Eötvös Loránd Tudományegyetem

Faculty of Informatics

Department of Algorithms and their Applications

Analyzing Bidirectional Reflectance Distribution Functions

Supervisor:
Csaba Bálint
PhD Student

Author:
Mohammed Ghaith Jassem Al-Mahdawi
Computer Science BSc

Budapest, 2022

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:

Student's Name: Al-Mahdawi Mohammed Ghaith

Student's Neptun code: ZYQSYJ

Course Data:

Student's Major: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: *Csaba Bálint*

Supervisor's Home Institution: *ELTE IK ALAL, Faculty of Informatics, Algorithms and their Applications department*

Address of Supervisor's Home Institution: *H-1117 Hungary, Budapest, Pázmány Péter sétány 1/C*

Supervisor's Position and Degree: *PhD Student, Masters degree*

Thesis Title: Analyzing Bidirectional Reflectance Distribution Functions

Topic of the Thesis:

(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

Physically-based rendering engines rely on accurate light and material interaction calculation to produce realistic images. This interaction is defined by the Bidirectional Reflectance Distribution Function or BRDF for short. The goal of BRDFs is to construct convincing material colors with reflection and specular highlights. Distinct BRDFs have several trade-offs, ranging from accuracy to speed. Hence, this thesis presents various BRDF models with demonstrations.

Some BRDF implementations are slow yet generate accurate results, while others converge quicker with less precision. Hence, diverse applications, for instance, game engines, movie production graphics, and scientific visualization require distinct material models. Consequently, our implementation features a lightweight rendering engine with a simple interface for exploring numerous existing BRDFs and tweaking their parameters, and even offers a testbed for creating a novel material model.

Regarding the practical implementation, we apply ray tracing with Monte-Carlo sampling on environment maps to estimate the rendering equation integration, we visualize objects along with the material textures and world skymap. Finally, we represent a variety of different BRDFs shifting from the physically theoretical into the empirical implementations.

In conclusion, we elaborate on existing BRDFs and implement them, and create an engine for researching, teaching, and testing purposes. Users will be able to establish a scene and tune the loaded object substance parameters. Furthermore, we use a low-level programming language such as C++, OpenGL Shading Language for GPU programming, and a low-overhead cross-platform graphical user interface.

Budapest, 2021. 12. 01.

Contents

Chapter 1

Introduction

1.1 Motivation

In every 3D application, it is required to have a convincing light-material interaction over the whole scene. However, many problems appear when we try to reconstruct a valid material in a 3D scene, starting from solving the visibility problem to computing the global illumination using numerous techniques [1].

An appealing realistic 3D constructed world materials must obey many laws of illumination to be comparable to the real world. Many valid approaches have been proposed and considered to solve many of these problems and many of them are considered standards now. Yet, papers are published yearly to give new ideas to solve a specific problem or improve one of the previous ideas.

In this thesis, we explore one of these problems and view some given solutions addressing it. It is how light interacts with the material to produce its color. Or, in other words, we seek to know the result color of the surface put under lighting extracted from an environment map [2]. Lastly, this study is inspired by the work of *Rosana Montes and Carlos Ureña* [3].

1.2 Overview

Simulating the interaction between light and material in a virtual environment based on a physically based model, as elaborated in Section 4.1, can be computationally expensive, time taking, and dependent on lots of parameters such as the wavelength of light, as well as, the structural and optical properties of the surface,

which is discussed in Section 3.2. The surface structure can be emissive, reflective, rough, transparent, and absorbent of energy and light.

Various scenarios require different approaches and implementations favoring speed over accuracy and vice versa. Many examples are found in physics simulations, the film industry, and game production. In physics simulators, it is a must to have a precise calculation up to very high accuracy, in the film industry we require enough precision to give a sense of realism, while in games production, it is the speed that matters the most.

Some physics-based model are not recommended to be implemented for a real-time application that does not require a lot of accuracy, such as a game. Therefore, the calculations can be approximated through empirical and numerical estimations of the physically based model, as demonstrated in Section 5.2.

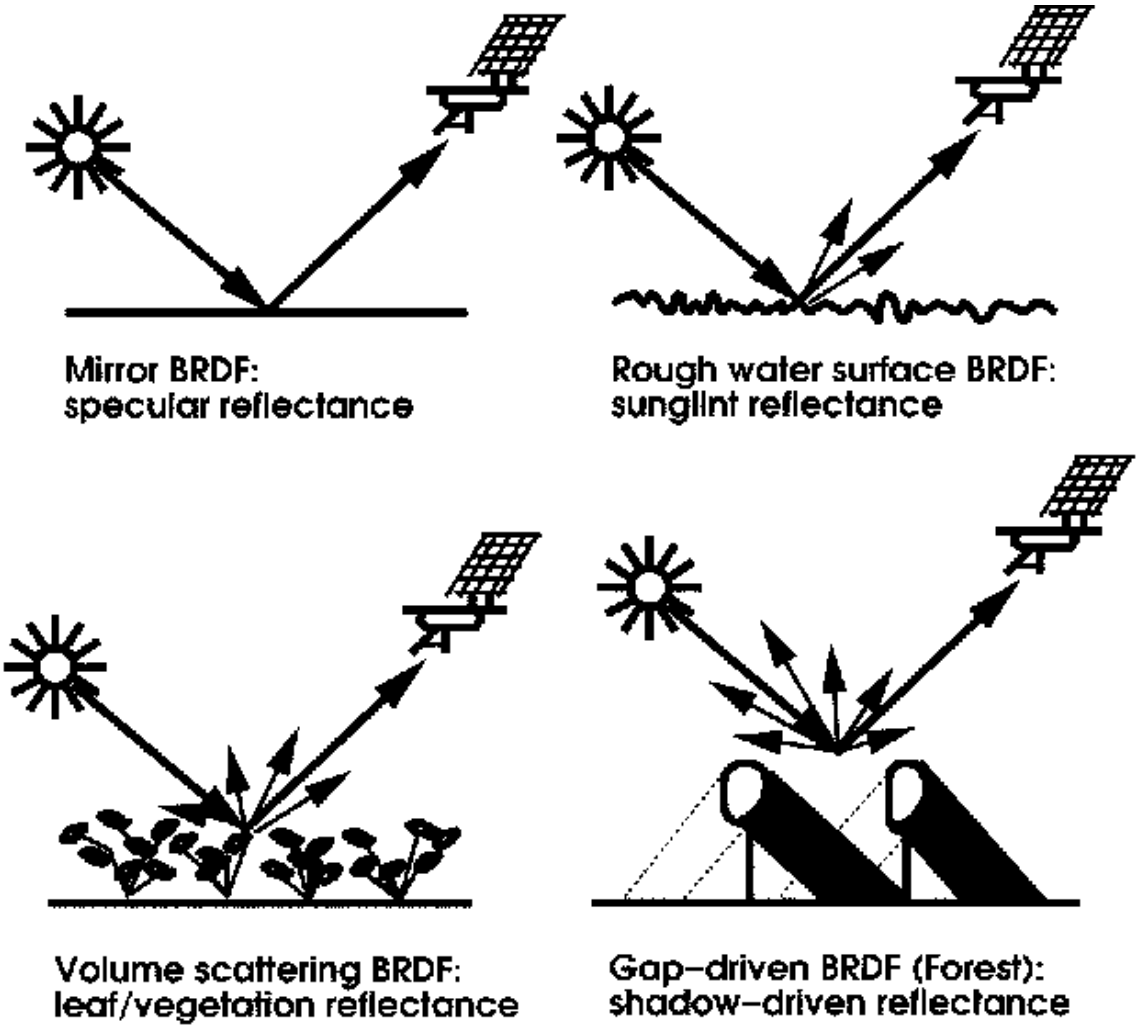


Figure 1.1: Types of Irrediance scattering based on the materials properties, such as roughness.
Proposed by Wolfgang Luch in 1997

The model name specifying the aforementioned phenomenon is called *Bidirectional Reflectance Distribution Function*, or so-called *BRDF*. It returns the ratio of reflected radiant flux, thus, defining the ratio of the color reflected toward a certain direction.

Hereafter, we investigate multiple BRDFs, which have been developed over time, in Chapter 5. We, also, deliver a rendering engine that allows the users to try and compare any BRDF, as well as, create their own, as explained in Chapter 2.

If we closely investigate the surface of an object, we observe the orientation of the surface irregularities. The surface reflectance can be described by the orientation of these irregularities, as seen in Figure 1.1. If the orientation is almost smooth, then the surface becomes reflective. However, if the orientation is rough and bumpy, the surface is scattering the light it receives into multiple directions and absorbs it, thus, the surface is opaque and not reflective. The model responsible for describing this phenomenon is known as the micro-facet model, and is described in details in Section 4.1. Extremely rough surfaces in this model are described as being diffuse, while smooth, or reflective, surfaces are considered specular or glossy. Furthermore, a surface can be both, diffuse and specular in many cases. For instance, plastic is a surface that is diffuse and specular at the same time.

Figure 1.2: Light source (Sun) is behind the viewer in the left image. The Sun is infront of the viewer on the right. Photograph by Don Deering

Another parameter to consider is the direction of the incoming light hitting the surface with respect to the direction of the viewing angle and the surface normal. The surface deformities may cause light to be reflected toward specific directions. For instance, *anisotropic* surfaces cause the light reflected radiance to favor some directions over other directions.

To rehearse, the surface color is dependent on the orientation, and the angle of radiance and view to the surface normal. To illustrate how these three parameters come together, see Figure 1.2.

1.3 Thesis Structure

This thesis consists of four chapters. It starts by describing the motive and an overview. Then, it delivers a guide on the software shipped with this thesis. After that, it holds an explanation of various BRDFs and their implementations. Finally, a summary and a conclusion.

Chapter 1 views the basic motivation and an overview of the phenomenon, which is being studied in this thesis, and defines it. As well as, stating the thesis structure.

Chapter 2 holds an explanation of the software substance of this thesis. The software is called the **BRDFA Engine** [4], which stands for *Bidirectional Reflectence Distribution Functions Analyzing Engine*. The explanation narrates the usage of the engine and its features. The chapter is divided into five sections. Sections 2.1, 2.2, 2.3 comprise the installation and guidance to use the tools provided by the engine. Section 2.4 addresses the engine shader implementation. Finally, Section 2.5 describes the testing and validity of the engine.

Chapter 3 remarks the irradiance computation and the physics concepts of light and materials. It revises the well-known rendering equation [5] and its digitization, and presents an overview of physics behind the light-material interaction.

Chapter 4 explains the theory behind the *Micro-Facets Based* models and its components, which form the *Physically-Based BRDFs*. Furthermore, it provides a thorough analysis of these components individually.

Chapter 5 addresses the main reason for writing this thesis, which is to analyze and explore some of the vastly used BRDFs in the industry. It dives deeper into elaborating the models analytically and visually. We use the **BRDFA Engine** [4], as well as, **Matlab**, which is software created to help engineers in enormous tasks

in a variety of ways, to demonstrate their behavior. This chapter is subdivided into two sections covering, both, the *empirical & physically-based (Microfacets-based) BRDFs*. Finally, Chapter 6 summarizes and concludes the thesis.

Chapter 2

BRDFA Engine

The subject software of this thesis has many useful features allowing testing, illustrating, visualizing, and capturing different BRDFs in real-time without the need to reload or restart the software. Researchers, teachers, students, and hobbyists can use this application for their purposes, from implementing a novel BRDF and testing how it behaves to illustrating existing BRDFs.

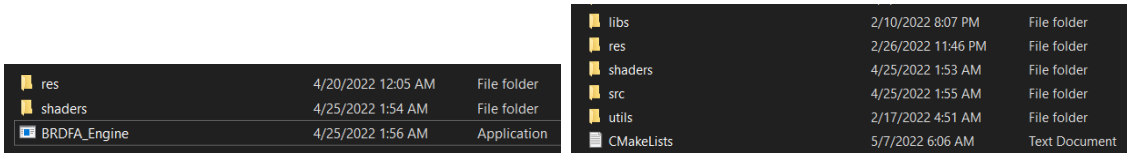
This chapter covers the installation of the software, and presents a user-guide to deliver an understanding of the application and its tools. Furthermore, the essential implementation details are presented. Finally, the testing and validations techniques are explained to assure the robustness of the program.

2.1 Installation

The engine developed with this thesis is called the BRDFA_Engine [4] and is hosted on *Github*, thus, making it widely accessible. The software installation can be done in two ways. Since the software is made to be an open-source project, the user can either download the built version of the software or download the source code and build the project. Furthermore, if the user wants to use an already built version of the software, it can be downloaded as well. The installation, and the building, steps are elaborated in the repository `readme.md` file. The extracted release built software comprises the folders and files shown in Figure 2.1a, while the engine source code is present in Figure 2.1b.

The `shaders/` folder contains the files used by the software shaders as seen in Figure 2.2. Shaders are programs that run on the GPU. The `shaders/cache/` sub-

directory holds the cached shaders which are saved to reduce the starting time of the program (See section 2.3.3) when many BRDFs are being used. The *shaders/brdfs/* subdirectory comprises the BRDFs that have been created.



(a) The application Files and Directories after installation

(b) The application source Files and Directories found in the project repository

Figure 2.1: Directories and Files of the **BRDFA_Engine**

The *res/* folder encapsulates all the default assets that are given with the engine. Multiple cubemaps, objects, and textures are created for the sake of illustration and basic testing. The engine can load any asset as seen in Section 2.2.3.

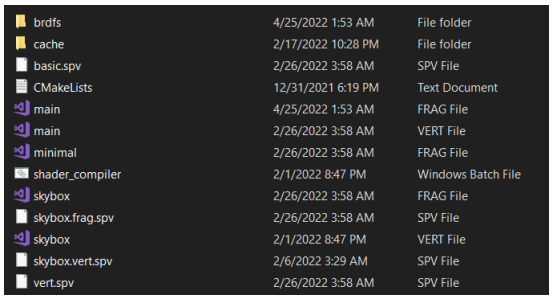
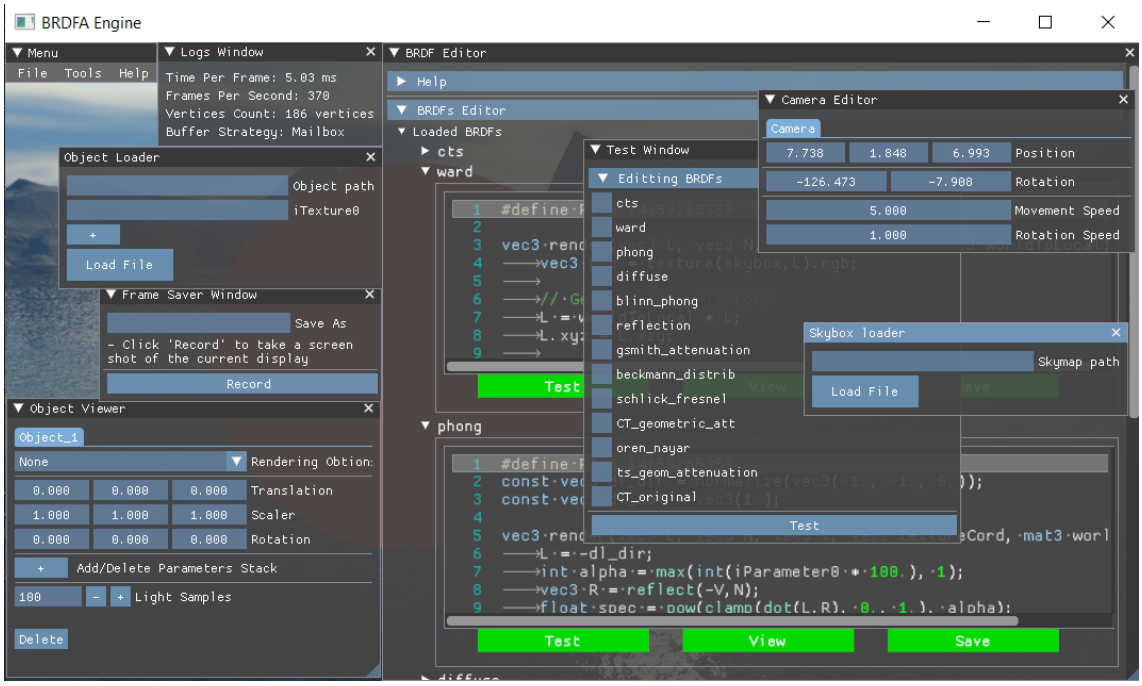


Figure 2.2: The directories and files presented inside the */shaders/* folder. This directory solely contains shader code in GLSL and Spir-V.

2.2 Basic User Guide

The user can start the software by executing the *BRDFA_Engine.exe* file. It initializes the application with the default behavior, which loads a window as seen in Figure 2.3a. The window contains a collapsible menu on the top left and the main canvas, which the scene is viewed through. The menu holds three main sub-menus, which are *File*, *Tools*, and *Help*. The *File* menu is explained in subsection 2.2.1. The *Tools* options belong to two categories (Basic and Advance), and are elaborated in Sections 2.2.3 and 2.3.1. Finally, the scene view canvas is demonstrated in Section 2.2.2, and the tool set is illustrated in Figure 2.3b and summarized in Table 2.1.

(a) The main window of the engine.



(b) The tools provided with the engine

Figure 2.3: A picture from within the **BRDFA Engine**. It shows the engine main window and the provided tools.

Tools Overview		
Tool Name	Accessed From	Overview
Object Loader	Menu > Open > Object	Allows the user to load a 3D object (*.Obj* file) and its textures. Four textures can be loaded, and the first texture must be loaded.
Skybox Loader	Menu > Open > Skymap	Allows the user to load a cubemap as a single image file.
Object Viewer	Menu > Tools > Objects Editor	Allows the user to edit the loaded 3D objects variables, such as the rendering function, extra parameters, transformation, and samples count per surface point.
Camera Editor	Menu > Tools > Camera Editor	Allows the user to specify the camera parameters, such as the movement and rotation speed.
BRDF Editor	Menu > Tools > BRDF Editor	The essential tool in the engine. It allows the user to write, load, save, and test *render* (BRDFs) functions, which are written in GLSL.
Log Window	Menu > Tools > Log Window	Provides extra data of the engine runtime state, such as the vertices count in the scene.
Test Window	Menu > Tools > Test Window	Presents a testing tool for the engine through a user interface. It is used to test a large amount of *render* functions asynchronously. The results are shown on an external sub-window.
Frame Saver	Menu > Tools > Frame Saver	Allows the user to capture the current rendered image into an file (*.bmp*) to be used in external analysis.

Table 2.1: A brief overview of the tools provided by the **BRDFA_Engine**.

2.2.1 File Menu

The *File* menu is composed of two options *Open* and *Close*. The *Open* serves for loading both **Objects** and **Skymaps** as seen in Figure 2.4 since this engine relies mainly on these two components to be useful. Objects resemble the mesh data, and skymaps are images that serve as an environment map (World map).

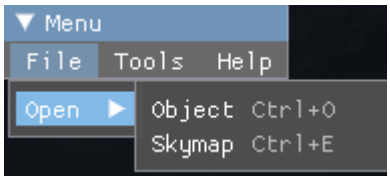


Figure 2.4: File Menu

The ***Object*** option views a subwindow in which the user can insert the path to the *.obj* 3D file, which are files containing the data of an object, and a texture related to that object as seen in Figure 2.5. Note that it is required for an object to have the first texture filled. Each object can have up to four textures attached to it, which can be referred to later within the shader code. To add another texture to an object, the user can click on the + button.

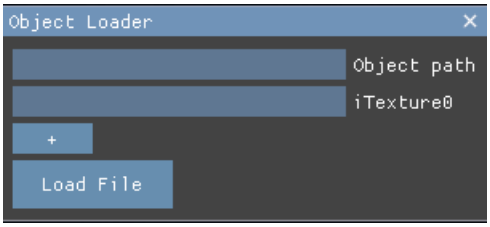


Figure 2.5: *Object loading* subwindow which allows the users to load 3D objects and their textures to the scene.

Similarly, Triggering the ***Skymap*** option activates another subwindow in which the user can insert the path to the environment image they want to load as seen in Figure 2.6. Remark, the engine only supports *Cubemaps* [6], which are images that hold data of the environment map, while other types of environment maps are not valid. Cubemaps are superior to *spherical mapping*, which is another technique used to render environment maps.

Cube-mapping [6] solves many problems that were related to spherical mapping, such as image distortion, viewpoint dependency, and computational inefficiency. Thus, the chosen environment mapping [2] technique for this software is cube mapping since it gives better results when calculating reflections and environmental radiance.

Figure 2.6: *Skymap loader* subwindow which allows the users to load cubemaps from an existing image on their machine.

2.2.2 Scene Window

This scene window shows the output of the rendering process that is done by the software. It presents an interactive three-dimensional environment (Refer to Figure 2.3a). The scene is viewed through a virtual camera constructed in the 3D world.

It is possible to navigate through the scene and view objects from different aspects. The movements involve the use, of both, the keyboard and the mouse. More specifically, the keys *W*, *A*, *S*, *D*, *Q*, *E* on the keyboard, as well as, the left mouse button. Table 2.2 presents information about each of the aforementioned input options for navigation, their intended type of interaction, and the impact on the movement of the camera within the rendered world space.

Navigation Information		
Input Key	Type of Interaction	Action
W	Click or hold	Move the camera forward
A	Click or hold	Move the camera leftward
S	Click or hold	Move the camera backward
D	Click or hold	Move the camera rightward
Q	Click or hold	Move the camera upward
E	Click or hold	Move the camera downward
Left Mouse Button	Hold and drag	Controls the camera direction

Table 2.2: Mapping of input keys, their intended type of interaction, and the impact on the camera movement.

By combining the use of the left mouse button and the other movement keys (W, A, S, D, E, Q), the user can navigate in any direction. The speed of movement

and the rotation speed can be changed through the ***Camera Editor*** (See Section 2.2.3). An example of distinct viewing points of the same object which illustrates the camera movement can be seen in Figure 2.7. It shows the *Viking Room* 3D model from different angles.

Figure 2.7: Different views of the vikings room 3D model from within the BRDFA Engine.

2.2.3 Basic Tools

This subsection covers the basic integrated tools that the user can use. The ***Tools*** menu contains **six** different integrated tools as seen in Figure 2.8. This section covers the ***Object Editor***, ***Camera Editor***, and ***Log Window***, while the rest of the tools are covered in Section 2.3.

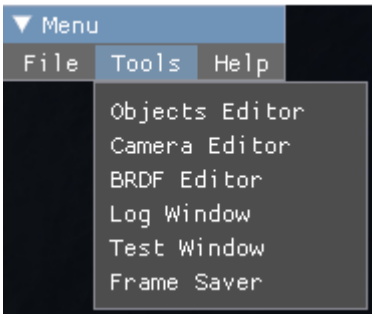


Figure 2.8: The integrated tools provided by the BRDFA Engine.

The ***Object Editor*** option triggers a new subwindow which is responsible for editing the scene objects parameters as seen in Figure 2.9. Each object in the scene has certain parameters that can be tweaked, starting from the object *rendering option* to deleting the object from the scene.

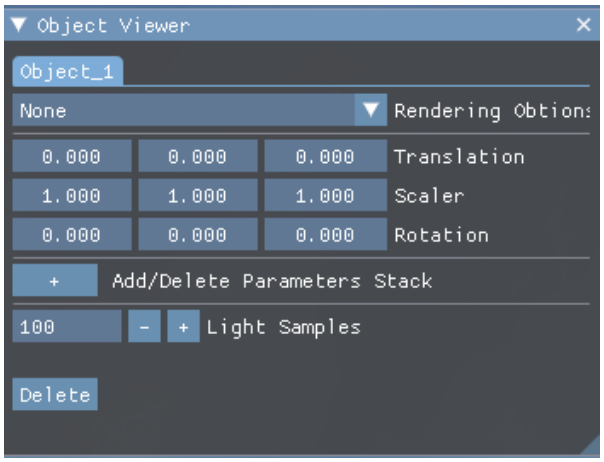


Figure 2.9: ***Object Editor*** tool which allows the users to view and edit the configuration of the existing objects in the scene.

The *Rendering Option* specifies which BRDF we are using to draw the material. The default behavior is *None*, which does not render any BRDF, instead, it renders the object with its texture as shown in Figure 2.3a.

The *Translation* defines the position of the object in the world space. The *Scalar* sets the objects scaling factors, which scales the objects on the given three axes. The *Rotation* parameters handle the object rotation. All the previous forms of transformation act on the three axes (x, y, z). Figure 2.10 illustrates the *Viking Room* 3D object with different *Rotation* parameters.

An object can have multiple additional parameters that can be attached to it. By clicking on the + button, a new custom parameter is added to the object and can be referred to within the shader code. The engine supports up to **nine** extra parameters. All the parameters are **normalized** floats (from *0.0* to *1.0*).

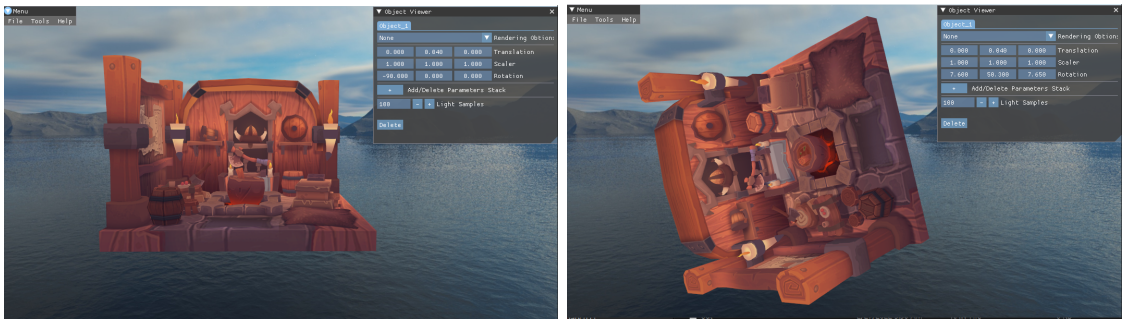


Figure 2.10: Different transformation setups for the Viking Room 3D model. This figure illustrates the usage of the ***Object Editor*** tool within the BRDFA Engine.

The *Light Samples* parameter controls the number of random light samples that are generated to estimate the irradiance. This controls how precise our estimation is since more samples output a better approximation of the population. More on the theory behind this parameter is covered in Section 2.4.

