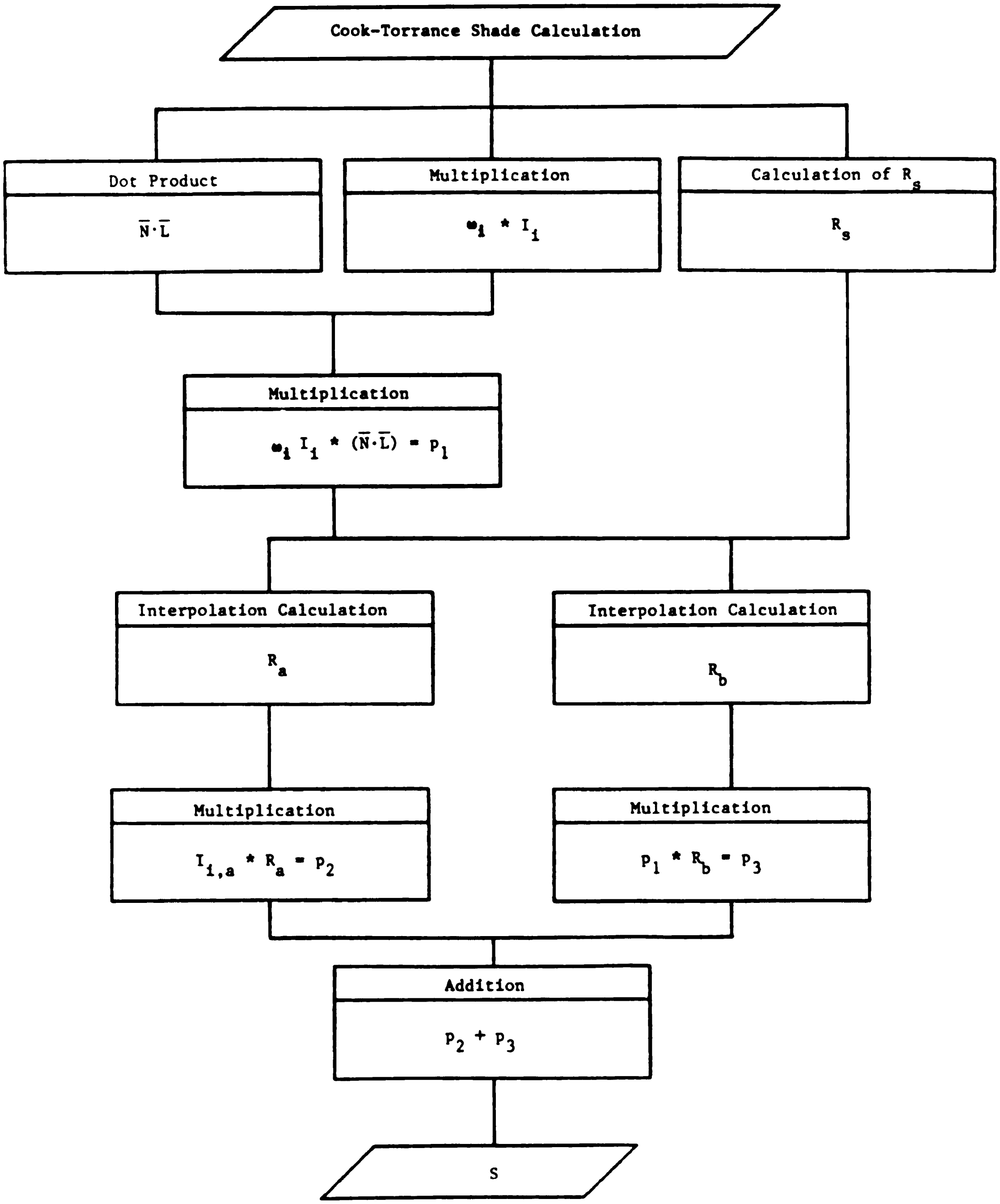

Figure 4.31: Functional-block diagram for the Cook and Torrance shading model

computation can be made into a five stage pipeline with a stage delay equal to that of division, 768 Δ_g. Thus, the total delay is (N + 4) 768 Δ_g plus a one time addition of 4105 Δ_g. Only three multipliers and one adder need to be added to the hardware. If three of these pipelines are used in parallel, three magnitude comparators, three complementors, 20 multipliers and one adder must be added to the system.

