MASTER THESIS GAME AND MEDIA TECHNOLOGY

Rendering Iridescent Objects In Real-time

Author:
Michel Sussenbach

Supervisor:
Dr. Robby T. Tan

THESIS NUMBER:
ICA-3856038

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

at the

DEPARTMENT OF INFORMATION AND COMPUTING SCIENCES
FACULTY OF SCIENCE,
UTRECHT UNIVERSITY

Universiteit Utrecht

November 6, 2013

Abstract

Current thin film interference rendering methods focus either solely on accuracy [8], performance [11] or render only a specific iridescent object [10]. We propose a method for rendering iridescent objects in real-time that strives to be as physically accurate as possible while providing intuitive parameters, generality and great performance. It is implemented as a shader running on the GPU in our custom physically-based OpenGL renderer. Our method is based on a multi-layered thin film interference algorithm for specular reflection and specular transmission [9], though we have made algorithmic changes in order to speed up the simulation process. We have also extended the robustness of the algorithm by considering the spatially varying nature of the thin film thickness across the surface of objects. We call this spatially varying thickness or SVT for short.

We validated the original algorithm proposed in [9] quantitatively using spectral reflectance graph comparisons and qualitatively using color data of soap bubble thin films to ensure we would base our method on a physically plausible algorithm. The performance of our method was measured in terms of rendering time in milliseconds. The measurements were done on a mid-range desktop PC. We have measured a significant increase in performance in our method compared to the original method. We have achieved real-time rendering performance for our iridescence algorithm and it is thus suitable for games and other interactive applications.

The remaining problem is that our method does not support semiconductor/metallic thin films (complex index of refractions) due to the algorithmic changes and assumptions we have made in order to speed up the original algorithm. Further research should be done to include (wavelength dependent) complex index of refractions in an efficient manner to preserve the real-time performance of the algorithm.

Contents

Chapter 1

Introduction

Iridescence is a unique view dependent light phenomenon that can produce a wide range of colors on the surface of objects. Iridescence can occur due to either diffraction or thin film interference. Amplitudes of light waves can cancel each other out or intensify each other. This is called the interference of light waves and is the reason for the wide spectrum of colors that are produced through diffraction and thin film interference. The colors of a soap bubble for instance are caused by thin film interference. In this thesis we will strictly focus on thin film interference.

Thin films have many different applications ranging from optical filters to protection from hazardous substances. For example, ultra thin OLED displays are made by placing a series of organic thin films between two conductors. Thin films are also used as a coating on sunglasses to reduce glare. Furthermore, water resistant thin film coatings exist to protect electronics against water damage. We will use the thin film interference concept to render iridescent objects like soap bubbles.

Physically based rendering algorithms are becoming more prevalent in games and interactive applications. However, existing thin film interference rendering methods focus either only on simulation accuracy or performance when in fact the focus should be on both. In this thesis we propose a robust physically-based thin film interference method that not only gives accurate results but also renders in real-time. Our method is suitable to use for rendering in interactive applications and games.

We have made the following contributions:

1. A real-time thin film interference algorithm implemented as a shader on the GPU.
2. An inexpensive and intuitive way to simulate spatially varying thin film thickness.

The following constraints apply to our method:

1. Materials used in the thin film system are strictly dielectric.
2. Materials used in the thin film system are wavelength invariant.

Chapter 2

Background Knowledge

2.1 Related Work

Methods for rendering different optical phenomena due to thin films or layered objects have been developed since 1990. Multiple film rendering models have been developed throughout these years:

1. Single-layer Film Primary Reflection and Refraction (SFPR).
2. Single-layer Film Multiple Reflection and Refraction (SFMR).
3. Multilayer Film Primary Reflection and Refraction (MFPR).
4. Multilayer Film Multiple Reflection and Refraction (MFMR).

Figure 2.1 shows these models schematically. The SFPR model was used to render single-layer

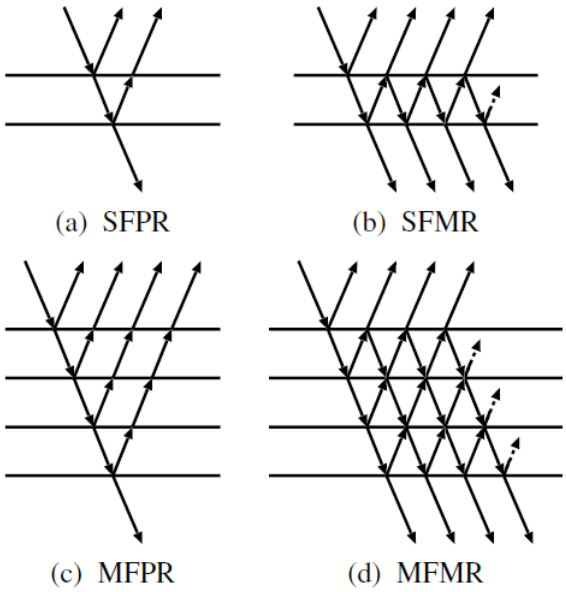

Figure 2.1: Different types of film rendering models.

