THESIS

ANALYSIS OF ROUGH SURFACE LIGHTING BEHAVIORS WITH OPENGL

by

Christopher P. Slattery

September 2001

Thesis Advisor:	Wolfgang Baer
Second Reader:	Samuel E. Buttrey

Report Documentation Page

Report Date 30 Sep 2001	**Report Type** N/A	**Dates Covered (from... to)** -

Title and Subtitle Analysis of Rough Surface Lighting Behavior With OpenGL	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Christopher Slattery	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey, Ca 93943-5138	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 95	

REPORT DOCUMENTATION PAGE				*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis		
4. TITLE AND SUBTITLE: **Analysis of Rough Surface Lighting Behavior With OpenGL**				**5. FUNDING NUMBERS**
6. AUTHOR Christopher Slattery				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000				**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A				**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited				**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

In the physical world, humans gather valuable information about objects through their sight. Information on shape, feel and composition are seen long before the object is touched. This information is generated by light reflecting off the surface of objects. Despite the advancement of computer graphics due to increased hardware rendering capacity, the fundamental equations, which draw three-dimensional scenes, lack the ability to truly model realistic objects. Whether it is smooth like highly polished metal or rough like the shag of a carpet, it is the reflection of light that tells humans what a surface feels like. The attempt taken in this thesis to implicitly model the roughness of textured surfaces through examination of an explicit model rendered with the OpenGL lighting equation. This approach has the potential to successfully increase the realism of computer graphics without increasing polygon count required for explicit surface generation. Through simulation of an explicitly constructed rough surface followed by the analysis of the behavior of its reflected light, the initial behaviors of textured surface reflections are identified. While these behaviors are not enough to create corrections to the OpenGL lighting equation, they lay the foundation for further development.

14. SUBJECT TERMS Rough Surface Graphics, OpenGL, Opposition Effect			**15. NUMBER OF PAGES** 94
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ANALYSIS OF ROUGH SURFACE LIGHTING BEHAVIORS WITH OPENGL

Christopher P. Slattery
Lieutenant, United States Navy
B.S., United States Naval Academy, 1994

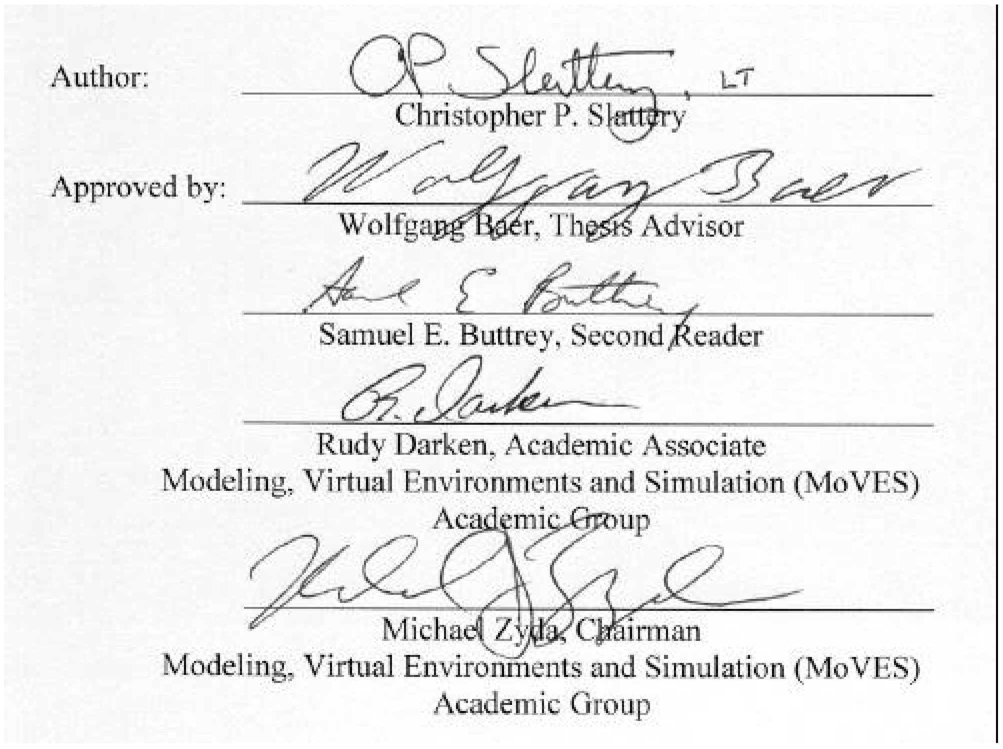
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN

MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION

from the

NAVAL POSTGRADUATE SCHOOL
September 2001



Author: [illegible], LT
Christopher P. Slattery

Approved by: [illegible]
Wolfgang Baer, Thesis Advisor

[illegible]
Samuel E. Buttrey, Second Reader

[illegible]
Rudy Darken, Academic Associate
Modeling, Virtual Environments and Simulation (MoVES)
Academic Group

[illegible]
Michael Zyda, Chairman
Modeling, Virtual Environments and Simulation (MoVES)
Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In the physical world, humans gather valuable information about objects through their sight. Information on shape, feel and composition are seen long before the object is touched. This information is generated by light reflecting off the surface of objects. Despite the advancement of computer graphics due to increased hardware rendering capacity, the fundamental equations, which draw three-dimensional scenes, lack the ability to truly model realistic objects. Whether it is smooth like highly polished metal or rough like the shag of a carpet, it is the reflection of light that tells humans what a surface feels like. The attempt taken in this thesis to implicitly model the roughness of textured surfaces through examination of an explicit model rendered with the OpenGL lighting equation. This approach has the potential to successfully increase the realism of computer graphics without increasing polygon count required for explicit surface generation. Through simulation of an explicitly constructed rough surface followed by the analysis of the behavior of its reflected light, the initial behaviors of textured surface reflections are identified. While these behaviors are not enough to create corrections to the OpenGL lighting equation, they lay the foundation for further development.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would sincerely like to thank Professor Wolfgang Baer for his guidance, and insight into this thesis. I am indebted to you for experience and feedback in this thesis. Working with you will forever change how I look at grassy field on a sunny day.

I would also like to thank Professor Sam Buttrey for his time and patience in this thesis. The results from the analysis would not have been possible without your involvement. I also owe you a great deal for the time spent editing, which transformed my thesis research into this document, of which I am immensely proud.

Finally, to my friend and fiancé, LT Julia Lopez, thank you for your support, guidance and patience, which has helped me throughout the entire thesis.

I. INTRODUCTION

Until a medium is created to display real three-dimensional images, three-dimensional graphics must continue to be presented through a two-dimensional medium. The true nature of three-dimensional graphics can then only be portrayed through interaction.(Moller, p.122) This same interaction is used to turn monocular cues into depth perception in the physical world. More specific data about object themselves can be extracted than simply positional information. This information can be gleamed from the manner in which light is reflected from an object as the viewpoint changes. It is in the reflection from an object that we can see its texture.

Carpets are an excellent example for illustrating the behaviors of illuminated textured surfaces. Figure 1 shows how a silk carpet that has both a textured and shiny surface can reflect light in completely different ways from opposite viewpoints. All carpets have a specific direction in which the threads lay. The apparent brightness of the carpet significantly changes by either pointing the light source into or away from the thread lay. However, if this same example was attempted in computer graphics program as a two-dimensional image, the image would appears exactly the same regardless of the position of the viewpoint.

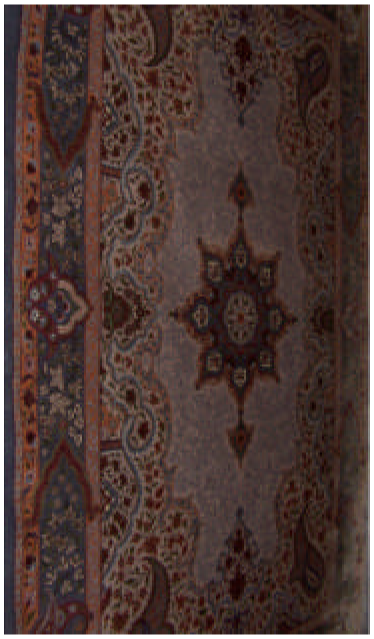
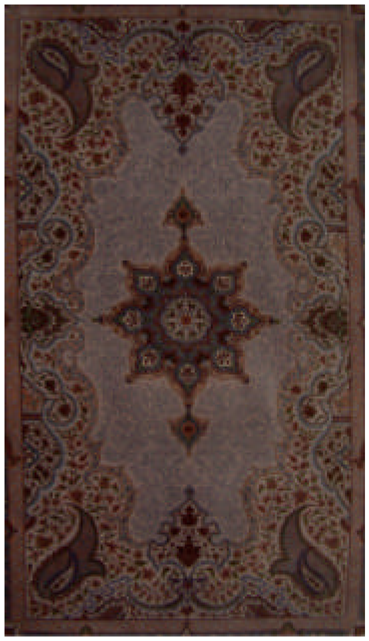
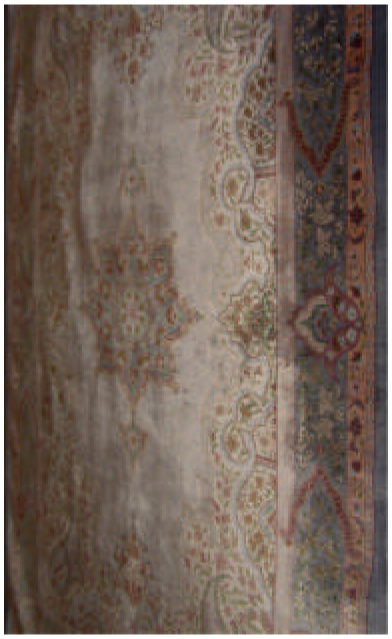
		
Into the threads	Perpendicular View	Along the threads

Figure 1: Example of Real World Textured Surface Reflection

In the real world, the laws of physics determine how object reflect light. In computer graphics, it is the rendering equations inside graphics libraries, like OpenGL and DirectX, which attempt to recreate these same reflections. If graphics libraries ignore an important part of the laws of physics, than they can hardly hope to accurately recreate it.

A. PROBLEM DEFINITION

Current graphics libraries do not intrinsically model rough surfaces. Much of the appearance of objects in the real world is a result of their textured surface reflecting light. In order for current graphics libraries to model realistic surfaces, they have to explicitly build the textured surface with a complex polyhedron. An accurately represented rough surface would require millions of individual polygons. This amount radically increases as the surface area or texture detail represented increases. Despite break-neck speeds found in today's computers, it is still not feasible to model rough surfaces explicitly.

An alternative to modeling rough surfaces explicitly is to develop a method for characterizing rough surfaces based on their interactions with light. Surfaces composed of relatively few polygons, which reflect light as if they were rough, would then be able to render realistic images at a fast enough frame rate to support scene interaction which is essential for realism.

This thesis will attempt to add intrinsic rough surface capability to OpenGL in the form of a correction component to its lighting model. To accomplish this task, the reflective behaviors of various computer generated rough surfaces will be examined through the development of a rough surface simulation program. The correction will attempt to quantify the difference in behavior between that of a smooth surface, which OpenGL already models quite well, and a textured surface.

This thesis will explore rough surface reflection using the following method:

1) Simulate a rough surface reflecting light via an explicitly computed generated textured model
2) Reflected light is captured, measured and recorded
3) Analyze the data for principle behaviors

The result will be the definition of a term for a correction to the OpenGL rendering equation that will serve as a first-order approximation for rough surfaces.

B. APPLICATIONS IN COMPUTER GRAPHICS

Interactive 3D computer graphics suffer from a cartoon-like feel which is a direct result of graphics libraries' failure to implicitly render textured surfaces. All surfaces reflect light with the same reflective behavior regardless of the intended composition of the surface or object. The realism of graphics largely depends on the lighting and how its behaviors matches those found in the real world. Realism is either gained or lost when computer graphics incorporate motion or interaction. It is easier to see the strengths or weaknesses of the lighting model when the viewer can look and move around. The observation that a rough object is reflecting light incorrectly may not necessarily be a conscious one, but it will detract from the realism of the scene.

Plugging the capability to model rough surfaces into a graphics library results in scenes where shiny surfaces can easily be distinguished from rough ones. This produces scenes that are more natural and realistic. Adding this functionality increases the complexity of the lighting model, thus slowing it down. The realism which is gained with the better lighting model may be lost if the frame rate slows down too much. Therefore, the computational nature of the correction must also be lightweight.

II. BACKGROUND

The human eye's perception of color, when looking at a surface in the physical world, depends on the distribution of photon energies that arrive and trigger the eye's cone cells. Those photons originate from one or many light sources, some photons are absorbed and some of which are reflected by the surface. Additionally, different surfaces have very different reflective and absorption properties – which is to say that a shiny surface tends to reflect light in a specific direction, while a dull surface tends to scatter incoming light equally in all directions. Most surfaces are somewhere in between. (Woo, p.177)

Objects in the real world all reflect light differently. Useful information about their size, shape and composition can be gathered from their appearances. The appearance of an object is a function of the how the object's surface reflects light from a particular light source. More specifically, it is the texture of the reflecting surface that determines its appearance when illuminated. This texture, which is really a rough surface, provides realism to objects. It is this same texture that is missing from objects rendered by computer graphics, thus limiting the perceived realism of the scene presented.

Without the ability to feel the texture on an object, our sight is the only means available to determine if an object is smooth or rough. In this way, perhaps a concept developed to understand objects appearing beyond our reach, can be used to increase the realism of computer graphics. Like space, computer graphics renders objects we can see, but not touch.

The scientific community has studied a phenomenon known as the *opposition effect*, which it uses to describe the behavior of light reflected from astronomical bodies. The same ideas that are the basis for the opposition effect can also be applied to a more general case; rough surfaces. Importing opposition effect behaviors into computer graphics is a reasonable method for simulating real three-dimensional textured surfaces.

Because this thesis will attempt to correct the OpenGL lighting equation by using an approximation of the opposition effect to model rough surface reflections, it is necessary to define the opposition effect and how it relates to the OpenGL lighting equation.

A. DEFINING ROUGH SURFACE REFLECTIONS THROUGH THE OPPOSITION EFFECT

The opposition effect describes why rough surfaces reflect light differently from smooth surfaces. It is the general assumption that a smooth surface with no refractive index will have an exodus angle equal and in a direction opposite to the incident light. That is to say, that a smooth reflective surface, that has no translucent depth, will reflect light away from the source at an equal angle. Rough surfaces, on the other hand, tend to reflect light back towards the light source. The opposition effect describes the fundamental principles of complex surfaces reflecting light.

The opposition effect derives its name from the fact that astronomical bodies appear at their brightest when the phase angle, which is the angular difference between the incidence and view angle, is zero for solar-system objects at astronomical opposition. Figure 2 illustrates the idea of phase-angle.

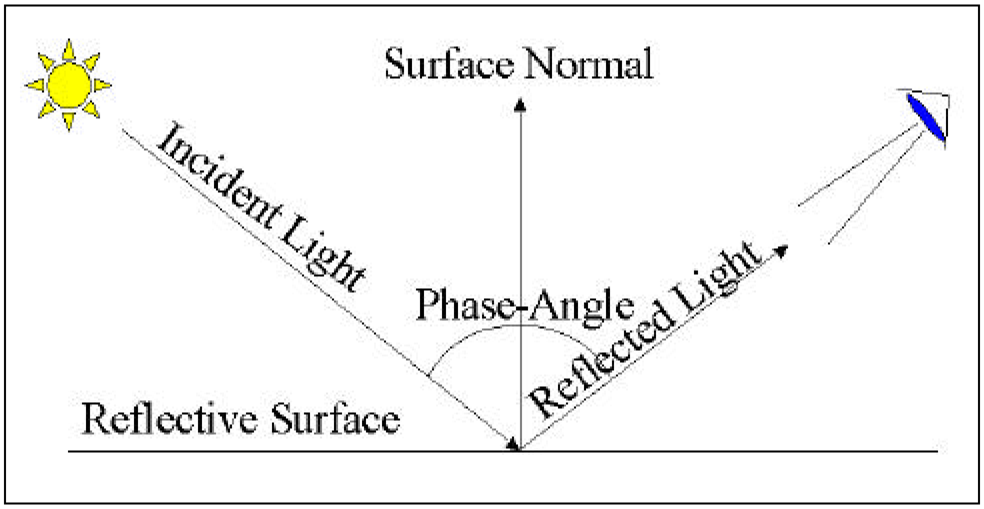


Figure 2: Phase Angle Diagram

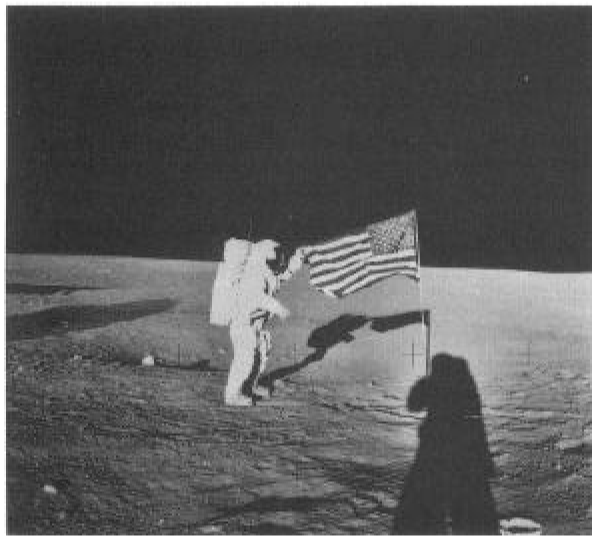
Planets, moons, etc, within the solar system are at astronomical opposition when the sun, the earth and the object are in that particular alignment. This effect can be observed on earth from an airplane as a bright halo around the shadow of the plane when the shadow falls on vegetation or soil. This halo typically disappears on worn pavement or other man-made surfaces for reasons that will be discussed later.

1. History of the Opposition Effect

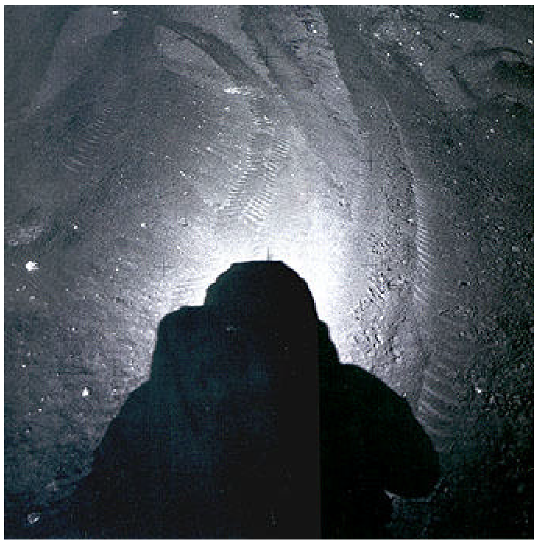
Seeliger (1887, 1895) first discovered the opposition effect when examining the light scattered by Saturn's rings. He correctly explained the phenomenon by stating that,

> In a medium in which the particles are large when compared with the wavelength, particles near the surface cast shadows on the deeper grains. These shadows are visible at large phase angles, but close to zero phase-angle they are hidden by the object that cast them. Thus, the effect may only be thought of as being caused by shadow hiding. (Hapke, p.217)

The moon's surface is considered a reflectively rough surface. Apollo astronauts, while on the surface of the moon, captured an example of light reflected by the opposition effect and is shown in Figure 3. A distinct halo can be seen around the shadow of the astronaut's head in Figure 3. This halo is the result of the sun's rays reflected in a specific direction.



On the Moon's Surface



Overhead on a Space Walk

Figure 3: Photos Of The Opposition Effect On The Moon

It is important to note that the light source, in this case the sun, is behind the astronauts taking the photos and not in front. The light from the sun is predominantly being reflected back and is brightest when the phase angle is near zero. This is the opposite of what happens on a smooth surface. The size of the halo is defined by the surface's reflective angular half width, its angular half-width, which is unique property of every rough surface. The angular half-width is a function the shininess of the rough surface; the more shiny it is, the smaller the half-width.