When applied to the image in Figure 4.2, the time added to the hidden-surface algorithm is still 2471 Δ_g. To calculate shading values for the 41 visible pixels requires 17,929 Δ_g where N = 14.

4.2.8 Fournier-Fussell-Carpenter Transformation

This last algorithm uses a technique called fractalization to alter the object modeling technique to make images of natural objects appear more realistic. It is presented in Section 3.8. The object to be modeled is composed of planar, triangular polygons. This algorithm divides the triangles into smaller triangles until none represent more than a single pixel in the raster.

The algorithm finds the midpoints of the three sides of a triangle then uses the points to break the triangle into four smaller triangles. The process is transformed into a functional diagram as in Figure 4.32. The total delay is 780 Δ_g per triangle that is divided. If three of these circuits are used in parallel, the extra hardware required by the process is 24 dividers and 19 adders.

After the new object model has been completed, the hidden-surface algorithm has to calculate the normals to the triangles so that any of the

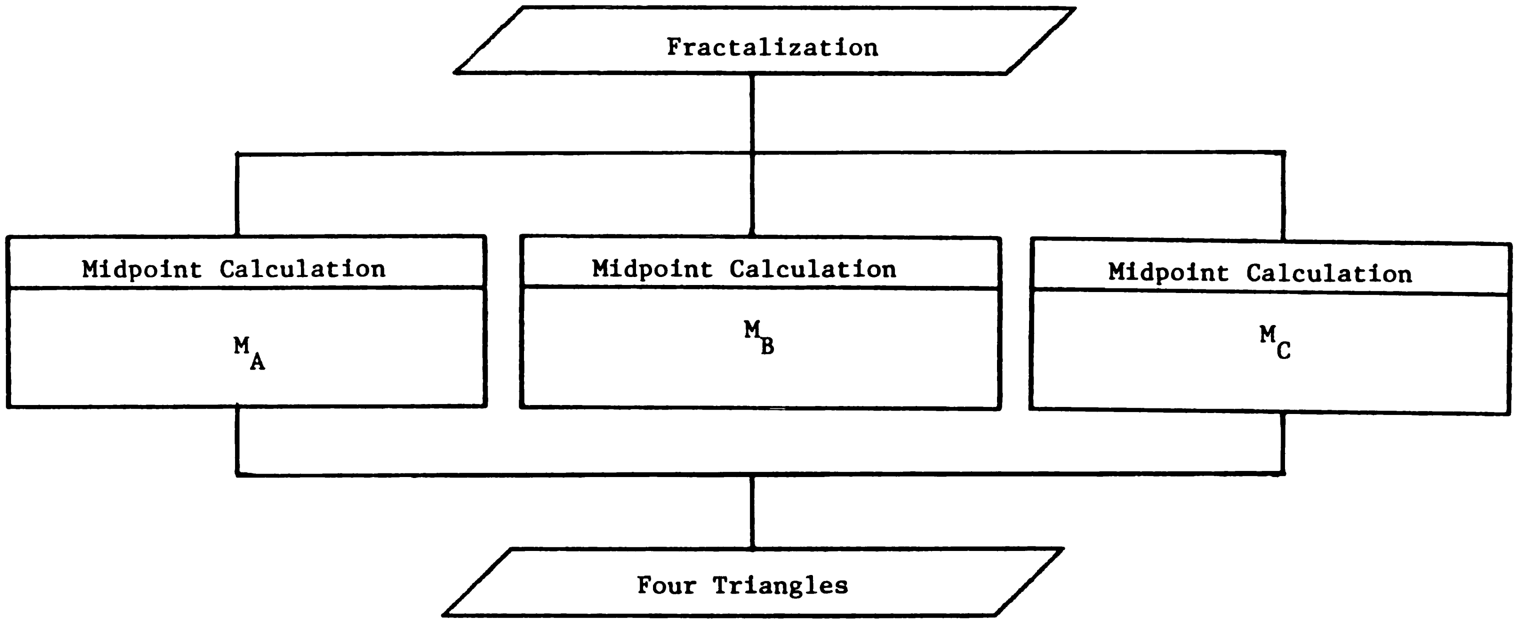

Figure 4.32: Functional-block diagram for fractalization calculations

shading models previously discussed can be applied. This process requires 161 Δ_g per normal calculated, or per triangle. The number of triangles is likely to get very large so three normal-calculator circuits are used in parrallel. No additional hardware is necessary.