All objects are stacked in the subwindow seen in Figure 2.9. Therefore, object_N refers to objects starting from the first object that was loaded to the scene to the last one respectively. Note that the engine requires at least one object, thus, the delete button is disabled when there is only one object in the scene.

The ***Camera Editor*** tool allows the user to alter the parameters of the camera as seen in Figure 2.11. The *Position* configure the camera placement in world space. The *Rotation* set up the camera direction by providing the polar coordinates of the direction vector. The first angle corresponds to looking left or right (*Y-Axis Rotation*), while the second angle corresponds to the up or down directions (*X-Axis Rotation*).

The *Movement Speed* and *Rotation Speed* control how fast or slow the camera is traversing the space. The speeds are bounded between some values to avoid infinite speed. This tool gives more freedom to the user to change the camera settings to what fits the scene.

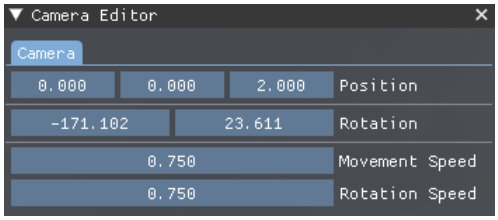


Figure 2.11: The ***Camera Editor*** tool integrated to the BRDFA Engine. This tool allows the user to change the configuration of the camera.

The ***Log Window*** tool displays the details regarding the rendering process and the scene content. It is used to measure the rendering speed in terms of *Frames Per Second* and *Milliseconds Per Frame*, the *Buffer Strategy* used method, and the number of loaded *Vertices*. See Figure 2.12.

The ***Frame Saver*** tool allows the user to save the current values of the pixels that are being rendered, to an external file as seen in Figure 2.13. This tool is a powerful analysis tool despite its simplicity because it provides the ability to output

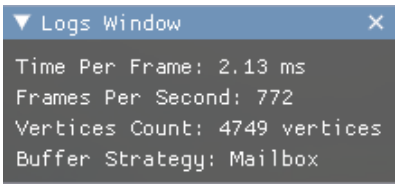


Figure 2.12: Logs Subwindow

the rendered data of different BRDFs into a file and then compare the rendered results in a post-processing analysis phase.

The user can write the desired file path to save the data. The path comprises the directory appended by the filename. The directory must exist or be created beforehand manually. After inputting the file path, the user can click on the *Record* button which creates a `.bmp` file, which is an image file with a simple format, and write the data of the image into it.

Figure 2.13: ***Frame Saver*** tool which allows the capturing a scene and save it into a file,

2.3 Advance User Guide

This section discusses the advance features integrated into the program. It requires a comprehension of *OpenGL Shading Language (GLSL)* and the functionality of fragment shaders. Furthermore, it covers the usage of the ***BRDF Editor*** (Section 2.3.1), ***Test Window*** (Section 2.3.2), as well as, the startup configuration of the engine (Section 2.3.3).

2.3.1 BRDF Editor Tool

This tool is the most essential in this software since it gives the user the ability to write and edit BRDFs in *GLSL* at run-time, as seen in Figure 2.14. There are two collapsible panels visible in this tool, the first describes the basics and the usage of the tool, as well as, the global variables that can be used or the textures that can be referenced. The second panel allows the user to edit, or add, new BRDFs

as seen in Figure 2.14. The ***BRDF Editor*** collapsible panel composes of two sub tools (*Loaded BRDFs* and *Custom BRDFs*).

Figure 2.14: ***BRDF Editor Tool*** which allows the users to create and edit BRDFs at run-time. TEST button compiles the code, the SAVE button saves the code and serializes it, and the VIEW button pushes the BRDF into the *Rendering Options*. If the implementation is valid, the buttons are turned *green*.

First of all, the *Loaded BRDFs* drop down contains all the loaded BRDFs from the source files when the engine is initialized. The loaded BRDFs are stored in the *shaders/brdfs/* directory that is found in the program folder (Section 2.1 and Figure 2.2). All of the shader source files are written in GLSL and they represent a part of the *fragment shader* used in the *rendering pipeline*. All the source files (*shaders/brdfs/*.brdf*) must contain the definition of the function so-called **render** which takes **five** arguments (*L*, *N*, *V*, *textureCord*, *worldToLocal*). It returns the color of the radiance toward the view direction (Details explained in Section 2.4).

The *L* represents a light sample toward a random direction on the surface hemisphere, the *N* presents the normalized facet normal, the *V* stands for the view direction, the *textureCord* is for the texture coordinates mapped to that point, and *worldToLocal* is a transformation matrix to transform the given vectors into local space if needed. The first three parameters are the ones usually used in a normal BRDF implementation, yet, two extra parameters are given for flexibility.

In each of the visible BRDFs in the *Loaded BRDFs* panel, there are **three** buttons with different usages. The VIEW is adding the current BRDF to the *Rendering Options* in the ***Object Viewer*** tool (revise Section 2.2.3). After submitting, the user can switch to it and see its behavior on objects.

The SAVE button writes the BRDF into the *shaders/brdf/*, as well as, cache the shader into *shaders/cache/*. This functionality allows the engine to save the source code of the BRDF into the disk and loads it in the next startup. Saving the code, also, creates a cached version of the whole fragment shader as a *Spir V*, which is a byte-code representation of the whole shader. It is worth noting that the **Vulkan API** requires the shaders to be in the *Spir-V* format. Furthermore, the engine can be configured to **NOT** load any cached data, thus, compile all the BRDFs code at startup (see Section 2.3.3). However, activating this configuration slows down the engine at startup.

The TEST button appends the written GLSL code to the built-in main fragment code and then compiles it. If there are errors, it prints the number of errors, as well as, the logs beneath the buttons. Otherwise, the BRDF can be added as a rendering option or saved to the disk.

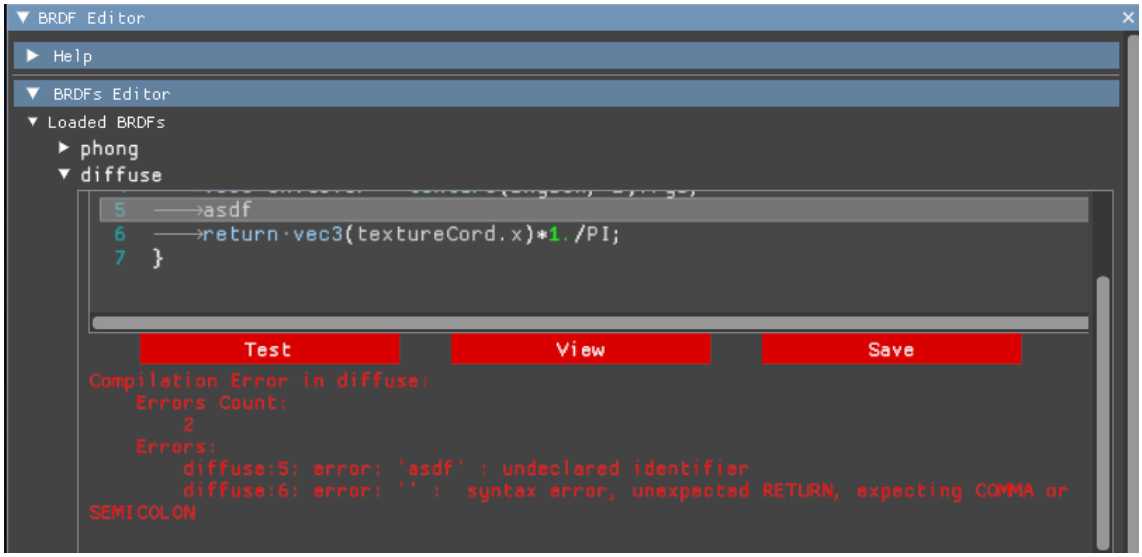


Figure 2.15: ***BRDF Editor*** tool Test and Validation layer visualization. Here a syntax error is inserted to the GLSL BRDF representation. Thus, this BRDF did not pass the testing and validation layers.

Secondly, below the *Loaded BRDFs*, there is another option called *Custom BRDFs* (see Figure 2.14). This option allows the user to create new BRDFs and add them to the ***BRDFA Editor***. The user must give a unique BRDF name, otherwise, it is discarded and a log message appears telling the user to insert a unique name. After inputting a BRDF name, the user clicks on the + button to create the new BRDF. The New BRDF composes of the function definition by default as seen in Figure 2.16.

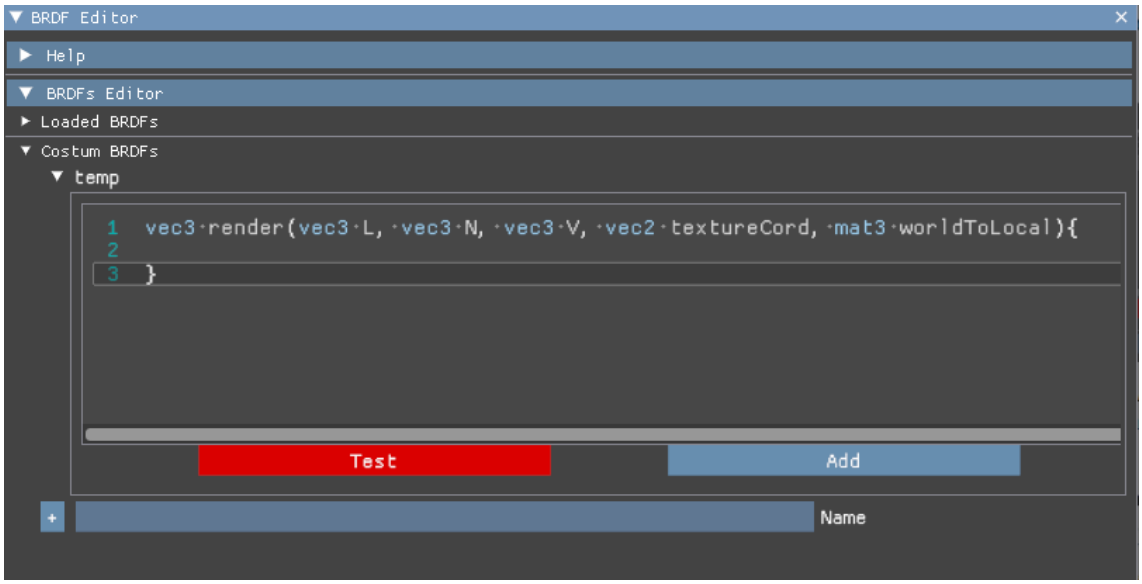


Figure 2.16: ***BRDF Editor*** tool visualization of adding custom BRDFs into the editor. *temp* is added by writing it in the *Name* input then pressing the + button.

The created BRDFs are **not** yet saved into the disk, it is only created inside the memory of the application. After the BRDF creation, the user must test and validate it, then press the ADD button to load it in the *Loaded BRDFs* panel and the rendering options. Note that, the ADD button does **not** save the BRDF into the disk.

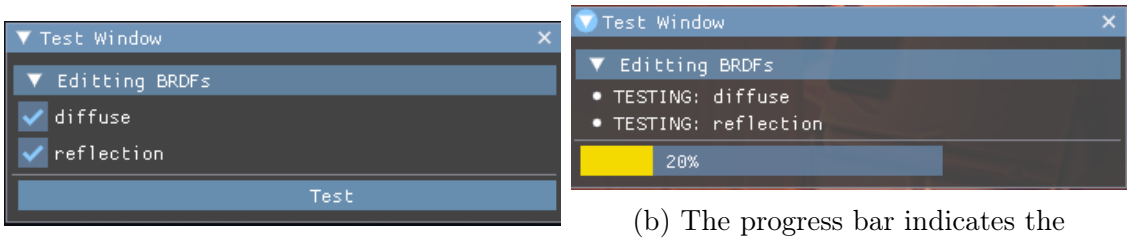
Finally, the ***BRDFA Editor*** is slower for testing lots of BRDFs at the same time. Therefore, an acceleration tool is introduced for maximizing the speed of testing and validating multiple BRDFs simultaneously (discussed in Section 2.3.2).

2.3.2 Test Window Tool

The ***Test Window*** tool provides asynchronous testing interface for multiple BRDFs, as seen in Figure 2.17a. The users choose what BRDFs they want to test and validate. After choosing the required BRDFs, they click on the TEST button at the bottom of the subwindow. A progress bar shows up indicating that the BRDFs are being tested as seen in Figure 2.17b.

Testing the implemented BRDFs in this tool gives the engine the ability to use multiple internal threads to fasten the testing. It makes it a powerful tool for asynchronous testing, validating, and debugging the implemented BRDFs. To rephrase, the ***BRDFA Editor*** provides a synchronised testing tool built into it by default

(see Figure 2.14), while this tool provides robust asynchronous automated testing for multiple BRDFs at the same time.



(a) ***Testing Tool***

(b) The progress bar indicates the percentage until all BRDFs are fully tested.

Figure 2.17: An illustration of the ***Testing Tool*** states.

After the testing is done and the progress bar finishes, a small window shows the log results of all the tested BRDFs. The logs pattern of each tested BRDF contains the name of the BRDF being tested, the number of errors if any exists, and the log of the test as seen in Figure 2.18.

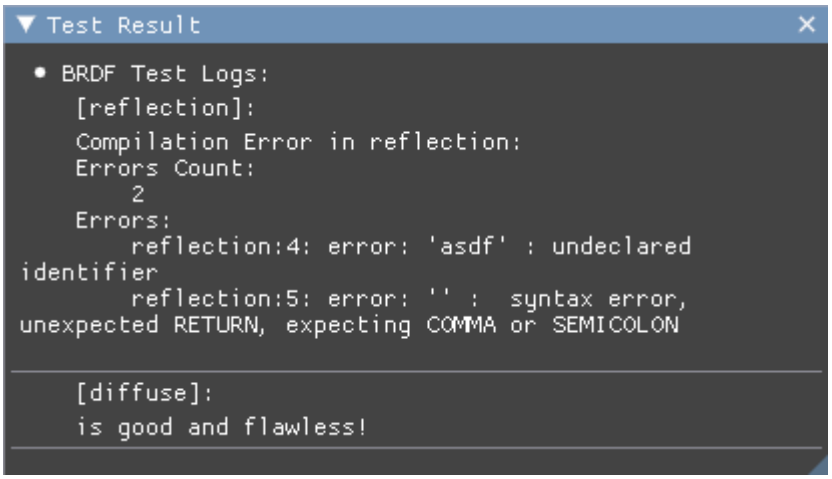


Figure 2.18: A subwindow holding the results of the ***Test Tool***. It shows all the BRDFs with their corresponding logs. In this testing session, the *reflection* implementation has two errors, while the *diffuse* passed the testing.

2.3.3 Startup Configuration

Additional configurations can be passed to the engine as flags (options) at the start of its execution. To check all the available flags supported by the engine, pass the `--help` or `-h` flag as in the example *./BRDFA_Engine.exe –help*. This command shows all the flags that the engine supports. Furthermore, there are two additional options other than the *–help*, which are *–no-cache-load , -ncl* and *–hot-load , -hl*.

The *–no-cache-load* flag forces the engine to ignore the saved serialized caches from previous sessions. Remark, the ***BRDF Editor Tool*** saves the compiled version of the GLSL code as a cached Spir-V file (see Section 2.3.1). By default, the Engine loads all caches from the previous executions if it finds any to avoid compiling the shaders, thus leading to a faster initialization. Note that if this flag is set, the engine attempts to compile all the shaders at startup since it does not load any cached files.

The *–hot-load* forces the engine not to compile any BRDF at startup. It only reads the BRDF source code and adds them to the *Loaded BRDFs*. The engine, by default, compiles and builds all the BRDFs if they are not cached.

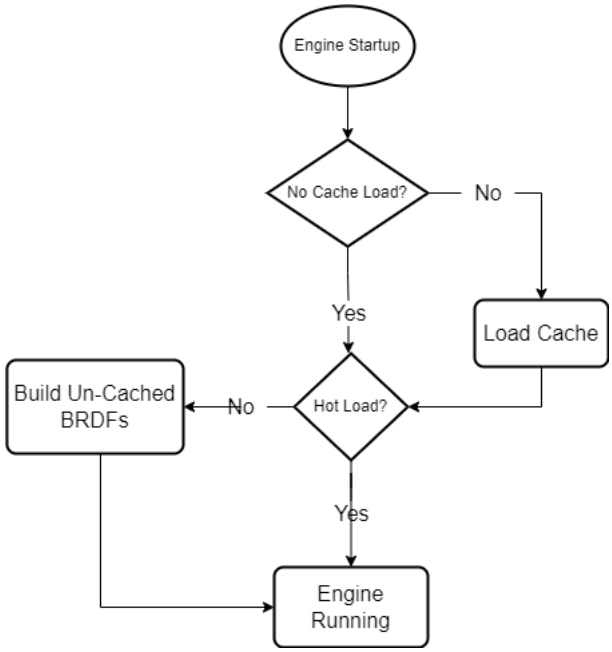


Figure 2.19: Simplistic flowchart showing the startup process of the engine with regards to the given flags.

The Engine starting flow is illustrated in Figure 2.19. As observed from the flowchart, the fastest way to start up the engine is by passing the *–hot-load* flag, which averts the BRDF compilation phase. Loading cached files is not resource nor time consuming, thus, its impact is relatively small.

The slowest method is achieved by offering *–no-cache-load* only, which prevents the engine from using cache. Consequently, it forces the engine to build all the BRDFs. Note that, the compilation is done asynchronously. However, if there are many stored BRDFs, it is better to pass *–hot-load*.

2.4 Implementation

This section covers the file structure and their roles seen in Table 2.3 and in Figure 2.3, libraries used in the engine, hardware API, and the implementation of the fragment shader used by the scene geometry. It requires an understanding of the *Monte Carlo Sampling Method* (see Section 3.1), as well as, basic concepts regarding the rendering process on the *graphical processing unit* (GPU).

The BRDFA_Engine works on *Windows Operating System* and it is written with *C++* and *GLSL*, while depending on *VulkanAPI*, *DearImGui*, and *GLFW*. The dependencies are found in the *./utils/** , while the source code of the project is found at *./src/** in the source code repository on Github (BRDFA_Engine).

C++ is the chosen programming language for this project since it contains lots of built-in and predefined data structures in its standard library, thus, giving more freedom to implement a variety of algorithms with a minimalist amount of code, yet preserving its low-level nature and high speed.

File Roles	
File Name	File Description
brdfa_callbacks	Holds all the callback functions that used in the project
brdfa_cons	Contain all the constants that are used in the project
brdfa_structs	Contain all the main and auxiliary structures that are used in the project
brdfa_engine	Contain the full engine class and its methods. Note that this application is a one class application
helpers/*_abs*	Abstracts Vulkan implementation, such as initiating vulkan objects, or manipulate them
helpers/functions	Includes all the *_abs* translation units

Table 2.3: Define the source code files and their uses in the engine.

Vulkan API resembles a lightweight graphical processing unit (GPU) API, which allows the developers, who use its software development kit (SDK), to create software that harnesses the power of the GPUs. *Vulkan* is the successor of *OpenGL* and both APIs are developed and maintained by the *Khronos Group*. There is no impact on choosing either of them to implement the software presented with this thesis.

However, since *Vulkan* is a relatively new API aiming to support the modest GPU architectures, it provides a long-term benefit. Therefore, it is the only GPU rendering API that the engine uses.

Since this thesis considers the exploration and exploiting of BRDFs, the fragment shaders are the part where the implementation matters the most. The user is only able to see and edit a subset of the shader, while the whole fragment shader is not editable within the engine run-time environment (***BRDF Editor***). The rest of this section delivers an understanding of the used fragment shader within the engine.

This software has two graphics pipelines, one for rendering the Cubemap, while the other renders all objects in the scene. The second graphics pipeline is relative to this thesis since the first pipeline is, only, used to draw the background of the scene.

The fragment shader of the second pipeline receives multiple uniform buffers, as well as, passed-in attributes from the vertex shader. These uniform buffers and attributes are discussed in Table 2.4. Clicking on the *Help* option in the ***BRDF Editor*** (Section 2.3.1) shows the uniform buffers. Through the usage of these defined values, sampling of the hemisphere at the surface normal of the desired point is performed to estimate the contribution of incoming light (Irradiance).

Fragment Shader Uniforms and Attributes		
Name	Type	Details
TextureCoordinate	Attribute	Resembles the texture coordinate map of the mesh, also called the u,v map. It is the same value we pass to the brdf(„,tc)
VertexNormal	Attribute	Resembles the normal of the surface at that fragment
VertexPosition	Attribute	Resembles the 3D point of the fragment in the world space
CameraPosition	Uniform	Camera Position in world space
SamplesCount	Uniform	Number of samples (used in Monte-Carlo sampling)
SkyboxTexture	Uniform (SamplerCube)	Texture that stores the CubeMap used in the program
iTexture[0-3]	Uniform (Sampler2D)	4 texture slots that can store objects texture. iTexture0 must always exist
iParameter[0-8]	Uniform (float)	Normalized float values (Between 0.0 to 1.0) that can be used arbitrary within the shader code

Table 2.4: All the uniform and attributes passed to the fragment shader of the **BRDFA Engine**.

Calculating all the contributions of the incoming light is computationally infeasible because computers are discrete, thus, performing finite number of calculations. The *Monte Carlo Sampling* technique (Section 3.1) solves this problem by estimating the contribution of indirect lights in the *Rendering Equation*. The creation of multiple uniformly distributed ray samples outgoing from the surface requires multiple steps, from the creation of a pseudo-random function to the transformation of the sample ray to the surface space.

GPU hardware cannot create a uniformly distributed sequence of random numbers (Such as the Halton Sequence [7]) and the drivers do not support it. Therefore, the creation of a fast random sequence generator function is required in the fragment shader. However, all valid generators require a *seed* to be passed as an input. Thus, the creation of a *seed* generator must be done before the creation of the uniform random sequence function itself.

For each fragment, a random seed must be created to avoid similar patterns on neighboring fragments. Thus, the seed creation function takes a fragment coordinate as an input and outputs a pseudo-random number. In other words, the fragment coordinate is hashed to a seed. Pseudo-random number generators can be created in many ways [8], starting from the usage of complex mathematical formulas to bits manipulation that gives an impression of randomness. Since the hashing is coded in the fragment shader, it is faster to use bits manipulation as implemented in Code 2.1.

```
uint base_hash(uvec2 p) {
  p = 1103515245U*((p >> 1U)^(p.yx));
  uint h32 = 1103515245U*((p.x)^(p.y>>3U));
  return h32^(h32 >> 16);
}

void main()
{
    int seed = int(base_hash(floatBitsToUint(gl_FragCoord.xy)));
}
```

Code 2.1: A demonstration of a random number generator function written in GLSL

After the seed creation, a sequence of two uniformly distributed random numbers

is required to create a sequence of light samples uniformly distributed along the circumference of the hemisphere of the surface. The **Halton Sequence** [7] generates a sequence of two random floats, between 0 and 1, from a given seed. Therefore, the Halton sequence helps in forming the uniformly distributed light sample. The implementation of a fast Halton method is illustrated in Code 2.2.

```
vec2 PseudoRandom2D(in int i){
  return fract(vec2(i*ivec2(12664745, 9560333))/exp2(24.0));
}

void main()
{
    for(int i = seed; i < samples + seed; i++)
        vec2 randomPoint2d = PseudoRandom2D(i);
}
```

Code 2.2: A Halton sequence function implemented in GLSL

The returned two values can be treated as the parametric coordinates [7] of a point on the hemisphere of the surface, which is then used to get the light sample direction. However, these parametric coordinates are uniformly distributed on a plane, thus the distribution is not uniform on the hemisphere. Fortunately, the plane surface can be treated as a cylinder surface. And, according to **Archimedes' Theorem** [9], the area is preserved in the projection between a cylinder and a sphere. Therefore, the *Halton* sequence output can be projected into the hemisphere, still, the distribution is preserved. Equation 2.1 and Code 2.3 demonstrate the projection of a point from the cylinder surface to the surface of the sphere. Remark, u and v are two given parametric coordinates.

$$\begin{aligned} x &= \cos(2v\pi)\sqrt{1-u^2} \\ y &= u \\ z &= \sin(2v\pi)\sqrt{1-u^2} \end{aligned} \tag{2.1}$$

```
// Where u and v are two given parametric coordinates.
void main()
{
    float x = sqrt(1. -  u*u)*cos(2.*PI *  v);
    float y = u;
    float z = sqrt(1 -  u*u)*sin(2.*PI * v);
```

```
7
8       vec3 sphereRandomPoint =  vec3(x, y, z);
9   }
```

Code 2.3: Forming a point on the surface of a sphere by using two given parametric coordinates.

The points on the hemisphere of the surface resemble direction vectors of light samples. However, they must be transformed from the local space to the point space on the surface, hence, a homogeneous transformation matrix is applied. The matrix is constructed based on the given *Normal* and *View* directions from the surface. The transformation matrix consists of three spanning vectors, one of which is the Normal of the surface. The other two spanning vectors must be calculated.

A *cross product* [10] is used between the *Normal* and the *View* vectors to form a new orthogonal vector. Since we have two orthogonal vectors in the point space, a new vector can be constructed using the same technique. Hereafter, the orthogonal vectors define the transformation needed from world space to the point space, as seen in Equation 2.2.

$$\begin{aligned} \vec{v} &= v\vec{iew} \times \vec{n} \\ \vec{u} &= \vec{v} \times \vec{n} \\ \mathbf{M} &= [\ \vec{v}^T \quad \vec{n}^T \quad \vec{u}^T\ ] \end{aligned} \tag{2.2}$$

After the transformation matrix is formalized, the light sample (Code 2.3 and Equation 2.1) can be transformed to the surface space. Nevertheless, acquiring the inverse of the matrix is crucial to map the light sample, back, from the point space to the local space. The inverse is achieved by using the *transpose* of the matrix [11]. Code 2.4 illustrates the implementation of generating the aforementioned matrices.

```
1 // Where: N and V are passed in attributes.
2 void main()
3 {
4     vec3 v = normalize(cross(V,N)); // e.g: (1,0,0)
5     vec3 u = normalize(cross(v,N)); // e.g: (0,0,1)
6     mat3 M = mat3(v, N, u); // e.g: [(1,0,0), (0,1,0), (0,0,1)]
7     mat3 M_ = transpose(M);
8 }
```

Code 2.4: An example of the implementation of the GLSL code representing the creation of a transformation matrix from the local space to the point space

Once the creation of a light sample is completed, the *Monte Carlo Sampling* approximates the irradiance as seen in Code 2.5 and elaborated in Section 3.1. Consequently, calls the render (*vec3 render(vec3 L, vec3 N, vec3 V, vec2 textureCord, mat3 worldToLocal)*) function that the user defines in the ***BRDF Editor*** tool (Section 2.3.1).

```
/*
Assumed to be given or calculated:
    N = Surface Normal in point space
    V = View Direction in point space
    seed = random number
    samples = the number of samples
    L = sampled light in point space
    Texture_Coordinate = The texture coordinate of the fragment
    pointToLocal = The inverse of the localToPoint matrix
*/
void main()
{
    /* Seed Creation */
    ...
    /* Transformation Matrix Creation */
    ...
    /* Monte Carlo Sampling of the Irradiance */
    vec3 accum = vec3(0.);
    for(int i = seed; i < samples + seed; i++){
        /* Light sample creation */
        ...
        /* Monte Carlo Sampling */
        accum += render(L, N, V, Texture_Coordinate, pointToLocal);
    }
    accum /= (float(samples));
    outColor = vec4(accum, 1.0);
}
```

Code 2.5: Monte Carlo Sampling technique implemented in GLSL.