An additional effect helps define the shape of the reflected light. Areas in the photo that have bright spots intermixed with visible shadows appear less bright even though those bright spots are just as intense as inside the halo. As the viewpoint rises above the incidence angle, the reflected light intensity dims. This is the same as looking at the bottom of the first photo in Figure 3. This dimming is a result of shadows, previously hidden in the rough cavities of the moon's surface, now becoming visible. The elements of shadow-hiding are illustrated in Figure 4 and is one of the principle parts of the opposition effect.

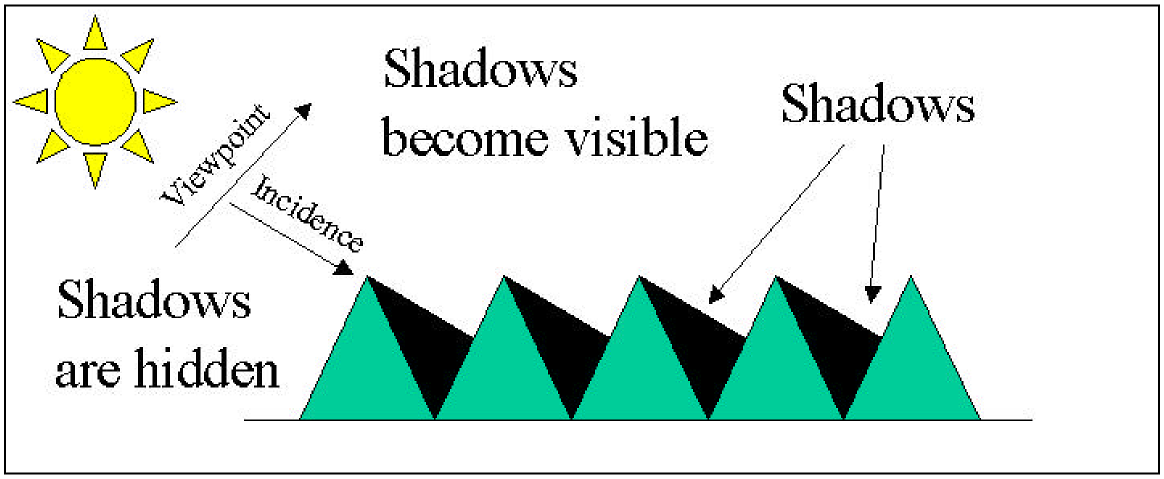


Figure 4: Illustration of Shadow Hiding

Computer graphics must incorporate these effects if they are to render rough or textured surfaces.

2. Defining Reflectance Models

It is more practical to talk about scenes in terms of reflectance models which are a light source and illuminated surface pair. Reflectance models allow specific definition of the scene pair. The reflective behavior of the scene does not remain constant. Identification of the specific reflectance model best illustrating the opposition effect will ensure proper development of a computer graphics simulation model.

The concept of reflection can be divided into components based on a surface's characteristic behavior to emit or scatter light. Bruce Hapke coins the terms *reflectance* and *reflectivity*, both terms referring to the fraction of incident light scattered or reflected by a material. Reflectivity refers to the specular type reflections produced by smooth surfaces. Reflectance, on the other hand, refers to the more diffuse type reflections produced by geometrically complex surfaces. (Hapke p.182) While other types of reflection models exists, the reflectance model is best suited for modeling the opposition effect because it is created to describe the scattering of light by rough surfaces.

There are many kinds of reflectance, depending on the geometry of the surface. It is important to qualify this defining term so that their meaning is unambiguous. In current terminology, reflectance is preceded by a pair of adjectives describing the degree of collimation of the light source and then that of the detector. Collimation describes how parallel or straight the rays of a particular light source are. Typically *directional*, *conical* and *hemispherical* are used to describe the level of collimation. If the case is such that the adjectives describing both source and detector are the same then the prefix bi- is substituted. Hence, a directional-directional reflectance model is then called a bi-directional reflectance model. (Hapke, p.182)

a) Directional Reflectance

Directional reflectance is best portrayed by sunlight on a cloudless day; hence being unfiltered or un-reflected light. The sun is considered a directional light source since its rays sub-tend less than 0.5^o on earth's surface. In practice, this yields shadows the same size as the casting object when the sun's rays are perpendicular to the earth's surface, therefore no actual shadow would be visible around the object. A camera whose lens is focused at infinity is an example of a directional detector.

b) Hemispherical Reflectance

Hemispherical reflectance is the opposite of directional reflectance. Due to multiple scattering and reflections, the rays of light no longer hold any particular direction. As the sun's rays pass through a cloud layer, the rays are scattered and produce a glowing light source. This is demonstrated by the seeming lack of shadows on a cloudy day.

c) Conical Reflectance

Conical Reflectance is the combination of both directional and hemispherical reflectance. In reality, all measured reflectances are bi-conical due to the fact that no naturally occurring reflectance can be either perfectly collimated, nor diffuse. However, many situations in nature are sufficiently close to collimated or diffuse for those models to be used as useful approximations.

3. The Bi-Directional Reflectance Model

A model that attempts to approximate the opposition effect must account for shadows. This requires a directional light source. Additionally, because the effect is better seen at longer distances, a directional detector is also required. A model connecting the light incident from one direction with the light observed from another direction is called a bi-directional reflectance model. Such a model is best suited to model the opposition effect.

4. The Bi-Directional Reflectance Function

The bi-directional reflectance function, or BDRF, is the implementation of the bi-directional reflectance model. The function will include the mathematical algorithm by which the lighting reflections are calculated. The BDRF can be thought of as a black box, which takes a series of parameter (incidence, viewpoint, roughness, etc…) and yields a light intensity value.

B. AN ALTERNATIVE METHOD FOR MODELING ROUGH SURFACES IN COMPUTER GRAPHICS

Another approach, which is widely viewed as the most accurate to date, seeks to incorporate microscopic rough surface reflection effects through a statistical/micro-facet model. While not explaining the specifics of the model, a model developed by Cook and Torrance, uses a linear combination of the ambient, diffuse and specular components.(Watt, P.58) The specular component of their model is based on the Fresnel Equation for reflections off a perfect surface, and then modified by a geometry attenuation term and a statistical micro-facet term. This model is similar to the Phong model in that it only models reflections from a directional light source. Specifically, this model was developed to improve the specular reflection off a highly polished metal surface, which they assume is not a perfectly flat surface, but rather a nearly flat faceted surface. Cook and Torrance's model also takes on a more physically based approach by dealing with light as wave energy and not simply geometric lines. In their model, it is possible to increase the roughness of a shiny surface by increasing the geometry attenuation term, which controls the diffuse light emitted from the surface. Unfortunately, this model does not affect the angular direction of the specular reflection lobe, which this thesis believes is necessary to accurately model rough surface reflections. This model is really better suited for modeling the subtleties of shiny surfaces than the reflective behaviors of textured surfaces. Furthermore, this shows that in addition to OpenGL's deficiency, Cook and Torrance's methodology is also insufficient for correctly modeling rough surface reflective behavior.

C. MODELING THE OPPOSITION EFFECT IN COMPUTER GRAPHICS

Having described why the opposition effect is a good method for describing reflection behaviors of rough surfaces, and why a BDRF is the appropriate model structure, it is now important to describe why the OpenGL graphics library should be used to simulate the model.

OpenGL is the most popular real time rendering model, and is also similar in form to the extensible VRML lighting model. The majority of computer graphics cards produced have hardware accelerators specifically designed to implement the equations defined by the OpenGL model. OpenGL captures all of the characteristic components of reflected light; *emission*, *ambient illumination*, and *diffuse* and *specular reflectivity.* Currently, OpenGL is capable of modeling light reflections from a facet or a flat surface comprised of a single polygon. A textured surface is comprised of many such facets. In order to produce the reflective behaviors of a textured surface, the surface must explicitly have these facets in order to allow OpenGL to make all lighting calculations. OpenGL as of yet, does not contain the ability to create the lighting effects in one polygon as if that polygon was comprised of many polygons. However, OpenGL's wide use in computer graphics and its open source accessibility makes it a good choice for simulating the opposition effect with a polyhedron.

1. The OpenGL Lighting Model

The OpenGL lighting model approximates light as if it can be broken into its primary component colors; red, green and blue. The color of a light source is characterized by the amount of red, green, and blue light it emits. The material of a surface is characterized by the percentages of the incoming red, green, and blue components that are reflected in various directions. (Woo p.177)

In the OpenGL model, light sources have effects only when there are surfaces configured to absorb or reflect that particular color of light. Each surface is composed of a material with various properties. Materials are able to emit their own light (such as headlights on an automobile), scatter incoming light in all directions (such as sidewalk made of concrete), or it might reflect some portion of the incoming light in a particular direction (such as a mirror or other shiny surface). Materials may also take on a combination of attributes, allowing for a multitude of possibilities.

The OpenGL lighting model considers light to be divided into four independent components: ambient, diffuse, specular, and emissive. All four components are computed independently and then added together. These four components can be correlated with the components mentioned in the reflectance sections. Both directional and hemispherical light sources can be used either singularly or as a group. Depending on how the light sources and reflecting materials are configured, different types of reflectance models can be simulated. The bi-directional reflectance model can be simulated in OpenGL by using directional lights and surfaces with primarily diffuse and specular components.

It is important to remember that the OpenGL lighting equations are just approximations and do not capture every behavior of light in the physical world. However, the model does work fairly well and, more importantly to computer graphics, it can be computed quickly and efficiently. If a task required a more accurate lighting model, then the calculations would have to be pushed up to the software level. Such software can be enormously complex, and dramatically slow down graphics rendering.

2. The Light Source / Reflecting Body Interaction

OpenGL's lighting model follows a two-part paradigm, the light source and the reflecting/absorbing-body pair. In order for a light interaction to exist, the light source and reflecting body must be parameterized to allow for such an interaction. If a red light source is paired with a body that reflects only blue or green, then the body will not be visible. The same holds true for the specular and diffuse characteristics. A reflecting body must be given ambient, diffuse or specular properties to be able to reflect light. Changing how an object's material interacts with light alters an its appearance. Furthermore, an object's material composition can be observed through its dynamic interaction with light.

3. The Components of the Reflective Light According to OpenGL

Reflected light is far more complex than its light source. Because of the variety of surfaces that all reflect light differently, the model must be broken down into several sub-components. Since objects in computer graphics have no physical or tactile properties, the only means of which an observer can determine their composition is through their appearance. Careful selection of an object's reflective properties is the only way to relay this level of detail. The reflective properties are broken down into three parts: *ambient*, *specular*, and *diffuse*. The OpenGL model also includes *emissive* light; however because this is property of the light source and not of reflecting body, it will be not be addressed hereafter.

The remaining three components that comprise the lighting equation all have a common axiom, in that each component has a light source and reflective surface pair. Without both parts of the pair, the component does not visually exist.

a) Ambient Component

Ambient illumination is light that has been scattered so much by the environment that its direction is impossible to determine – it seems to come from all directions. Back lighting in a room has a large ambient component, since most of the light that reaches your eye has first bounced off many surfaces. Rooms lit by ambient illumination tend to have lamps that reflect light off several surfaces before being viewable. This produces a uniform glow in the room. In OpenGL, ambient light intensity is independent of incidence angle and viewpoint. Therefore, it is not useful to include ambient light in the rough surface model.

b) Diffuse Component

The diffuse component is light that comes from one particular direction, however far enough away so that its light rays are parallel. The result is light that is brighter if it strikes squarely down on a surface than if it strikes a surface at a glancing angle. Once directional light strikes a surface, it is scattered equally in all directions. The level of brightness remains constant regardless of the view angle, so as long as the surface is viewable. Any light coming from a particular position or direction most likely has a diffuse component.

c) Specular Component

Like directional light, specular light also comes from a particular direction, but tends to reflect off the surface in a preferred direction. If a laser beam, which is composed of highly collimated light, is reflected off a high-quality mirror, it produces an almost perfect specular reflection. Shiny metals or plastics have a high specular component, where as chalk or carpet have almost none. The level of shininess that the material exhibits drives the width of the specular reflection.

D. APPLYING THE OPENGL LIGHTING EQUATION

Now that the OpenGL lighting equation has been identified as the proper tool for simulating the opposition effect, it is important to take a closer look at the workings of the equation and why it is applicable to simulate a real world effect. As mentioned before, light in OpenGL is broken into three components: red, green and blue. In OpenGL, each pixel carries an intensity value, ranging from 0 to 255, in each of the three colors. An extension of the Phone Lighting model, shown in Figure 5, is used to calculate each color component. Mathematically, it is given by the following equation:

$$N_{R,G,B} = T(u,v) * \{m_a J_a + m_e + m_d J_d \max(\underline{l}\cdot\underline{n},0) + m_s J_s [\max(\underline{s}\cdot\underline{n},0)]^{\zeta}\}$$

Figure 5: Phong Lighting Equation (Bui-Tong)

In this equation, $T(u,v)$ deals with texture coordinates and is not particularly influential as a reflecting property. J_a, J_d, and J_s are the normalized ambient, diffuse and specular light intensities, and have values ranging from 0 to 1. m_a, m_e, m_d, and m_s are the ambient, emissive, diffuse, and specular color intensities of a given object and range from 0 to 255. These intensities can be thought of as potential reflectiveness for each property. ζ is the shininess parameter and ranges from 0 to 128, which determines the angular half-width of the specular reflection lobe. Finally, the light vectors, $\underline{l}$, $\underline{n}$, and $\underline{s}$, are defined respectively as the incidence, surface normal and the reflection vectors. Figure 6 illustrates the interactions of the lighting model on a simple surface.

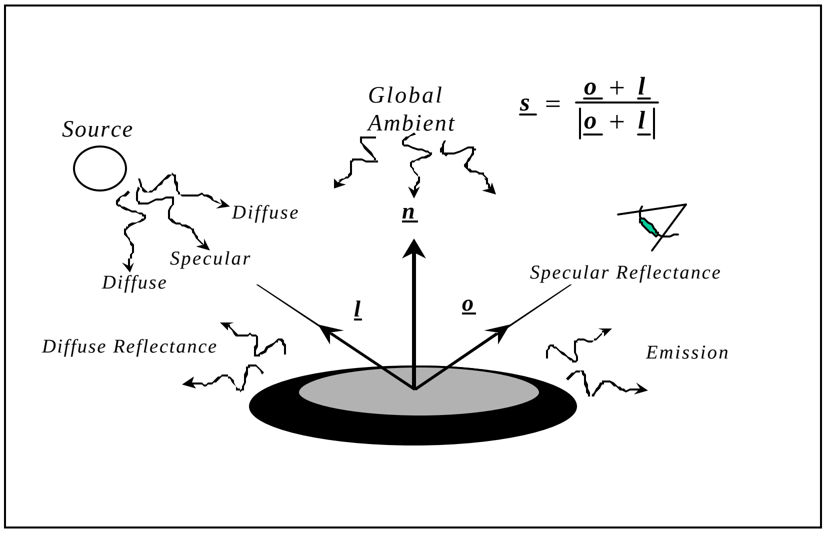


Figure 6: Geometric definitions associated with the OpenGL lighting model

1. OpenGL Lighting Equation Limitations

This lighting equation qualitatively captures the geometric behavior of emission and reflection from a surface. There is no direct mapping between units in the physical world and the OpenGL lighting equation, but this does not preclude the light equation from being a valuable tool with which to simulate behaviors found in the real world. The deficiencies of the equation must be kept in mind to prevent incorrect assumptions from being made. These deficiencies include:

- Non-linear color intensity summation
- Non-physically based parameters
- No opposition or self shadow effect
- Statistical two-dimensional texture function

Despite these limitations, the OpenGL lighting equation is a good starting point for a semi-empirical surface-rendering model, because it approximates the reflection from a single flat homogeneous surface facet with reasonable accuracy.

2. The Principle Basis of the OpenGL Equation

Even though there are several areas where the OpenGL equation attempts to model different light interactions in the physical world, this thesis will focus on the intensity of a single pixel which has been illuminated by an incident light. Therefore, several components of the lighting equation will be held constant so as to not affect the reflection model. This eliminates the emissive term, and turns the texture parameter into a constant.

The term *pixel* is used to represent both the detector element, which makes a measurement of the reflected light, and the emissive element, which generates the visual energy seen on a computer monitor. For both an ideal measurement and for the display system, the pixel is simply two sides of the same area. The same light pattern that is measured should be viewed by an observer with no difference. This is to say that the virtual reality scene should visually be no different from the actual scene.

The mathematical definition of the BDRF function is:

$N_{R,G,B} = BDRF * I_{R,G,B}$

Where:

- $I_{R,G,B}$ - Input Light Intensity
- BDRF – Bi-directional Reflectance Function

A single BDRF, which parallels the OpenGL lighting equation, is expressed in terms of physical parameters is given by:

$BDRF = \rho_l/\pi + \{\rho_s(1 + \zeta) [\max(\underline{s}\cdot\underline{n},0)]^{\zeta} / [4\pi\cos(i)]\}$

Where:

- ρ_l - Lambertian reflectivity
- ρ_s - Specular reflectivity.

(Watt, p.24)

This particular form is the basis by which surfaces are characterized in the OpenGL lighting model. It is a function of three parameters: *Lambertian reflectivity*, *specular reflectivity*, and *shininess*. These three parameters relate directly the bi-directional reflectance model of the facets we hope to use as individual elements in our simulation of the opposition effect.

This equation can be physically interpreted by going back to the two fundamental mechanisms by which light interacts with a surface. The Lambertian term corresponds to light that is absorbed and uniformly re-emitted. There is no viewpoint angular dependency in the Lambertian term since the re-emission is *isotropic*; meaning light is reflected equally in all directions. (Webster) The Lambertian portion of the BDRF is identical to OpenGL's diffuse light component.

The specular term represents light reflecting from a surface as a whole in the form of a wave. Specular reflection is a function of reflection geometry and the index of refraction, which is known as the Fresnel coefficient (ρ_s). (Wesley, p.24) For most solid surfaces, this coefficient is treated as constant. However, for more accurate models or models that attempt to handle liquid surfaces, this coefficient itself would be a function.

3. Visual Effects of the OpenGL Lighting Components

The Lambertian and specular components result in two different lighting effects, but both are important parts in modeling rough surface reflectance. The light intensity of the Lambertian component is independent of view angle. Therefore, the Lambertian component provides a surface with an unchanging light intensity as long as the light source does not change position. This unchanging light source can be interpreted as a glow that is cast by the surface.

The specular component is the light of the source seen "through" the surface. For a perfectly flat shinny surface (i.e. a mirror which has a very large shininess value) the image appears to be inside the surface. Instead of the uniform distribution given by the Lambertian component, the specular component defines a narrow lobe. As the surface becomes rougher and the shininess value decreases, the reflecting lobe becomes wider. Despite the change in width of the reflecting lobe, the reflected color intensity remains the same value as the source. It is important to note that the reflection of the specular component is driven by the reflection geometry, which so far only considers smooth surfaces.

4. The OpenGL Bi-Directional Reflectance Function

The OpenGL BDRF works best when scaling permits many pixel elements to represent a single polygon and that polygon represents only one flat homogeneous surface. Computer graphic artists create impressive images by remaining within these constraints. Polygon size can be decreased, thus increasing the number which represents a given area, in order to accommodate more complex materials and/or geometric surfaces. Unfortunately, as the polygon count increases, the pixel-to-polygon ratio will eventually decrease to the point where one pixel represents more than one polygon. The This is best illustrated when computer objects are viewed up close and then moved into the distance. The same object can now be represented with fewer pixels. An algorithm of one type or another must be incorporated to properly represent multiple polygons with one pixels in order maintain the object's correct appearance when viewed at long distances. Hence, the experienced computer graphic artist can create models that minimize negative affects that appear as a result of combining polygons. Adding the implicit correction for rough surfaces would reduce the number of polygons needed, thus reducing these undesirable effects.