Before the algorithm can be applied to the image in Figure 4.2, the polygons must be broken into triangles. Any polygon can be broken into triangles by starting at one vertex and connecting it to all of the other vertices except the two immediately beside the first. Applying this to the image in Figure 4.2, a total of 12 triangles are formed from both visible and hidden surfaces. There are a total of 81 pixels covered by all of these surfaces so there must be at least 81 triangles formed. This will require 22 fractalizations which will use 6240 Δ_g. The normal calculations will be performed for only the 41 visible triangles so they use 2254 Δ_g.

4.3 Comparisons

This chapter discussed the eight algorithms on the basis of their images' realism. Although realism is a subjective quality, it is important because it is often the basis of most viewers' opinions of the images. The most important criterion for raster graphics systems is the speed of execution because if successive frames take too long to generate, the quality of the image degrades to the point where flicker can be annoying. Section 4.1 described basic processes that could be used to transform the algorithms into functional-block diagrams to help analyze their execution times. The hardware requirements of these processes were assessed and formed the basic processor for the system. The functional-block diagrams were applied to various architectures to attempt to minimize the execution times. This often required additional hardware to be added to the system. Sometimes the algorithms added extra time and/or hardware to the hidden-surface removal or object modeling portions of the entire display process. All of these execution times and hardware requirements are summarized in Table 4.2.

Based on the time analyses performed, the Newell-Sancha algorithm requires both the shortest delays and the minimal extra hardware. This may be the reason why the images are not as realistic as any from the other algorithms. Gouraud's images are quite good yet the delays are not too great and the extra hardware is the second minimal. Gouraud utilized the cosine function rather than a dot product during the hidden-surface algorithm to compute contributions of light from the source; most of the other algorithms used dot products. The cosine requires more time to calculate than do dot products. Newell, et al., use the cosine function but calculate much fewer shading values overall so its extra time is not significant. Gouraud's extra time in the hidden-surface algorithm could be reduced but the cosine is not used to calculate all shading values so switching to the dot product may not generate too great of time-savings.

However, Gouraud's algorithm does not simulate specular reflections very realistically. Blinn and Phong improve on Gouraud's method with Blinn's algorithm being the faster and requiring less extra hardware. Still, these do not accurately simulate effects of transparency. Whitted's algorithm is capable of such effects and produces extremely realistic images. Whitted's shading model did not reproduce specular reflections as well as Blinn's or Phong's as the surface becomes less smooth but it does

Table 4.2: Summary of execution times and additional hardware

ALGORITHM	EXECUTION TIMES			ADDITIONAL HARDWARE						PORTION OF ALGORITHM
	HSR	MODEL	SHADE	*	÷	+	COMPL	S.REG	MAG.CMP	REQUIRING EXTRA HARDWARE
Gouraud	26633	Ø	9228	Ø	Ø	6	3	Ø	Ø	- Shading
Phong	26112	Ø	29184	Ø	9	13	3	Ø	Ø	- HSR (Hidden-surface Removal)
				11	Ø	6	Ø	3	Ø	- Shading
Blinn	28583	Ø	16696	2	12	17	3	1	Ø	- HSR
				Ø	Ø	Ø	Ø	Ø	1	- Shading
Newell	Ø	Ø	12288	Ø	Ø	Ø	Ø	1	1	- Shading
Whitted	43590	Ø	2480	2	3	7	Ø	3	Ø	- HSR
Catmull	3668	12600	---	179	Ø	145	Ø	Ø	Ø	- HSR
Cook	2471	Ø	17929	20	Ø	1	3	Ø	3	- Shading
Fournier	2254	6240	---	Ø	27	19	Ø	Ø	Ø	- Object Modeling

-- in the SHADE column implies that any other shading algorithm may be added for this function.

account for light reflected among the objects in the scene. However, his preprocessing before applying the shading model is time-consuming. Cook and Torrance's algorithm required less time than Blinn's and less overall hardware even though they were quite similar. Catmull's algorithm required significantly more hardware than any of the others due to approximating normals for the curved patches. For this reason this method for modeling is ruled out. Its delay is relatively short but this does not include any time for shading. The fractal method does not fare too badly in either time or hardware but does not account for the shading process either.