The final color of the point is the average of all the output of the *render(...)* function. Examples of countable implemented *render* functions can be found in the project files at *shaders/brdfs/** . These examples are shipped with the engine and can be viewed from within the engine.

Note. In the actual implementation, the first ray direction is shot toward the perfect reflection direction to ensure better results, faster convergence, and lesser noise.

2.5 Testing and Validation

User interface tests are performed, which target any user-level, with or without prior knowledge or familiarity with the subject of this thesis. They involve any input mean operated by the user and the response of the program to any action triggered by such interactions.

All the interactions executed by the use of the mouse and keyboard to the Scene Window (Sections 2.2.2), are widely used by the gaming and simulation industry. This was implemented to provide a familiar interface to more experienced users, while still maintaining its simplicity for others. The aforementioned interactions were included and heavily tested as part of the **BRDFA Engine**.

Another major aspect for the quality of user experience is the interface responsiveness. Features, such as testing multiple BRDFs, were developed in an asynchronous manner to insure the responsiveness of the user interface even when many events occur, while providing feedback through the progress bars, which are synchronized and display the same data (Section 2.3.2). Tests with different input mesh files, environment maps, textures, and BRDFs, were performed to guarantee the robustness of the program.

Moreover, all the basic and advanced interactions with the subwindows, such as expanding and hiding sections of all tabs, and modifying values, were tested to a high extent both individually and in conjunction with others.

Stress tests were performed on buttons and all other triggers (eg. hide/expand sections) to avoid deadlocks and other unwanted reactions, when facing adverse user interactions. In addition, input components such as buttons are concealed to prevent the aforementioned unauthorized use.

Lastly, for every user interaction, the program provides a feedback. When attempted to test multiple BRDFs, for instance, the user is prompted with a progress

bar and receives a real-time updates on the time-taking procedures, as seen in Figure 2.17b. Every tool expresses the feedback differently based on the tool functionality and usage. For instance, when feeding incorrect file paths to the file loaders, an instructive message appears explaining the reason for the error, as seen in Figure 2.15. And when a major task is completed, such as the automated tests routine, the progress bar is hidden, and a new window is shown containing the logs of the process, as seen in Figure 2.18. Figures 2.20,2.18, 2.14 and 2.15 show some of the aforementioned feedback: the incorrect file prompt, the finished automated tests sequence, and the buttons colors, indicating the validity of the `render(..)` implementation.

Figure 2.20: Inputting a non-existing file to the skybox (skymap) loader

Advanced tests and validation layers are incorporated into the program to test its usability and strength. Furthermore, distinct BRDFs are imported and displayed by the engine, as seen, countable times, in Chapter 5, to assure the usability and resilience of the software under stressful conditions.

Basic file handling tests were performed by supplying non-existent and existing files. If an existing file has the right format, it is loaded. Otherwise, an error notice appears. The identical error notice appears when attempting to load non-existent files. As a result, the engine operates as predicted, handling all situations and exceptions.

The automated tests are integrated into the ***Test Window*** tool, as described in Section 2.3.2. The tool allows for simultaneous testing of several BRDFs in order to validate their validity. Furthermore, only genuine BRDFs may be inspected or reliably stored on the disk. Any modification to the textual representation in the ***BRDF Editor*** tool (Section 2.3.1) prevents the BRDF from being saved or viewed. The modified BRDF must go through the testing routine to guarantee its validity after any textual changes to its content.

Finally, throughout the thesis, the software is put to the test in a demanding environment. Numerous `render(...)` functions are implemented and viewed for single

and multiple objects. Consequently, we may infer that the program is stable under long-term use.

Chapter 3

Theoretical Background

The idea of drawing an image that represents the world from a certain perspective was created centuries ago and many artistic techniques were proposed. For instance, the *Ray Tracing* technique was employed as early as the 16th century and described by ***Albrecht Dürer*** in his **Four Books on Measurement** [12]. Many offline renderers tend to utilize Ray Tracing to simulate very realistic scenes, and the first known application of Ray Tracing was accomplished by ***Arthur Appel*** in the year 1968 [13]. Ray tracing is just one of many approaches that have been developed to address the visibility issue in computer graphics. Ray tracing and rasterization, which entail the 3D projection of primitives onto the screen, are the most widely used and reliable solutions for solving the visualization problem today. This chapter reviews the rendering equation and how to approximate it, then going through the fundamental concepts of light and how it interacts with specific material attributes such as roughness.

3.1 Rendering Equation

The **Rendering Equation** was presented by ***Kajiya, James T.*** in 1986 [5] as a way to generalize the many rendering approaches and techniques. In computer graphics, all rendering approaches aim to solve or approximate this equation. The rendering equation calculates the color of each point in the scene, taking into account all of the surrounding light (irradiance) that affects it, as shown in Figure 3.1. Energy conservation, linearity, and homogeneity are the most significant characteristics of the rendering equation.

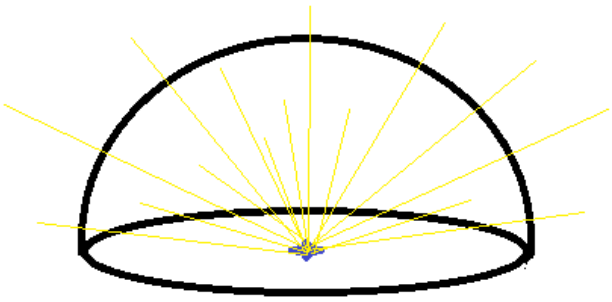


Figure 3.1: Contribution of all incoming lights determines the color and the shade of a particular point in the scene

The formal representation of the rendering equation is given in Equation 3.1. However, other quantities, such as the wavelength, are left out of the equation for ease of implementation. Equation 3.2 describes the final representation in most circumstances. The wavelength can be essential in physics simulations, therefore the desired equation can be changed based on the application.

$$L_0(\vec{x}, \vec{w}_0, \lambda, t) = L_e(\vec{x}, \vec{w}_0, \lambda, t) + \int_\Omega brdf(\vec{x}, \vec{w}_0, \vec{w}_i, \lambda, t) * L_i(\vec{x}, \vec{w}_i, \lambda, t) * (\vec{w}_i \cdot \vec{n})\, d\vec{w}_i \tag{3.1}$$

$$L_0(\vec{x}, \vec{w}_0) = L_e(\vec{x}, \vec{w}_0) + \int_\Omega brdf(\vec{x}, \vec{w}_0, \vec{w}_i) * L_i(\vec{x}, \vec{w}_i) * (\vec{w}_i \cdot \vec{n})\, d\vec{w}_i \tag{3.2}$$

where

- L_0 is the total light of the point $\vec{w}$
- $\vec{x}$ is the point position in space
- $\vec{w}_0$ is the view direction in which the point is viewed from
- λ is the light wavelength
- t is the time
- L_e is the emitted light from the point $\vec{w}_0$
- $\int_\Omega$ is the continuous sum of all incoming light or the irradiance
- Ω is the unit hemisphere centered around $\vec{n}$ *brdf* is the Bidirectional Reflectance Distribution Function
- $\vec{n}$ is the surface normal

It is impossible to solve the rendering equation analytically, and no closed solution exists. Many techniques and methodologies, however, can be utilized to estimate it by converting it from an analog signal to a sampled discrete signal that can be

practically implemented on computers. *Monte-Carlo Sampling* is one of these techniques, and it is used in the software subject of this thesis.

The rendering equation is an analog continuous function over the domain of Ω, which necessitates sampling to compute the outcome of such a function. *Radiosity* and *Monte-Carlo* methods are two sampling approaches used in computer graphics to approximate the rendering equation. Both strategies have their own set of benefits and drawbacks.

The *Radiosity* sampling method [14] attempts to solve the *Rendering Equation* by patching the mesh into numerous smaller subsurfaces and determining the energy contribution between them. The energy on the patches converges into a value after enough iterations. Equation 3.3 shows the mathematical representation of the described approach.

$$B_i = E_i + \rho_i \Sigma_{j=1}^{n} F_{ji} B_j \tag{3.3}$$

where

- B is the radiosity emitted from a patch
- E is the emission energy of the patch
- ρ is the reflectivity factor of a patch
- n is the number of patches in the scene
- F_iJ is the view factor between two patches. It defines how much patch i contribute to patch j

The *Radiosity* method is a strong tool, although it is computationally intensive and has certain drawbacks. It solely takes into account diffuse global illumination rendering, thus, all the surfaces in the scene are displayed opaque. This approach does not allow for the use of specular reflections. However, a recent publication [15] blends *Radiosity* with *Neural Rendering [16]* to create specular reflections.

The *Monte Carlo* model predicts the light contribution over a spot in the scene by shooting sample rays from it as seen in Figure 3.2. Each sample ray contributes to the brightness and color of the surface based on the material it is comprised of. According to the Monte-Carlo method, the more samples used from a population, the more precise the population is approximated.

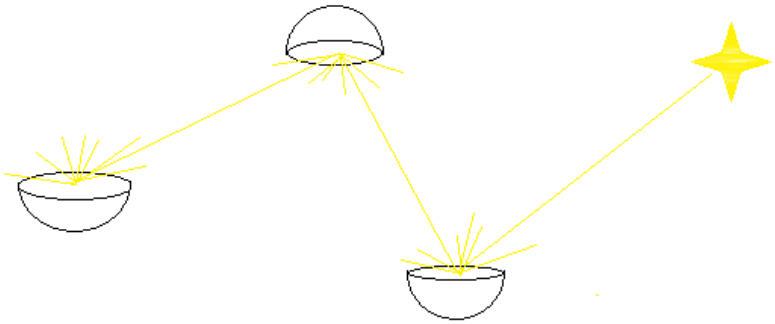


Figure 3.2: Monte-Carlo sampling with 3 bounces depth. Each point color is calculated by the contribution of all light samples corresponding to the point.

Monte-Carlo sampling is dependent on the number of samples which equates to $samples^{bounces}$, hence, the fewer samples, the faster it computes. Consequently, there are strategies to reduce sample count and retain the precision, one of which is *Importance Sampling* [17, 18].

3.2 Physics of Light and Material

The properties of, both, the material and the incoming light, influence the appearance of the surface. Thus, the reflected light color and energy are different due to the impact of the material on it. This section provides an insight into the physics of light and materials interaction.

3.2.1 Electromagnetic Waves

Light is an electromagnetic wave traveling through space. Electromagnetic waves are comprised of two waves, which are electric and magnetic, traversing space perpendicularly to each other and orthogonal to the field of motion [19, 20] as illustrated in Figure 3.3.

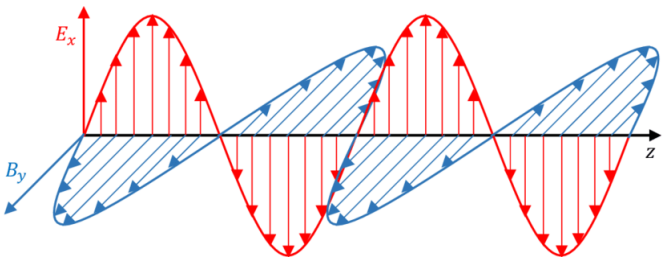


Figure 3.3: An illustration of an electromagnetic wave traversing space in direction x. Both, electric and magnetic, fields are described as E *and* B.

All charged particles exert an electric field on their surrounding [21]. Particles with similar charges repel, while particles with different charges attract each other. Figure 3.4 illustrates the exerted electric fields of two charged particles.

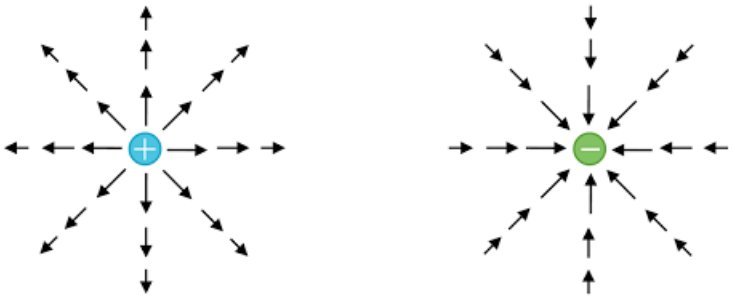


Figure 3.4: Electric field direction and force exerted from/on two charged particles. A charged particle exerts a force that may be repulsive or attractive depending on the charge of the other particle.

The movement of charged particles causes its electric field to shift, resulting in a magnetic field that rotates around the direction of motion [22]. Consequently, when magnetic fields change, electric fields shift as well.

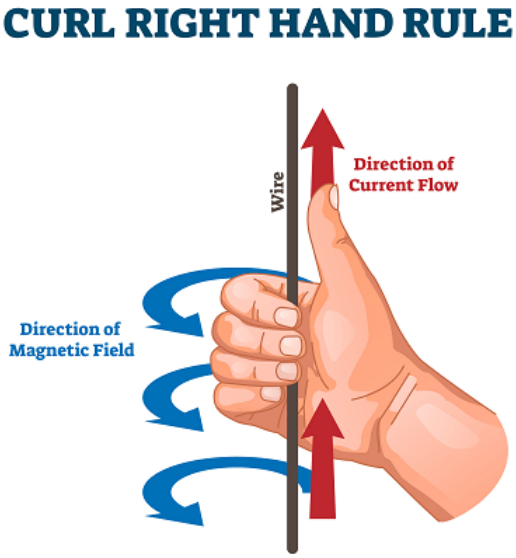


Figure 3.5: Right Hand Rule for determining the magnetic field direction of positively charged particles. Pointing the thumb toward the direction of motion and folding the fingers inwards toward the palm gives the direction of the produced magnetic field by that particle. Figure credits to The PASCO Foundation [23].

The direction of the magnetic field cycle (counterclockwise or clockwise) is determined by the charge of the particle and its field of motion, and it can be determined by using the right-hand rule as shown in Figure 3.5.

All light are electromagnetic radiations that traverse space with different rates of oscillations of its electric and magnetic fields. Human eyes evolved to perceive visible light which has an oscillating frequency between 4×10^{14} to 8×10^{14} cycles per second [24].

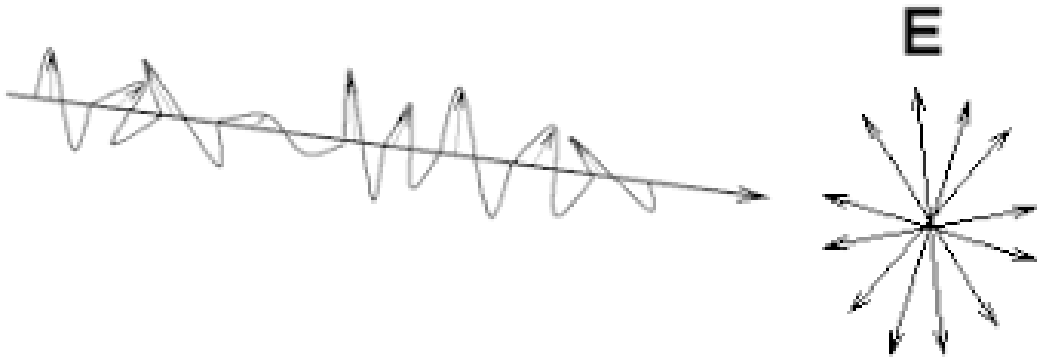


Figure 3.6: An upolarized electromagnetic wave, which comprises a variety of frequencies. The wave is visualized only by its electric field instead of both fields for the ease of visualization.

Light waves can be either *monochromatic* or *polychromatic* [25]. *Monochromatic* is a simple linearly polarized light wave as seen in Figure 3.3 with constant frequency. However, most light waves encountered in practice are *polychromatic* (unpolarized) light waves with a changing frequency as seen in Figure 3.6.

3.2.2 Material Properties Effect on Reflection

Materials affect light differently according to its frequency. For instance, refraction varies regarding the light frequencies, as in *optical prisms* (Figure 3.7). Light-Material interaction can be defined using geometric optics [26, 27], wave optics [28, 29], or quantum theory of light [30] models.

Wave Optics presents an adequate explanation of the refraction and reflection phenomenon. Electromagnetic waves may collide constructively or destructively with each other based on the interference property. Light interaction with material causes the velocity of light to change due to the interference resulting from the absorption

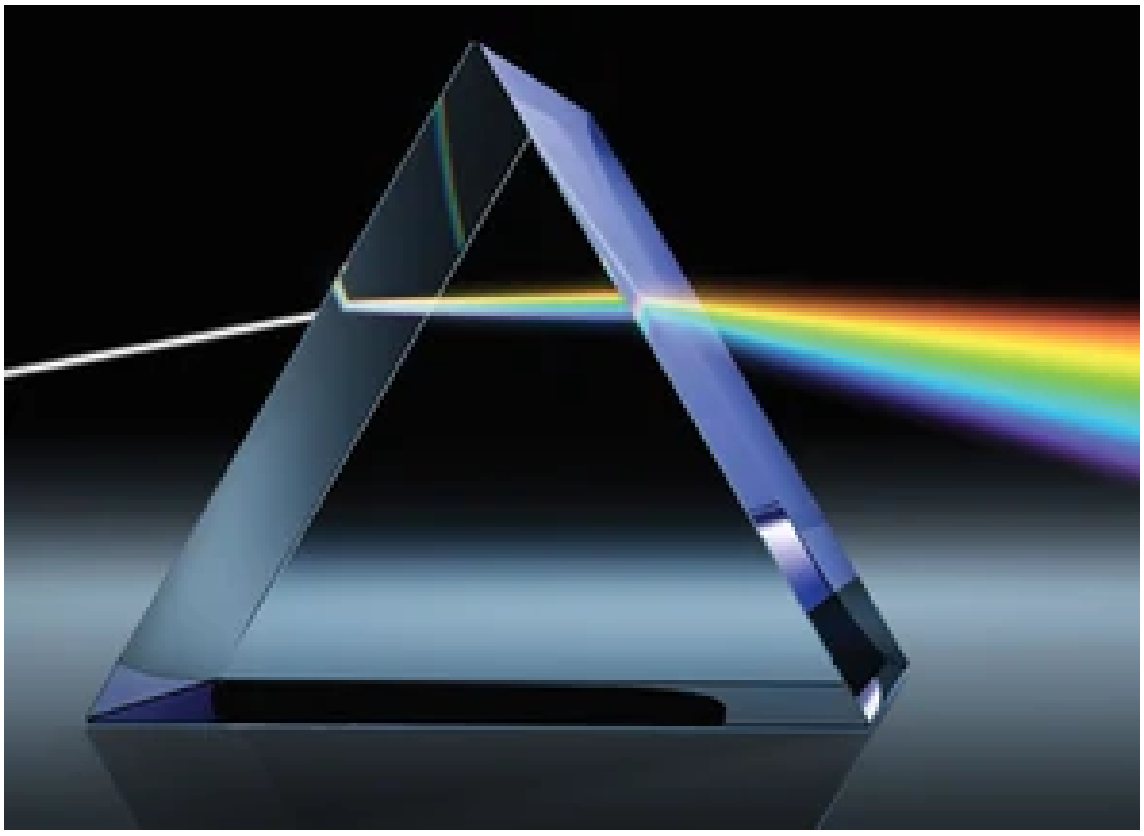


Figure 3.7: A 3D model of an optical prism illustrating the refraction dependency on the incoming frequency of the light wave. Figure source: *https://optical-illusions.fandom.com/wiki/Prism.*

and re-emission of electromagnetic waves inside the medium [29]. The absorption and re-emission of light energy are called *scattering*.

Light velocity coincides with the medium of propagation. The ratio of change in the light velocity is denoted by the *index of refraction* (IOR) or the *refractive index*. Some materials are absorbent, thus, reducing the energy of the electromagnetic wave. The absorption ratio is denoted by the *attenuation index*.

in *Geometric Optics*, light is treated as rays, which simplifies the calculations of the interaction between the light and the medium. However, it does not explain the refraction or the reflection phenomenon. Nonetheless, it is widely used in Computer Graphics because of its simplicity [27]. A comparison between the two mentioned models is shown in Figure 3.8.

To be able to use the *Geometric Optics* model to calculate the refraction and reflection direction and intensity through **Snell's Laws** and **Fresnel Equations**, the *refraction index* of the medium must be known. The *refraction index* abstracts the molecular level details of light-material interaction.

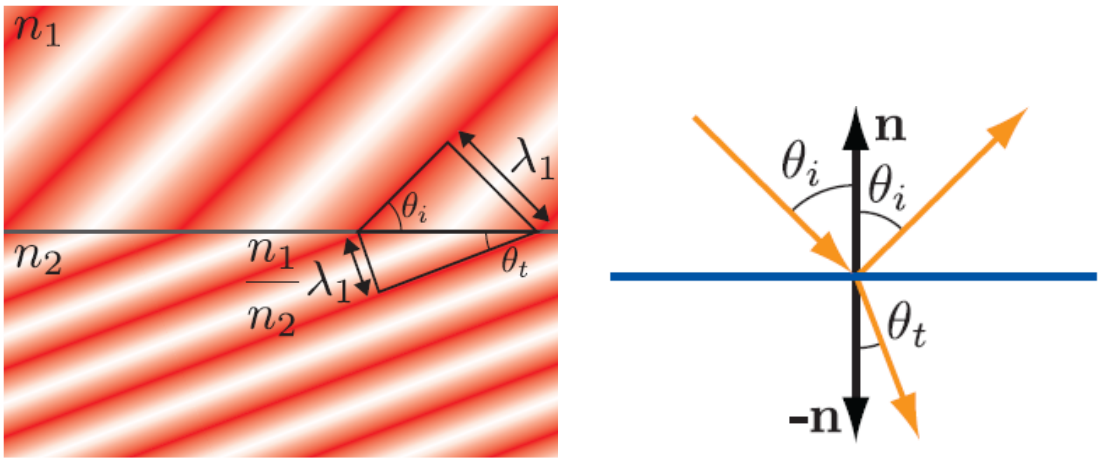


Figure 3.8: On the left is an *optic waves* representation of the refraction and reflection phenomenon of light waves, where the light is coming from the upper left corner and its phase is illustrated by reddish color. On the right a *geometric optics* representation is used. Figure credits to Real-Time Rendering [31] Figure No. 9.9

Finally, two types of materials exist, which are *Dielectrics* and *Conductors*. *Dielectrics*, such as water, refract light energy and have a low absorption rate. On the other hand, *Conductors*, such as metals, have a higher absorption rate, which rapidly reduces the incoming light energy. Simply put, *Conductors* have a substantially higher absorption attenuation than *Dielectrics*.

Chapter 4

Micro-Facets Theory

This chapter provides an overview of the Micro-Facets theory. Furthermore, it covers the individual components of the *microfacets-based BRDFs*.

4.1 Overview

Light may scatter within a medium until it is absorbed (transferred to heat) or exits, as illustrated in Figure illustrated in Figure 4.1. The distance between the entrance and exit of the ray is denoted by the *scattering distance*.

The goal of rendering in computer graphics is to determine the color of pixels on the screen. Depending on how far away an object is, a pixel may cover a huge or tiny amount of a surface. The area that a pixel covers is called the *shading scale*. Local shading techniques, such as BRDFs, are utilized when the *shading scale* is larger than the *scattering distance*. If the *scattering distance* is greater, global illumination methods are applied, which imitate the physical phenomenon of subsurface scattering [32].

The laws of physics impose two constraints on any BRDF, which are *Helmholtz reciprocity* and *conservation of energy*. In interactive applications, however, when an approximation is sufficient to generate a sense of realism, these laws may be ignored. Offline renderers, on the other hand, do not break these constraints because their primary goal is to provide a realistic scene.

The *Shading Scale* of a pixel may cover numerous surface irregularities which impact the pixel color. Rough surfaces, for example, have more surface variations than smooth surfaces. Between 1977 and 1982, *Blinn, Cook, Sparrow, and Torrance*

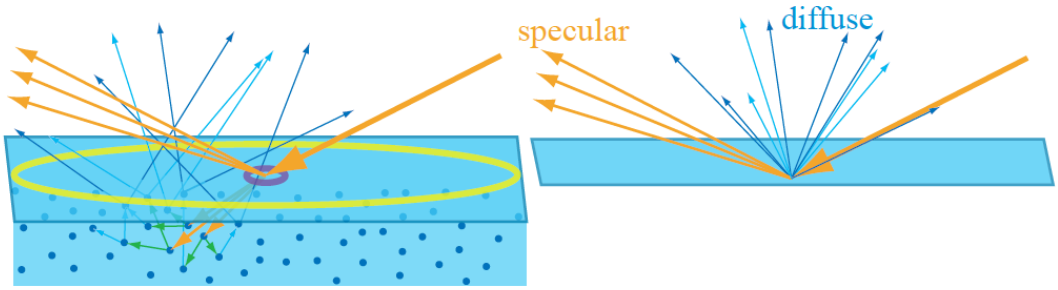


Figure 4.1: On the right, a far view of the scattering and reflection phenomenon, while on the left, a closer inspection of the same phenomenon is shown. The color of the rays resemble their wavelength. As observed, some light wavelengths are absorbed and others are scattered. The yellow and purple circles represent the *scattering distance* of the incident ray. Figure 9.15 in Real-Time Rendering [31].