E. SUMMARY

In order to enhance the realism of computer graphics, we must extend the OpenGL BDRF model to accommodate more complex reflection effects while not drastically slowing down its rendering process. This extension can be accomplished through a correction to the lighting model which models behavior similar to that exhibited by the opposition effect. Handling this extension implicitly will maintain the high frame rate required for realistic interaction. Adapting OpenGL for this interaction will result in the addition semi-empirical corrections to its BDRF in order to improve its accuracy.

III. EXPLICIT ROUGH SURFACE MODELING AND SIMULATION

Computer graphics can be used to model the behavior of light reflecting off surfaces by modeling the texture of real surfaces with many small facets. The validity of the output depends on the number of polygon's representing the surface and the accuracy of which each facet is rendered. OpenGL, which has a credible model for representing simple light interactions, can be used to calculate light reflected from a single facet. The appearance of the textured surface will be the sum total of calculations of each and every facet's reflection. However, OpenGL is only a part of the process; an entire simulation program must be written to build, manipulate and measure such a surface.

In order to simulate and analyze rough surfaces in computer graphics, design and development of software tools is necessary. This chapter will discuss the considerations that have gone into the design of the rough surface simulator, including the choice of programming language, the architecture of the tool's algorithm, and finally the rough surface itself.

A. DESIGNING THE ROUGH SURFACE SIMULATOR

Various computer programs and tools are currently available that support the idea of modeling rough surfaces in a computer environment. Many computer languages have imbedded libraries that support three-dimensional graphic development. Despite the attractiveness of higher-level tools that produce extremely realistic images, their rendering engines are not open source, thus preventing the level of inspection and interaction that this thesis requires. Open source algorithms are particularly important when trying to understand and validate the output of the model.

1. Base Line Assumptions

Before the design and development of any tool, it is imperative to establish the expectations of the program. The simulator is built with the following assumptions in mind:

a) Incorporate a Bi-Directional Reflectance Function Model

The opposition effect requires that a BDRF model be implemented in the simulator. This requires that the characteristics of the light source be the same as those of light infinitely far away; that is, they must be collimated light. Additionally, the detector, in the case of the simulator the viewpoint, must have a focal view of infinity to prevent any warping or bending of light rays as they are collected. These requirements can be met will existing components of OpenGL lighting model.

b) Provide Variability to the Rough Surface

Capturing the behavior of a rough surface goes beyond simply varying the incidence and viewpoint angles. True understanding of the behaviors requires examining different types of rough surfaces. The simulator should allow basic manipulation of the surface, enabling some surface variability.

c) Provide an Experimental and Control Model

All properly formed scientific experiments require that a control be established alongside the experiment. Even though this experiment is entirely inside a computer, a scientific approach is still applicable. In this case, the control will be a single polygon the size of the rough surface. All facets in the rough surface will inherently have the same diffuse and specular reflective properties as the control surface. Therefore, if the parameters of the rough surface are set such that the surface becomes smooth, then the output of the simulation should match the output of the control surface. This will also enable comparison of the rough surface model to OpenGL's current capability, which will be implemented in a control surface.

2. Using Java & Java3D (OpenGL Variant)

The purpose of the simulator is to create rough surface reflection behavior through modeling the opposition effect with many facets each of which is rendered with the OpenGL graphics library. While it is possible that another graphics library will yield results similar to those from OpenGL, it would be inappropriate to modify the OpenGL lighting equation based on those results. Therefore, this narrows the selection of a programming language to one that supports the OpenGL library.

Several languages support the OpenGL graphics library for 3D programming. Two of the more commonly used languages are C and Java3D. Each language has strong and weak points. The C language has been around much longer than Java 3D and has been used in industry on many projects. It allows for in-depth control of the lighting model as well as providing faster results. Java 3D is Sun Microsystems' variant of the network-based Java language. It is a stronger typed and structured language than C, making for stream-lined programming. Unfortunately, Java3D does not benefit from the level of certification resulting from repeated use in industry that C offers.

Taking that all points into consideration, the Java 3D language was selected to develop the simulation tool for three reasons:

- The OpenGL Architecture Review Board (ARB) has award to Java3D the rights to bear its trademark, signifying the validation of the library implementation.
- Java by nature is a platform independent language; therefore, the expectation is that differing hardware will have no impact on program output. This was tested and proven correct on computers using different graphics cards.
- By nature Java is an easier language to decipher, thus allowing other researchers continuing the development the simulator, thus extending its software lifecycle.

3. Program Design

The purpose of the simulation program is to measure the light intensity reflected from a rough surface at various incidence and view angles. The reflecting surface is a rough surface explicitly constructed by macroscopic polyhedral structure. The light source and capture instrument are collimated as specified by the BDRF model.

The simulation program will generate two surfaces of the same material composition, light source and capture device. The reflective distributions of both surfaces measured from all possible viewpoint elevation and azimuth angles. This will be accomplished by orbiting the capture device over the surface along a specific azimuth. When that orbit is completed, the capture device will be rotated a predefined azimuth increment, and then orbited again. This will repeated, until the entire surface has been measured. Figure 7 illustrates this algorithm for measuring the reflective distribution of a surface.

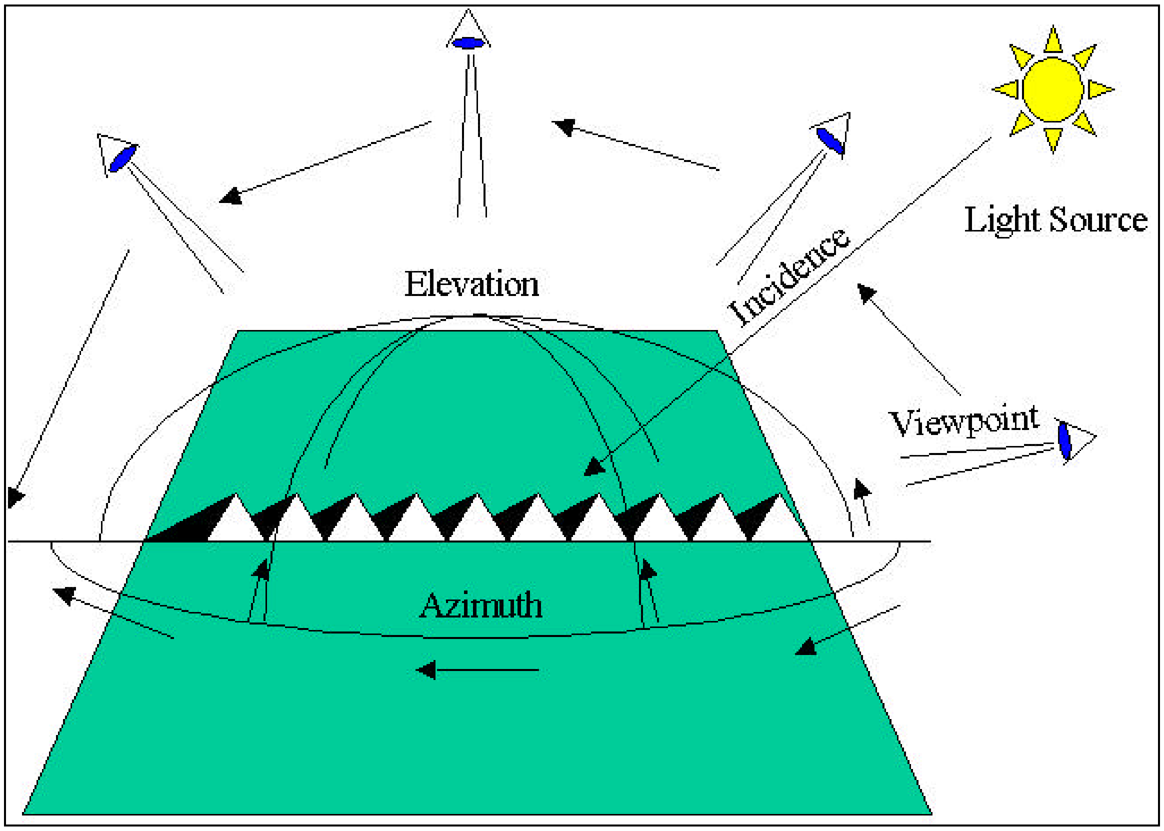


Figure 7: Illustration of Rough Surface Simulation Program

To accomplish the purpose of the simulator program, the program will require direct interaction with the graphics card's buffers. It will be useful for multiple surfaces of different characteristics to be rendered and measured at the same time, thus requiring the program to manipulate and control multiple threads. Graphics User Interface (GUI) front ends are required, since the simulator should also offer some level of interaction for user-defined surface inspection.

To accomplish an automated process of measuring a given rough surface with a specific incidence angle over all view angles, the following algorithm has been developed:

- Both Experiment & Control scene graphs render a new frame.
- Execution threads for scene graphs are paused and wait for image capture routines to complete execution.
- Image capture routine pulls information from buffers and notifies scene graphs to resume.
- Rendered light intensity values for frame are calculated and written to database.
- Thread control increments elevation/azimuth rotation.
- Start process over again.

Figure 8 is a graphically flow diagram of this process, and highlights interaction management that must occur for proper automation to occur.

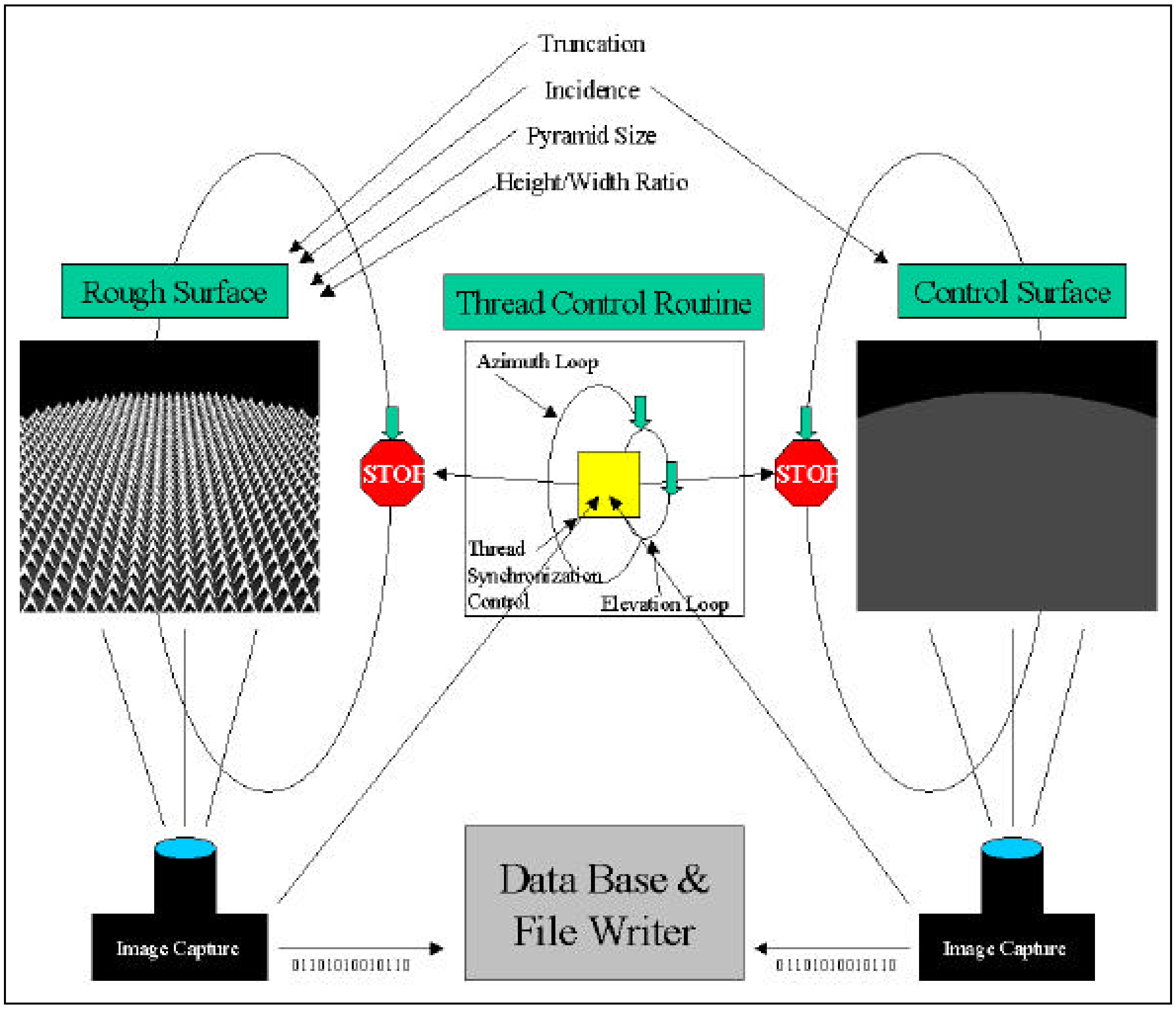


Figure 8: Flow Diagram for Rough Surface Simulator

4. Developing the Rough Surface

The key to simulating a rough surface is to accurately build the undulations of a surface from many small polygons. Each geometric detail contributing to the overall surface roughness is modeled by many small polygons. In order for the simulator to measure how different types of rough surfaces reflect light, the polyhedrons must be constructed for easy manipulation.

a) Basic Geometric Building Block

The simulator provides a means to examine the reflection behaviors of rough surfaces. This thesis attempts to capture these behaviors, and modify OpenGL's lighting components such that the behaviors can be distilled into a texture form. A key element in the texture concept is that the rough surface behavior must be uniform and constructed from a single material if it is to replace many polygons with only one. This uniformity requires a surface constructed from simple objects, which combine to produce a rough surface. Defining this requirement further, the uniqueness is achieved through repetitive use of one geometric object.

The simplest object is a pyramid. First, it has no curves. Its four-sided base lends to easy grouping and organization. The slope of the reflecting faces is determined by the height of the pyramid, making it relatively easy to adjust the roughness of the overall surface. A shadow cast by a pyramid is the simplest geometric shape: a triangle.

The height of the pyramid can also be truncated to simulate surfaces which exhibit the behavior of a rough surface, but also incorporates the behavior of a worn surface. This additional behavior is found in most real-world textured surfaces and key to developing an accurate model.

b) Inner Shadowing

The crux of the opposition effect rests on the fact that shadows are normally seen at large phase angles and are obscured at phase angles near zero. The opposition effect requires inner shadows in order to be modeled correctly.

Java3D does not provide inherent shadowing capability; therefore the simulator, that is the programmer, must explicitly cast shadows from one geometric object to the next. This is a relatively easy task when the surface is built of regular objects at regular intervals. In this case the shadows can be drawn on each pyramid independent of its placement on the surface. Figure 9, shows the simulator's shadowing capability at various incidence angles as well as handling shadows for truncated pyramids. Each picture is from a viewpoint directly in front of the pyramid and with no elevation. Although not visible, the shadow casted on these pyramids is from an identical pyramid directly in front the ones visible.

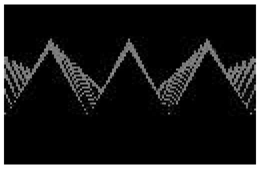
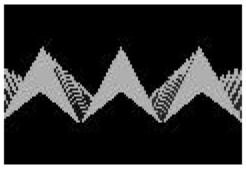
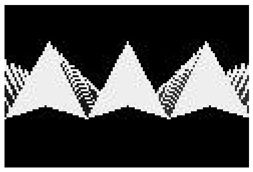
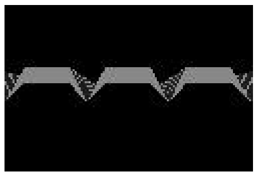
 10^o Incidence	 40^o Incidence
 55^o Incidence	 30^o Incidence with 45% Height Truncation

Figure 9: Images of Shadows at different Incidence Angles in the Simulation Tool

Unfortunately, calculating shadows beyond these cases becomes extremely difficult.

c) Gouraud Shading

OpenGL uses the Gouraud shading model to calculate the amount of light that a surface will reflect. As mentioned in Chapter II, the amount of light reflected from a diffuse surface is driven only by the incidence angle and independent of the viewpoint. The Gouraud shading model is rather simple and shown in the following equation:

$$I_{reflected} = m_{source} * \cos(\text{incidence angle})$$

For example, if the incidence angle is 30^o and the intensity of the light source is 128, the resulting diffuse reflection will be 64.

d) The Light Source

As stated in the design goals for the simulator program, the BDRF model requires a non-subtending light source to properly portray the specular reflection. Java3D, or rather OpenGL, provides a Directional Light class, which meets this requirement and as a single light source provides illumination for both diffuse and specular reflections. Conversely, the material properties of the polygons constructing the rough surface must also be properly set to interact with this light source correctly. Additionally, the direction of the directional light is variable allowing all incidence angles to be simulated.

5. Measuring Light Reflected by a Surface

The simulator relies on the OpenGL functions to render all images. The graphics program, and in this case the graphics card, renders the image and writes the resulting data to a buffer. The data stored in the buffer is considered the final product. It is from the color values stored in this buffer that the video signals are generated. By inheriting from Java3D's Canvas3D class, the simulator can gain access the specific portion of the buffer that holds the data for each surface.

The buffer itself is set up so that each pixel on the screen is represented as color triplet ranging from 0 to 255. This is the same color triplet that OpenGL uses as mentioned in Chapter II. Because all of the polygons in both surfaces in the simulation tool are either white, black or a shade of gray, the individual values in the color triplet are always the same.

The important question which now arises is which pixel in the buffer to select as the representative of the light reflected from a surface. It is also important to keep in mind that the capture window captures the intensity values of pixels on the monitor and not the polygons that make up the surfaces. If the surface viewed from the edge, it may not be visible on the screen. However, as the viewpoint rotates to a perpendicular position, then the surface becomes visible in the view plane; hence the portion of the buffer that holds the surface's data increases. Figure 10 shows this effect.

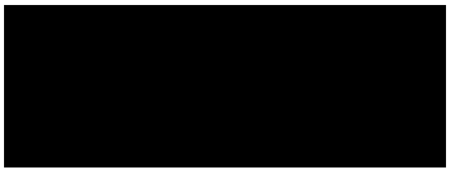
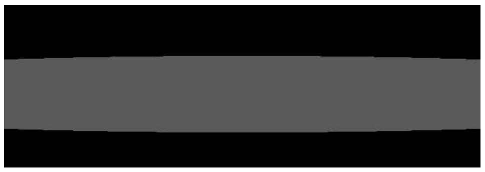
	
Surface viewed 0^o elevation	Surface viewed at 10^o elevation

Figure 10: Surface Views at Near Zero Angles

It is therefore not possible to take all pixels in the buffer as representing light emanating from the surface. A small capture window, which selects only those pixels projecting from the simulated surface, is required. When developing the process to select the dimensions of this capture window, two possibilities are considered. The first is a capture window of fixed height and width located in the center of the surface's window; the second is a window that increases and decreases in height as the elevation is incremented. This method attempts to minimize the error induced by measuring pixels representing the horizon and maximizing the amount of the surface measured. The first image in Figure 11 shows the pixels measured in the fixed capture window. This box does not change as the elevation is increased. The second and third images show the variable height capture window and how it attempts to follow the surface's edges as the viewpoint rotates.

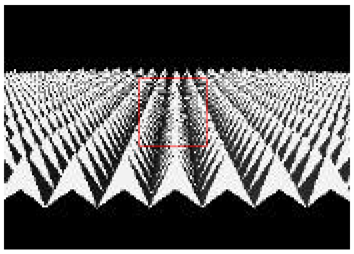
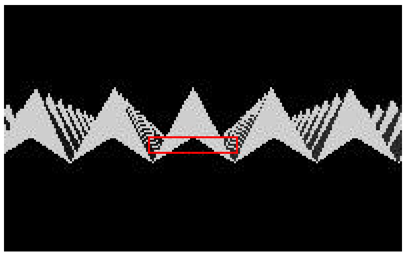
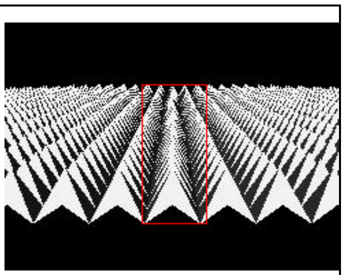
		
Fixed Capture Window	Variable Capture Window at 0^o elevation	Variable Capture Window at 15^o elevation

Figure 11: Views depicting Fixed and Variable Capture Windows

While the variable height capture window produced smoother graphs, its effects on the measurements caused by a wider range of view angles are not entirely obvious. The simulator defines the view angle to be a single angle between the view vector and the surface. That angle is only true for the pixel at the center of the surface. If the capture window is wide enough, then a variety of view angles are captured which could lead to the data being shifted in one direction or another. The degree of this shift is not known. Figure 12 illustrates the how multiple view angles are actually seen when viewing a surface. It is expected that the inclusion of so many actual view angles, actually washes out what truly is happening at the center of the surface where the primary interest lies. Avoidance of too many view angles is also the same reason that the fixed both types of capture windows' widths are relatively narrow.