All things considered, Cook and Torrance's algorithm is the best all around method but it still does not simulate some effects, such as transparency. The very best solution is to combine several of these methods. Cook and Torrance's reflectance model could be used in most instances. Whitted's ray-tracing could be used to determine the effects of object reflections in other objects as well as transparent surfaces. And when mountains or other naturally random objects are displayed, the algorithm of Fournier, et al., could alter the object model for more realistic images. Of course, this demands much more hardware. Also, the control of such an implementation would be extensive to determine when to use which algorithm. But the system would display a wide variety of objects most realistically.

CHAPTER V

CONCLUSION

5.1 Summary

The purpose of this reseach was to present shading algorithms for computer graphics systems and to compare them first on the basis of speed of execution and, second, by the overall quality of the images generated. The report began by generally discussing the field of computer graphics. Graphics terminology was explained and display systems were described, particularly the raster graphics system. Problems of high-performance graphics were presented and discussed narrowing in on the problems associated with shading displayed images.

The third chapter presented eight algorithms available in the current literature that attempt to solve the shading problem. The shading process is closely related to the object-modeling and hidden-surface removal processes; so, the algorithms suggested improvements in one or more of these procedures. Generally, the object-modeling technique determines the method to model objects using numerical representations. There are two basic types, polygon-mesh and curved-patch. Most of the algorithms discussed were applied to objects modeled with polygons. The hidden-surface algorithm processes the model of the object to determine which parts of the image are visible to the viewer and eliminates those that are not. This processing often included calculating general information about the object's surfaces, such as computing the surface normals. Most of the algorithms performed some processing here. The shading process included using a shading or reflectance model to determine how much of the source illumination is being reflected from the object and in what manner. For example, most objects under certain circumstances exhibit specular reflections or highlights. The algorithms and their relationships among these three processes is depicted in Figure 5.1.

The algorithms were transformed into functional-block architectural representations to enable an assessment of their speed of execution. This is a critical characteristic for raster graphics systems if they are to operate in real-time. The functional-block transformations were applied to various architectures in an attempt to minimize the execution times of the processing. Then, all of the shading systems were used to shade the same simple image as a standard for comparison. The execution times and additional hardware requirements resulting from shading the image were tabulated in Table 4.2.

Results of this investigation suggest that the best single algorithm is that of Cook and Torrance because it adequately simulated effects of diffuse and specular reflections without requiring extremely long delays or excessive extra hardware. However, the best shading implementation is a combination of algorithms so that a broader range of objects and reflectance effects can be more realistically simulated and displayed.

5.2 Future Research

Several topics touched upon in this report may be researched further. First of all, one of the assumptions made before comparing the speeds of the

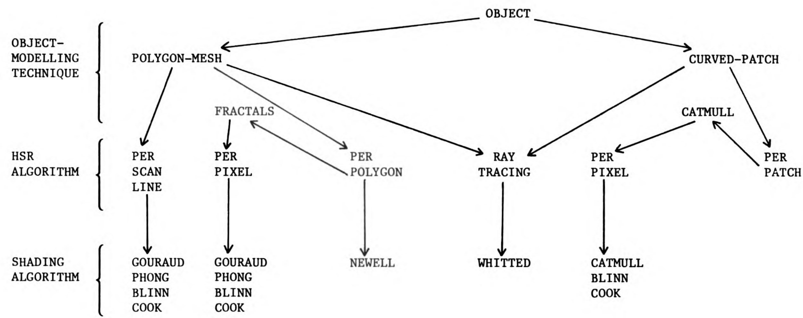

Figure 5.1: The relationships between the three processes and the algorithms

algorithms was that the memory did not affect the execution of these algorithms. This is not true but the assumption was made to simplify the comparisons. The relationship between the processor and the memory is crucial to the performance of a raster graphics system. Besides just the architecture, the way in which the memory is mapped affects how the information will be available for the algorithms to update the pixels.