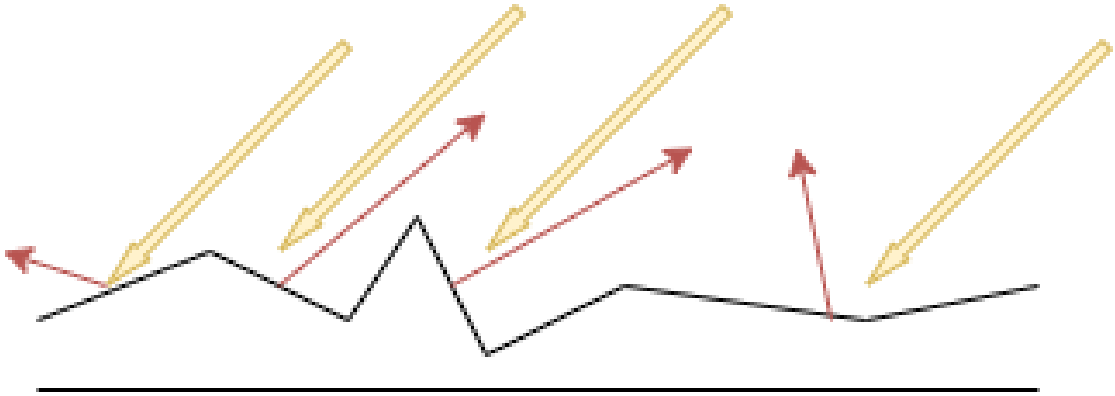


Figure 4.2: Micro-Geometric representation of a surface. Yellow rays represent the incoming radiance, while the red arrows resemble the direction of reflection. Each subsurface represents a small tilt of the surface normal toward a certain direction.

[33, 34, 35] introduced *micro-geometry* and *nano-geometry* concepts that represent surfaces. *Micro-geometry* and *nano-geometry* representations are used to establish physically accurate BRDFs.

Since each deformity is considered as a tilted mirror in the *micro-geometry* model, geometric optics laws can be used to define the interaction between light and material. The *micro-geometry* representation of a surface is shown in Figure 4.2. Furthermore, *micro-geometry* cannot be employed to produce thin films, as in *Huygen's slit experiment* [36]. The wave optics description of light is used in *nano-geometry* models, allowing light waves to interfere constructively or destructively [29, 37, 38]. In this thesis, only *microgeometry-based* models are investigated.

The *micro-facet* models emerged from the *micro-geometric* representation of the surfaces. As a result, they are considered *physically based*, yet they may be physically

inaccurate [39]. Three key components constitute these *microfacet-based* BRDFs [34]. The *Fresnel Reflectance Function* (Section 4.2) deals with light reflected intensity, the *Geometric Attenuation Model* (Section 4.4) with shadowing and masking between light and view direction, and the *Distribution Function of Normals* (Section 4.3) with the ratio of normals to the given direction.

4.2 Fresnel Reflectance and Schlick Approximation

Augustin-Jean Fresnel introduced a set of equations to determine the intensity of the reflected and refracted light based on the *geometric optics model*. These equations are based on the assumption that the surface is perfectly flat.

Although *Snell's* rules of reflection are used to determine the direction of reflection, as illustrated in Equation 4.1, the equation does not define the intensity. On the other hand, *Fresnel* equations use the *refraction indices* and the *reflection angle* to measure the intensity of the reflection.

$$\vec{r_I} = 2(\vec{n} \cdot \vec{I})\vec{n} - \vec{I} \tag{4.1}$$

Where $\vec{r}$ is the reflection direction, $\vec{n}$ is the surface normal, and $\vec{I}$ is the incident direction.

The base color of the specular reflection is depicted when the incoming light hits the surface at *normal incidence* ($\theta_i = 0°$), and it is indicated by F_0. Furthermore, as the incident angle increases, the Fresnel function output for all light frequencies rises to 1. As shown in Equation 4.2 and Figure 4.3, if $\theta_i = 90°$, then $F(\theta_i) = 1$, where $F(\theta_i)$ is the Fresnel function.

$$\lim_{\theta \Longrightarrow 90°} F(\theta) = 1 \;\; \& \;\; \lim_{\theta \Longrightarrow 0°} F(\theta) = F_0 \tag{4.2}$$

For unknown F_0 values, equation 4.3 can be used.

$$F_0 = (\frac{n_1 - n_2}{n_1 + n_2})^2 \tag{4.3}$$

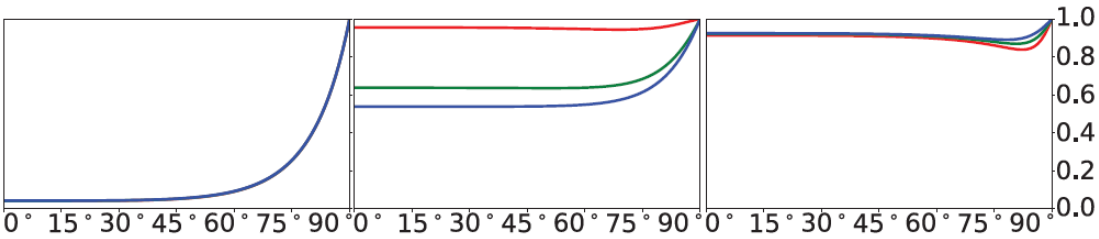


Figure 4.3: RGB values of Fresnel reflectance of glass, copper and aluminum (from left to right). The RGB triplets of glass line up because glass Fresnel reflectance is colorless. Figure credits to Real-time Rendering Book [31] Figure No. 9.20

Schlick approximation [40], seen in Equation 4.4 and visualized in Figure 4.4, is employed in interactive applications because it works with RGB triplets, making it simpler to implement. Additionally, the *Schlick* model can be expressed as in Equation 4.5, allowing for greater artistic flexibility. This also permits the artist to construct materials that do not fit into the Fresnel calculations, such as dusty materials.

$$F(\vec{n}, \vec{I}) \approx F_0 + (1 - F_0)(1 - (\vec{n} \cdot \vec{I}))^5 \tag{4.4}$$

$$F(\vec{n}, \vec{I}) \approx F_0 + (F_{90} - F_0)(1 - (\vec{n} \cdot \vec{I}))^{1/p} \tag{4.5}$$

The outcome of the *Schlick* model is fairly realistic. However, in some extreme cases, such as Zinc [40, 29], the inaccuracy is high, although in most cases, this error is ignored. In addition, *Gulbrandsen* [41] presented a more precise model but is more computationally intensive than the *Schlick* method. *Lagarde* [42] also discusses the Fresnel equations as well as several approximations.

When light interacts with less dense material (n1 > n2), *internal reflection* occurs. *Snell's* laws of reflection also predict the *critical angle*, which is the angle at which light impacting the surface with a greater angle is entirely reflected. *Total internal reflectance* is a phenomenon that occurs in dielectrics since conductors absorb light energy rapidly. The *critical angle* equation can be seen in Equation 4.6 [35, 29]. In this scenario, the Fresnel function progressively approaches perfect reflection in which $F(\theta) = 1$ & $\theta >= \theta_c$.

$$\sin\theta_c = \frac{1}{n} = \frac{n_2}{n_1} = \frac{1 - \sqrt{F_0}}{1 + \sqrt{F_0}} \tag{4.6}$$

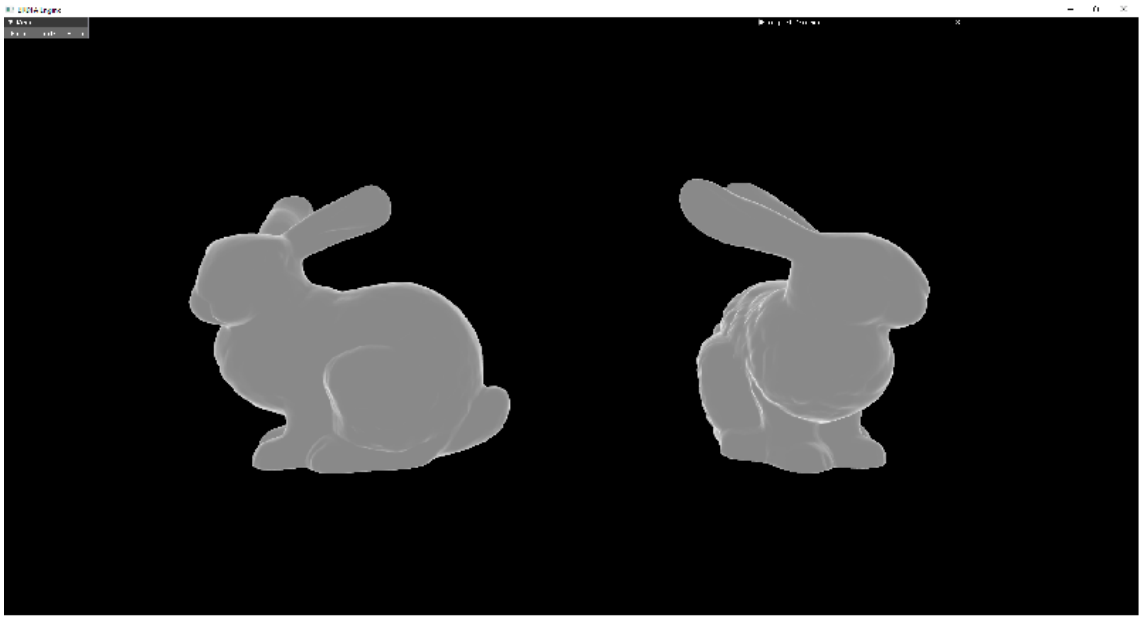


Figure 4.4: An illustration of the Fresnel coefficient (factor) with base refractive index of 0.25 ($F_0 = 0.25$).

4.3 Distribution Function

All of the micro facets on mirror-like surfaces face the normal, whereas rough surfaces have micro facets oriented in random directions. The ratio of microfacets facing a particular direction is described by the *Normal Distribution Function. Brian Karis* studies a range of distribution functions [43] and states that *Trowbridge-Reitz GGX* is the function utilized in Unreal Engine 4, as seen in Equation 4.7.

$$D(\vec{n}, \vec{h}, r)_{Trowbridge-ReitzGGX} = \frac{r^2}{\pi((\vec{n} \cdot \vec{h})^2(r^2 - 1) + 1)^2} \tag{4.7}$$

The normal direction, intended direction, and surface roughness, are the three arguments of the distribution function. As observed from Figure 4.6, it returns a ratio of the faces towards the given direction.

The $\vec{h}$ vector is the halfway vector between the view and incidence directions. Obtaining its distribution entails acquiring the sub-surfaces contribution ratio to the reflection.

Finally, a distribution function must be energy conservative in order to be legitimate. Therefore, the sum of the distribution function for all given directions must equal 1, as seen in Equation 4.8. In addition, Figure 4.5 depicts a set of well-known distribution functions.

$$\int_{\vec{h}} D(\vec{n}, \vec{h}, r).(\vec{n} \cdot \vec{h}) d\vec{h} = 1 \tag{4.8}$$

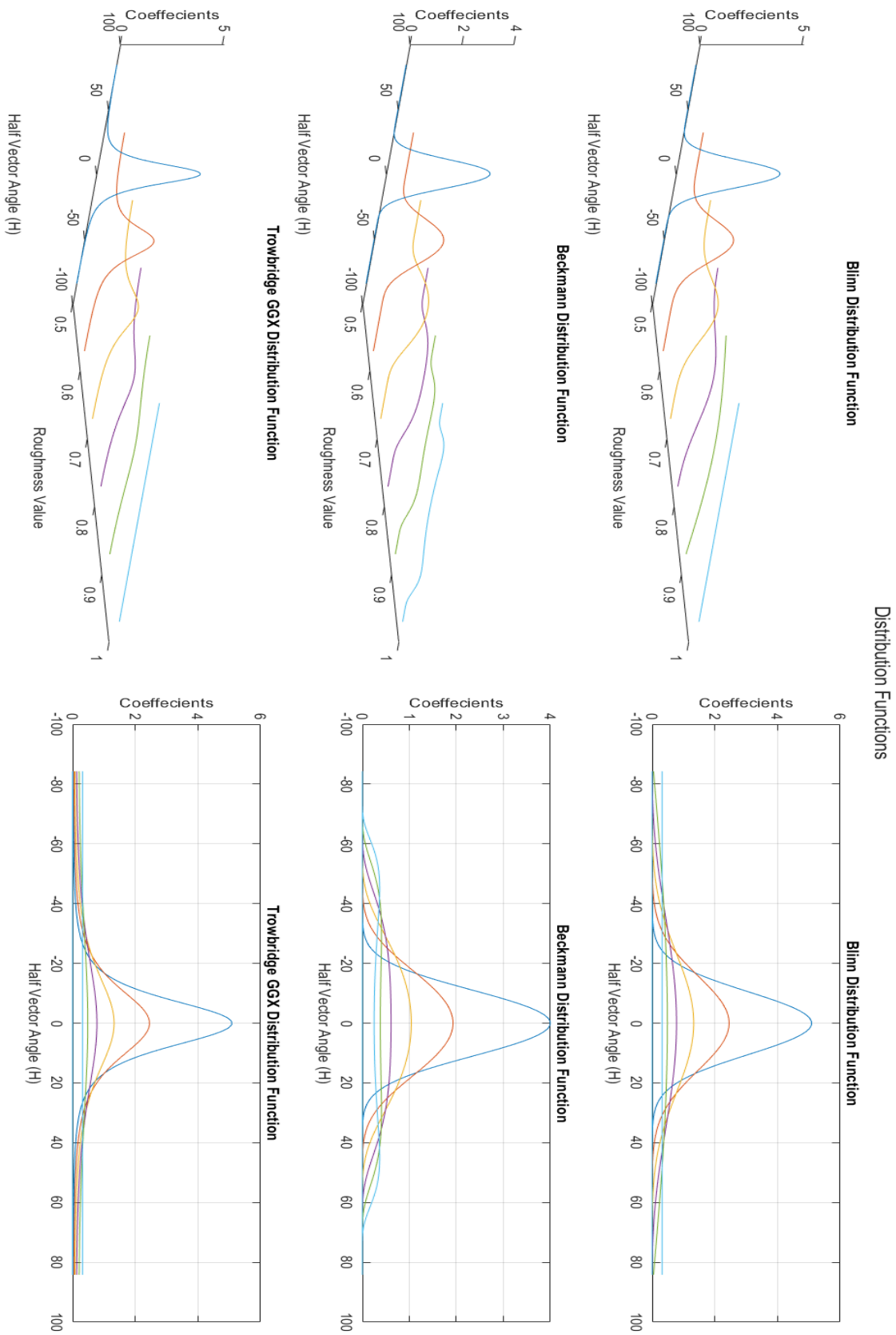


Figure 4.5: An illustration of several isotropic distribution functions. Each distribution has two images describing it, one to depict the roughness values, while the other demonstrates the shape of the slope.

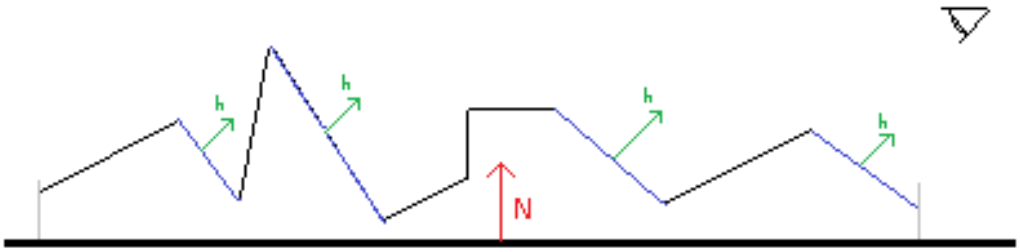


Figure 4.6: The green arrows shows the faces normals, the red arrow shows the surface normal. Given an h vector, the distribution function returns the ratio of micro-surfaces (blue lines) facing toward that vector

4.4 Geometric Attenuation Factor

Since the distribution function returns the orientation of the sub-surfaces toward a given direction, it does not consider the overlap of these sub-surfaces. Therefore, another factor is added to *microfacets-based* BRDFs to address the problem and is denoted by the *Geometric Attenuation Factor*. It was first addressed by *Torrance and Sparrow* [34] in their investigation of the *off-specular peak* phenomenon, as described in Section 5.1.1.

The *geometric attenuation factor* controls the masking and shadowing that occurs between the incoming light and the direction being viewed. Figure 4.8 shows surface *Self-Masking*, also known as *Geometric Obstruction*, which results in a reduction of light intensity when viewed from certain angles. Simply defined, reflected rays from one sub-surface may collide with another sub-surface before reaching the eye (viewer), scattering or reflecting the light in a different direction. Reflected light, in principle, continues to scatter and reflect until it loses energy or escapes the sub-surface valley. In practice, however, because of the small error, this situation is ignored.

Similar to the distribution functions, several geometric functions are used in production, however, the one utilized in *Epic Games Unreal Engine 4* is *Smith's Schlick-GGX* function, given in Equation 4.9. According to *Heitz* [39], *Smith's Schlick-GGX* model proved to be the most accurate among all. Furthermore, various geometric

functions are depicted, some of which are roughness independent as seen in Figure 4.7, while others rely on roughness as seen in Figures 4.9, 4.10, and 4.11.

$$G(\vec{n}, \vec{v}, k)_{SchlickGGX} = \frac{\vec{n} \cdot \vec{v}}{\vec{n} \cdot \vec{v}(1-k) + k} \tag{4.9}$$

k is a mapped value of roughness, which can be $\frac{(r+1)^2}{8}$ or $\frac{r^2}{2}$, where r is the surface roughness. Since the geometric attenuation function must account for shadowing and masking, the final representation is seen in Equation 4.10.

$$G(\vec{n}, \vec{v}, \vec{l}, k) = G(\vec{n}, \vec{l}, k)_{SchlickGGX} * G(\vec{n}, \vec{v}, k)_{SchlickGGX} \tag{4.10}$$

Finally, all the plotted functions are rendered and visualized. Roughness independent functions are shown in the Figures Collection 4.12, while the methods that rely on the roughness are shown in Figures 4.13.

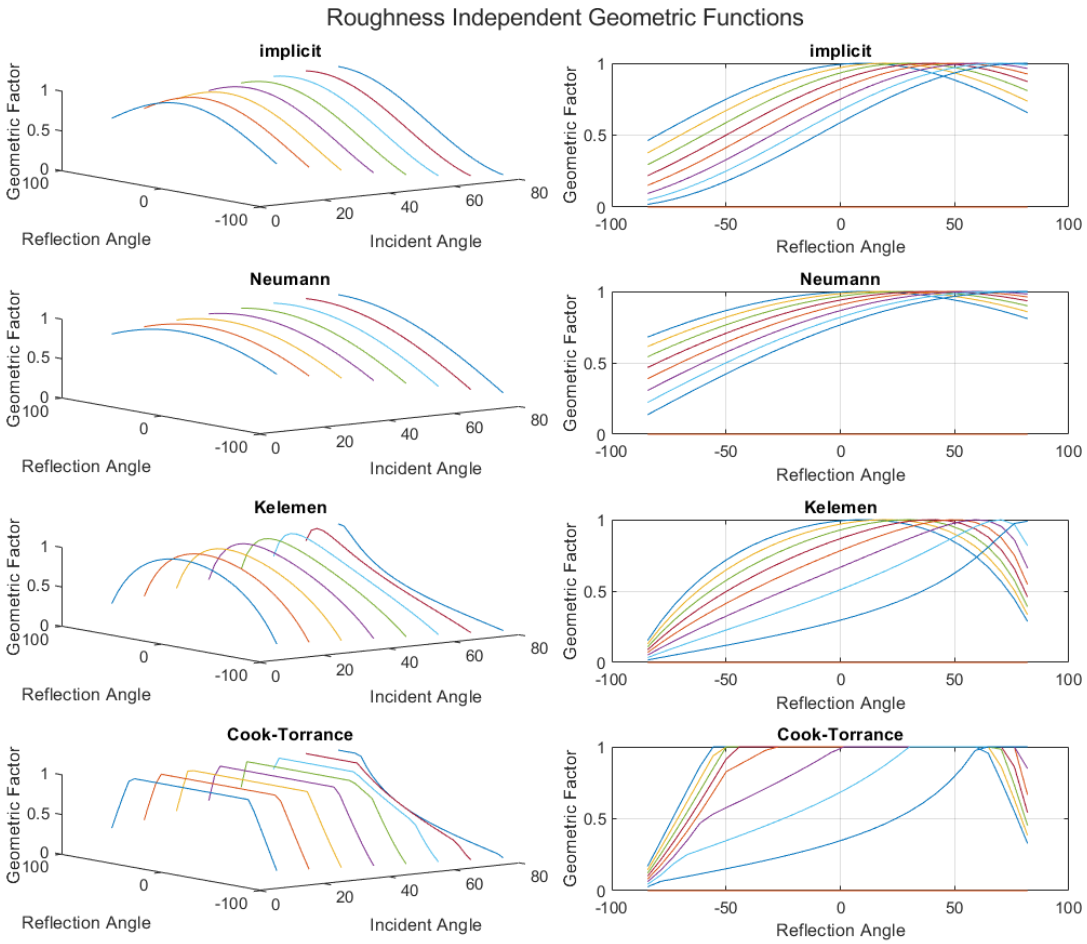


Figure 4.7: Plots of several geometric attenuation functions which are roughness independent. The left part emphasizes the angles and the overall shape, while the right part focuses on the reflection intensity.

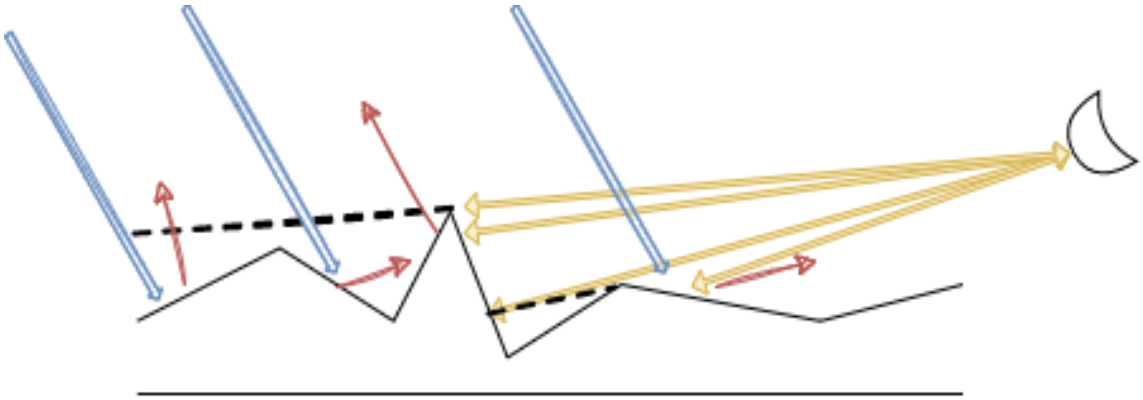


Figure 4.8: Illustrates the contribution of sub-surfaces masking other parts of sub-surfaces. Blue lines resemble incoming light, red lines are the reflection direction, and the yellow arrows are the viewer observation. The image illustrates self-shadowing (Incoming light can not reach the viewer due to valley scattering), and self-masking (The viewer can not view the sub-surfaces behind some specific sub-surface)

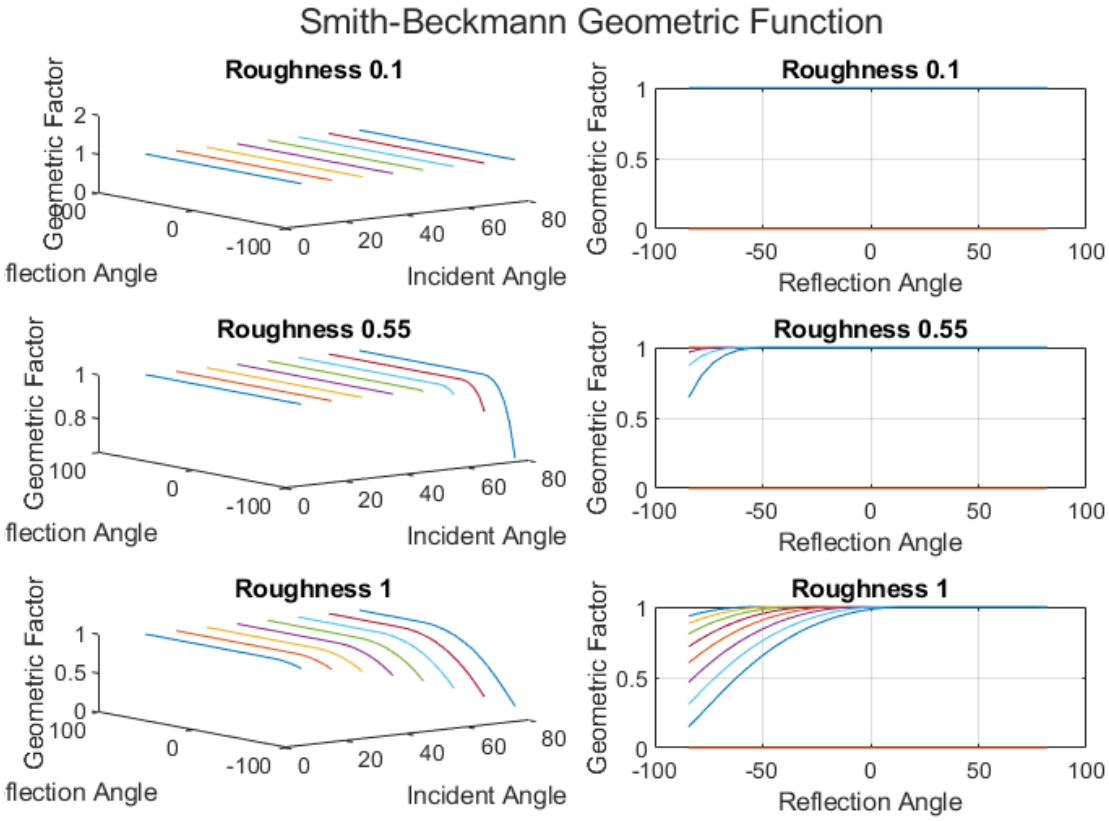


Figure 4.9: A visualization of the Smith-Beckmann Geometric function.