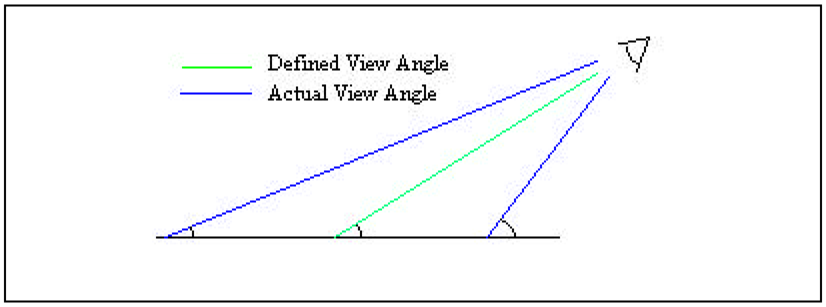


Figure 12: Shows Differing View Angles

In order to minimize the angular width, the fixed height capture window was used in favor of the variable height capture window. The final fixed capture window captures the center 40 x 40 pixels. Each surface renders into a screen window of 300 x 400 pixels.

The final light intensity for each pixel is the averaged of all pixels in the capture window. The simulator stores this average value in the database as the reflected light intensity for that particular elevation and azimuth angle.

6. Incorporating Low Grazing Angle Reflections

Many rough surfaces appear very shiny at grazing angles. Grazing angle reflections occur when the peaks on a rough surface are slightly rounded. This rounding will cause light at low incidence to be reflected forward. This effect is only visible at similarly low view angles that look into the direction of the light source. Only the rounded tops of a rough surface are visible at such angles, resulting in a rough surface, which has a behavior similar to a polished one. The simulator attempts to capture the behavior of grazing angle reflections, which manifest differently, but occur on all rough surfaces. To simulate this behavior, the simulator truncates pyramids, giving them a flat horizontal surface. Figure 13 shows a rough surface with truncated tops and how light is reflected at low incidence levels.

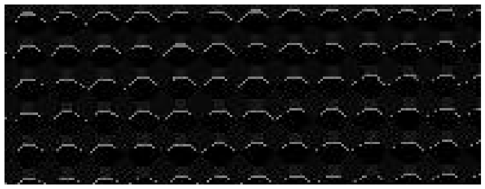
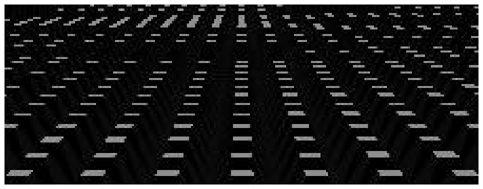
 Light reflected backwards from the front facet of the pyramid	 Light reflected forwards off the top facet from the pyramids

Figure 13: Example Of Near Angle Reflection

The level of truncation is variable which results in a varying amount of light reflected forward. As the truncation increases, the rough surface gradually becomes a flat surface in both appearance and behavior. As the truncation decreases, the rough surface reflects less and less light forward. This behavior is not observed if roughness is controlled by pyramid aspect instead of truncation.

IV. SIMULATION LIMITATIONS

It is vital to keep in mind that the quality of the output relates directly to the accuracy of the model. Despite adequate software design and testing, it has become apparent that the existing simulator has in it severe limitations, which make using it to model rough surfaces problematic at best. These limitations did not become apparent until examining the simulator's output data. If sufficient time allowed for each limitation to be address and handled, then the simulation tool's credibility would be much improved.

A. ALIASING AND PIXELATION

Computer graphics, like all parts of computers, works on a discrete set of numbers. No matter how large the buffers, or how big a value can be handled, there will come a point when precision is lost because the very last digit can only be a 1 or a 0. This same limitation can be found in images rendered by computers. Two problems have arisen which are attributed directly to this discreteness. While these problems are not directly responsible for the simulation tool's inability to satisfactorily model rough surfaces, they did contribute to a noticeable difference between real world observations and the computer simulation.

1. Pixelation

When viewing the ocean at a relative low altitude it is easy to make out whitecaps of individual waves and the deep blue of the sea. As you increase your altitude, the whitecaps begin to blur into the blue of the sea, but not completely disappearing. Finally, you will reach an altitude when sea blurs into a singular color. However, the color of the sea at high altitude is different on a windy day than a calm one. The white of the breaking waves and the blue of the seas combine into grayish blue. In effect, as the viewing distance increases between a set of objects and the viewpoint, a blurring or rather combining of color levels takes place. Unfortunately, this is not the case in computer graphics unless very specific code is designed to pixel blend. OpenGL does not automatically handle this situation. Figure 14 attempts to illustrate the effect caused by pixelation.

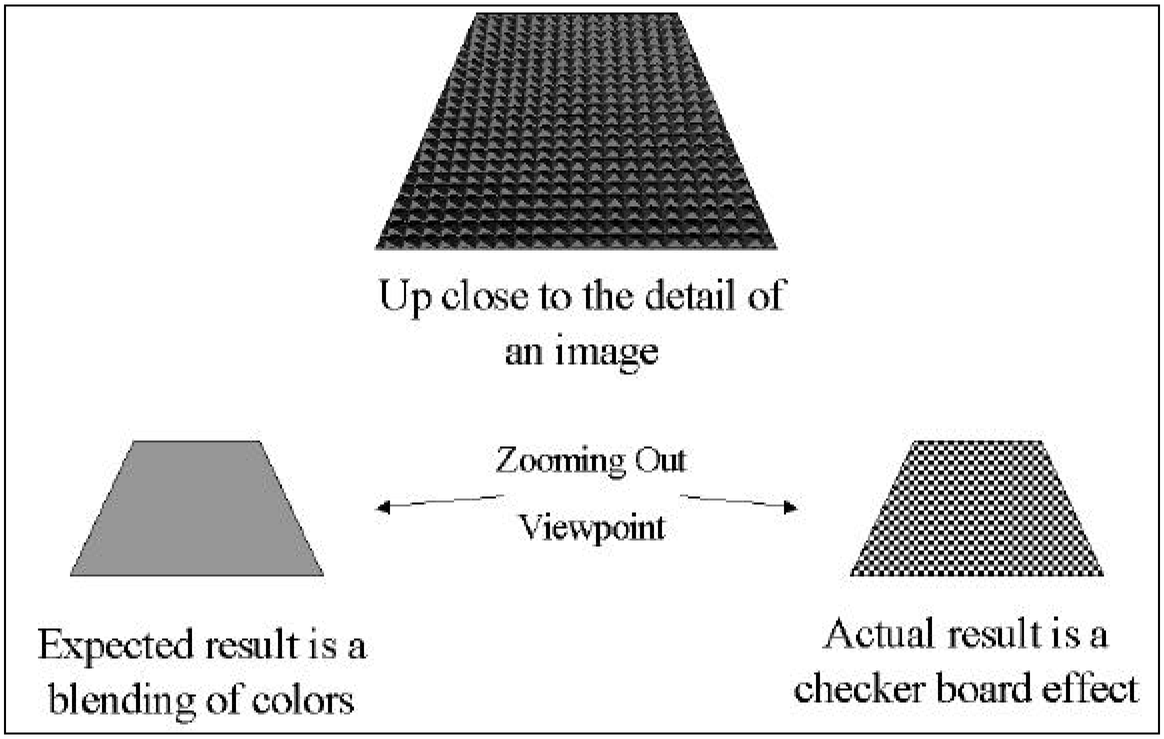

Up close to the detail of an image
Zooming Out
Viewpoint
Expected result is a blending of colors
Actual result is a checker board effect

Figure 14: Effects of Pixelation on Image Detail

The simulator suffered from pixelation when the pyramid size was reduced enough that multiple pyramid facets could be represented in a single pixel. Instead of OpenGL averaging the color value of all of the polygons to be rendered in that pixel, it takes the value of the closest one as part of its culling algorithm. When this happens over a large portions of the image, a mesh of various colors instead of nice uniform shade, begins to appear. The result is an image that appears significantly different from one that would result in the physical world.

Another example of pixelation in the simulator is the incorrect manner in which the tops of pyramids are draw in Figure 15. In this figure, red circles high-light areas that are affected by pixelation: it looks as if the tops of some of the pyramids are connected.

This is an artificiality created by rounding and floating-point error in computations carried out by OpenGL during the rendering process. The pyramids in the bottom of the image, which are also the closet, are affected less than those further away. It is only when the viewable portions of partially hidden polygons are smaller than the pixel size that incorrect pixel coloring can be seen.

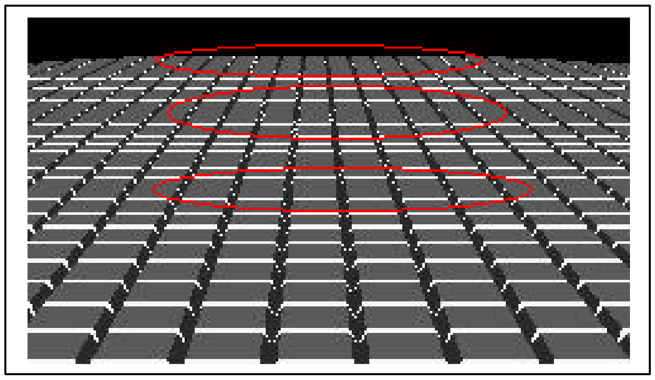


Figure 15: Example of Pixelation in Rough Surface Rendering

Over a group of images, this error becomes even more noticeable as flashing lines appear resulting from pixels being colored different from image to image. Pixelation of this type translates directly into the data as jagged peaks over a range of values. These jagged peaks introduced error, which hampers regression efforts.

2. Aliasing

Aliasing is a problem similar to pixelation, but it has different effects on the rendered image. Aliasing results when a line, which is not vertical or horizontal, is draw on a computer screen; the line appears jagged. This is a result of discrete pixels attempting to represent a non-discrete object. OpenGL supports various methods for eliminating aliasing effects. Such methods include using Fog algorithms or shading techniques. (Woo, p.233) Unfortunately, these techniques only apply to lines and not polygons; therefore they are unable to smooth the edges of the simulator's pyramids. The result is various geometric patterns emerging on the rough surface, which would not appear in the physical world.

The following image in Figure 16 is an example of aliasing. Like pixelation, aliasing is more prevalent for objects further away from the viewpoint than closer. Patterns resulting from aliasing appear in the section above the red line, while little or no patterns are seen in the lower half.

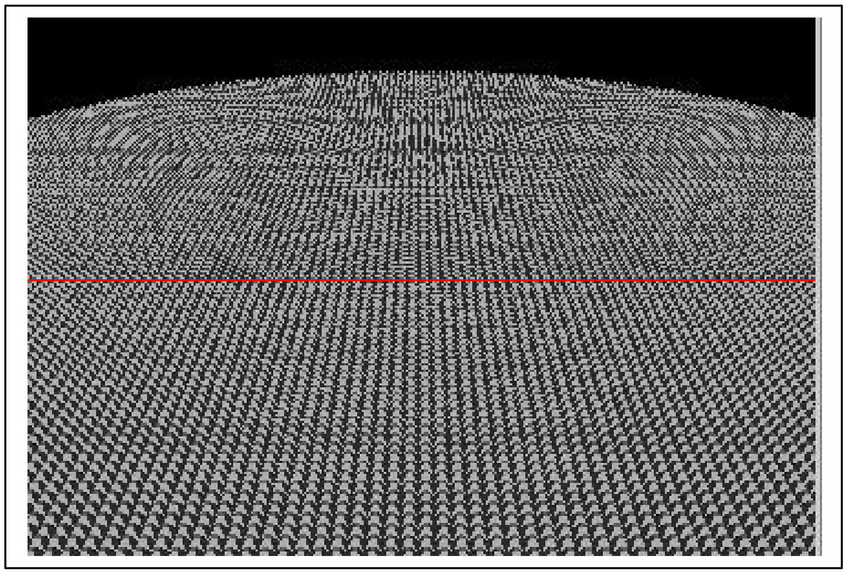


Figure 16: Example of Aliasing Effects in the Simulation Tool

While it is possible to see such patterns emerging from macroscopic rough surfaces, the intended correction to the OpenGL lighting equation is for uniformly distributed microscopic roughness. It is not entirely understood how significant aliasing and pixelation affect the simulations tool's measurements, but an image rendered with aliasing effects does fail to pass a visual test.

B. PROBLEMS RESULTING FROM MAXIMUM LIGHTING CONDITIONS IN OPENGL

Unlike the scientific community's practice of describing light in units of measure, ranging from 0 to infinity typically in watts/cm^2, OpenGL chooses to describe light in a unit-less fashion as well as limiting intensity levels to a maximum of 255. The first difficulty, which already has been mentioned in Chapter II, is that no direct mapping is possible from the physical world to the OpenGL lighting model. The second, and more important in terms of the simulation tool, is that it is possible to generate values greater than 255 in the specular portion of the OpenGL light equation. This breach of the 255 maximum is not possible for ambient and diffuse since the ambient value is a fractional function and the diffuse value is a cosine function of the source, and never exceeds a multiple of 1.0. However, the specular term's exponential nature does allow intensity values to exceed the maximum level even when the light source intensity is as low as 127. The result is a rather severe truncation of specular reflections when the light source's intensity approaches 255. Figure 17 is the result of measuring the intensities of both specular and diffuse reflection from two identically flat surfaces, one being illuminated by a light source of 127, the other by a light source of 255. The truncation can been seen when the reflecting light level increases beyond OpenGL's maximum value.

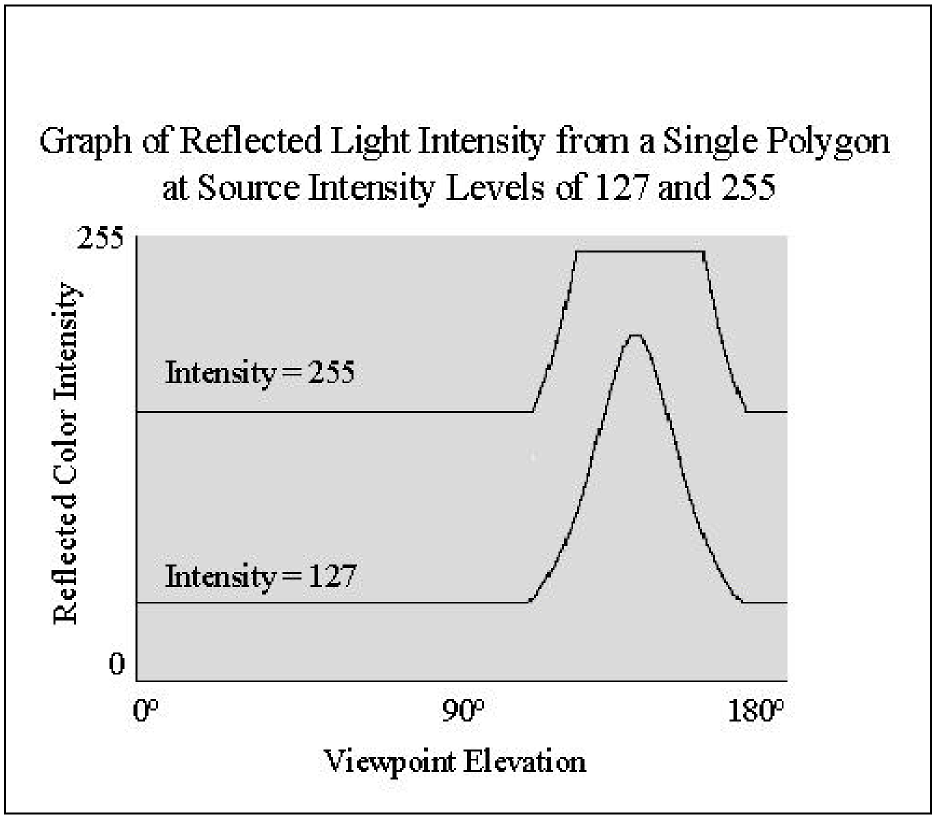


Figure 17: Example of Truncation of Specular Lobe at Maximum Source Intensity

This truncation significantly affects the behavior of specular reflections if a maximum light source setting is used. To avoid truncation error it is necessary to run simulations at low light levels. Since the reflected intensity of specular reflections is linear with respect to light source intensity level, varying light levels to avoid truncation will not adversely affect the results of the simulator.

C. PYRAMIDS POORLY REPRESENT ROUGH SURFACES

When designing the simulator, several geometric shapes were considered for use. In the end, a truncatable pyramid was selected based on the ease of which shadows could be calculated and the aspects of the pyramid adjusted. Unfortunately, using pyramids to model rough surfaces introduced several artificialities, which significantly affected the analysis of the simulation tool's output data. These artificialities are:

1. Non-Random Reflecting Normals

It is the general assumption that rough surfaces have an inherent uniform random distribution of surface normals.(Baer, 2001) This holds true for surfaces with a uniform roughness, since the actual rough texture is microscopic. OpenGL relies heavily on surface normals because it calculates specular and diffuse reflections with them. The pyramids used in the simulation tool's rough surface, were not constructed with any randomness in order to maintain reasonable shadow calculations. Unfortunately, this results in a rough surface with a single surface normal, and reflection behavior inconsistent with a physically based rough surface. If the aspects of the pyramids had a more random distribution, the resulting data would have more closely followed observations from physical rough surfaces.

2. Rectangular Reflection Patterns

In the physically based world, directional light that strikes a uniformly rough surface, like those found in nature reflects in the shape of a circular lobe. Figure 18 shows the circular shape of the backward reflection of light off of grass.

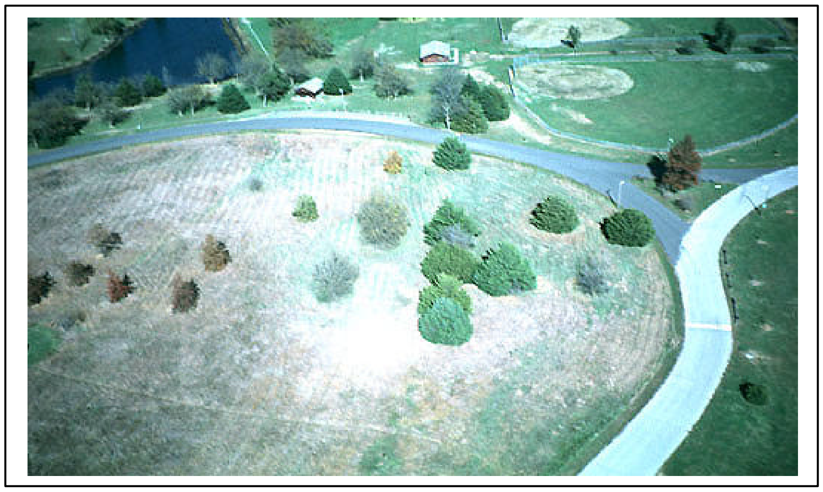


Figure 18: Halo Effect As Seen From An Airplane On A Grassy Field

The circular shape of the reflecting lobe is the result of the fall off created by the random distribution of surface normals characteristic of rough surfaces. This same circular pattern is captured in the OpenGL specular component by using a cosine function. The reflections produced by rough surfaces in the simulator do not have the same circular patterns found in the physically based world. This is due to the simulation tool's use of pyramids to model rough surfaces. The resulting reflections produced by the simulation tool takes on a rectangular shape, which significantly deviates from physically based rough surfaces when the viewpoint azimuth begins to shift either to the left or right of the light source. Figure 19 attempts to show this phase shift through a set of graphs. Below are artificial top-down contour plots, created to help show this phase shift from a different view aspect. It is the phase shift of the peak reflection lobe which significantly degrades the quality of the simulator's data, as the viewpoint changes azimuth away from the incidence.