Another area that should be investigated further is to find faster implementations of the arithmetic functions used as the functional blocks during the algorithm transformations. Both parallel and pipeline architectures might lead to faster or more efficient implementations. The regularity of these functions and architectures should be investigated to assess the suitability for VLSI or VHSIC implementations.

As stated, the algorithms were compared on the basis of first speed and then realism. Other criteria such as cost or area usage should also be used to compare these algorithms. And of course, new algorithms can always be included in the comparisons.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. D. R. Baldauf, "The Workhorse CRT: New Life", IEEE Spectrum, Vol. 22, No. 7. July 1985, pp. 67-73.

2. C. R. Baugh and B. A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm", IEEE Transactions on Computers, Vol 22, No. 12. December 1973, pp. 1045-1047.

3. J. F. Blinn, "Models of Light Reflection for Computer Synthesized Pictures", Computer Graphics, Vol. 11, No. 2. 1977, pp. 192-198.

4. R. R. Castleberry, "High-speed D/A Converters Yield Precision Graphics", Computer Design, Vol. 22, No. 11. November 1984, pp. 175-181.

5. R. Castleberry, "High-speed Video D/A Converters Simplify Graphics-display Designs", Engineering Design News. May 31, 1984, pp. 185-190.

6. E. Catmull, "Computer Display of Curved Surfaces", Computer Graphics, Pattern Recognition, and Data Structure Proceedings. May 1975, pp. 11-17.

7. E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces", Technical Report, Computer Science Department, University of Utah, December 1974.

8. J. H. Clark, "Designing Surfaces in 3-D", Communications of the ACM, Vol. 19, No. 8. August 1976, pp. 454-460.

9. J. H. Clark, "The Geometry Engine: A VLSI Geometry System for Graphics", Computer Graphics, Vol. 16, No. 3. July 1982, pp. 127-133.

10. R. L. Cook and K. E. Torrance, "A Reflective Model for Computer Graphics", Computer Graphics, Vol. 15, No. 3. August 1981, pp. 307-316.

11. M. J. Drumm, J. B. Harris and M. Ebertin, "Dual-port Static RAMs Can Remedy Contention Problems", Computer Design, Vol. 23, No. 9. August 1984, pp. 145-151.

12. J. D. Foley and A. van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Company, Reading, Massachusetts, March 1983.

13. J. A. B. Fortes, K. S. Fu and B. W. Wah, "Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays", International Conference on Acoustics, Speech, and Signal Processing. March 1985, pp. 300-303.

14. A. Fournier, D. Fussell, and L. Carpenter, "Computer Rendering of Stochastic Models", Communications of the ACM, Vol. 25, No. 6. June 1982, pp. 371-384.

15. H. Fuchs and B. W. Johnson, "An Expandable Multiprocessor Architecture for Video Graphics", Proceedings of the 6th Annual Symposium on Computer Arichitecture. April 1979, pp. 58-67.

16. H. Fuchs and J. Poulton, "Pixel-planes: A VLSI-oriented Design for a Raster Graphics Engine", VLSI Design. 1981, pp. 20-28.

17. A. Fujimoto, C. G. Perrott and K. Iwata, "A 3-D Graphics Display System with Depth Buffer and Pipeline Processor", IEEE Computer Graphics and Applications, Vol. 4, No. 6. June 1984, pp. 11-23.

18. H. Gouraud, "Continuous Shading of Curved Surfaces", IEEE Transactions on Computers, Vol. 20, No. 6. June 1971, pp. 623-629.

19. D. W. Gulley, "Joining Text and Graphics Enhances Video Performance", Computer Design, Vol. 23, No. 9. August 1984, pp. 121-129.

20. K. Hwang, Computer Arithmetic: Principles, Architecture and Design, John Wiley and Sons, New York, 1979.

21. J. H. Lawitzke, "The CORDIC Algorithm", Technical Report, Michigan State University, November 1984, Unpublished.

22. C. Machover and W. Myers, "Interactive Computer Graphics", IEEE Computer, Vol. 17, No. 10. October 1984, pp. 145-161.