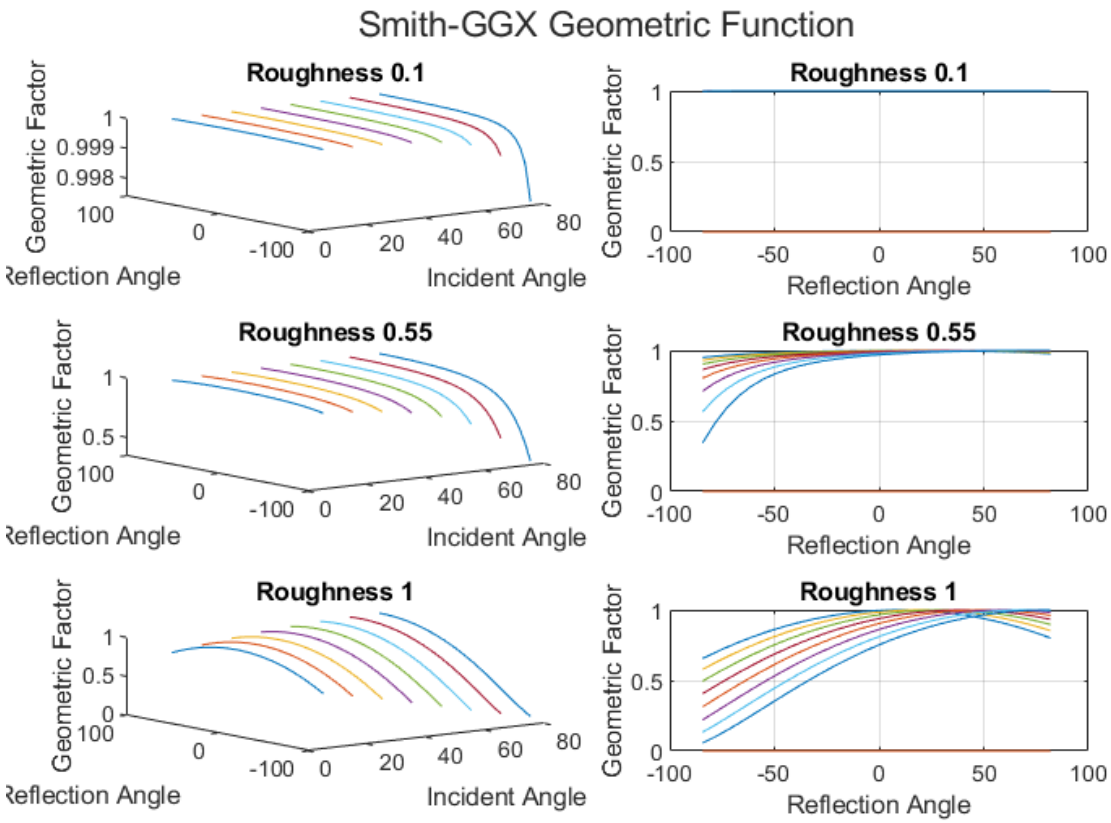


Figure 4.10: A visualization of the Smith-GGX Geometric function.

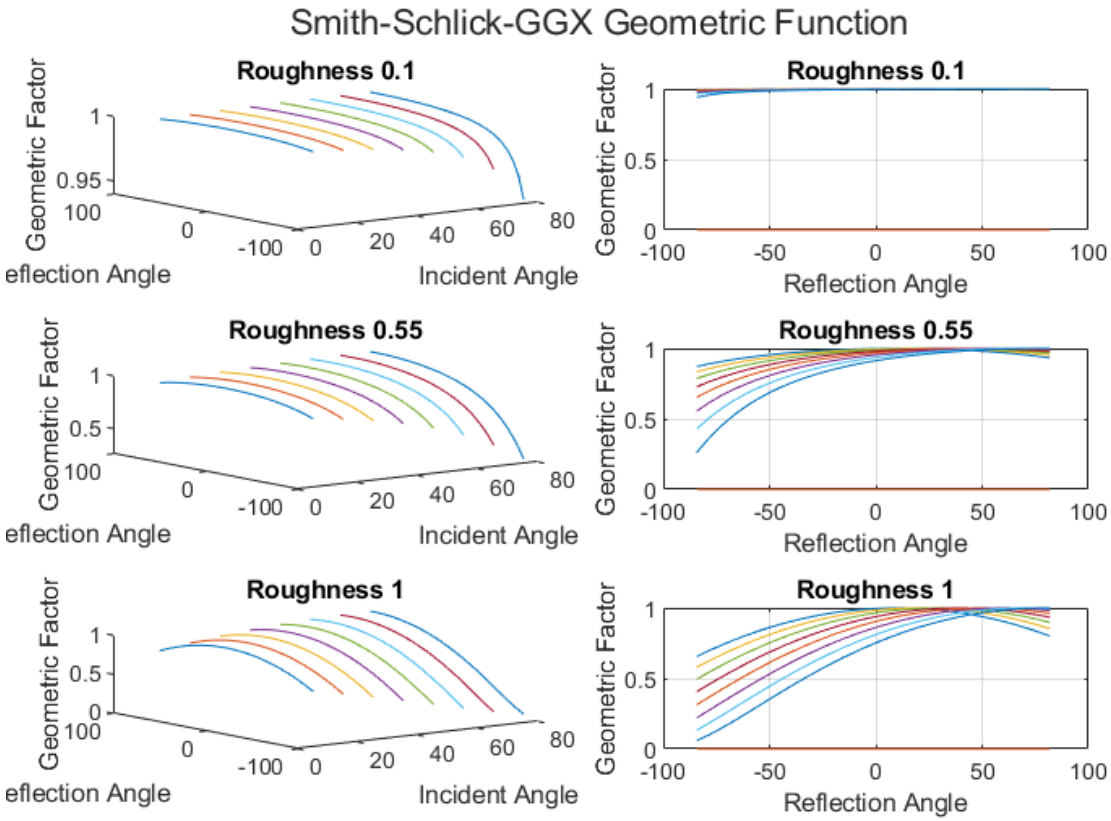
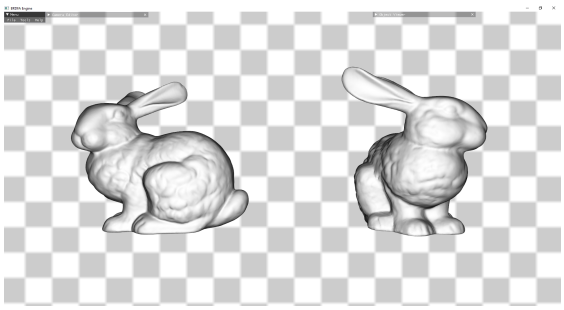
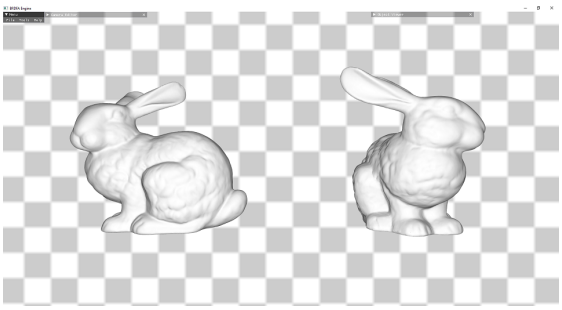


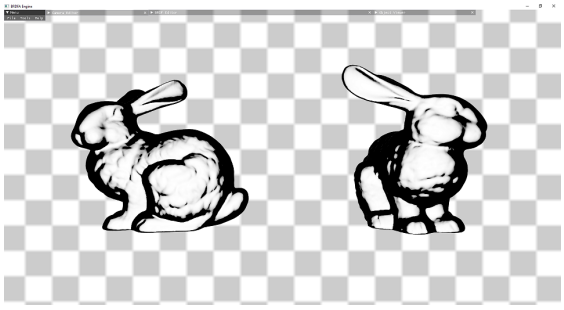
Figure 4.11: A visualization of the Smith-Schlick-GGX Geometric function.



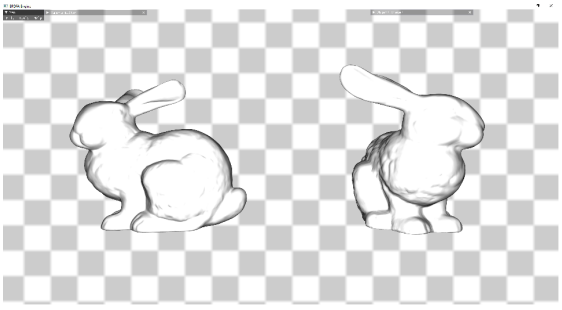
(a) Implicit



(b) Neumann

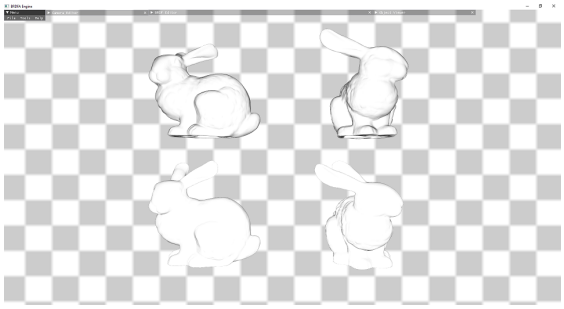


(c) Kelemen

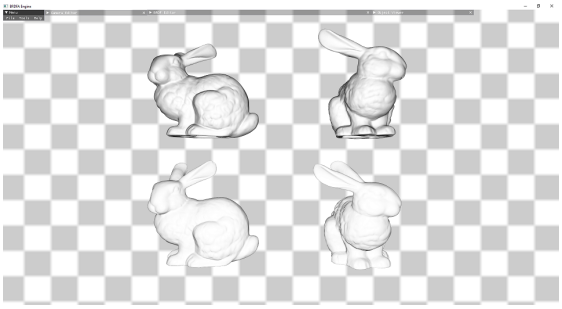


(d) Cook-Torrance-Sparrow

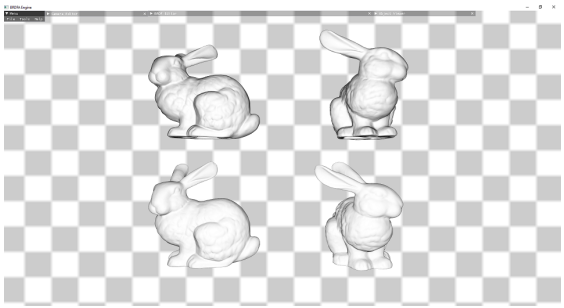
Figure 4.12: Visualization of roughness independent geometric attenucation functions.



(a) Smith-Beckmann



(b) Smith-GGX



(c) Smith-Schlick-GGX

Figure 4.13: Roughness dependent geometric attenuation functions. The first row has roughness of 1, while the second one has roughness of 0.7.

Chapter 5

Bidirectional Reflectance Distribution Functions Analysis

This chapter surveys several *BRDFs* models, which are empirical and physically-based. It illustrates and analyse the behavior of these models through utilizing, both, **Matlab** and the **BRDFA_Engine** [4].

5.1 Micro-Facet Based BRDF Models

microfacet-based BRDFs are proved fast and realistic, despite not being completely accurate. The distribution function of the normals described in Section 4.3, the geometric factor mentioned in Section 4.4, and the reflection intensity defined by the Fresnel factor explained in Section 4.2 are the three fundamental components that characterize the behavior of these BRDFs. This section provides specular and diffuse *microfacet-based BRDFs*.

5.1.1 Torrance-Sparrow Model

Torrance and Sparrow provided the first realistic specular micro-facet model in their paper *Theory for off-specular reflection from roughened surfaces* [34] in 1966. Earlier models existed and were mentioned in the same paper, such as *Pokrowsky-Schulz* and *Middleton and Mungall* models, yet, they turned out to be inaccurate, deficient, and unrealistic.

According to *Fresnel* equations, as the view angle increases, the reflectance rises. In other words, the BRDFs result rises when the surface is viewed from grazing

angles. Rough surfaces, however, tend to look dimmer than expected when viewed from grazing angles. This phenomenon is denoted by the *Off-Specular Peak* of reflection. Figure 5.1 illustrates some BRDFs calculated for real surfaces emphasizing the aforementioned phenomenon.

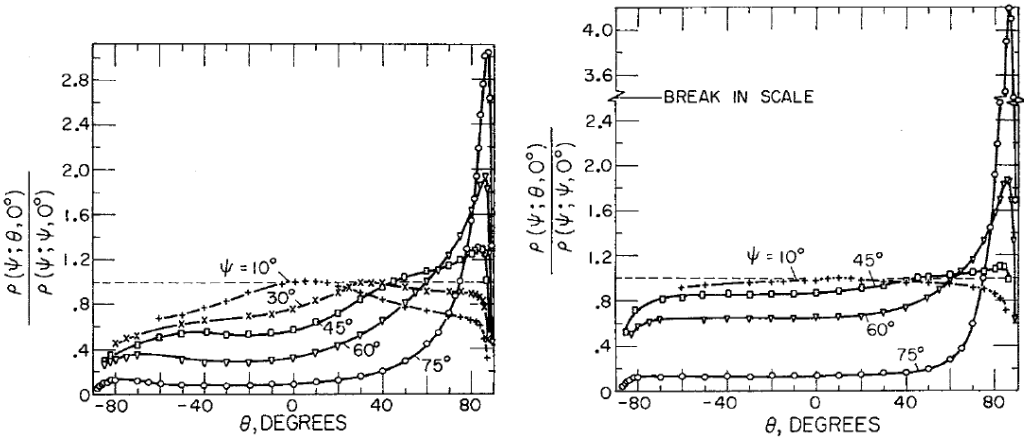


Figure 5.1: BRDF results in the plane of incidence for various angles. On the left, Aluminum (2024-T4), aluminum coated, is described, while on the right, Magnesium oxide ceramic is presented. Figure credits to *Theory for off-specular reflection from roughened surfaces* [34] by *Torrance and Sparrow*

Torrance and Sparrow investigated the issue of *off-specular peaks* and proposed the first realistic *physically-based BRDF* model. Furthermore, they described the source of the anomaly and presented the first Geometric attenuation function with adequate results, as shown in Equations 5.1 and 5.2. Figure 5.2 shows the symbols that are utilized.

$$G_a(\psi_p, \theta_p) = 1 - \frac{1 - (1 - A(\psi_p, \theta_p)^2)^{1/2}}{A(\psi_p, \theta_p)} \tag{5.1}$$

$$A(\psi_p, \theta_p) = \frac{\sin\theta_p{}^2 - \cos\left((\theta_p - \psi_p)/2\right)^2}{\cos\left(\theta_p - \psi_p\right)/2^2 - \cos\theta_p - \psi_p \sin\theta_p{}^2}$$

Since the sample ray may be shadowed, masked, both, or none, depending on the view and sample directions, a variety of scenarios must be addressed. Equation 5.2, which was developed by *Torrance and Sparrow* [34], handles all the mentioned cases.

$$G(\psi_p, \theta_p) = \begin{cases} G_a(\psi_p, \theta_p) & \text{if } 0 \leq \psi_p \leq \frac{\pi}{4} \text{ and } -\frac{\pi}{2} \leq \theta_p \leq \frac{\psi_p - \pi}{3} \\ G_a(\psi_p, \theta_p) & \text{if } 0 \leq \psi_p \leq \frac{\pi}{4} \text{ and } \frac{\psi_p + \pi}{3} \leq \theta_p \leq \frac{\pi}{2} \\ 1 & \text{if } 0 \leq \psi_p \leq \frac{\pi}{4} \text{ and } \frac{\psi_p - \pi}{3} \leq \theta_p \leq \frac{\psi_p + \pi}{3} \\ G_a(\psi_p, \theta_p) & \text{if } \frac{\pi}{4} \leq \psi_p \leq \frac{\pi}{2} \text{ and } -\frac{\pi}{2} \leq \theta_p \leq -\psi_p \\ G_a(\theta_p, \psi_p) & \text{if } \frac{\pi}{4} \leq \psi_p \leq \frac{\pi}{2} \text{ and } -\psi_p \leq \theta_p \leq 3\psi_p - \pi \\ 1 & \text{if } \frac{\pi}{4} \leq \psi_p \leq \frac{\pi}{2} \text{ and } 3\psi_p - \pi \leq \theta_p \leq \frac{\psi_p + \pi}{3} \\ G_a(\psi_p, \theta_p) & \text{if } \frac{\pi}{4} \leq \psi_p \leq \frac{\pi}{2} \text{ and } \frac{\psi_p + \pi}{3} \leq \theta_p \leq \frac{\pi}{2} \end{cases} \tag{5.2}$$

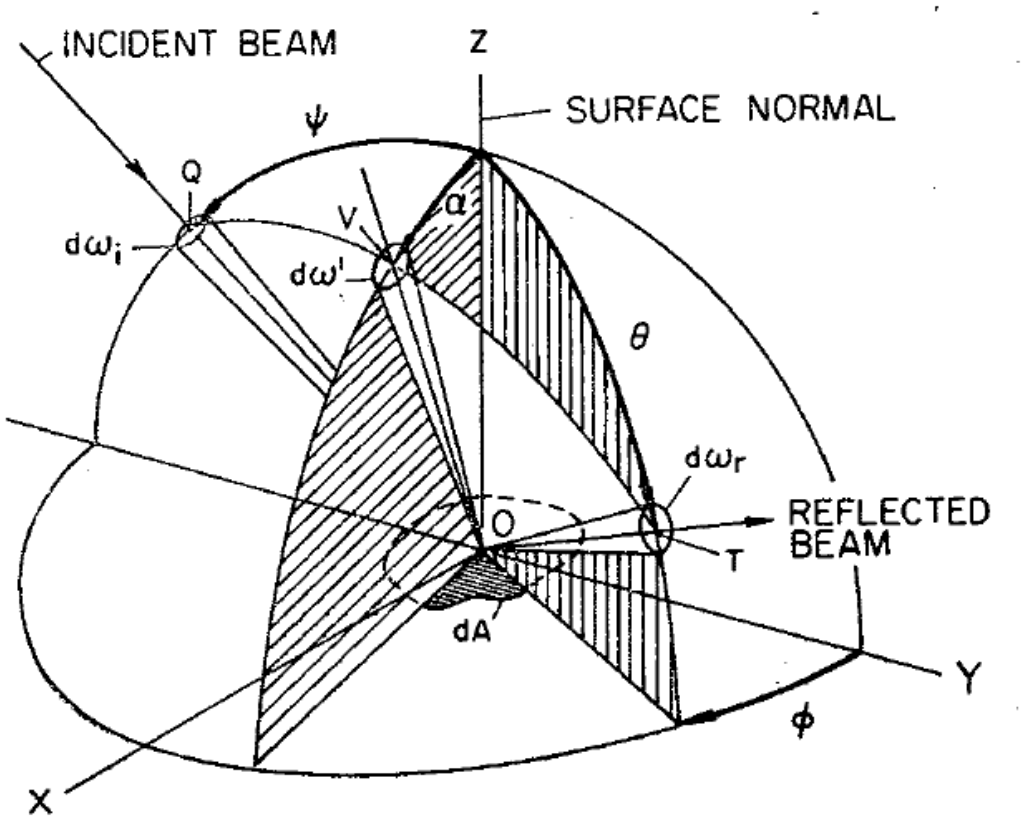


Figure 5.2: Illustration of the used symbols. Figure credits to *Theory for off-specular reflection from roughened surfaces* by *Torrance and Sparrow* [34]

In the *Torrance-Sparrow* study of the *Geometric Attenuation Function*, angles are used to define the directions. However, because vector arithmetic is computationally faster than trigonometric functions, the geometric attenuation function can be expressed as Equation 5.4 using the trigonometric identities shown in Equation 5.3. Note that vector arithmetic is valid only if the vectors are normalized.

$$\begin{aligned}
&\cos(\theta) = \vec{v_\theta} \cdot \vec{n} \\
&\sin^2(\theta) = 1 - \cos^2(\theta) = 1 - (\vec{v_\theta} \cdot \vec{n})^2 \\
&\cos(\theta_a - \theta_b) = \vec{a} \cdot \vec{b} \\
&\cos^2(\frac{\theta_a - \theta_b}{2}) = \frac{1 + \vec{a} \cdot \vec{b}}{2}
\end{aligned} \tag{5.3}$$

Equation 5.1 can be represented as:

$$\begin{aligned}
&G_a(\psi_p, \theta_p) = 1 - \frac{1 - (1 - A(\psi_p, \theta_p)^2)^{1/2}}{A(\psi_p, \theta_p)} \\
&A(\theta_i, \theta_r) = \frac{C_1 - \frac{1}{2}(1 + C_2)}{\frac{1}{2}(1 + C_2) - C_2 C_1} \\
&C_1 = \sin^2 \theta_r = 1 - (\vec{r} \cdot \vec{n})^2 \\
&C_2 = \cos(\theta_r - \theta_i) = \vec{r} \cdot \vec{i}
\end{aligned} \tag{5.4}$$

Furthermore, the final $A(\vec{v_i}, \vec{v_r})$ function can be simplified, as seen in Equation 5.5. Since the shortened version is faster to compute than the function provided in the article, it is recommended in a production application. The simplified version was not presented in the paper, thus, it has been derived, tested and compared to the original version using *Matlab*. The original $A(\vec{i}, \vec{r})$ as well as the simplified vector form implementations are found in the project directory (BRDFA_Engine) [4].

$$A(\vec{i}, \vec{r}) = \frac{2(\vec{r} \cdot \vec{n})^2 + (\vec{r} \cdot \vec{i}) - 1}{2(\vec{r} \cdot \vec{i})(\vec{r} \cdot \vec{n})^2 - (\vec{r} \cdot \vec{i}) + 1} \tag{5.5}$$

Figure 5.4 presents an illustration of the Geometric attenuation function for different angles of reflection and incidence. In practical implementation, the incidence angle may represent the view direction, while the reflected angles represent the sample directions. A comparison of the *Torrance-Sparrow* and *Smith-Schlick-GGX* geometric functions is provided in Figure 5.3.

Equation 5.6 describes the rendering equation using the terms of radiometry. This notation was used since the general form of the rendering equation was not established yet (Section 3.1).

$$\begin{aligned}
&dL_r = dL_{r,s} + dL_{r,d} \\
&dL_r(\psi; \theta, \phi) = brdf_s(\psi; \theta, \phi).dL_i + brdf_d(\psi; \theta, \phi).dL_i
\end{aligned} \tag{5.6}$$

where

- dL_r is the reflected radiance
- $dL_{r,d}$ is the diffuse radiance
- $dL_{r,s}$ is the specular radiance
- $brdf_d(\psi;\theta,\phi)$ is the diffuse BRDF model
- $brdf_s(\psi;\theta,\phi)$ is the specular BRDF model

Equations 5.7 and 5.8 demonstrate the model presented in the paper.

$$brdf_s(\psi;\theta,\phi) = \frac{F(\hat{\psi},\vec{h}).G(\psi_p,\theta_p).f.b.\exp(-c^2a^2).d\omega_i}{4\cos\theta} \tag{5.7}$$

Where

- $\vec{h}$ midway vector between the view and the incident
- $d\omega_i = \frac{1}{\cos\psi}$
- a, b are variables used in the Gaussian Distribution
- f is facets size

Since the used *Distribution Function* in Equation 5.7 is the *Gaussian Distribution Function*, a general model can be formulated as Equation 5.8.

$$brdf_s(\psi;\theta,\phi) = \frac{F(..).G(..).D(..)}{4\cos\theta\cos\psi} \tag{5.8}$$

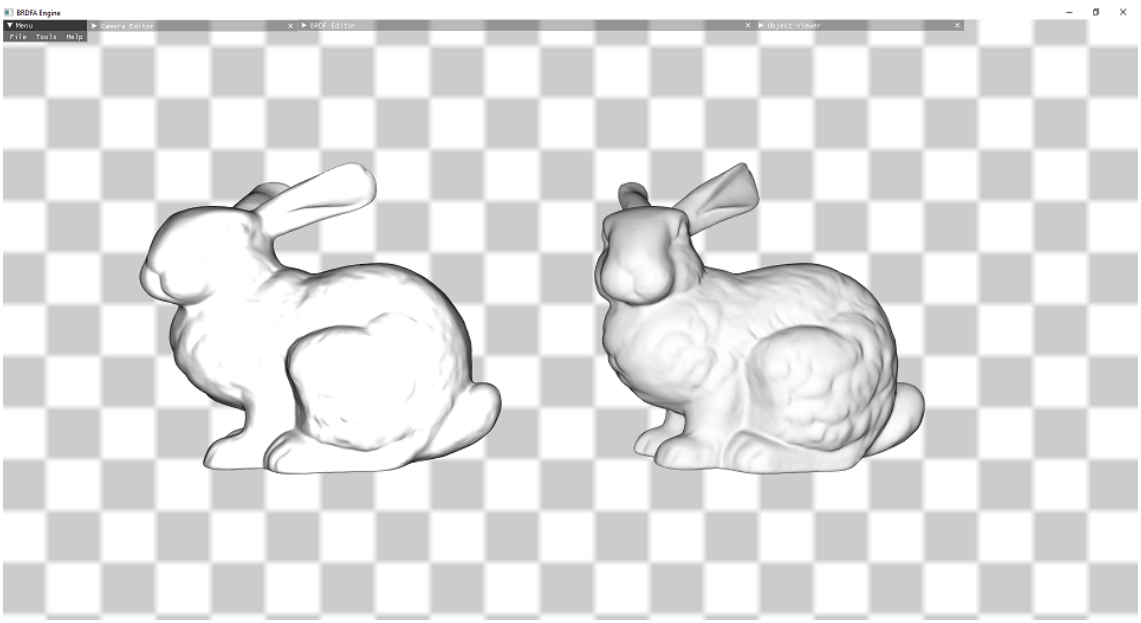


Figure 5.3: On the left, the torrance-sparrow geometric attenuation factor is provided, while on the right, the smith-schlick geometric attenuation factor is shown with roughness parameter of 1.0.