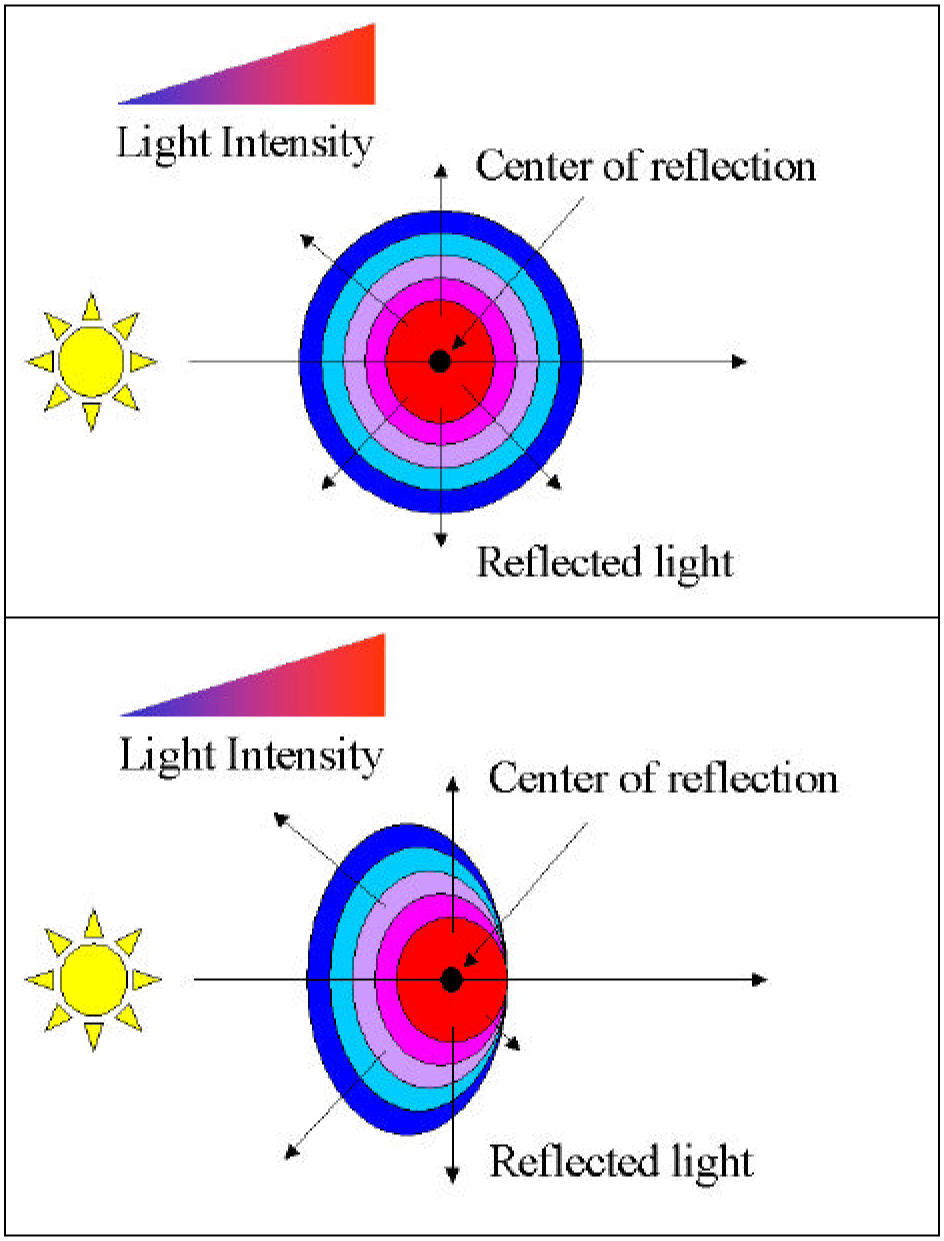


Figure 19: Charts Illustrating Phase Shift in Simulation Tool

D. VALIDITY OF THE SIMULATION TOOL

While significant limitations do exist in the simulator, it does provide a framework for improved rough surface representation. It shows that careful consideration is needed when modeling the rough surface as shown in the limitations inherent in using pyramids. However, it is also the case where the pyramid's limitations are minimized, that the expected behavior can be seen. Should the pyramid be replaced with a more suitable object or method, more accurate measurements might be possible.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DATA ANALYSIS

The emphasis of this thesis is the design and development of the rough surface simulator. With the simulator, various types of rough surfaces can be modeled and analyzed to examine their reflective behaviors. Several sets of data were collected from the simulator for analysis. The input parameters for these runs where configured to minimize the error induced by the simulator's limitations.

A. GOALS OF ANALYSIS

This analysis intended to develop a correction for OpenGL's light equation to approximate rough surface reflections. Due to the simulator's limitations, the confidence of the data allowed for only a limited analysis. The analysis did begin to identify reflection behaviors of the surface's specular and diffuse components. More importantly, the analysis helped identify many of the problems and limitations found in the simulator, which will improve its usefulness for future work.

B. SELECTING SIMULATION RESULTS

The simulator was set up to produced data varying two of its five input parameters:

- Pyramid Truncation- 0.0 to 1.0 in 0.1 increments
- Pyramid Aspect- 0.0 to 1.0 in 0.1 increments

Due to the limitation surrounding the characteristics of a pyramid, the incidence was held constant. Even though all incidence angles were examined to some extent, only one incidence angle allows the correct behavior to be seen. A model approximating the opposition effect should reflect the most light at 0^o phase-angle. Since the majority of the simulator's runs set up the pyramids for a 1.0 aspect, the surface normal of the facet with the brightest reflection is 30^{o} back in the direction of the light source. Therefore, the incidence angle that best produces the opposition effect behavior is also 30^{o}.

The simulator took measurements of the reflecting surface every degree of viewpoint elevation and every 5 degrees of viewpoint azimuth. Due to the deterministic nature of the simulator's measurements, duplicate runs for statistical analysis were not required. Examination of each surface produced roughly 6,500 points of data allowing for good data resolution.

C. USE OF DATA REGRESSION TO DEVELOPMENT MODEL

The data produced by the rough surface simulator was analyzed through a technique called *data regression*. Data regression is the process of examining the error between a data set and a purposed model. Through finding specific behaviors in the error, improvements to the overall model can be made. The process of regressing data is by no means automated and often requires human intuition and interaction. The result is a model or group of models that fit a data set to within an allowable margin error as set by the analyst. *Regression With Graphics* by Lawrence Hamilton is an excellent reference for further reading on data regression.

MathSoft's S-Plus data regression software package was used for the model development in this thesis. The details of the data regression will not be included, however, the values used to compare the fit of models to the data set, are included in Tables 1 & 2. S-plus' *General Linear Model* which uses a *Iterated Re-weighted Least Squares* regression function was used to estimate the errors of each model. The specific errors of the linear model examined are the intercept (b0), the slope (b1), and the Residual Sum of Squares (RSS). A model that properly fits the data set would have a b0 value near zero, and a b1 value near one. Models with smaller RSS values are thought to be a better fit. These values were satisfactory metrics for comparing the various models in this thesis.

It is important to point out that when data sets are typically regressed, models are developed against the idea of an ultimate truth; that is somewhere out there, unbeknownst to the analysis, a perfect model exists. In search of that truth, the analyst must develop models against the imperfect data points representing the truth. This thesis approached the process of data regression differently, in that the data itself is assumed to be the truth, and that models are developed to fit the data perfectly. The assumption that the data is the truth is willingly made knowing that the limitations inherent in the simulator corrupts the output enough to misrepresent the real behaviors to some level or degree. This assumption is acceptable knowing that the desired validity of the resulting model is only for an initial behavioral examination, and not one that completely defines the real truth. Furthermore, it is possible to pull some beneficial information about how rough surfaces really reflect light from a regression conducted in this manner.

D. DIVIDING THE LIGHTING MODEL

Chapter II illustrated the relationship between the OpenGL lighting equation and the Opposition Effect. It is the assumption of this thesis that manipulating the diffuse and specular terms in the OpenGL equation can approximate the opposition effect. Furthermore, we assume that the manipulation can be described as one or more mathematical functions. Therefore, the analysis will attempt to individually extract the behaviors of the diffuse and specular components. These individual components should still behave correctly when combined together, since under OpenGL the diffuse and specular components are combined using simple addition.

A special set of runs was required in order to separate the light intensity reflected by diffuse and specular components. While this did not adversely affect the behavior of both reflections, it did remove all effects of inner shadows from the specular reflections. The effects of the inner shadows were then examined in the diffuse reflection data. It was not possible through the simulator to maintain the effects of the inner shadows on the specular component while separating the diffuse component.

E. THE SPECULAR COMPONENT

The simplest rough surface is a smooth surface. In this case, specular reflection will result in a forward reflecting lobe. This means that the lobe will continue to travel away from the light source. Figure 20 is the specular reflection on a smooth surface for incidence angles varying from 0 to 90 degrees. The intensity level in this figure is 127. Since the intensity of all the reflections remained at 127, the intensity of the reflected light is dependent on the light source and not on the incidence angle. It is important to note that the measured intensity of reflected light from a surface is an average of many pixels taken from the center of the surface. Secondly, because light is broken into its three-color components in OpenGL, and since the surface is has only black, gray or white colors, the terms light and color can be used interchangeably.

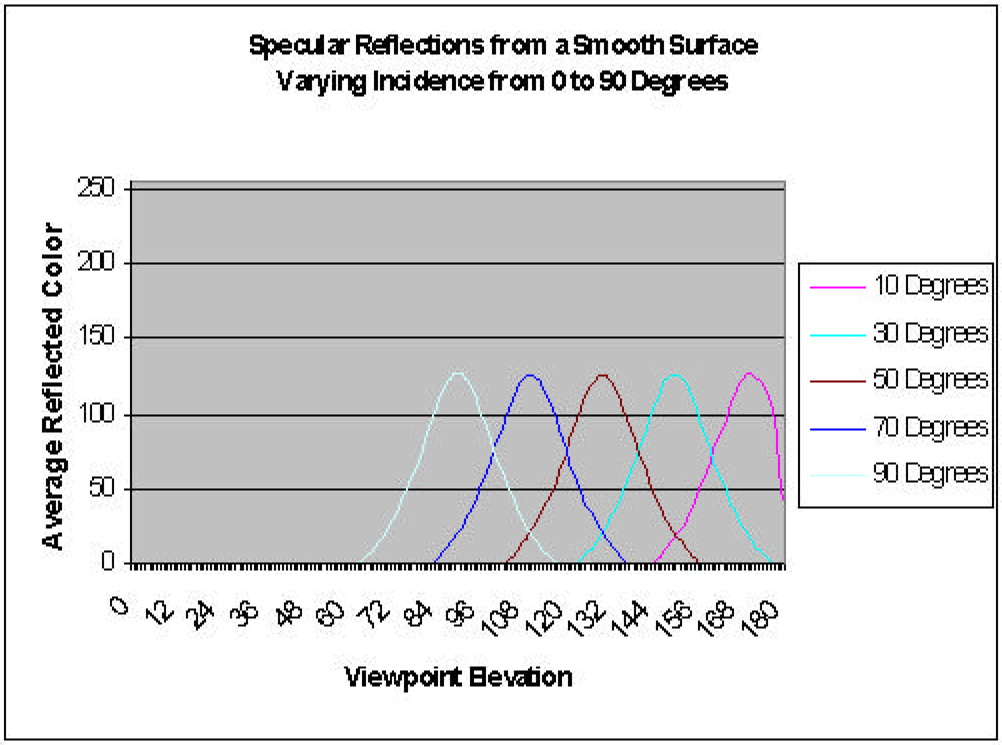


Figure 20: Specular Reflection On A Smooth Surface

The forward specular reflection decreases as the roughness increases while modifying either the pyramid's truncation value or the height-to-width aspect ratio. These modifications have different effects on how the specular reflection changes.

1. Effects of Truncation on Specular Reflections

A rough surface constructed of truncated pyramids results in two specular reflections; one forward and one back. When the surface is nearly smooth, the majority of the light is reflected forward. As the truncation value increases, and the cavities between pyramid tops grow larger, a backwards reflection lobe also appears. Graphing the reflected light as truncation increases from 0% to 100% shows an inverse relation between the intensities of the forwards and backwards reflected light. Figure 21 illustrates this relationship.

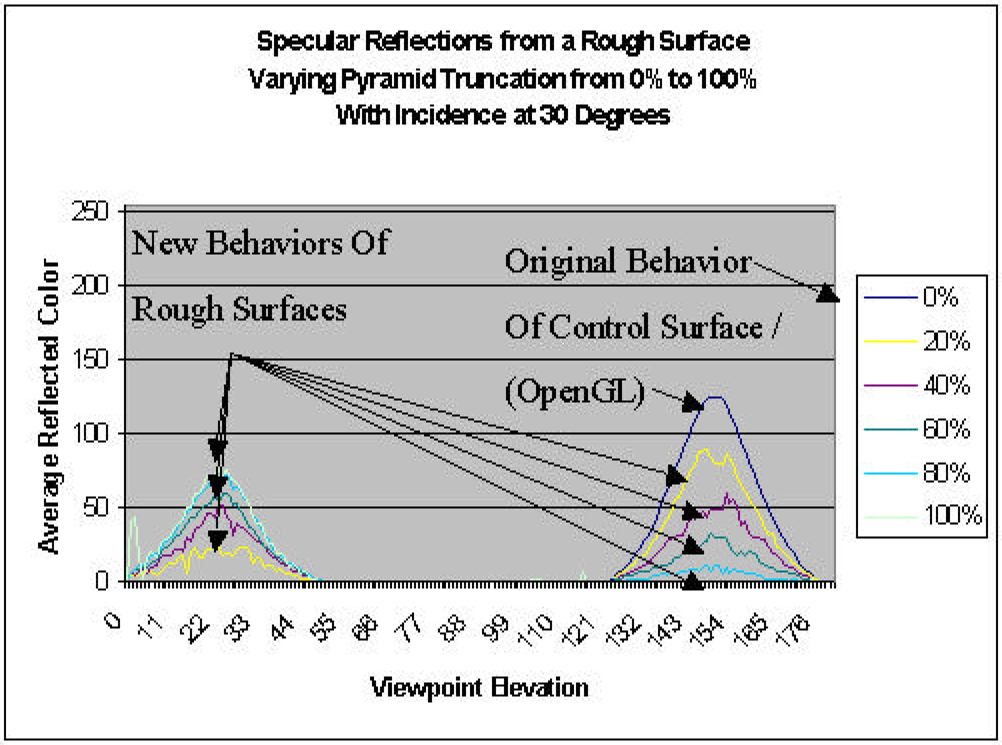


Figure 21: Specular Reflection Varying Truncation Value

The light reflected back towards the light source, illustrated by the left portion of the graph in Figure 21, is centered at 30^o elevation, which is also the zero phase angle since the incidence angle is 30^o. The forward specular reflection is also correctly centered at 150^o elevation. Some quick math shows that 150^o elevation is also 120^o phase angle, which is also twice the angular distance between the incidence angle and the surface normal. Figure 22 illustrates the behaviors displayed in Figure 21 into a more logical form.

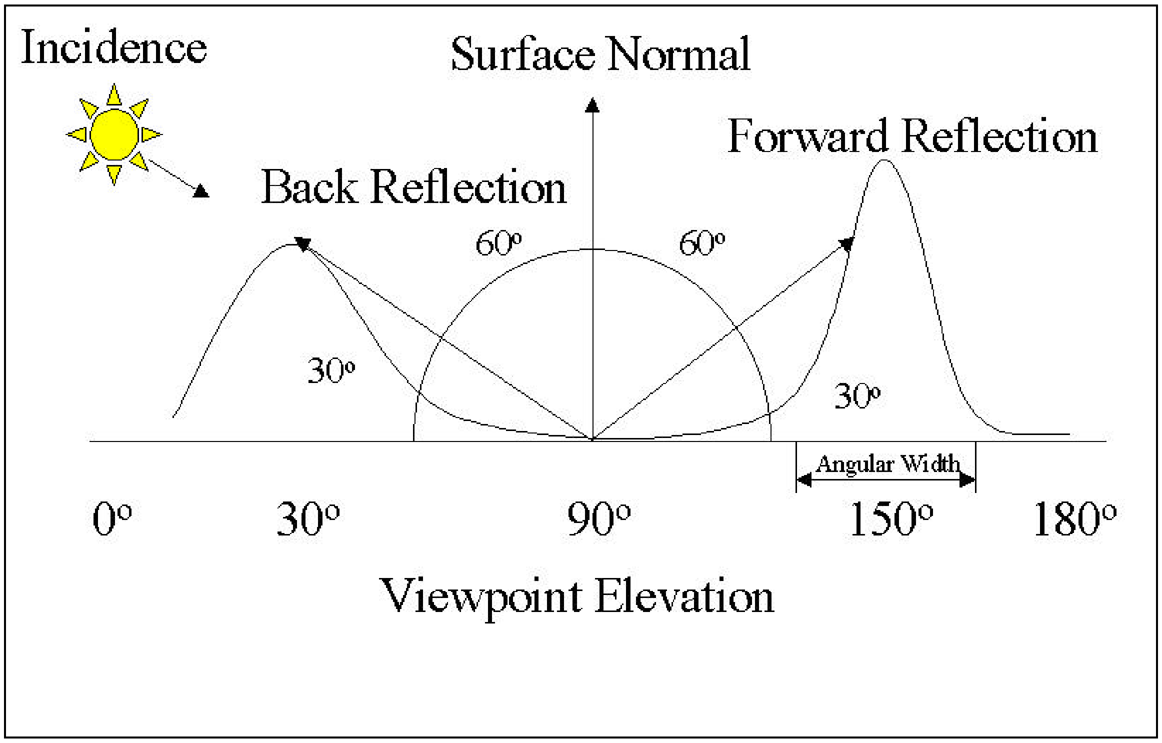


Figure 22: Elements of the Reflection Graph

a) Developing a Model for the Forward Specular Reflection

Even though the forward and back reflections have very similar characteristics, it is helpful to analyze the forward reflection first. Since the forward reflection already exists on smooth surface, starting with the original specular model was a logical step. It is clear from the behavior of the forward reflection lobe in Figure 21that the angular width of the lobe does not change when varying truncation values; only the reflected intensity changes. Therefore, a model attempting incorporate this behavior should focus only on changing the intensity of the reflection only.

At first glance, it looked as if the function for calculating the surface area of the top of the pyramid as a function of truncation level would serve as a good model for varying reflected intensity. For brevity, this function will be called the *square area* function from the fact that it calculates the area of the pyramid's top facet, which is a square, based on the level of truncation. This function is stated below:

$$I_{\text{reflected}} = m_{\text{source}} * \{2 * [(1 - t) / \tan(\theta)]\}^2$$

Where:

- t is the truncation value
- θ is the angle of the pyramid faces

If the light reflected by the pyramid's top was the only source of reflected light, this equation would work. However, there is a small amount of light which is reflected from the sides of the pyramid, thus increasing the overall intensity. A simple model, developed through data regression, captures the light intensity add from the sides of the pyramid.

$$I_{\text{reflected}} = m_{\text{source}} - [m_{\text{source}} * \sin(t * \pi/2)]$$

Figure 23 compares the data collected from the simulator to both the square area function and the regression model. In this comparison, roughness is attributed to varying the truncation level.