23. M. E. Newell, R. G. Newell and T. L. Sancha, "A Solution to the Hidden Surface Problem", ACM Proceedings. 1972, pp. 443-450.

24. M. E. Newell, R. G. Newell and T. L. Sancha, "The Economic Application of a Half-tone Picture Generating Algorithm", Computer Aided Design, IEE Conference Publication 86, Southampton. April 1972.

25. M. B. Phillips and G. M. Odell, "An Algorithm for Locating and Displaying the Intersection of Two Arbitrary Surfaces", IEEE Computer Graphics and Applications, Vol. 4, No. 9. September 1984, pp. 48-58.

26. B. T. Phong, "Illumination for Computer Generated Pictures", Communications of the ACM, Vol. 18, No. 6. June 1975, pp. 311-317.

27. R. Pinkham, M. Novak and K. Guttag, "Video RAM Excels at Fast

Graphics", Electronic Design. August 18, 1983, pp. 161-172.

28. W. H. Righter, "CMOS 256-Kbit RAMs Are Fast and Use Less Power", Computer Design, Vol. 23, No. 9. August 1984, pp. 133-140.

29. M. Shanblatt, EE809 Class Notes, Michigan State University, 1984, Unpublished.

30. S. L. Stepoway, D. L. Wells and G. R. Kane, "A Multiprocessor Architecture for Generating Fractal Surfaces", IEEE Transactions on Computers, Vol. c-33, No. 11. November 1984, pp. 1041-1045.

31. I. E. Sutherland, R. F. Sproull and R. A. Schumacker, "A Characterization of Ten Hidden-surface Algorithms", Computing Surveys, Vol. 6, No. 1. March 1974, pp. 1-55.

32. J. E. Volder, "The CORDIC Trigonometric Computing Technique", IRE Transactions on Electronic Computers. September 1959, pp. 330-334.

33. J. S. Walther, "A Unified Algorithm for Elementary Functions", Spring Joint Computer Conference. 1971, pp. 379-385.

34. D. S. Whelan, "A Rectangular Area Filling Display System Architecture", Computer Graphics, Vol. 16, No. 3. July 1982, pp. 147-153.

35. T. Whitted, "An Improved Illumination Model for Shaded Display", Communications of the ACM, Vol. 23, No. 6. June 1980, pp. 343-349.

36. M. C. Whitton, "Memory Design for Raster Graphics Displays", IEEE Computer Graphics and Applications, Vol. 4, No. 3. March 1984, pp. 48-65.

37. T. Williams, "Semiconductor Memories: Density and Diversity", Computer Design, Vol. 23, No. 9. August 1984, pp. 105-116.

38. M. S. Young, "Use a CRT-controller Chip to Mix Text and Graphics", Engineering Design News. May 31, 1984, pp. 153-170.

39. "Computer Graphics Comes of Age, An Interview with Andries van Dam", Communications of the ACM, Vol. 27, No. 7. July 1984, pp. 638-648.

GENERAL REFERENCES

1. J. Acquah, J. Foley, J. Sibert and P. Wenner, "A Conceptual Model of Raster Graphics Systems", Computer Graphics, Vol. 16, No. 3. July 1982, pp. 321-328.

2. H. Fuchs, G. D. Abram and E. D. Grant, "Near Real-time Shaded Display of Rigid Objects", Computer Graphics, Vol. 17, No. 3. July 1983, pp. 65-72.

3. S. Harrington, Computer Graphics: A Programming Approach, McGraw-Hill, Inc., New York, 1983.

4. T. Ikedo, "High-Speed Techniques for a 3-D Color Graphics Terminal", IEEE Computer Graphics and Applications, Vol. 4, No. 5. May 1984, pp. 46-58.

5. M. M. Mano, Digital Logic and Computer Design, Prentice-Hall, Inc., New Jersey, 1979.

6. M. Novak and R. Pinkam, "Inside Graphics Systems, from Top to Bottom", Electronic Design. July 21, 1983, pp. 183-188.

7. R. Pike, "Graphics in Overlapping Bitmap Layers", Computer Graphics, Vol. 17, No. 3. July 1983, pp. 331-356.

8. D. S. Roark, "Toward Real-time Interactive Color Graphics", Computer Design, Vol. 21, No. 7. July 1982, pp. 167-173.