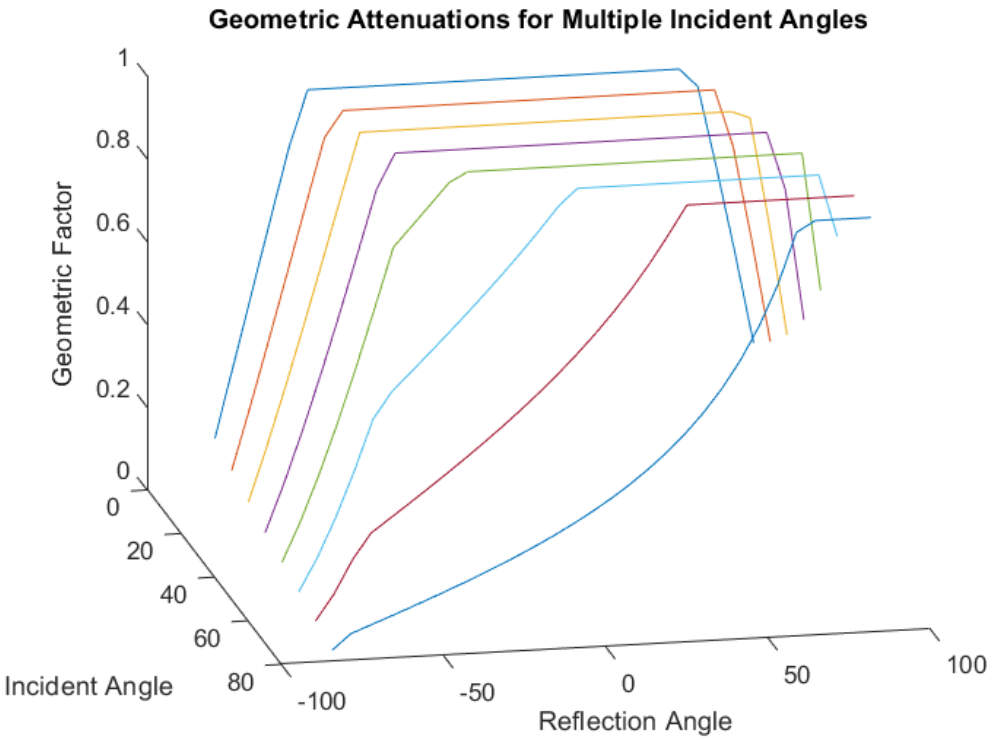


Figure 5.4: Illustration of the Torrance-Sparrow Geometric Attenuation function for multiple incidence and reflection angles (with respect to surface normal).

The closer curves in Figure 5.4 show the results when the surface is viewed from near grazing angle. In this case, samples that are directed toward the view direction are observed, while other samples are obstructed and/or shadowed.

5.1.2 Cook-Torrance Model

Torrance and Cook [35] provided an alternative distribution, Fresnel, and geometric attenuation functions to the *Torrance-Sparrow* BRDF (Section 5.1.1). Remark, the *Torrance-Sparrow* model is presented in Equations 5.9 and 5.8.

$$brdf_s(\vec{L}, \vec{V}, \vec{N}) = \frac{FDG}{4(\vec{V} \cdot \vec{N})(\vec{L} \cdot \vec{N})} \tag{5.9}$$

Various distribution functions may be considered to be used in this model, some of which are computationally intensive while others are wavelengths dependent. Similarly, any geometric attenuation and Fresnel approximations can be used.

The Fresnel function used by *cook and Torrance* is illustrated in Equation 5.10. Since it is derived from the original Fresnel equation, in which the *absorption atten-*

uation is put to zero (k = 0), it is wavelength dependent. Different wavelengths may have different base reflection factors F_0 and refraction indices. Furthermore, Figure 5.5 presents a visual comparison between *Schlick* (Equation 4.5) and *Cook-Torrance* approximations of the Fresnel function.

$$F_\lambda(\vec{V},\vec{H}) = \frac{1}{2}.(\frac{g-c}{g+c})^2.[1+(\frac{c(g+c)-1}{c(g-c)+1})^2] \tag{5.10}$$
$$g^2 = n^2 + c^2 - 1$$
$$c = \cos\theta = \vec{V}\cdot\vec{H}$$

where $\vec{V}$ is the view direction, $\vec{H}$ is the halfway direction, and n is the refraction index.

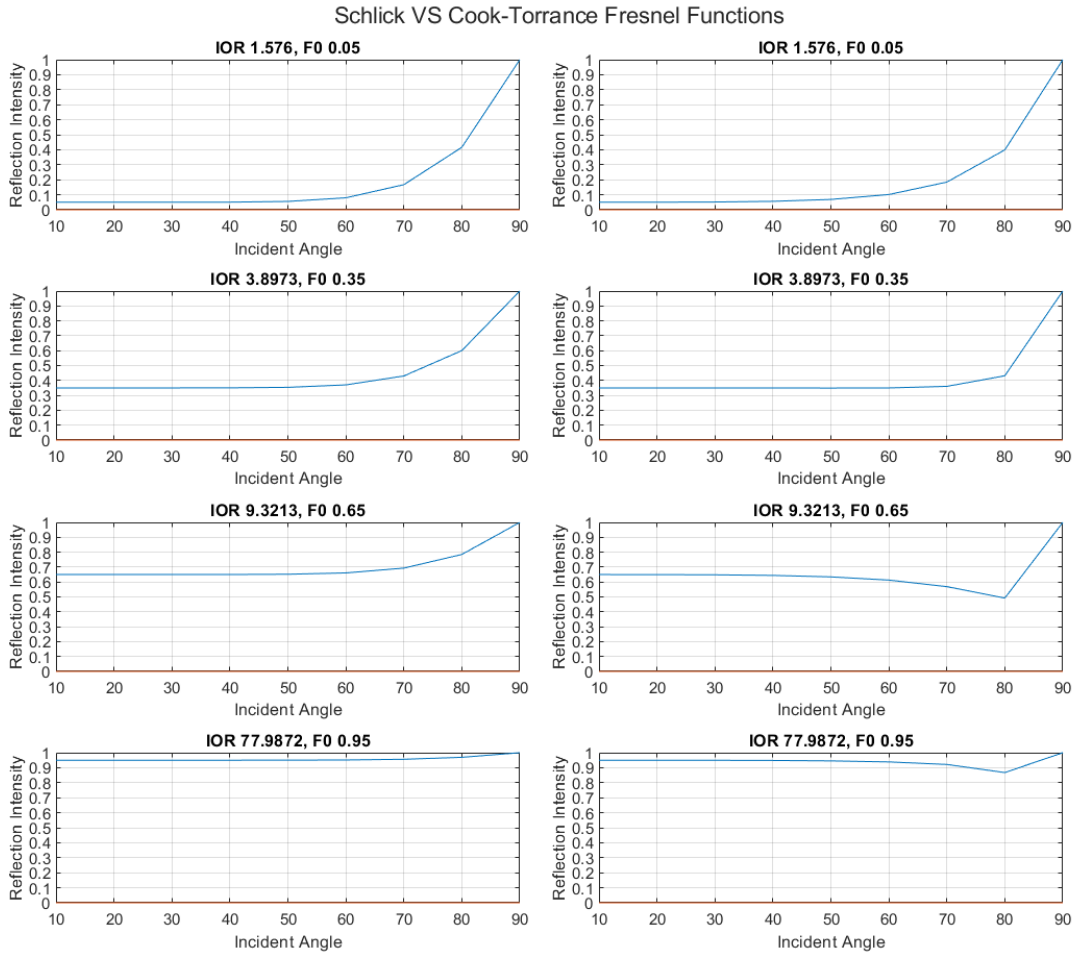


Figure 5.5: A comparison between *Cook-Torrance* and *Schlick* approximations of the fresnel function. The left set of plots belong to *Schlick*, while the right ones are from the Cook-Torrnace model.

Cook and Torrance employed the geometric attenuation function, as shown in Equation 5.11 and visualized in Figure 5.6, as detailed by *Blinn and Torrance* [34,

33, 44]. It returns $G = 1$ for non-blocked reflections, otherwise, partial masking, shadowing, or both, is returned. It is worth noting that this method is incorrect since it returns 1 for perfect reflection directions. However, to remedy the issue, the sample ray is reflected around the surface normal to achieve the expected results seen in Figure 5.4.

$$G_{cook-torrance} = \min\{1, \frac{2(\vec{N} \cdot \vec{H})(\vec{N} \cdot \vec{V})}{\vec{V} \cdot \vec{H}}, \frac{2(\vec{N} \cdot \vec{H})(\vec{N} \cdot \vec{L})}{\vec{V} \cdot \vec{H}}\} \tag{5.11}$$

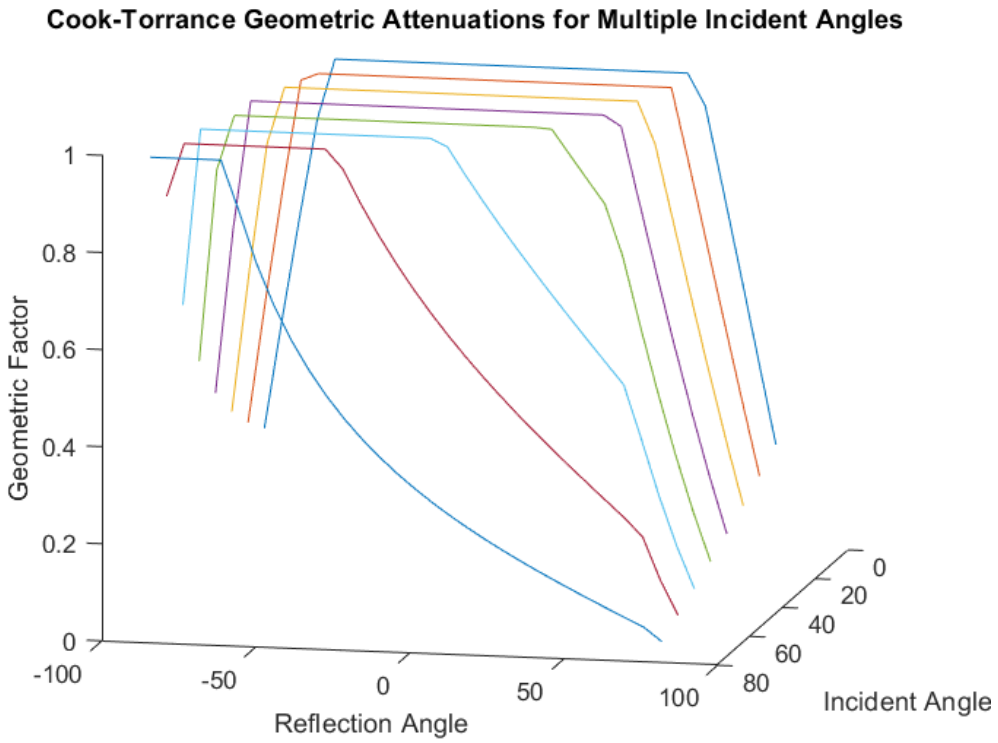


Figure 5.6: Geometric attenuation function used by cook-torrance specular BRDF model.

various *Normal Distribution Functions* have been discussed by *Blinn [33, 44], Torrance and Sparrow [34], Davies [45], and Bennett and Porteus [46]*, yet, *Cook and Torrance* chose the *Beckmann* distribution function, which is shown in Equation 5.12 and depicted in Figure 5.7, since it is applicable for different surface conditions. Furthermore, a full discussion on the distribution functions is presented by *Brian Karis [43] and Heitz [39]*. Additionally, *Cook-Torrance* model is inspected with various settings in Figures 5.21 and 5.8.

$$D_{beckmann}(\theta_H) = \frac{e^{-(\tan\theta_H/\alpha)^2}}{4.\alpha^2.(\cos\theta_H)^4} \tag{5.12}$$

Or can be rewritten in vector arithmetic as:

$$D_{beckmann}(H) = \frac{1}{4\alpha^2(\vec{H}\cdot\vec{N})^4} exp^{(\frac{(\vec{H}\cdot\vec{N})^2-1}{\alpha^2(\vec{H}\cdot\vec{N})^2})} \tag{5.13}$$

Where

- α is a mapped value of roughness
- Unreal Engine 4 uses the following mapping : $\alpha = (roughness)^2$ [43]

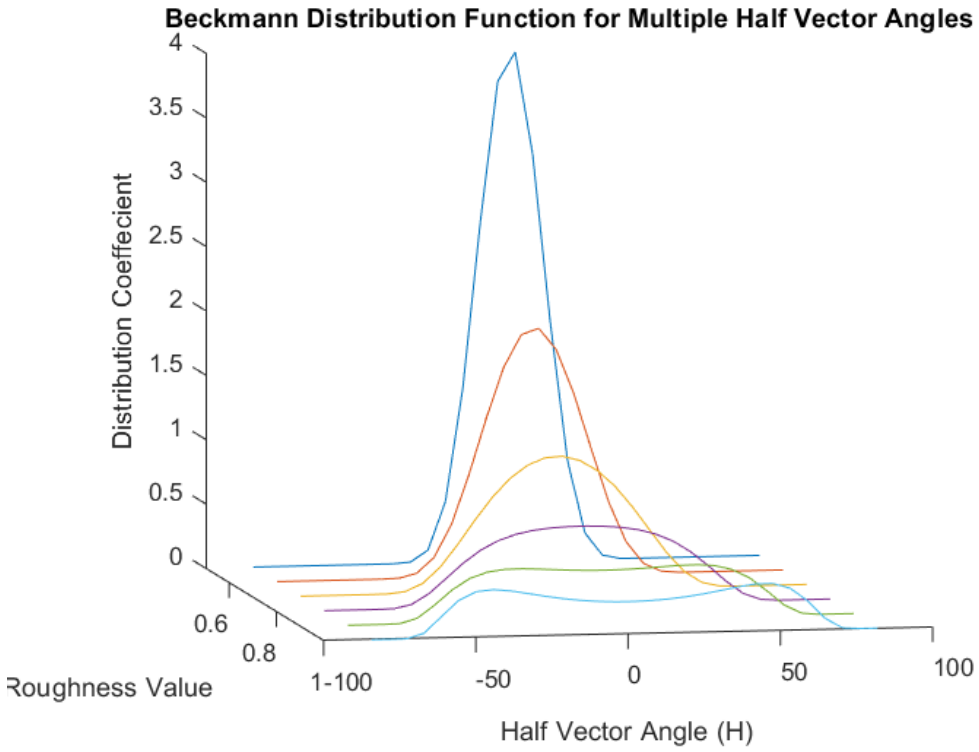


Figure 5.7: Beckman distribution function for multiple roughness values. Roughness values are mapped into α by $\alpha = r^2$. Observably, the smoother the surface, the more microfacets facing the normal direction.

.

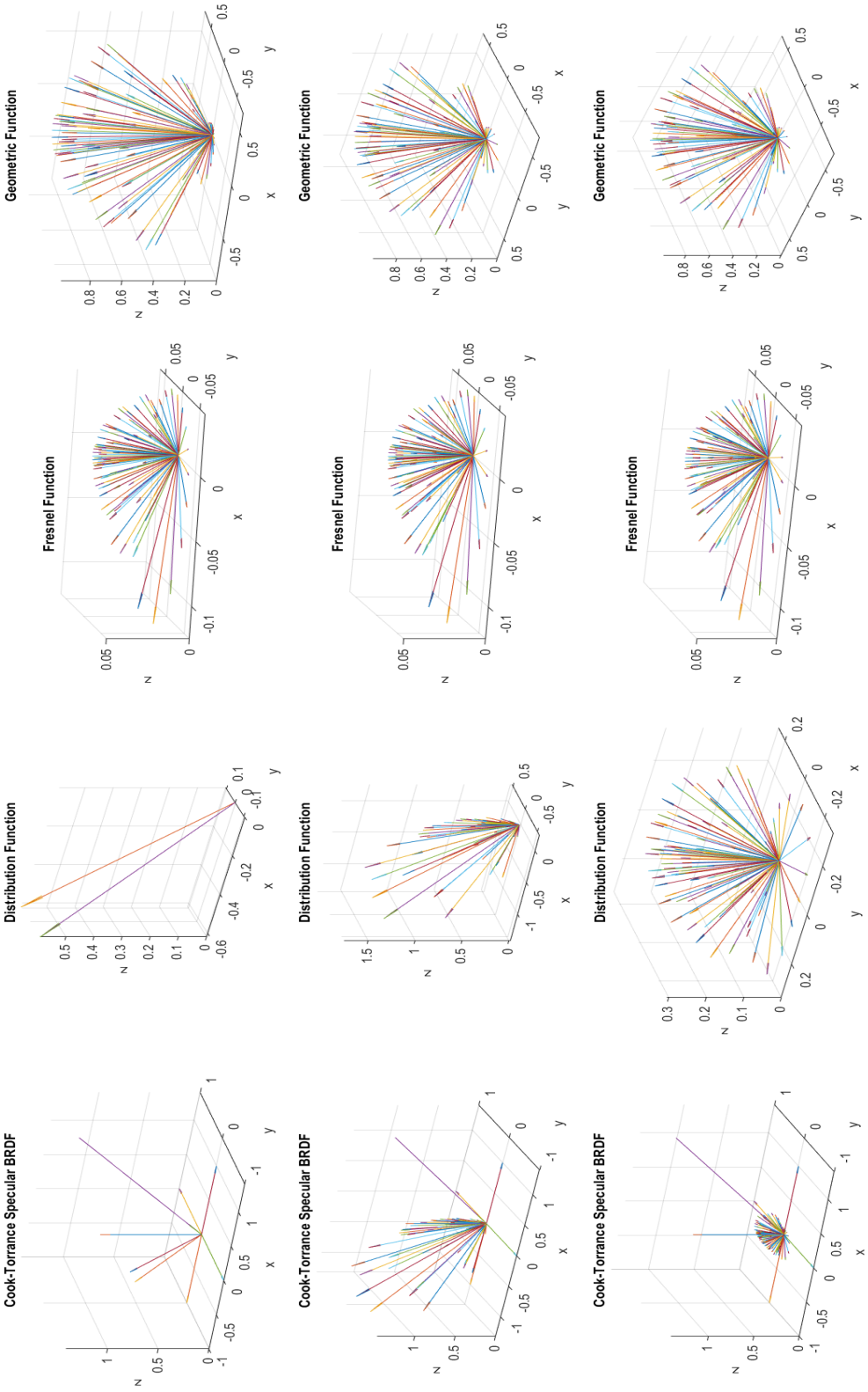


Figure 5.8: Cook-Torrance specular BRDF Visualization concerning multiple roughness values. Roughness values are 0.2, 0.6, and 1.0 respectively. And, the used refraction index is 1.576 (Glass). The view angle is 45°.

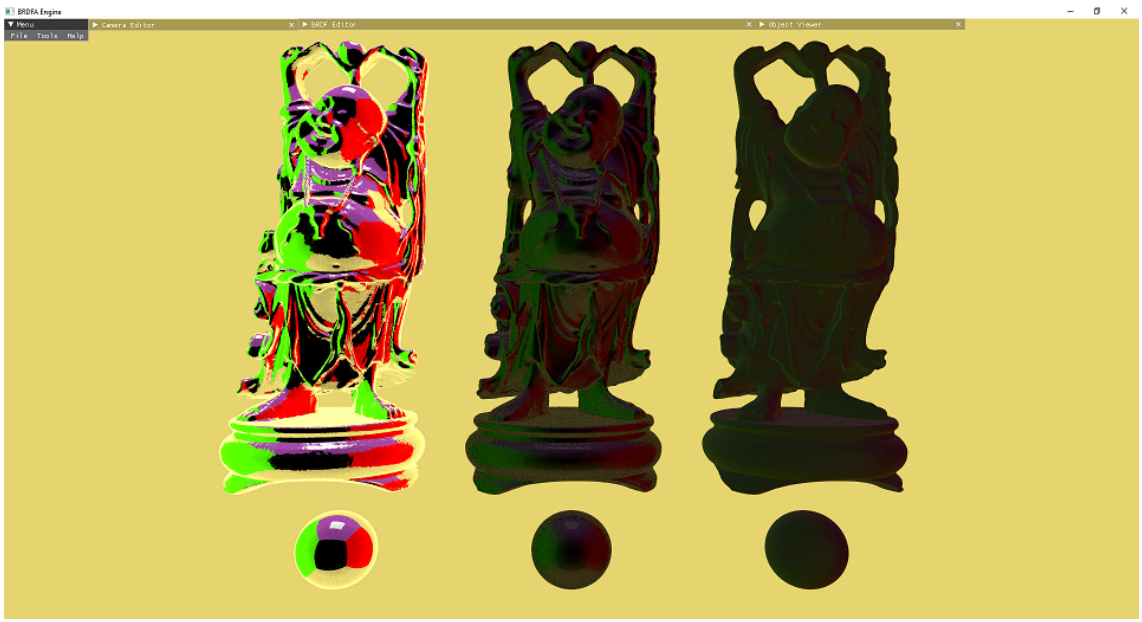


Figure 5.9: Illustration of multiple Cook-Torrance model. Images roughness settings are 0.18, 0.5, and 1.0 respectively. Fresnel coefficient is calculated through The the schlick fresnel approximation with Base refractive index used for all of them is 0.25 ($F_0 = 0.25$). The *Cornell* environment map is used, and the sample count per point is 30.

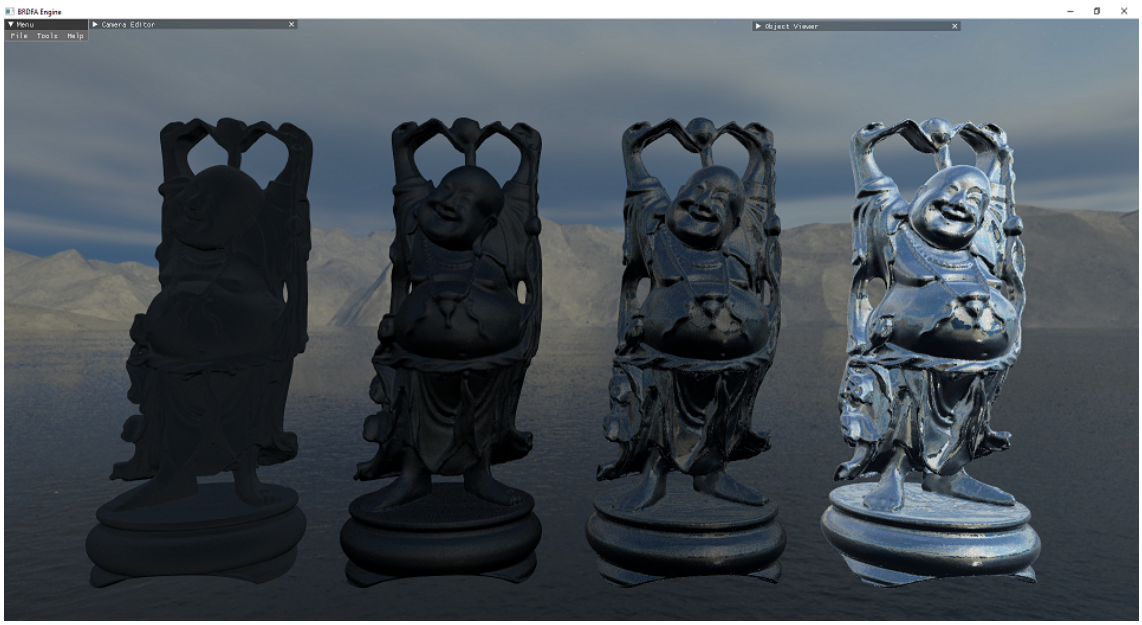


Figure 5.10: An illustration of Cook-Torrance model with roughness settings of 0.18, 0.3 , 0.6, and 1.0. Fresnel coefficient is calculated through the the schlick fresnel approximation with Base refractive index used for all of them is 0.25 ($F_0 = 0.25$). The sample count per point is 30.

5.1.3 Oren-Nayar Model

Oren and Nayar [47] proposed a generalization of the *Lambert* diffuse model for extremely rough surfaces, such as concrete and sand, based on the microfacet theory proposed by *Torrance and Sparrow* [34]. The model is seen in Equations 5.14 and 5.15, and visualized in Figure 5.11.

$$L_{r,d} = \int_{\Omega} brdf_d(\theta_i, \theta_r, \phi_i, \phi_r).L_i.di \tag{5.14}$$

$$brdf_d(\theta_i, \theta_r, \phi_i, \phi_r) = \frac{\rho}{\pi}\cos(\theta_i)(A + B.\max\{0, \cos(\phi_i - \phi_r)\}.\sin\alpha.\tan\beta) \tag{5.15}$$

$$A = 1 - \frac{r^2}{2(r^2 + 0.33)}$$

$$B = 0.45\frac{r^2}{r^2 + 0.09}$$

$$\alpha = \max(\theta_i, \theta_r)$$

$$\beta = \min(\theta_i, \theta_r)$$

The angles θ & ϕ resemble the angles of the vector from the normal and the surface tangent, respectively. The *Oren-Nayar* model returns the *Lambertian* model for smooth surfaces where $A = 1$ & $B = 0$, but it reduces irradiance by adding an absorption factor for rough surfaces.

The equation can be implemented based on vector arithmetic, as seen in Equation 5.16 and detailed in the *Physically Based Rendering:From Theory To Implementation* [48]. Furthermore, the implementation is viewed and tested n Figure 5.11. Observably, *Oren-Nayar* model tends to absorb more light for rough surfaces than smooth surfaces, which makes rougher surfaces look dimmer.

$$brdf(\vec{V}, \vec{L})_{diffuse} = \rho.(\vec{L}\cdot\vec{N}).\{A + B.\max(0, C).D.E\} \tag{5.16}$$

$$A = 1 - \frac{r^2}{2(r^2 + 0.33)}$$

$$B = 0.45\frac{r^2}{r^2 + 0.09}$$

$$C = \frac{\vec{V}.x}{\sqrt{1 - (\vec{V}\cdot\vec{N})^2}}\frac{\vec{L}.x}{\sqrt{1 - (\vec{L}\cdot\vec{N})^2}} + \frac{\vec{V}.y}{\sqrt{1 - (\vec{V}\cdot\vec{N})^2}}\frac{\vec{L}.y}{\sqrt{1 - (\vec{L}\cdot\vec{N})^2}}$$

$$D = \sqrt{1 - \min((\vec{V}\cdot\vec{N}), (\vec{L}\cdot\vec{N}))^2}$$

$$E = \frac{\sin(\min(\theta_v, \theta_l))}{\cos(\min(\theta_v, \theta_l))} = \frac{\sqrt{1 - \max((\vec{V}\cdot\vec{N}), (\vec{L}\cdot\vec{N}))^2}}{\max((\vec{V}\cdot\vec{N}), (\vec{L}\cdot\vec{N}))}$$

The minimum angle between the ray direction and the normal is equivalent to the maximum length of projection between the ray vector and the surface normal.