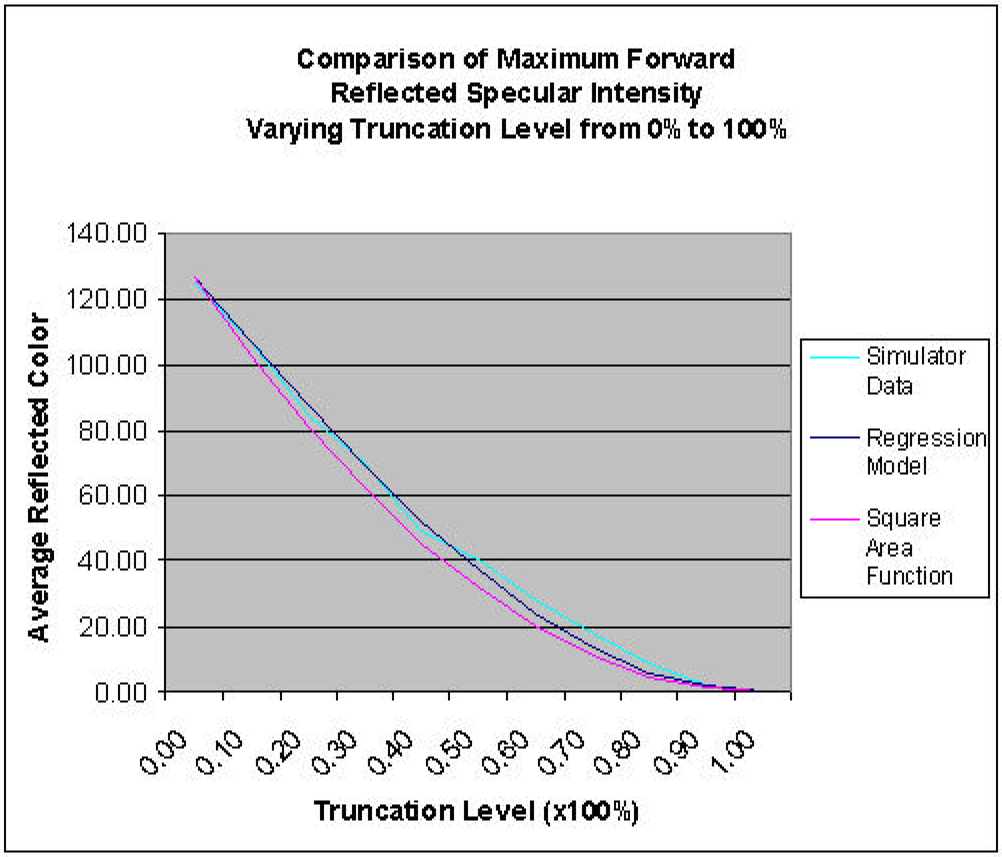


Figure 23: Comparison Of Forward Reflection Models To Simulation Data

Both models are good approximations of the behavior of the forward reflecting lobe. While not being a perfect fit, the regressed model is a better fit. Table 1 shows the errors of the models to the collected data.

	Mean Abs Error	Standard Deviation	b0	b1	RSS
Square Area Function	4.10	3.08	-4.684	1.012	122.407
Regressed Model	2.35	1.88	-2.268	1.032	51.918

Table 1: Comparison Of Statistical Results For Forwards Reflection

b) Adapting the model for to Backwards Specular Reflection

Having found a model that reasonably captures the behaviors of forward reflection light on a rough surface created with truncated pyramids, the model should be extended to handle back reflections. Referring back to Figure 21, the correlation between the forward and back reflection lobes is evident. This figure shows that as truncation level increases, the back reflection intensity increases as well, which is inverse to the behavior of the forward's reflection. It is also evident from the data that the angular width of the back reflection lobe is the same as the forward lobe. It seems plausible to use a model similar to the forward lobe. However, the model would have to take into account the decreasing intensity levels seen in the back reflection data.

The decrease in reflection intensity is due to the mathematical nature of the reflecting surface. In the forward reflection, light reflects off the tops of the pyramids, which have a square shape. Light reflected off the front face of a pyramid reflects from generally a triangular shape. The areas of both a triangle and square are driven by the truncation value. A graph showing the change in area of each shape as the truncation level changes is shown in Figure 24.

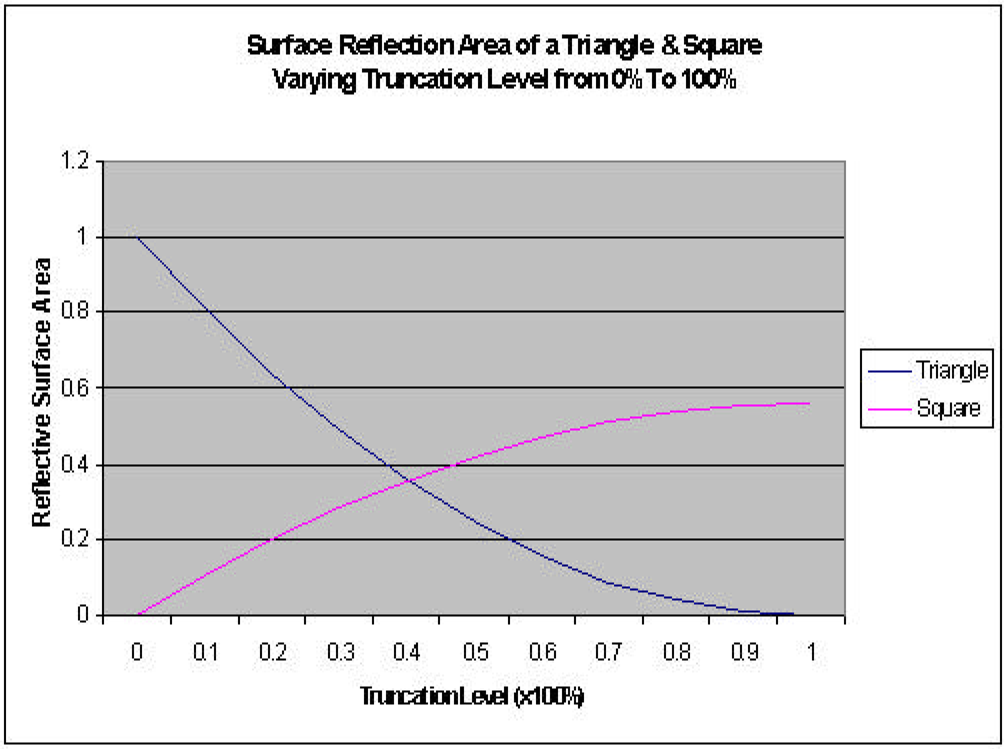


Figure 24: Effects of Truncation Level On Surface Reflection Areas

This graph illustrates the two elements behind the change in intensity as truncation levels change. First, when truncation level is 0.0 the majority of light reflected is from the square reflector (the top facet of the pyramid) and, that as the truncation level increase to 1.0, the triangle shape of the front facet dominates. Second, the ratio of area of the square at 0.0 truncation to the triangle at 1.0 truncation is the same as the ratio of the forward lobe at 0.0 truncation and the back lobe at 1.0 truncation.

Figure 25 shows that the behavior of the backward reflection is similar to that of the forward reflection. The mathematical function formula determining the area of a triangle is:

$$I_{\text{reflected}} = m_{\text{source}} * \{a - [a * (1 - t)^2]\}$$

Where:

- t is the truncation value

- *a* is the surface area of a pyramid facet

  $= \text{height}/[2 * \sin(\tan^{-1}(\text{height}/\text{width}))]$

As with the function for finding the area of the top pyramid facet based on truncation level, this function will be called the *triangle area* function. The data measured was tested against a model similar to the one developed for the forward lobe, which is:

$$I_{\text{reflected}} = m_{\text{source}} * \sin(t * \pi/2) / 2$$

Figure 25 and Table 2 shows the improved fit of the regressed model over the triangle area function.

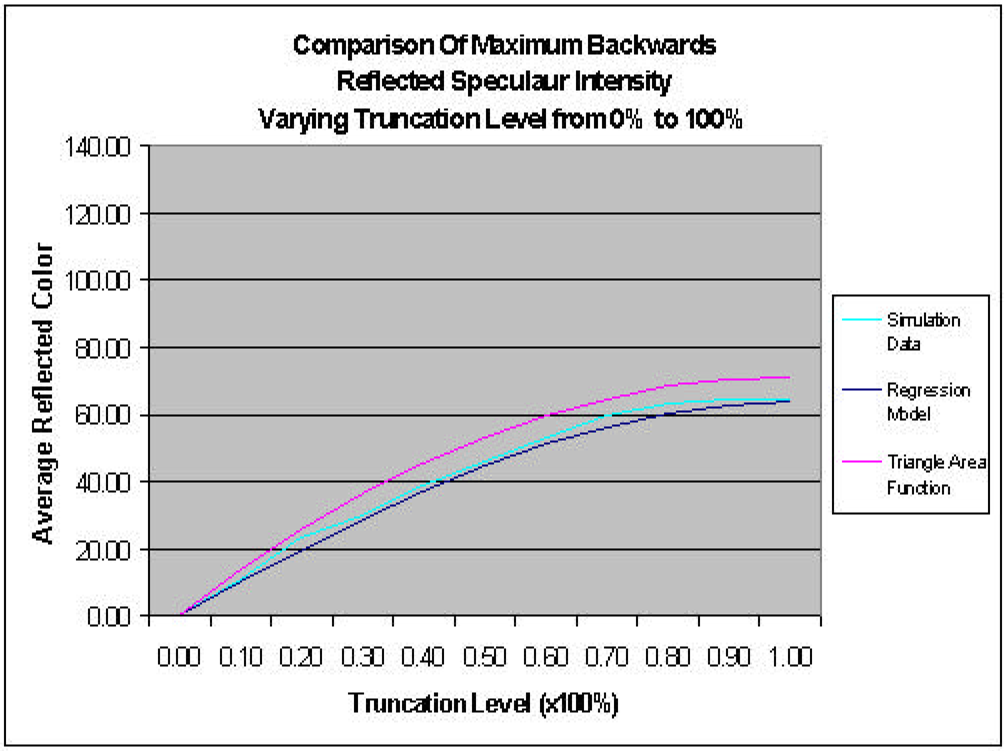


Figure 25: Comparison Of Backward Reflection Models To Simulation Data

	Mean Abs Error	Standard Deviation	b0	b1	RSS
Triangle Area Function	4.96	2.30	1.760	1.077	21.883
Regressed Model	1.71	2.40	-0.92	0.980	8.858

Table 2: Comparison Of Statistical Results For Backwards Reflection

The regression model has a closer fit to the data taken from the simulator than the straight mathematical function for determining the area of the triangular reflection based on truncation level.

2. Effects of Height-to-Width Aspect Ratio on Specular Reflections

Another method for varying surface roughness is varying the height-to-width ratio of the pyramid. A pyramid with a 0.0 aspect ratio will be completely flat, where a pyramid with an aspect ratio of 1.0 produces extremely rough surface. Changing the aspect ratio of the pyramid has a profoundly different effect on the behaviors of the two reflections. Where truncation level affected reflection intensity only and did not change the angular centers of those reflections, changing the aspect ratio affects both the intensity and the angular center of that intensity. This is reasonable, since changing the aspect ratio changes the surface normal of the primary reflection facet. As previous stated the effects of varying aspect ratio were not examined in depth. Figure 26 shows the shifting of the angular center as well as the change in intensity of the specular reflection as the aspect ratio increases.

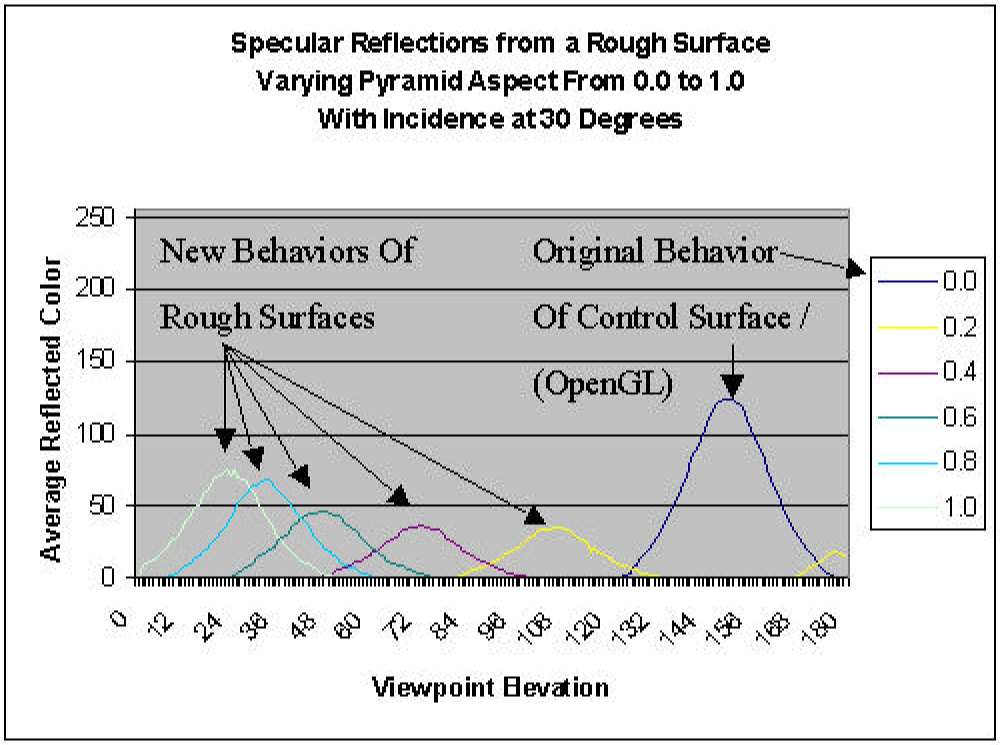


Figure 26: Specular Reflection Varying Aspect Ratio

The second lobe seen in Figure 26 when the truncation level is at 0.2 is actually the specular reflection off the backside of the pyramid. This reflection is seen only at very low aspect levels. This effect, if viewed at much smaller intervals of aspect ratio, would show that the forward specular reflection on a smooth surface actually divides into the two separate lobes. This division rapidly decrease in reflected intensity since the reflection break up into four smaller surfaces, each of which are reflecting light into a different direction. No further examination into this effect was conducted.

F. THE DIFFUSE COMPONENT

The diffuse component is far simpler than its specular counterpart: one need know only the incidence angle to determine the percentage of light reflected. In smooth surfaces, viewpoint is irrelevant, but on rough surfaces viewpoint is necessary for determining the intensity of the reflection. Despite its simplicity, we will see that the lack of shadow calculations in OpenGL will cause major errors in diffuse component reflection intensities.

1. The Effects of Truncation Level on Diffuse Reflections

The graph of the diffuse component on a smooth surface is rather uninteresting. Figure 27 shows the independent relationship of viewpoint and reflection intensity.

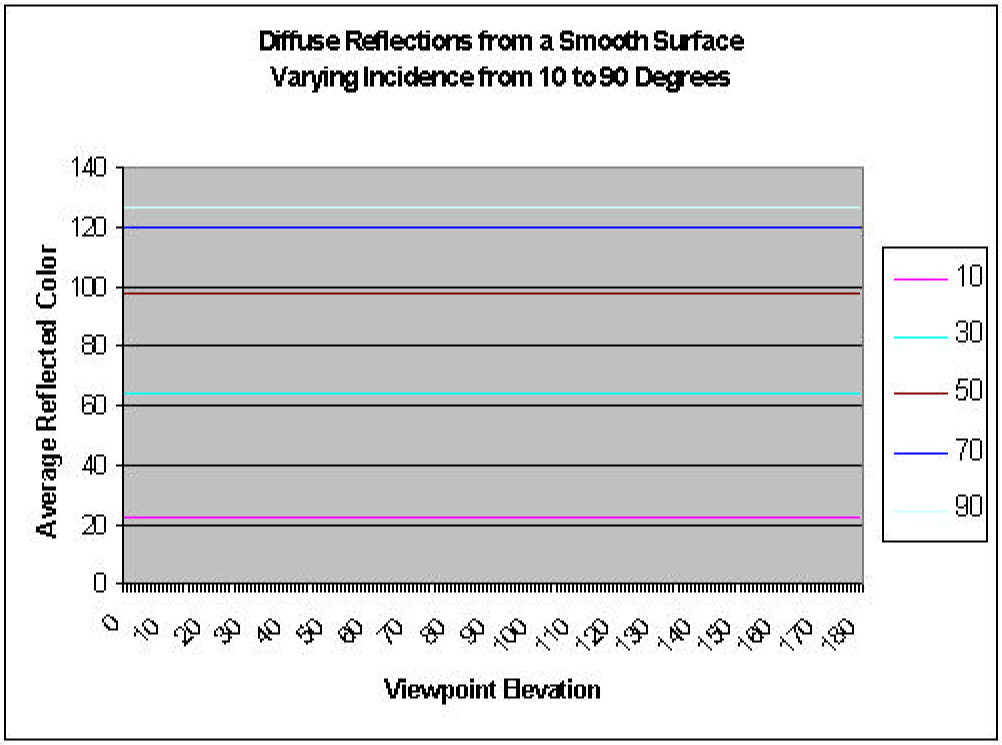


Figure 27: Reflection Intensity Of Diffuse Component On A Smooth Surface

Figure 28 shows how the intensity of the diffuse reflection changes as the surface becomes rougher by increasing the truncation level. A rough surface with a truncation level of 0% is identical to a smooth surface. The effects of truncation level on the diffuse reflection are more complex than with the specular component. As mentioned before, the value of the diffuse component remains constant for all viewpoints for a specific incidence angle when the surface is smooth. When the surface becomes rough, this no longer holds true; the intensity of the diffuse reflection is now viewpoint-dependent.

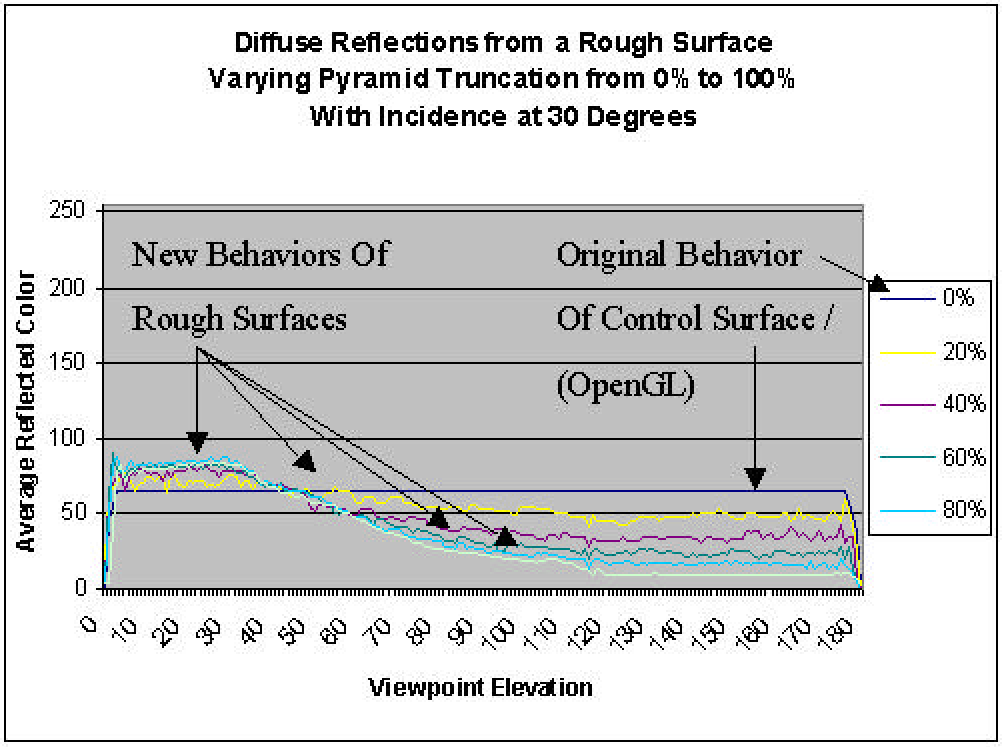


Figure 28: Diffuse Reflection Intensity Varying Truncation Level

The non-continuous behavior of the graph in Figure 28 suggests that the model describing rough surface diffuse reflection takes the form of three-part model. Even thought specific formulas were not derived to describe this complex behavior, it was possible to pull out the individual behaviors in each part of the model. Since all facets of the pyramid produce diffuse reflections, their specific intensities are constant throughout the range of viewpoint elevations. The overall intensity is derived from the portions of each facet that are viewable are each viewpoint. So in essence, the function describes how the view of the pyramid changes.

The model can be broken into three parts, which are described below. The domain of the model is described in terms of viewpoint elevation.

1. 0^o to incidence angle (zero phase angle)
2. Incidence angle to the angle at which the front face is no longer visible
3. From that angle to 180^o

Due to the nature of the pyramids used in the simulation tool and the desirability of a 30^o angle of incidence for the specular reflection, the domain is specifically broken down into 0^o – 30^o, 31^o – 120^o, and 121^o – 180^o.

The reflected intensities of all three parts are essentially driven by the amount pyramid's facet occupies a given view. For the first and third parts, the view is primarily occupied by the front and back facet respectively, and the top facet if the pyramid is truncated. Since the light reflected is diffuse, changing viewpoint with in this part does not change the reflected intensity of each facet, therefore, the overall reflected intensity remains constant. The second part incorporates the same facet comparison as the first and third parts, but the reflected intensity decreases from the inner shadows that are only visible in the second part. The effects of the inner shadows take on an exponential decay of the difference between the intensity levels of the first and third parts. This exponential decay could then be added to the intensity of the third part to produce an overall reflected intensity for the second part. It is unlikely that a continuous function exists that describes the all of behaviors of varying truncation level on the diffuse component.

2. Effects of Height-to-width Aspect Ratio on Diffuse Reflections

Changing the height-to-width aspect ratio has effects very similar to those from varying truncation level. It is expected that a similar model could be applied to the effects of varying aspect ratio as that applied to varying truncation level. Figure 29 shows behavior similar to that seen in Figure 28. The correlation between these effects was not examined further.

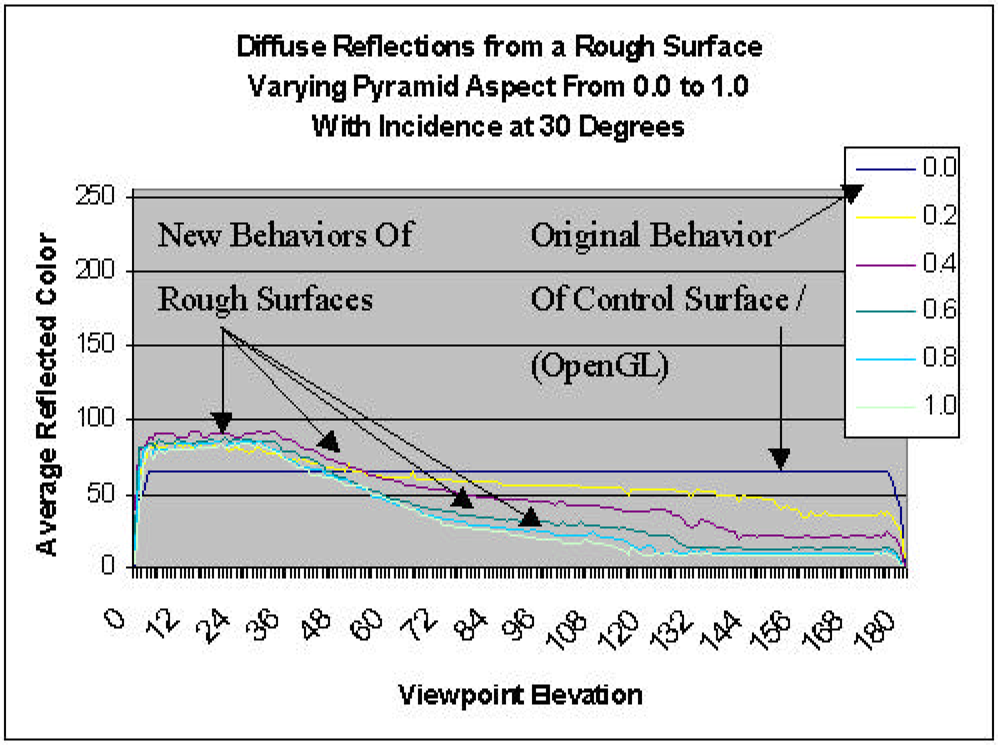


Figure 29: Diffuse Reflections Varying Height-to-width Aspect Ratio

G. COMBINING THE EFFECTS

This thesis did not examine the effects on the specular and diffuse reflections, as truncation level and height-to-width aspect ratio were varied simultaneously. This was considered beyond the scope of this thesis.

H. SUMMARY

It is evident that reflective behavior of surfaces change significantly when the surface is represented by more than one polygon and become more complex. The behaviors of both diffuse and specular reflections show considerable change when roughness is added through an increase of either truncation level or aspect ratio. This alone indicates the need for correcting OpenGL's lighting equation for rendering implicit rough surface behavior.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

The addition of rough surfaces to computer graphics is not a simple task. Rough surfaces have an infinite range of possibilities, even without considering those provided by nature. However, advancing computer graphics through the implementation of rough surface reflections still has its merit. Many in the computer graphics business attempt to implement rough surfaces through increasing the number of polygons rendered every second. They seek to create realism through explicitly giving objects texture with millions of polygons. Another method, with an aim similar to this one, seeks to attach the same realism through an implicit texturing rather than an explicit polyhedron. In essence, this seeks to change the math behind the lighting calculations to achieve the same result achieved by adding millions of polygons. Neither approach is necessarily better or worse than the other; however the implicit method is achievable on today's hardware, and the latter must wait.

This thesis suggests that adding an semi-empirical correction to the models used computer graphics is a viable method for significantly improving their over all realism, and provides a simulation tool to quantify rough surface behaviors. The behavior of rough surfaces has been compared to an astronomical phenomenon known as the opposition effect, in attempt to qualify the behavior of a rough surface and begin to understand how to model that behavior. Graphics libraries can already approximate the opposition effect explicitly. These same graphics libraries would benefit great if they could render the same behavior implicitly.

A computer graphics program using the OpenGL library was created to examine the behaviors of light reflection off rough surfaces. In the design and development of this program it became very clear that the task of creating an explicitly generated rough surface that match the reflection behaviors of real textured surfaces is not simple. The most basic element in the simulator also became the biggest limitation:

1. Rough surfaces, as the human eye sees it, are continuous. Even the microscopic details of the surface have perfect color and shape. Computer graphics on the other hand is not capable of rendering continuous forms.

Its discrete nature will only ever be able to approximate that which is continuous. Due to the complexity of rough surface reflections, that approximation carries over into undesirable artifacts that dilute the real behaviors. Thus making modeling such behaviors more difficult.

2. Rough surfaces can be thought of as surfaces composed of randomly placed facets, which have, when taken as a whole, have a uniform appearance. Any attempt to simulate rough surfaces must incorporate this random nature.
3. Discrete geometric forms are not the best building blocks for constructing a rough surface. Even if these forms are reduced in size small enough to become microscopic, their geometry will still carry though and dominate, thus inducing further error into the data.

A simulation, if able to overcome these stumbling points, would be a power tool for examining the reflection behavior of simulated rough surfaces. Unfortunately, this is only the start for developing a model. Because the algorithms used to draw the effects of lighting in computer graphics are mathematically based, so must the model be.

With data provided by the simulator, specific behaviors were examined. These behaviors were characterized into the two types of reflected light: diffuse and specular. Each component is a function of the incidence angle, viewpoint, and the two methods in the simulation program for varying the rough surface: height-to-width aspect ratio and truncation level. Despite the problems inherent in the data, which resulted from limitations in the simulation program, specific behaviors were extracted and models developed for the simplest behavior.

1. It is possible for two separate specular reflections to occur, depending on pyramid truncation level.
    a. For the forward reflection:
        i. The angular center of its reflection will always be twice the angular difference between the incidence and surface normal.
        ii. The reflection intensity can be modeled by a decreasing sine function of truncation level and source intensity.
    b. For the backwards reflection:
        i. The angular center of its reflection is driven by pyramid's height-to-width aspect ratio.

   ii. The reflection intensity can be modeled by the source intensity and either an increasing sine function of the truncation level decreasing cosine function of the aspect ratio.

2. The diffuse reflection was found to have a three part model. This model worked for rough surfaces varying either truncation level or height-to-width aspect ratio. No explicit mathematical function was found; however, a simple behavioral model was developed with the following attributes:

   a. The first and third steps acted like the original Gouraud shading function driven by the primary viewable facet.

   b. The second step exhibited an exponential decay of the difference between the intensities of the first and third parts.

3. Combining the effects of varying truncation levels and aspect levels was not examined.

While no final model exhibiting the characteristics of rough surface reflections was created, important behavioral patterns were identified. Correcting the limitations of the simulation and re-examining the data thus produced, should lead to better-defined behaviors, and more refined models.

Despite the advancement of computer hardware and non-real time graphics rendering, this approach may be the only real method for making realistic three-dimensional graphics work in a real time interactive environment.

Finally, one rule is always paramount in computer graphics, which is also the driving force behind this thesis:

IF IT LOOKS GOOD, IT IS GOOD.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: GLOSSARY OF TERMS

Ambient – Ambient light is non-directional and distributed uniformly throughout space. Ambient light falling upon a surface approaches from all directions. The light is reflected from the object independent of surface location and orientation with equal intensity in all directions. (Woo, p685)

Antialiasing – A rendering technique that assigns to pixels the color of the primitive being rendered, regardless of whether that primitive covers all or only a portion of the pixel's area. This results in jagged edges. (Woo p.686)

Culling – The process of eliminating polygons from being rendered either from hidden surface removal or level of detail management. (Woo, p.690)

Diffuse - Diffuse lighting and reflection accounts for the direction of a light source. The intensity of light striking a surface varies with the angle between the orientation of the object and the direction of the source. A diffuse material scatters that light evenly in all directions. (Woo, p.691)

Incidence Angle – The angle that a line (as a ray of light) falling on a surface or interface makes with the normal drawn at the point of incidence. (Webster)

Lambertian – see *diffuse*.

Opposition Effect – In a medium in which the particles are large when compared with the wavelength, particles near the surface case shadows on the deeper grains. These shadows are visible at large phase angles, but close to zero phase-angle, they are hidden by the object that cast them. Thus, the effect may only be thought of as being caused by shadow hiding. (Hapke, p.217)

Phase Angle – The angular difference between the *incidence* and viewpoint angles. In vector form, the phase angle can be calculated by: $\theta = \cos^{-1} \{(\underline{\mathbf{v}} \cdot \underline{\mathbf{w}})/([\underline{\mathbf{v}}][\underline{\mathbf{w}}])\}$

Specular - Specular lighting and reflection incorporates reflections off shiny objects and the position of the viewer. Maximum specular reflectance occurs when the angle between the viewer and the direction of the reflected light is zero. A specular material scatters light with greatest intensity in the direction of the reflection, and it is brightness decays, based upon the exponential value shininess. (Woo, p.702)

APPENDIX B: OPERATION MANUAL FOR ROUGH SURFACE SIMULATOR

Program Requirements

- Java Runtime Environment (rev 1.3.0 or newer)
- Java3D w/ OpenGL Library (rev 1.2.0 or newer)
- Rough_Surface_Simulator.jar

System Requirement

- At least 800 MHz processor, 1GHz recommended
- At least 512 Mbytes Ram, 1 Gbytes recommended
- nVidia GeForce w/ 32 Mbytes VRam (or equivalent)

Installation and Setup Directions

1. Install Java Runtime Environment (JRE)
2. Install Java3D – installer should automatically install graphics library into JRE directory structure.
3. Create a "c:\RSSimulator" directory
4. Put "rssimulator.jar", & "Graphics User Interface.bat" & "Batch file.bat" into c:\RSSimulator
5. Create a "c:\RSSimulator\data" directory
6. Create a "c:\RSSimulator\images" directory
7. Shutdown and Reboot Computer

Running the Rough Surface Simulator from the Command Line

The rough surface simulator allows a command line option for program execution to allow for batch file operation. There are some additional Java command-line options that are required in order to properly set up the JRE, these options must be included. All program options must also have a value. The following is a list of options:

- Pyramid Height-to-width Aspect Ratio (recommended 0.0 or greater)
- Pyramid Base Size (recommend 0.5 or less)
- Pyramid Truncation Height Level (recommend 0.0 to 1.0)
- Incidence Angle (0 to 90 degrees)
- Image Capture switch (“Image” or “NoImage”)
- Run Once Switch (“Once” or “NotOnce”)
- Elevation Rotation Increment (1 to 180 degrees)
- Azimuth Rotation Increment (1 to 180 degrees)

Examples:

```
java –mx512m –cp rssimulator.jar rssimulator.GUI 1.0 1.0 0.5 30 Image Once 1 5
```

This command line would run the program with 512 Mbyte memory model, use classes stored in the rssimulator.jar file, an aspect ratio 1.0, a truncation level of 100%, a pyramid base of 0.5, incidence angle of 30 degrees and the program would capture each image rendered and save it as a file in the image directory.

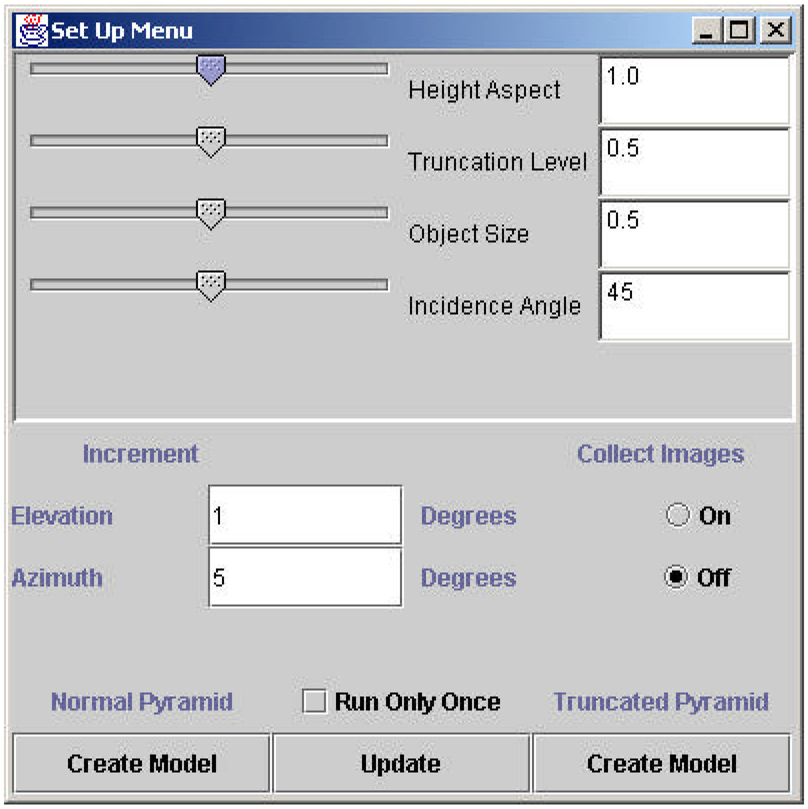
Running the Rough Surface Simulator from a Batch File

The program maybe run as part of a batch file to automate a large data run. In the “c:\RSSimulator” directory, “Batch File.bat” is an example batch file. All parameters must be appropriately filled as in the Command-Line execution. Any number of runs can be added to the batch file. It recommended that all runs in a batch file, have the

"NoImage" switch, as the images from a single run can take up several Mbytes of hard drive space.

Running Rough Surface Simulator as a GUI Application

Running the Rough Surface Simulator is the preferred method, but allows for only one run at time. Executing the "Graphics User Interface.bat" file (which is located in "c:\RSSimulator" directory) will start the opening menu, which will ask the user to set the parameters for aspect ratio, truncation level and incidence angle of the rough surface. The following picture shows the layout of the opening menu.



Height Aspect 1.0

Truncation Level 0.5

Object Size 0.5

Incidence Angle 45

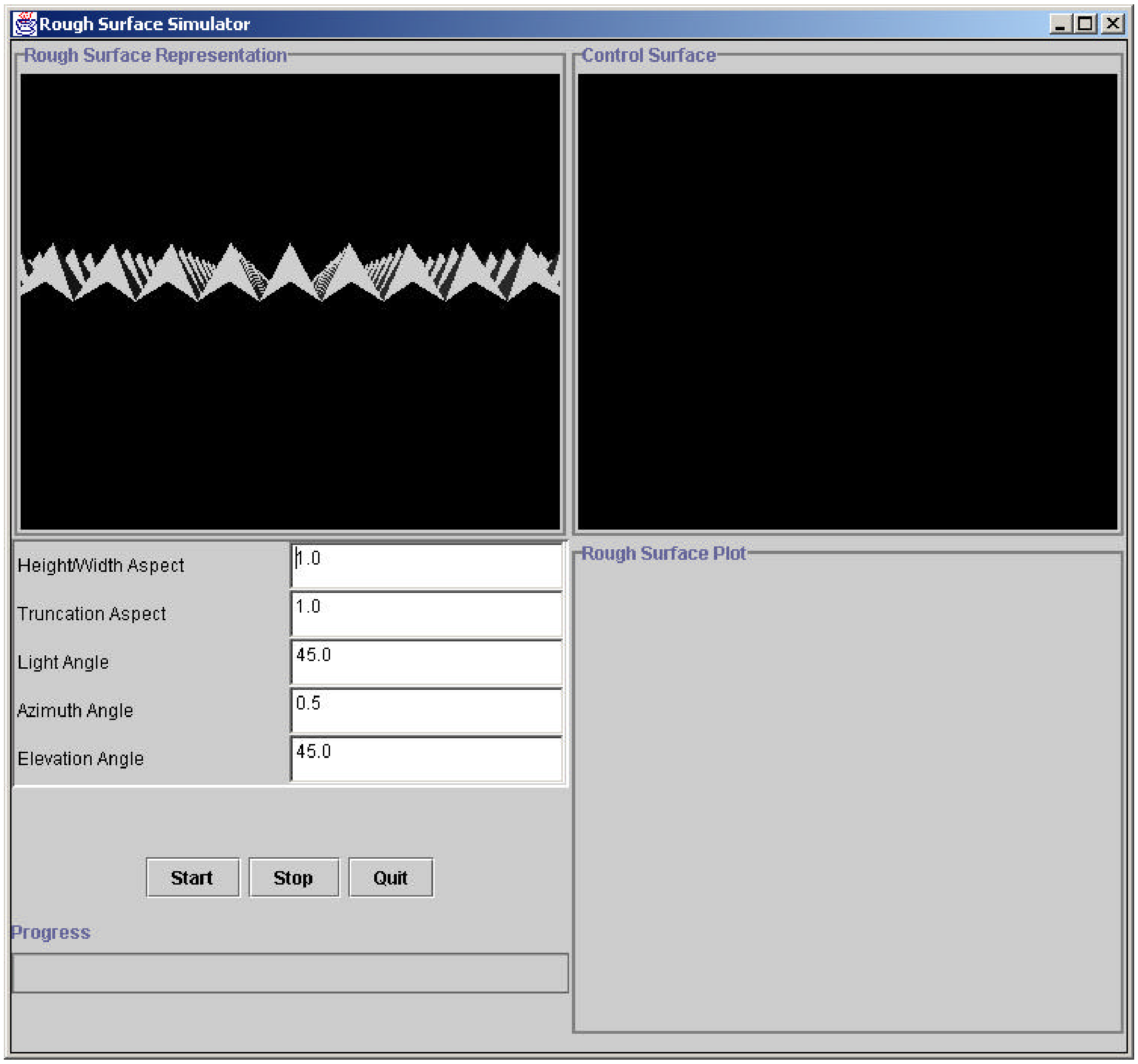
Elevation 1 Degrees

Azimuth 5 Degrees

On [✓ Off]

☐ Run Only Once

Currently, this rough surface is set for an aspect ratio of 1.0, truncation level of 50%, an object size of 0.5 and an incidence angle of 45 degrees, the option to capture images of the rough surface is turned off and the elevation and azimuth rotations will increment by 1 and 5 respectively. If either rotation increment is modified, the user will need to click the "Update" button to ensure changes are registered in the program. After setting the desired parameters, the rough surface can be generated by clicking on 1 of 2 "Create Model" buttons. The *Pyramid Model* button will disregard any truncation level value set and render a scene of full pyramids. The *Truncated Pyramid Model* will generate a surface with the set truncation level. This window will be replaced with the simulator window.