Figure 5.11: A comparison between Oren-Nayar and lambertian diffuse models. On the left, the lambartian model is shown, while the Oren-Nayar is shown in the middle and right. The middle object has roughness of 0, while the object on the right has roughness of 1.

5.2 Empirical BRDF Models

On the contrary, to physically (Microfacet) based models such as *Oren-Nayar* and *Cook-Torrance*, empirical models approximate the BRDF numerically. These models are quicker to compute, yet, the results might not be realistic. As a result, fine-tuning the parameters of these models to create a realistic scene is more difficult, and their use may not be appropriate in all contexts.

5.2.1 Lambertian Model

Johann Heinrich Lambert introduced the concept of perfect opaque surfaces in his book *Photometria* in 1760. Based on the *Geometric Optics*, he stated that the incoming light to a surface is distributed along the covered area. Therefore, if the surface is tilted, the same beam of light covers more area, as viewed in Figure 5.12. Consequently, a point on the surface looks dimmer since the light energy is distributed over more points.

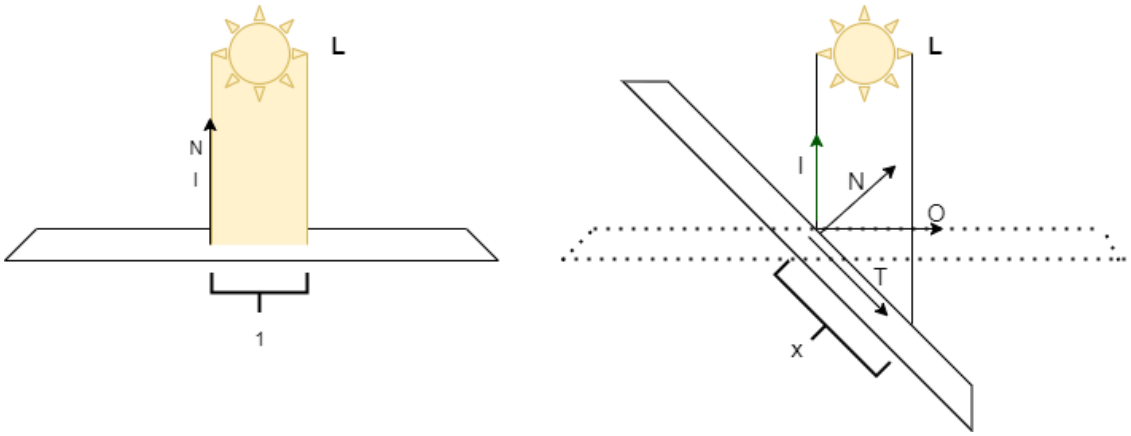


Figure 5.12: An illustration of the received radiance on a surface. L represents the radiance (Color), N, I, O and T are vectors presenting the normal direction, incident direction, original tangent (Before rotation) direction, and the new tangent direction. X resembles the area that the light is covering when the surface is tilted.

Figure 5.13: A fully diffused sphere rendered using the Lambertian model in the BRDFA_Engine.

The surface receives all incoming light when it is facing toward the incident direction. However, if the surface is tilted, as shown in Figure 5.12, the received light is distributed over an area $\{L * \frac{1}{X}\}$. The $\{\frac{1}{X}\}$ term is the *Cosine* of the angle between T & O, or N & I since they are equivalents, and is denoted by the *Geometric Term*. The final diffuse BRDF is shown in Equation 5.17, and rendered in Figure 5.13.

$$brdf_d(\theta_i, \theta_v) = L.\cos(\theta_i) \tag{5.17}$$

Similarly, through using vector arithmetic:

$$brdf_d(\vec{l}, \vec{v}) = L.(\vec{l} \cdot \vec{n}) \tag{5.18}$$

5.2.2 Phong and Blinn-Phong Models

> Bui Tuong Phong: "We do not expect to be able to display the object exactly as it would appear in reality, with texture, overcast shadows, etc. We hope only to display an image that approximates the real object closely enough to provide a certain degree of realism."

Bui Tuong Phong, a talented computer graphics researcher, developed one of the first empirical models to display the glossiness of plastic-like objects. As seen in Equation 5.19, the *Phong* model is dependent on a few adjustable parameters. It is crucial to highlight that these parameters have no physical meaning.

$$L = L_a + \Sigma_i^m L_c.k_d.(\vec{L_i}, \vec{N}) + k_s.(\vec{R}, \vec{V})^n \tag{5.19}$$

where

- $R = reflect(\vec{L}) = 2(\vec{L} \cdot \vec{N})\vec{N} - \vec{L}$
- L_c is the incident color
- L_a is the ambient color
- k_d is the surface color
- k_s is the reflection color
- n is the shininess of the reflection

Since the n parameter has no physical meaning, the artist must try different values until the desired result is achieved. Figures 5.14 and 5.15 show the different values of n for multiple incidents and view angles, and the Phong model is rendered in Figure 5.16.

Despite its resemblance to a distribution function, the Phong model is not normalized. As a result, it is not energy conservative (Equation 4.8 fails), and it cannot be used for weighted multi-sampling techniques, including Monte-Carlo sampling. However, it can be used to calculate the direct illumination. Finally, Table 5.1 displays the area under the curve for the same parameters used in Figure 5.14, while Equation 5.20 describes the behavior of the distribution when the parameter (n) increases or decreases.

$$\lim_{n\longrightarrow\infty}\{\int_{-\pi/2}^{\pi/2}\cos^n(\theta)d\theta\}=0 \quad , \quad \lim_{n\longrightarrow 1}\{\int_{-\pi/2}^{\pi/2}\cos^n(\theta)d\theta\}=2 \tag{5.20}$$

Calculated Area	
Integration	Area
$\int_{-\pi/2}^{\pi/2}\cos^1(\theta)d\theta$	2
$\int_{-\pi/2}^{\pi/2}\cos^{234}(\theta)d\theta$	0.16368...
$\int_{-\pi/2}^{\pi/2}\cos^{467}(\theta)d\theta$	0.11593...
$\int_{-\pi/2}^{\pi/2}\cos^{700}(\theta)d\theta$	0.09470...

Table 5.1: The *Phong* distribution area for multiple parameters.

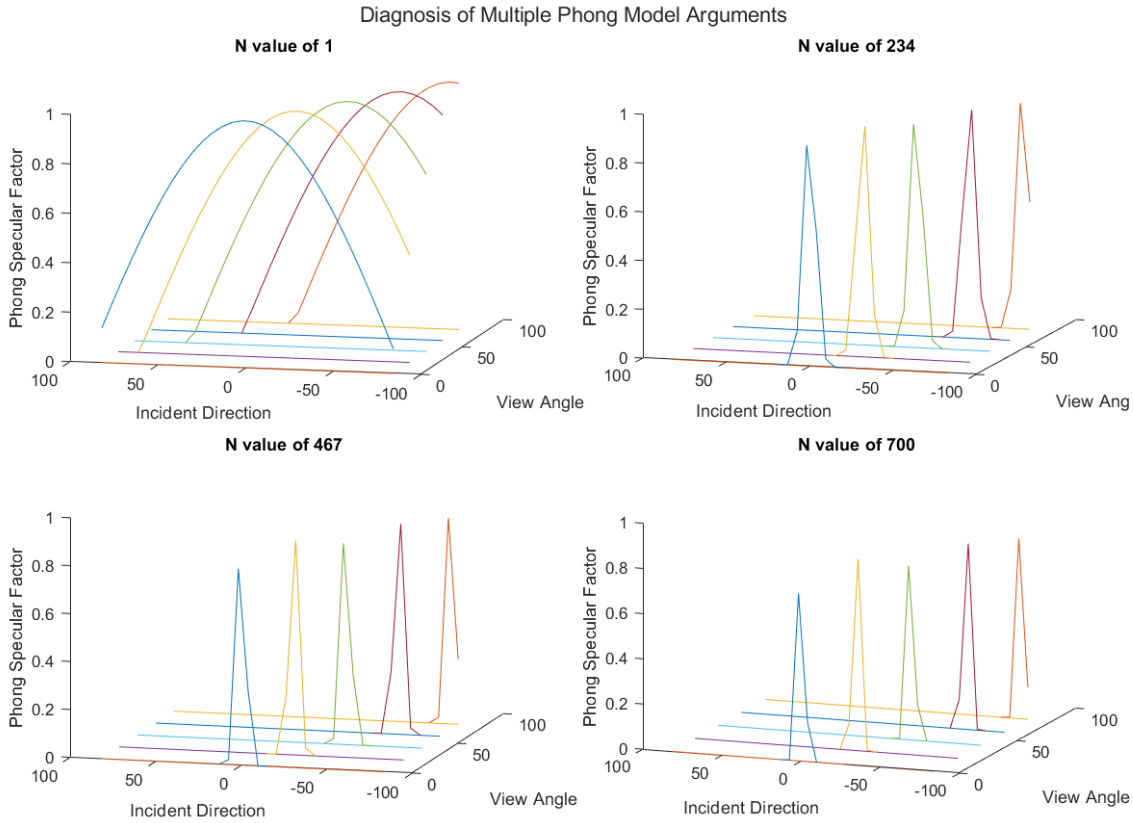


Figure 5.14: A diagnosis of Phong specular term with four different n arguments for multiple incident and view directions.

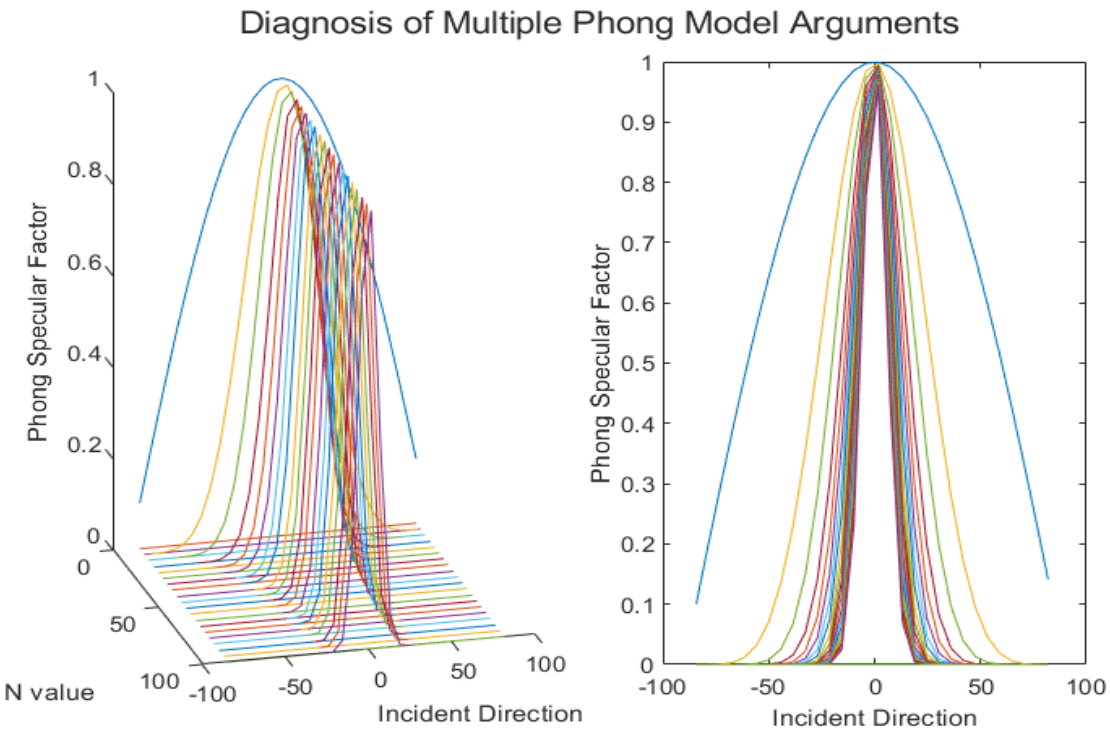


Figure 5.15: A diagnosis of Phong specular term with multiple different shininess (n) arguments for multiple incident directions

Figure 5.16: An illustration of Phong specular BRDF model with different n arguments. The objects from right to left have shininess values of 1, 10, and 50 respectively.

Blinn proposes changing the specular term to get a slightly better outcome. Instead of projecting the reflection vector to the view direction, *Blinn* recommended projecting the half vector to the surface normal. This adjustment corrects the specu-

lar lopes seen in the Phong model. Equation 5.21 illustrates the *Blinn-Phong* model, while Figures 5.18 and 5.17 visualize it.

$$L = L_a + \Sigma_i^m L_c.k_d.(\vec{L_i}, \vec{N}) + k_s.(\vec{H}, \vec{N})^n \tag{5.21}$$

where

- $H = \frac{\vec{L}+\vec{V}}{||\vec{L}+\vec{V}||}$
- L_c is the Incident color
- L_a is the Ambient light
- k_d is the Surface color
- k_s is the Reflection color
- Σ_i^m is the Direct lights

Because *Phong and Blinn-Phong* models lack adequate normalization, they are not suited for Monte-Carlo sampling. Since they do not yield correct energy balance, they do not provide meaningful physical results. Furthermore, only direct illumination can be considered by Phong and Blinn-Phong models. However, other empirical models, such as *Lafortune* [49] and Lewis [50], add energy conservation to the *Phong and Blinn-Phong* models, making them usable for multi-sampling techniques.

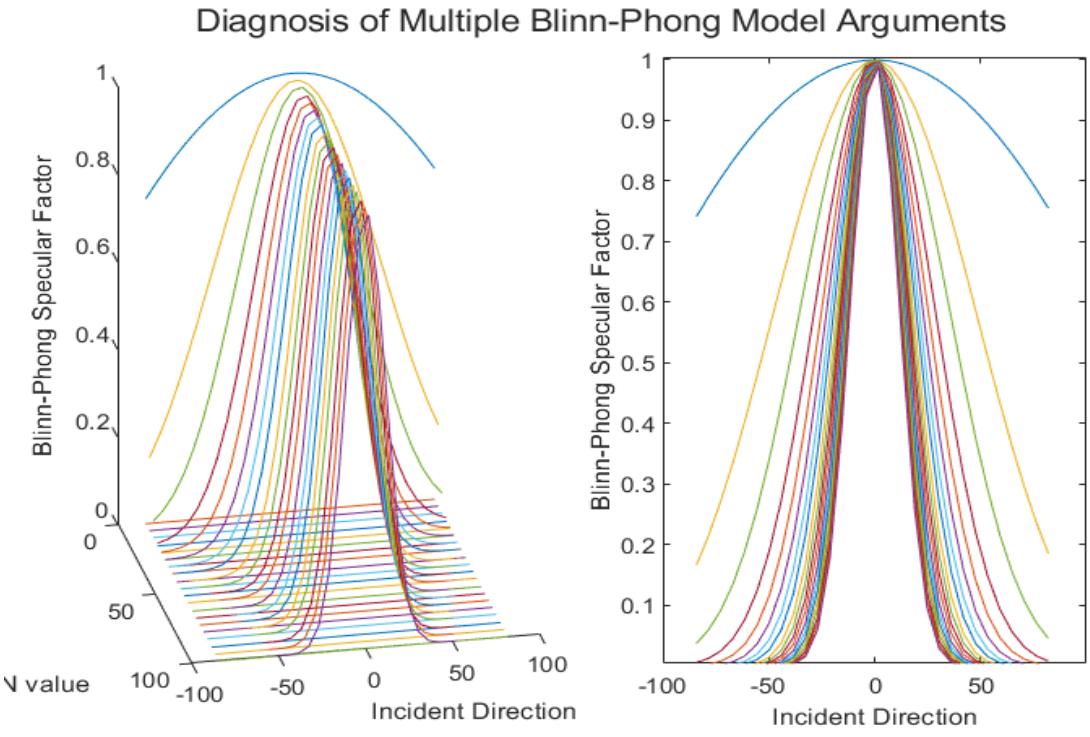


Figure 5.17: A diagnosis of Blinn-Phong specular term with multiple different n arguments for multiple incident directions

(a) Phong Model

(b) Blinn-Phong Model

Figure 5.18: Comparison between Blinn-Phong model and Phong Model. Only direct illumination has been considered with one light source. Multiple n values are showing different results for both models. The shininess values considered were 1, 50, and 100 from right to left respectively.

5.2.3 Lafortune Model

Based on the Phong model of reflectance, *Lafortune* improved upon it and introduced a normalization term, thus, achieving the energy conservation constraint. In other words, the added term allow the Phong model to be usable in multi-sampling techniques, hence, making it plausible. The *Lafortune* model is given in Equation

5.22, while diagnosed and visualized in Figures 5.19 and 5.20.

$$brdf_s(\vec{v},\vec{l}) = ks.\frac{n+2}{2\pi}(\vec{n}\cdot\vec{h})^n \tag{5.22}$$

where

- ks defines the reflection color (Specular)
- n phong model parameter determining the shininess of the reflection
- h the half vector: $normalize(\vec{v}+\vec{l})$

Overall, this model exhibits the same behavior as other distribution functions seen in Figure 4.5. However, in this model, fine-tuning the shininess parameter is a hard task since it does not resemble any physical meaning.

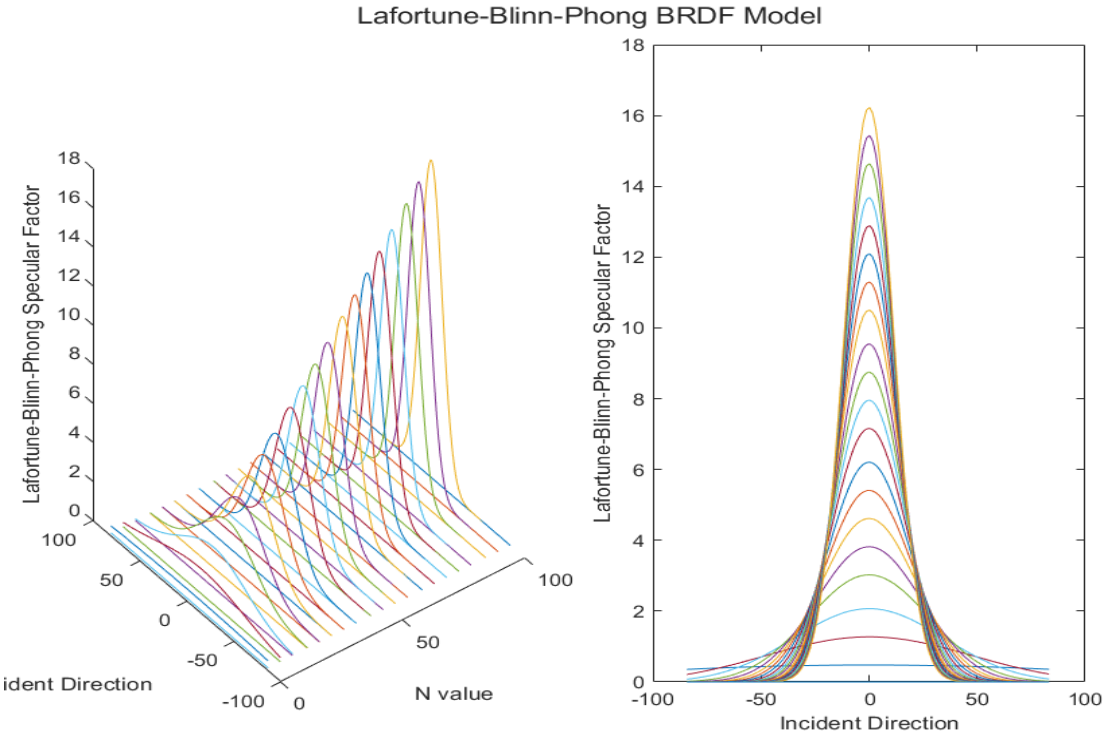


Figure 5.19: A representation of the Lafortune normalization term for the Blinn-Phong model. Applying the correcting term allows the BRDF to be used in multi-sampling applications.

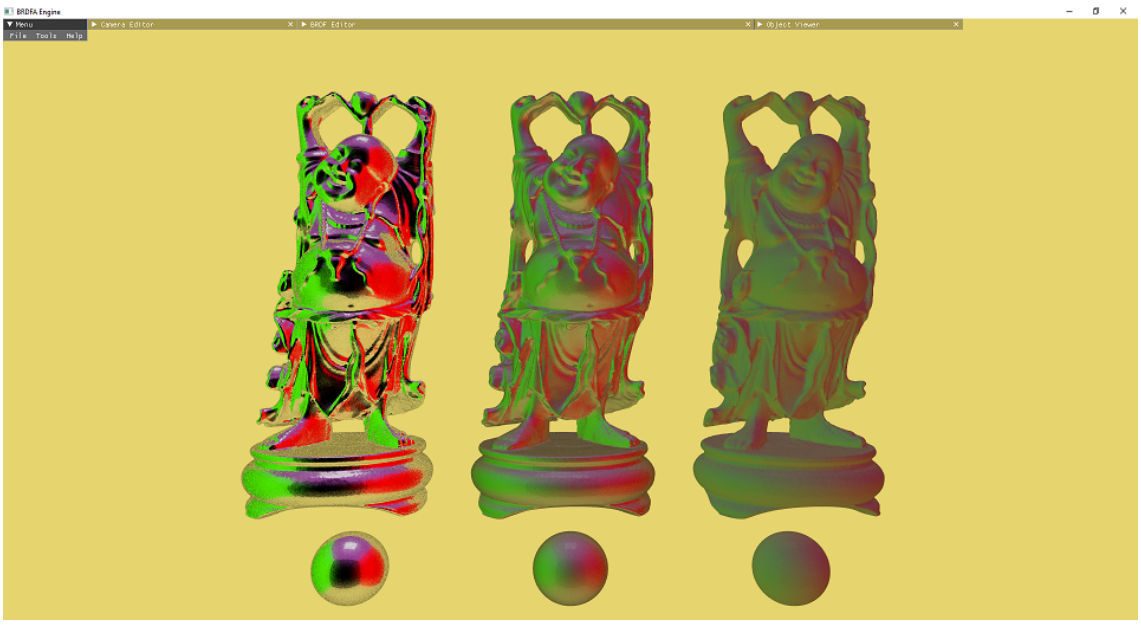


Figure 5.20: A visualization of the Lafortune model with different shininess parameters. The shininess values from right to left are 1, 10, and 100. The used environment map is the Cornell box.

Figure 5.21: An illustration of multiple images rendered with Lafortune model with shininess settings of 1, 10 , 70, and 100, and sample count of 30.

5.2.4 Ward Model

Ward in 1994 [51] defined an empirical specular BRDF seen in Equation 5.23, which is derived from the Gaussian distribution and handles anisotropic and isotropic surfaces, yet, holds physically adequate results. Although *Ward* presented an approx-

imation to the model (*Ward* 1994, Equation 5b [51]), *Walter* [52] showed that the vector form seen in Equation 5.24 is exact and cheaper than the provided estimation.

$$brdf_s(\theta_v, \theta_i, \theta_h, \phi_h) = \frac{\rho_s}{4.\pi.\alpha.\beta.\sqrt{\cos\theta_v \cos\theta_i}}.e^{-\tan^2\theta_h(\frac{\cos^2\phi_h}{\alpha^2}+\frac{\sin^2\phi_h}{\beta^2})} \tag{5.23}$$

Similarly written in vector arithmetic:

$$brdf_s(\vec{v}, \vec{l}) = \frac{\rho_s}{4.\pi.\alpha.\beta.\sqrt{(\vec{l}\cdot\vec{n})(\vec{v}\cdot\vec{n})}}.e^{-(\frac{(\vec{h}_x/\alpha)^2+(\vec{h}_y/\beta)^2}{(\vec{h}\cdot\vec{n})^2}} \tag{5.24}$$

where

- ρ_s used changes the magnitude of the Lobe
- α, β used to alter the width of the Lobe.
- θ is the angle of vectors from normal
- ϕ is the angle of vectors from the surface tangent

Since the model can handle anisotropic and isotropic surfaces, it requires two separate parameters to change the lobe width. The surface becomes isotropic when α equals β; otherwise, it is considered anisotropic. Furthermore, the ρ adjusts the length of the lobe.

The parameters α & β implies the roughness of the material toward one of the surface axes. Increasing either of the values changes the orientation of the irregularities of the surface. Thus, increasing the α & β introduces more masking and shadowing. The ρ parameter defines the surface absorption of the incoming energy.

Finally, Figures 5.23 and 5.24 demonstrate the behavior of the Ward model for a variety of parameters. Furthermore, anisotropic and isotropic surfaces are rendered with different values in Figures 5.25, 5.27, 5.22, and 5.26.

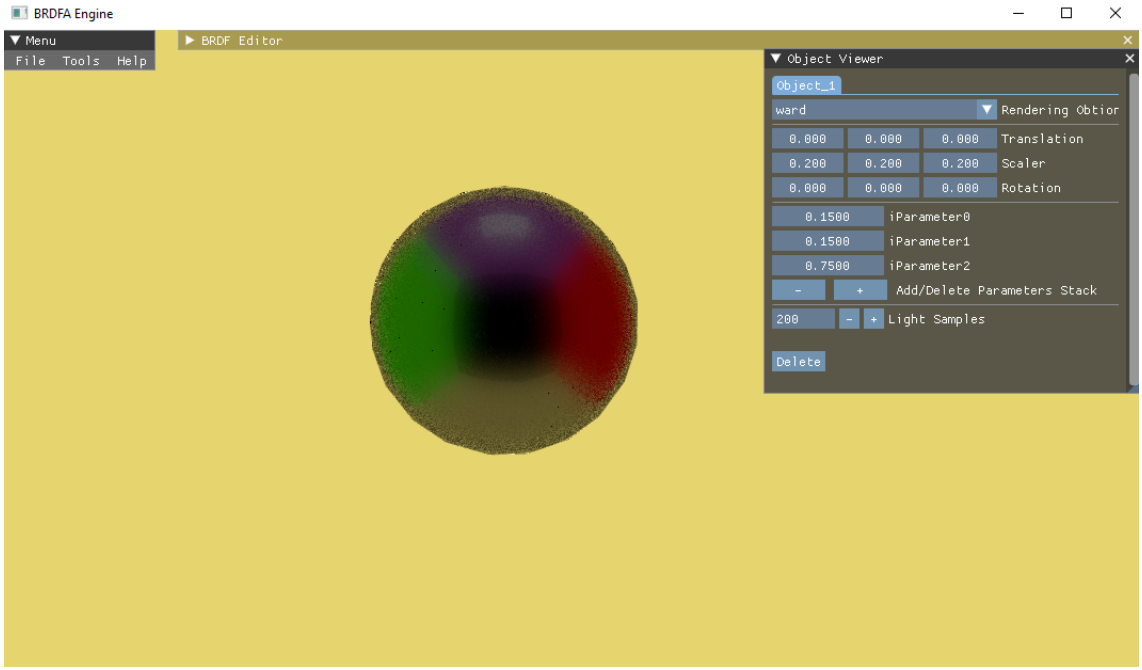


Figure 5.22: The image is a mimic of the presented image in the *Notes on the Ward BRDF* by *Walter* [52], in which the settings are set to $\alpha = \beta = 0.15$ & $\rho = 0.75$.