Height/Width Aspect	1.0
Truncation Aspect	1.0
Light Angle	45.0
Azimuth Angle	0.5
Elevation Angle	45.0

From here, clicking on the Start button begins execution of the simulator. If for some reason it is desirable to stop the application the click the **Stop** button. It is important note, that once the application is stopped, it cannot be restarted. This window must then be closed by clicking on the **Quit** button. All data stored up to this point will be saved in an appropriately titled file and stored in the data directory. The program will close all windows on its own.

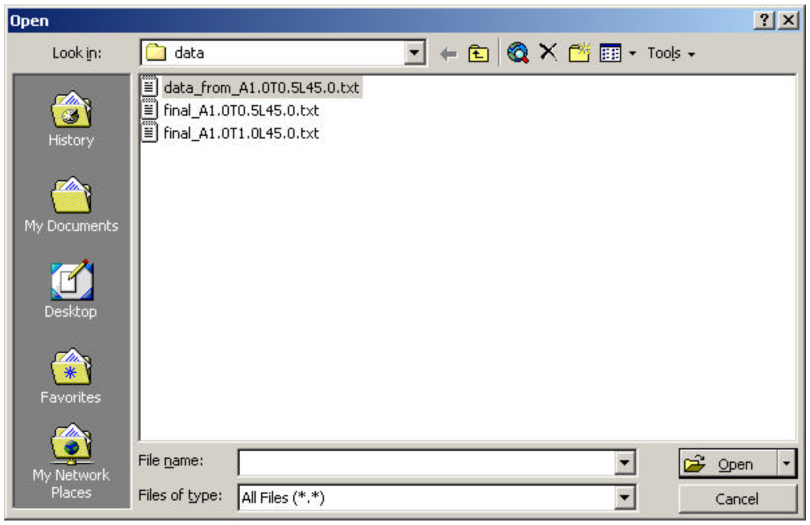
Converting Simulation Results into an Excel Spread Sheet

After a run has been completed, the data will be stored in the "c:\rssimulation\data" directory. An example file for a simulation run with a height-to-width aspect ratio of 1.0, a truncation level 50% and a incidence angle of 45 degrees would be named "example_data_from_A1.0_T0.5_L45.0.txt". The contents of the file would look something like:

```
AzGamma ElGamma Roughness Incidence Control Rough Final Azimuth Elevation

0.0     -45.0   0.5       45.0      0.0     27.81 27.81 0.0     0.0

0.0     -44.0   0.5       45.0      0.0     27.81 27.81 0.0     1.0

0.0     -43.0   0.5       45.0      63.0    80.22 80.22 0.0     2.0

0.0     -42.0   0.5       45.0      76.5    56.69 56.69 0.0     3.0
```

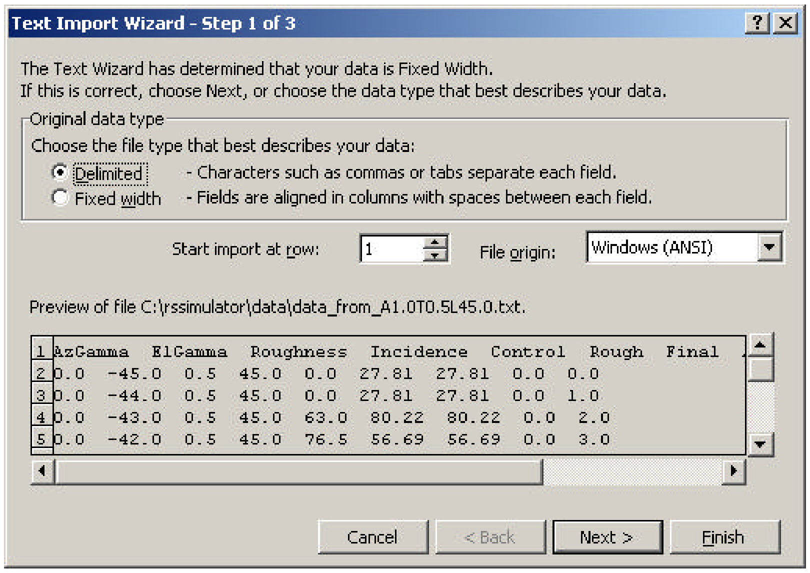
To convert this file into a working Microsoft Excel work sheet, follow the step-by-step procedure:

1. Open Microsoft Excel
2. Under **Files**, select the **Open** function
3. Navigate to the "c:\rssimulator\data" directory and select the desired data file
4. Change the file type to be opened to **All Files *.***
5. Click on the **Open** button (see the illustration on the next page for help)



data_from_A1.0T0.5L45.0.txt
final_A1.0T0.5L45.0.txt
final_A1.0T1.0L45.0.txt

6. For *Original Data Type*, select **Delimited**, and click on **Next**



The Text Wizard has determined that your data is Fixed Width.
If this is correct, choose Next, or choose the data type that best describes your data.

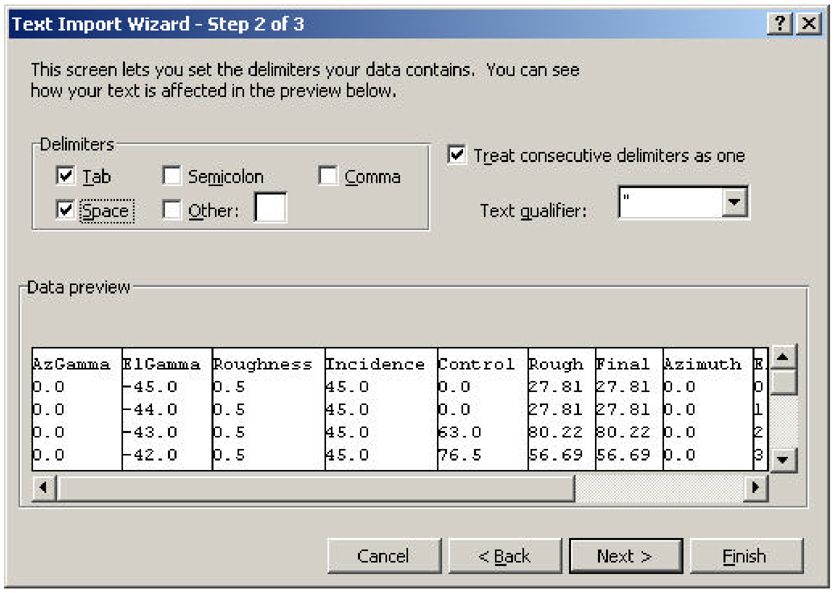
Original data type
Choose the file type that best describes your data:
[✓ Delimited] - Characters such as commas or tabs separate each field.
Fixed width - Fields are aligned in columns with spaces between each field.

Start import at row: 1 File origin: Windows (ANSI)

Preview of file C:\rssimulator\data\data_from_A1.0T0.5L45.0.txt.

```
1 AzGamma  ElGamma  Roughness  Incidence  Control  Rough  Final
2 0.0   -45.0   0.5   45.0   0.0   27.81   27.81   0.0   0.0
3 0.0   -44.0   0.5   45.0   0.0   27.81   27.81   0.0   1.0
4 0.0   -43.0   0.5   45.0   63.0   80.22   80.22   0.0   2.0
5 0.0   -42.0   0.5   45.0   76.5   56.69   56.69   0.0   3.0
```

7. Now, add **Spaces** to the range of *Delimiters*, then click on **Finish**



Text Import Wizard - Step 2 of 3

This screen lets you set the delimiters your data contains. You can see how your text is affected in the preview below.

Delimiters: Tab [✓], Semicolon [ ], Comma [ ], Space [✓], Other: [ ]

Treat consecutive delimiters as one [✓]

Text qualifier: "

Data preview

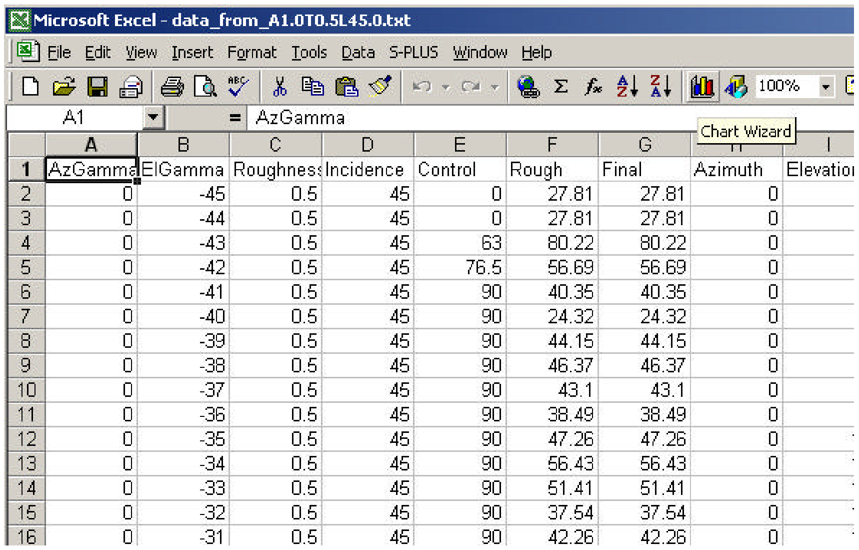
```
AzGamma ElGamma Roughness Incidence Control Rough Final Azimuth E
0.0     -45.0   0.5       45.0      0.0     27.81 27.81 0.0     0
0.0     -44.0   0.5       45.0      0.0     27.81 27.81 0.0     1
0.0     -43.0   0.5       45.0      63.0    80.22 80.22 0.0     2
0.0     -42.0   0.5       45.0      76.5    56.69 56.69 0.0     3
```

8. Your now have the data represented in a working Excel spreadsheet

Creating a Standard Chart of the data

To recreate charts as seen in Chapter V, complete the following steps:

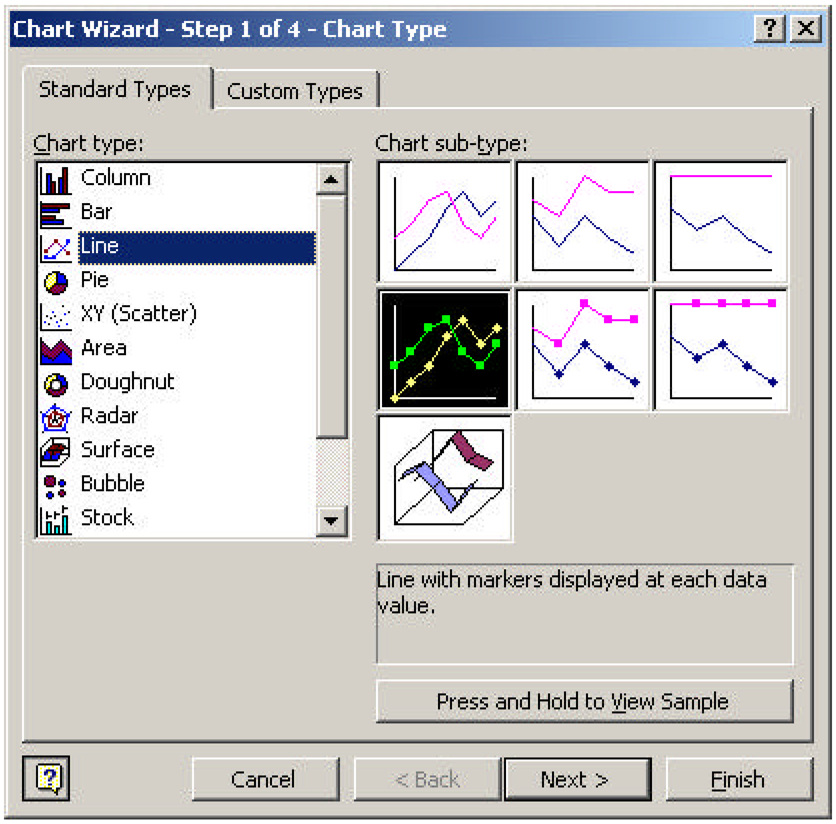
1. Select the Chart Wizard, which is the button with the blue, yellow and red bar graph on it



Microsoft Excel - data_from_A1.0T0.5L45.0.txt

	A	B	C	D	E	F	G	H	I
1	AzGamma	ElGamma	Roughness	Incidence	Control	Rough	Final	Azimuth	Elevatio
2	0	-45	0.5	45	0	27.81	27.81	0	
3	0	-44	0.5	45	0	27.81	27.81	0	
4	0	-43	0.5	45	63	80.22	80.22	0	
5	0	-42	0.5	45	76.5	56.69	56.69	0	
6	0	-41	0.5	45	90	40.35	40.35	0	
7	0	-40	0.5	45	90	24.32	24.32	0	
8	0	-39	0.5	45	90	44.15	44.15	0	
9	0	-38	0.5	45	90	46.37	46.37	0	
10	0	-37	0.5	45	90	43.1	43.1	0	
11	0	-36	0.5	45	90	38.49	38.49	0	
12	0	-35	0.5	45	90	47.26	47.26	0	
13	0	-34	0.5	45	90	56.43	56.43	0	
14	0	-33	0.5	45	90	51.41	51.41	0	
15	0	-32	0.5	45	90	37.54	37.54	0	
16	0	-31	0.5	45	90	42.26	42.26	0	

2. Select the Line Graph option

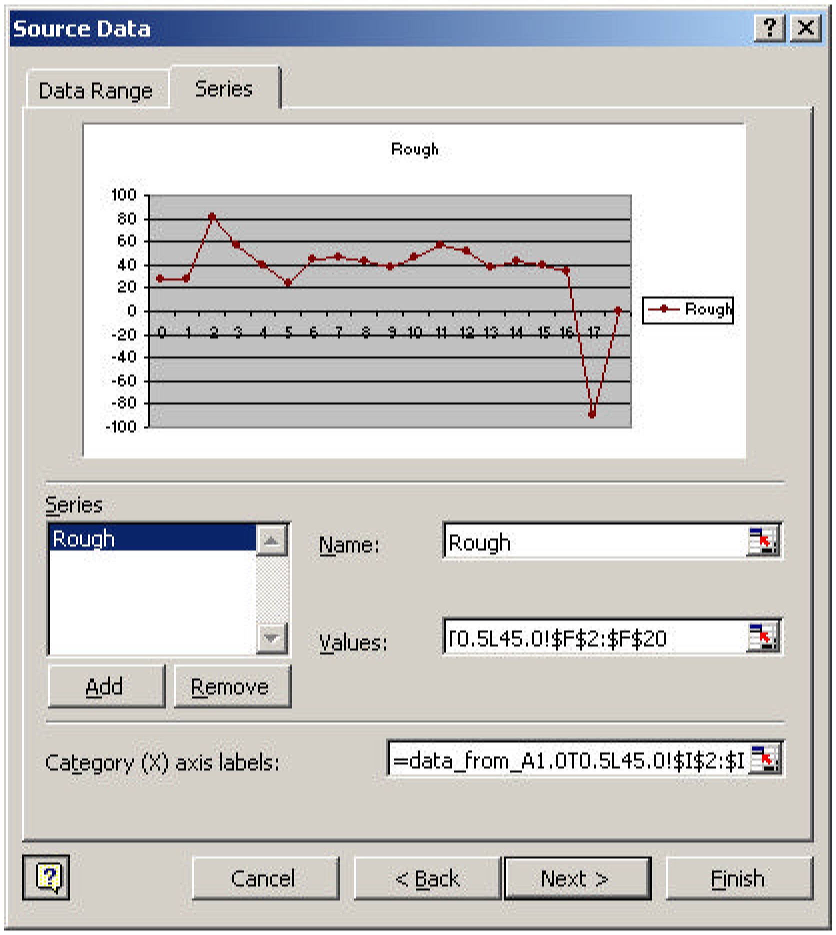


Line with markers displayed at each data value.

3. Select the **Series** tab on the next window

4. Remove all undesired columns from the series window

5. Add the Elevation column to the **Category(X)axis labels:**

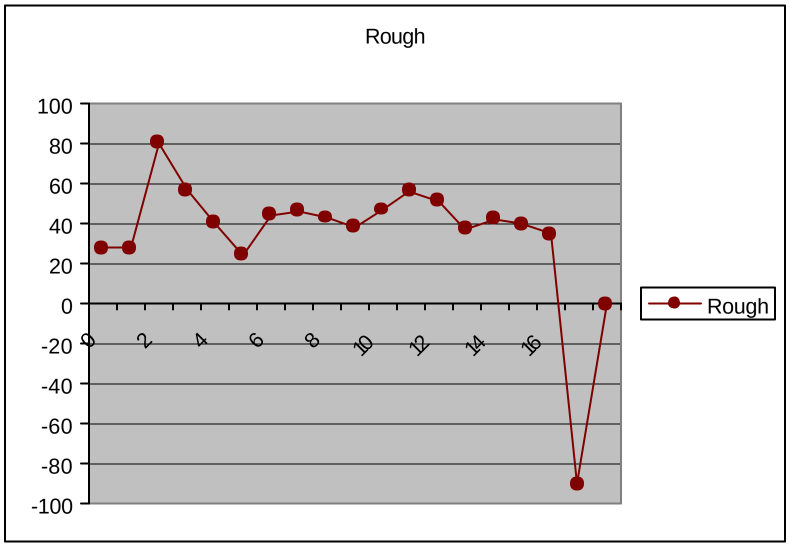


Name: Rough

Values: T0.5L45.0!$F$2:$F$20

Category (X) axis labels: =data_from_A1.0T0.5L45.0!$I$2:$I

6. Click the **Finish** button, and you are done



7. Of course axis and charts labels need to be added, this can be accomplished through clicking on the graph with the mouse, clicking the right mouse button, selecting **Edit Chart Object** and then **Chart Options**
8. The rest should be self-explanatory

LIST OF REFERENCES

1. Baer, W. et al, *Toward Standard Rendering Equation For Intrinsic Earth Surface Classification*, 2000 Spring Simulation Interoperability Workshop, Workshop Paper 00S-SIW-70 March, 2000

2. Baer, W., Personal Communications, September 2000 – August 2001

3. Bui-Tuong, Phong, *Illumination for Computer-generated Pictures*, Comm. *ACM*, 18(6), June 1975

4. Hamilton, L., *Regression With Graphics: A Second Course in Applied Statistics*, Duxbury Press, 1992

5. Hapke, B., *Theory of Reflectance and Emittance Spectroscopy*, Cambridge University Press, August 1993

6. Moller, T. and Haines E., *Real-Time Rendering*, A K Peters Ltd, 1999

7. Ryan T., *Modern Regression Methods*, Wiley Inter-Science, 1997

8. Sun Microsystems Inc, http://www.javasoft.com/, accessed October 2000 – August 2001

9. University of Dayton, http://www.udayton.edu/, accessed January – June 2001

10. Watt A. and Watt M., *Advanced Animation and Rendering Techniques and Practice*, Addison Wesley, 1994

11. Webster Dictionary, http://www.webster.com/, accessed August 2001

12. Woo M. et al. (OpenGL Architecture Review Board), *OpenGL Programming Guide*, Addison Wesley, July 1999

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Professor Wolfgang Baer, Code (CS)
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5101

4. Professor Samuel Buttrey, Code (OR/5b)
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93943-5101

5. Professor Eric Bachmann, Code (MV)
   MOVES Academic Group
   Naval Postgraduate School
   Monterey, CA 93943-5101

6. LT Christopher P. Slattery
   171 Irish Settlement Road
   Underhill, Vermont 05489

THIS PAGE INTENTIONALLY LEFT BLANK