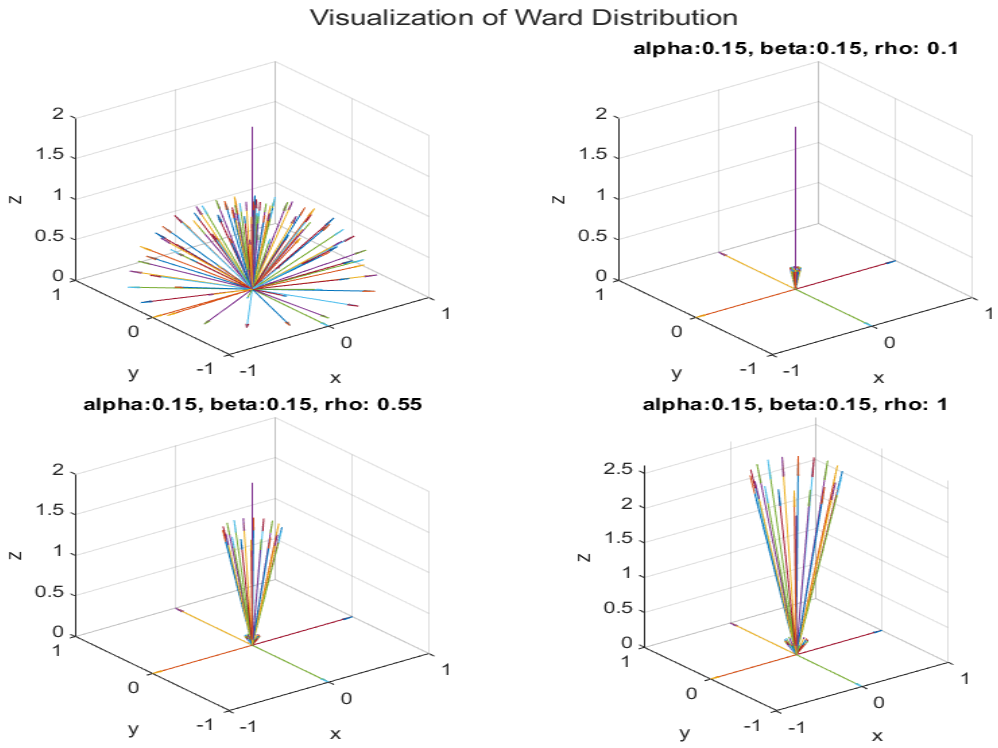


Figure 5.23: Visualization of the vectors distributions before and after using ward model with different magnitude values (ρ spelled as rho), and fixed α & β values at $\alpha = \beta = 0.15$. ρ resembles the strength of the reflection, while α & β resembles the roughness along one of the surface axes. The view vector is aligning with the surface normal, and distribution weights are clamped to 5 ($D \leq 5$)

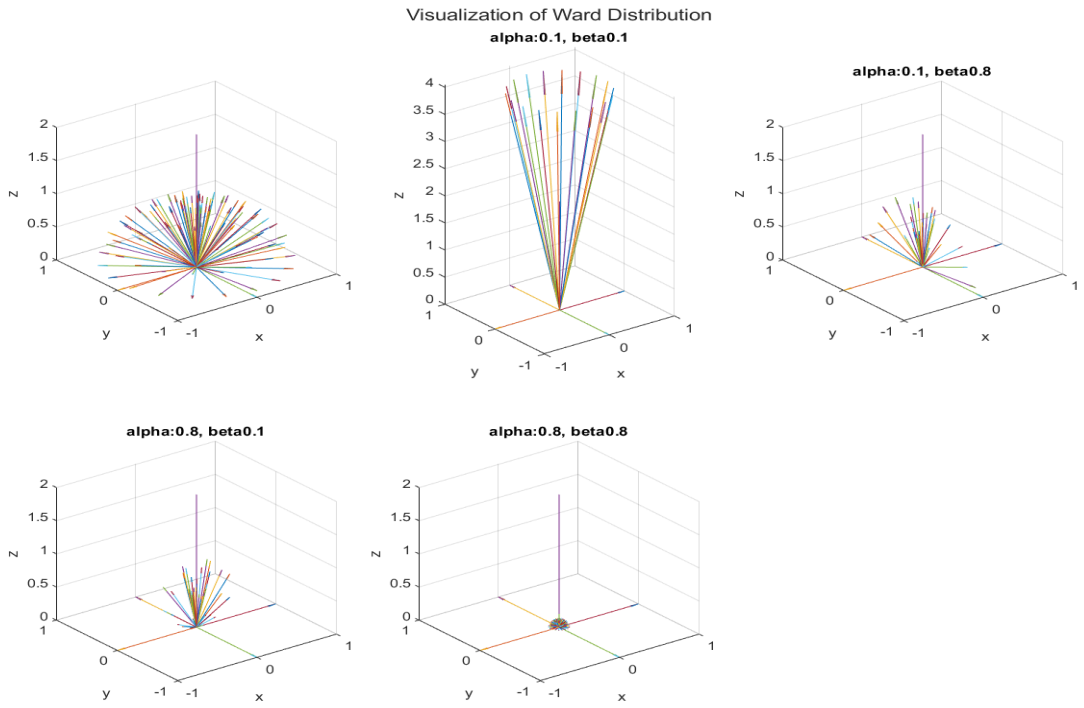


Figure 5.24: Visualization of the vectors distributions before and after using ward model with different α & β values , and fixed (ρ spelled as rho) value at $\rho = 1.0$. ρ resembles the strength of the reflection, while α & β resembles the roughness along one of the surface axes. The view vector is aligning with the surface normal, and the distribution weights are clamped to 5 ($D \leq 5$)

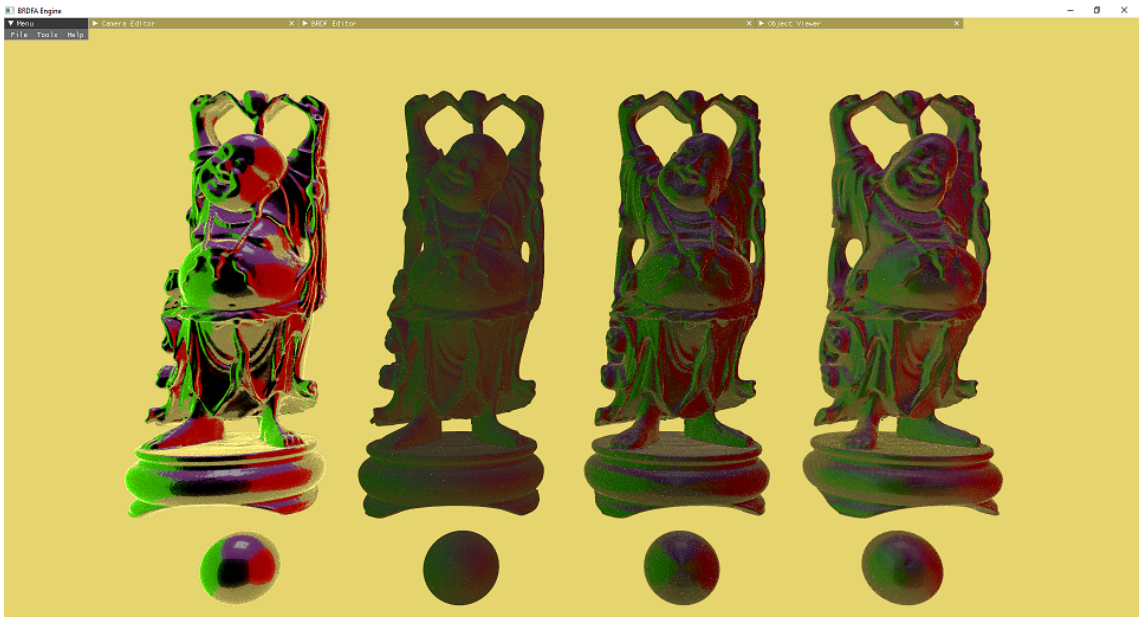


Figure 5.25: A visualization of the Ward model on multiple objects inside the Cornell box (Used as a skymap) with different settings. The settings are $\alpha = \{0.1, 0.9\}$ & $\beta = \{0.1, 0.9\}$ while $\rho = 1.0$. The two objects on the right describe the case of when $\alpha \neq \beta$ (Anisotropic), while the ones on the left resemble the settings $\alpha = \beta$ (Isotropic).

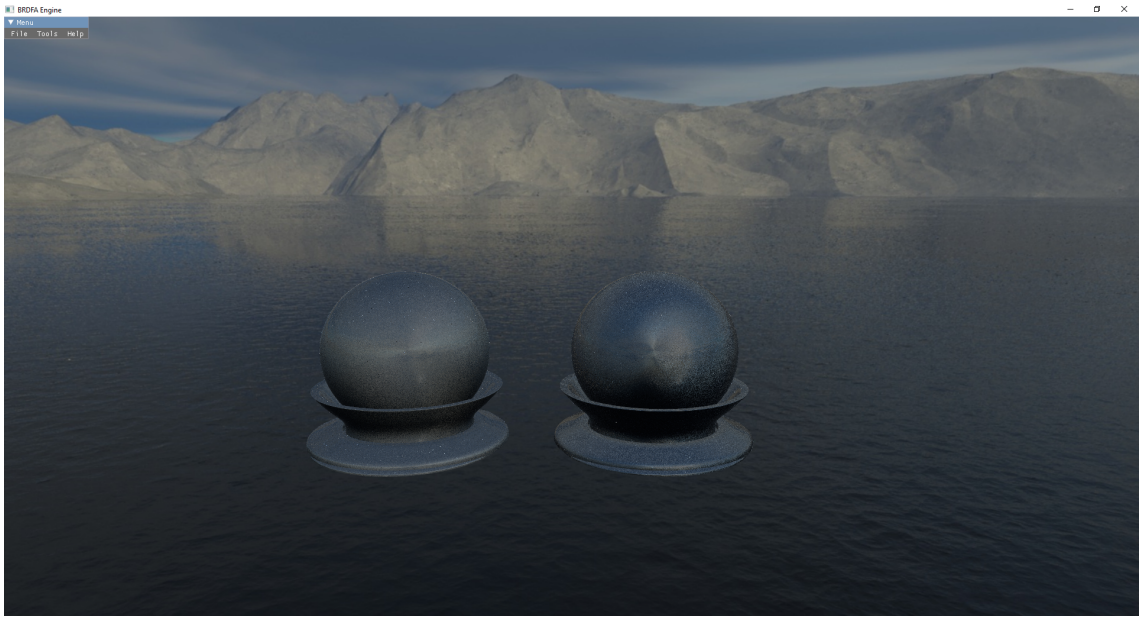


Figure 5.26: A visualization of the Ward model on two anisotropic spheres rendered outdoor. The settings are: $\rho = 1, \alpha != \beta, \alpha, \beta \in [0.1, 1.0]$.

Figure 5.27: A visualization of the Ward model on multiple isotropic objects rendered outdoor with different settings. The settings are $\rho = 1, \alpha = \beta = [0.1, 0.3, 0.6, 1.0]$ from right to left respectively.

Chapter 6

Conclusion

In this thesis, we present a comprehensive survey of several empirical and physically based.*Bidirectional Reflactance Distribution Functions*, as well as, providing numerous illustrations and examples. This study was inspired by the work of *Rosana Montes and Carlos Ureña* [3] that briefly overview *BRDFs* models, yet, they are not illustrated nor compared. Furthermore, a utilized engine for rendering, displaying, and investigating *Bidirectional Reflactance Distribution Functions* is shipped with the thesis and discussed thoroughly.

A brief overview of the physics of light and material is given, which describes the foundation and fundamentals of the physically based *Bidirectional Reflactance Distribution Functions*. Furthermore, an in-depth explanation, with visualizations of the distinct components, of the physically-based *BRDFs* is provided.

The discussed *BRDFs* are elaborated in two sections (Micro-facet based and empirical based), focusing on separating the description of the empirical from the physically-based models. Thus, delivering a cogent experience to the reader.

We provide an engine developed solely to analyze *BRDFs* by harnessing the GPU capabilities through the *VulkanAPI*. The software has a set of tools that the user may utilize, ranging from viewing the rendering state to saving a screenshot of the scene. Furthermore, the ***BRDF Editor*** is the most important tool among all, and it is explained in details. It allows the users to add, edit, save, and test existing *BRDFs* implementations.

Further improvements can be done and more tools may be added to the engine, which may provide an in-depth analysis in real-time. Moreover, improving the sampling technique, by introducing *Importance Sampling* for instance, is a valid

consideration.

Finally, Figure 6.1 summarizes the studied *specular BRDFs*. However, this study does not cover the entirety of existent *BRDFs*, yet, it provides a demonstration for many.

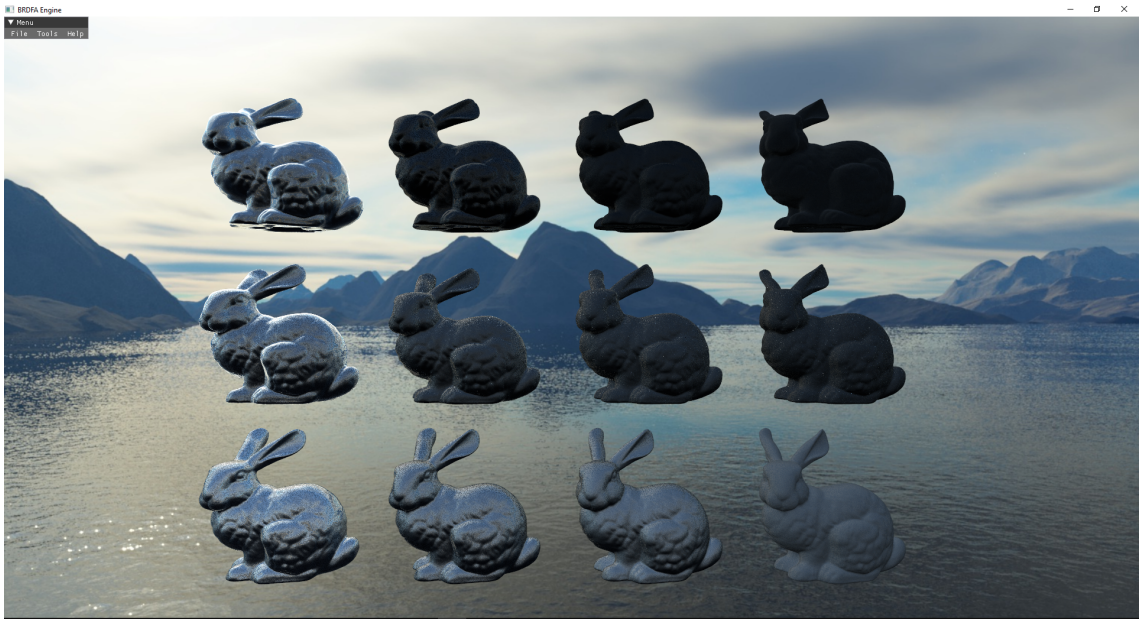


Figure 6.1: An image encapsulating the studied *specular BRDFs* in this thesis. Each row is a different BRDF. First row is rendered with the *Cook-Torrance* model, the second on is with the *Ward* model, and the last one is with *Lafortune* model.

Bibliography

[1] FLORICA MOLDOVEANU CRISTIAN LAMBRU ANCA MORAR. "Comparative Analysis of Real-Time Global Illumination Techniques in Current Game Engines". In: (2021). DOI: 10.1109/ACCESS.2021.3109663. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9527241.

[2] James F. Blinn and Martin E. Newell. "Texture and Reflection in Computer Generated Images". In: *Commun. ACM* 19.10 (1976), 542–547. ISSN: 0001-0782. DOI: 10.1145/360349.360353. URL: https://doi.org/10.1145/360349.360353.

[3] Rosana Montes and Carlos Ureña. "An Overview of BRDF Models". In: (2012). URL: https://digibug.ugr.es/bitstream/handle/10481/19751/rmontes_LSI-2012-001TR.pdf.

[4] Mohammed Ghaith Jassem Al-Mahdawi. *BRDFA Engine*. 2022. URL: https://github.com/Mohido/BRDFA_Engine.

[5] James T. Kajiya. "The Rendering Equation". In: *SIGGRAPH Comput. Graph.* 20.4 (1986), 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. URL: https://doi.org/10.1145/15886.15902.

[6] Ned Greene. *Environment Mapping and Other Applications of World Projections*. 1986. DOI: 10.1109/MCG.1986.276658.

[7] Rio de Janeiro. *Differential Geometry of Curves and Surfaces*. URL: http://www2.ing.unipi.it/griff/files/dC.pdf.

[8] F. James. "A review of pseudorandom number generators". In: *Computer Physics Communications* 60.3 (1990), pp. 329–344. ISSN: 0010-4655. DOI: https://doi.org/10.1016/0010-4655(90)90032-V. URL: https://www.sciencedirect.com/science/article/pii/001046559090032V.

[9] Ph.D. K Hong. *Uniform Distribution of Points on the Surface of a Sphere*. URL: https://www.bogotobogo.com/Algorithms/uniform_distribution_sphere.php.

[10] Eric W. Weisstein. *Cross Product*. URL: https://mathworld.wolfram.com/CrossProduct.html.

[11] *Orthogonal Matrix*. URL: https://en.wikipedia.org/wiki/Orthogonal_matrix.

[12] Georg Rainer Hofmann. "Who Invented Ray Tracing?" In: *Vis. Comput.* 6.3 (1990), 120–124. ISSN: 0178-2789. DOI: 10.1007/BF01911003. URL: https://doi.org/10.1007/BF01911003.

[13] Arthur Appel. "Some Techniques for Shading Machine Renderings of Solids". In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. AFIPS '68 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1968, 37–45. ISBN: 9781450378970. DOI: 10.1145/1468075.1468082. URL: https://doi.org/10.1145/1468075.1468082.

[14] Donald P. Greenberg Cindy M. Goral Kenneth E. Torrance and Bennett Battaile. *Modeling the Interaction of Light Between Diffuse Surfaces*. 1984. URL: http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S07/lectures/goral.pdf.

[15] Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. "Neural radiosity". In: *ACM Transactions on Graphics* 40.6 (2021), pp. 1–11. DOI: 10.1145/3478513.3480569. URL: https://doi.org/10.1145%2F3478513.3480569.

[16] Károly Zsolnai-Fehér, Peter Wonka, and Michael Wimmer. "Gaussian material synthesis". In: *ACM Trans. Graph.* 37.4 (2018), 76:1–76:14. DOI: 10.1145/3197517.3201307. URL: https://doi.org/10.1145/3197517.3201307.

[17] Teun Kloek, Tuen Kloek, and Herman Van Dijk. "Bayesian Estimates of Equation System Parameters: An Application of Integration by Monte Carlo". In: *Econometrica* 46 (Feb. 1978), pp. 1–19. DOI: 10.2307/1913641.

[18] *Importance Sampling*. URL: https://en.wikipedia.org/wiki/Importance_sampling.

[19] *Introduction to Waves*. URL: https://www.nde-ed.org/Physics/Waves/introwaves.xhtml.

[20] *Electromagnetic Radiation*. URL: https://en.wikipedia.org/wiki/Electromagnetic_radiation.

[21] John Roche. "Introducing electric fields". In: *Physics Education* 51.5 (2016), p. 055005. DOI: 10.1088/0031-9120/51/5/055005. URL: https://doi.org/10.1088/0031-9120/51/5/055005.

[22] *Magnetic Fields*. URL: https://en.wikipedia.org/wiki/Magnetic_field.

[23] *Right Hand Rule*. URL: https://www.pasco.com/products/guides/right-hand-rule.

[24] NASA Science. *Visible Light*. 2016. URL: http://science.nasa.gov/ems/09_visiblelight.

[25] Prof. Glenn Stark. *Unpolarized Light*. URL: https://www.britannica.com/science/light/Unpolarized-light.

[26] Prof. Glenn Stark. *Geometrical Optics Light as Rays*. URL: https://www.britannica.com/science/light/Light-rays.

[27] Roland Winston, Juan C. Miñano, and Pablo Benítez. "2 - SOME BASIC IDEAS IN GEOMETRICAL OPTICS". In: *Nonimaging Optics*. Ed. by Roland Winston, Juan C. Miñano, and Pablo Benítez. Burlington: Academic Press, 2005, pp. 7–23. ISBN: 978-0-12-759751-5. DOI: https://doi.org/10.1016/B978-012759751-5/50002-6. URL: https://www.sciencedirect.com/science/article/pii/B9780127597515500026.

[28] Prof. Glenn Stark. *Geometrical Optics Light as Rays*. URL: https://www.britannica.com/science/light/Characteristics-of-waves.

[29] Tomas Akenine-Mller, Eric Haines, and Naty Hoffman. "Ch. 9, Physics of Light". In: *Real-Time Rendering, Fourth Edition*. 4th. Burlington: A. K. Peters, Ltd., 2018, pp. 293–305. ISBN: 0134997832.

[30] Prof. Glenn Stark. *Quantum Theory of Light*. URL: https://www.britannica.com/science/light/Quantum-theory-of-light.

[31] Tomas Akenine-Mller, Eric Haines, and Naty Hoffman. *Real-Time Rendering, Fourth Edition*. 4th. USA: A. K. Peters, Ltd., 2018. ISBN: 0134997832.

[32] Clara Asmail. "Bidirectional Scattering Distribution Function (BSDF): A Systematized Bibliography". In: *National Institute of Standards and Technology Journal of Research* 96 (Mar. 1991), pp. 215–223. DOI: 10.6028/jres.096.010.

[33] Jim Blinn. "Models of Light Reflection for Computer Synthesized Pictures". In: (1977). URL: https://www.microsoft.com/en-us/research/publication/models-of-light-reflection-for-computer-synthesized-pictures/.

[34] K. E. Torrance and E. M. Sparrow. "Theory for Off-Specular Reflection From Roughened Surfaces*". In: *J. Opt. Soc. Am.* 57.9 (1967), pp. 1105–1114. DOI: 10.1364/JOSA.57.001105. URL: http://opg.optica.org/abstract.cfm?URI=josa-57-9-1105.

[35] R. L. Cook and K. E. Torrance. "A Reflectance Model for Computer Graphics". In: *ACM Trans. Graph.* 1.1 (1982), 7–24. ISSN: 0730-0301. DOI: 10.1145/357290.357293. URL: https://doi.org/10.1145/357290.357293.

[36] C. Huygens. *Treatise on Light*. Echo Library, 2007. ISBN: 9781406813753. URL: https://books.google.hu/books?id=Xq5EjjlgV6gC.

[37] *Huygens–Fresnel principle*. URL: https://en.wikipedia.org/wiki/Huygens%E2%80%93Fresnel_principle.

[38] Naty Hoffman. *Physics and Math of Shading*. 2016. URL: https://www.youtube.com/watch?v=j-A0mwsJRmk&ab_channel=ACMSIGGRAPH.

[39] Eric Heitz. "Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs". In: *Journal of Computer Graphics Techniques (JCGT)* 3.2 (2014), pp. 48–107. ISSN: 2331-7418. URL: http://jcgt.org/published/0003/02/03/.

[40] Christopher M. Schlick. "An Inexpensive BRDF Model for Physically-based Rendering". In: *Computer Graphics Forum* 13 (1994).

[41] Ole Gulbrandsen. "Artist Friendly Metallic Fresnel". In: (Dec. 2014).

[42] Sébastien Lagarde. In: (2013). URL: https://seblagarde.wordpress.com/2013/04/29/memo-on-fresnel-equations/.

[43] Brian Karis from Epic Games. *Specular BRDF Reference*. 2013. URL: http://graphicrants.blogspot.com/2013/08/specular-brdf-reference.html.

[44] James Frederick Blinn. "Computer Display of Curved Surfaces." AAI7909865. PhD thesis. 1978.

[45] H. Davies. "The reflection of electromagnetic waves from a rough surface". In: 1954.

[46] H. E. Bennett and J. O. Porteus. "Relation Between Surface Roughness and Specular Reflectance at Normal Incidence". In: *J. Opt. Soc. Am.* 51.2 (1961), pp. 123–129. DOI: 10.1364/JOSA.51.000123. URL: http://opg.optica.org/abstract.cfm?URI=josa-51-2-123.

[47] Michael Oren and Shree K Nayar. "Generalization of Lambert's reflectance model". In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 1994, pp. 239–246.

[48] Wenzel Jakob Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory To Implementation: Ch.8.4 Microfacet Models*. URL: https://www.pbr-book.org/3ed-2018/Reflection_Models/Microfacet_Models.

[49] Eric P. Lafortune and Yves D. Willems. *Using the modified Phong reflectance model for physically based rendering*. Report CW. Celestijnenlaan 200A, 3001 Heverlee, Belgium, 1994.

[50] Robert R. Lewis. "Making Shaders More Physically Plausible". In: *Computer Graphics Forum* (1994). ISSN: 1467-8659. DOI: 10.1111/1467-8659.1320109.

[51] Gregory J. Ward. "Measuring and Modeling Anisotropic Reflection". In: *SIGGRAPH Comput. Graph.* 26.2 (1992), 265–272. ISSN: 0097-8930. DOI: 10.1145/142920.134078. URL: https://doi.org/10.1145/142920.134078.

[52] Bruce Walter. "Notes on the Ward BRDF". In: (Jan. 2005). URL: https://www.graphics.cornell.edu/~bjw/wardnotes.pdf.