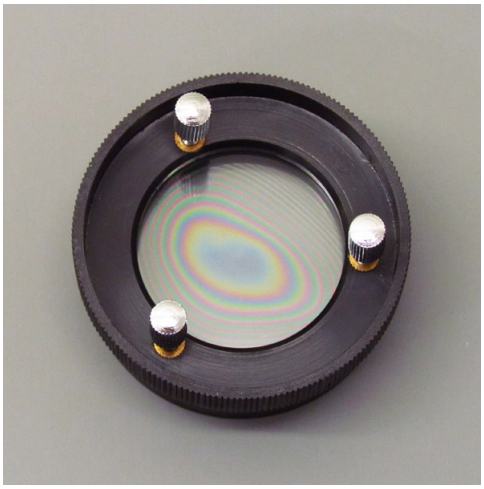

Figure 2.2: Real life example of Newton's ring.

thin film interference effects of Newton's ring [18]. Figure 2.2 shows a real life example of Newton's ring. The SFMR model for rendering single-layered thin film systems is more accurate compared to the SFPR model since it takes into account multiple reflection and refraction of light inside the film. This model was used in [6] to render iridescent paints and plastic. In 1997, a MFPR model was proposed to render the unique appearance of pearls [13]. A generalized MFMR thin film interference model was proposed in [9] which is based on wave optics. It takes into account multiple reflections and refractions of light inside the thin film layers and also considers the interference of light in every layer. The model can render both dielectric and semiconductor thin films due to the use of complex refractive indices. However, it could only be used to strictly render smooth surfaces coated with thin films. The authors improved the model in [8] where they introduced an illumination model to render multi-layered thin films coated on rough surfaces. In recent years, several different GPU implementations for rendering thin film interference effects were proposed. In [4] such an implementation is proposed, though it only renders single-layered thin film systems. Furthermore, their method is not physically-based since they use the Phong reflectance model as their illumination model. An implementation for rendering soap bubbles in real time on the GPU was proposed in [10]. However, this implementation focuses solely on rendering (single-layered) soap bubbles and thus the method lacks generality. A real-time multi-layered thin film interference method on the GPU based on [9] was proposed in [11]. However, it does not take into account the transmissivity of light in the simulation and thus cannot be used to render thin film interference effects of (semi) transparent objects like soap bubbles. Furthermore, the GPU is only used for rendering the simulation data; the actual thin film interference simulation is done off-line on the CPU.

2.2 Ray Tracing

Ray tracing is one of the most commonly used techniques in the field of Computer Graphics. It can be used to accurately simulate light transport. The basic ideas of ray tracing will be discussed using the PBRT ray tracer as our example.

2.2.1 PBRT

PBRT stands for "Physically Based Ray Tracer" and is software that can be used to render images using a variety of different rendering techniques. The software strives to be as flexible and extensible as possible and thus was a well suited fit for this project. PBRT is mainly suited for rendering of photo-realistic scenes by accurately simulating the physics of light and its interaction with matter.

Rendering using rays

Simulating the propagation of light rays through a scene efficiently is the main benefit of ray tracing. There are three different ways of tracing rays through a scene:

1. Tracing rays from the light source
2. Tracing rays from the camera
3. Tracing rays from both the light source and the camera

The first case is generally used in the photon mapping algorithm to construct the photon map. The second case is the general case for most scenes using for example a direct lighting or path tracing algorithm. The third case is used in bidirectional path tracing where rays are generated from both camera and light source and their paths are connected somewhere in between. By doing this, the amount of rays necessary to render an image is less when compared to rendering with a standard path tracer; this is assuming they generate equal variance in the output. We will focus on explaining the second case where rays are only shot from the camera, since PBRT does not support bidirectional path tracing by default. In Figure 2.3 the basic idea of ray tracing

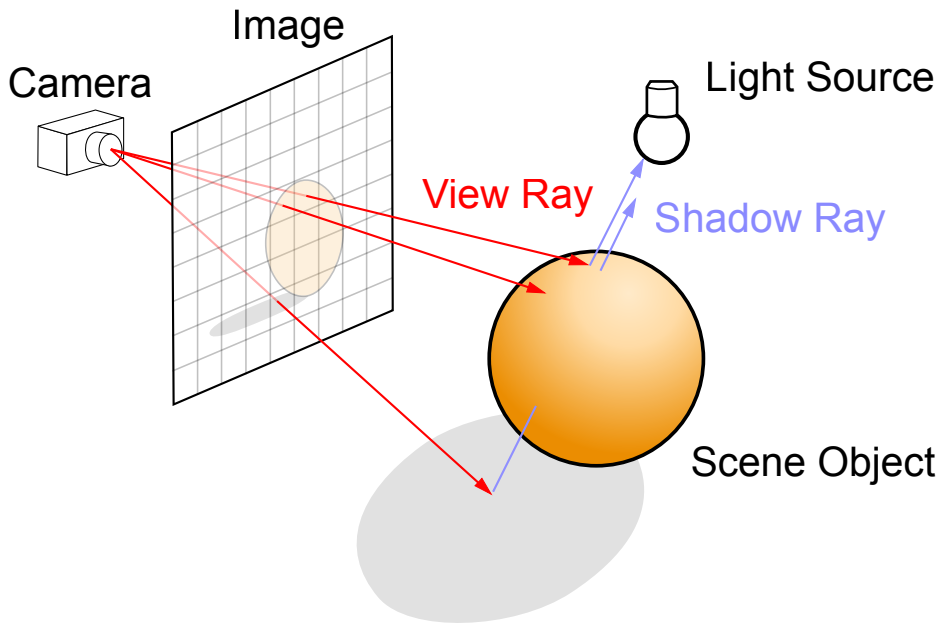

Figure 2.3: Rays propagating through the scene.

is shown. A scene consisting of a camera, object and light source is defined. The camera records the portion of the scene that needs to be rendered to the image. A virtual image is placed at

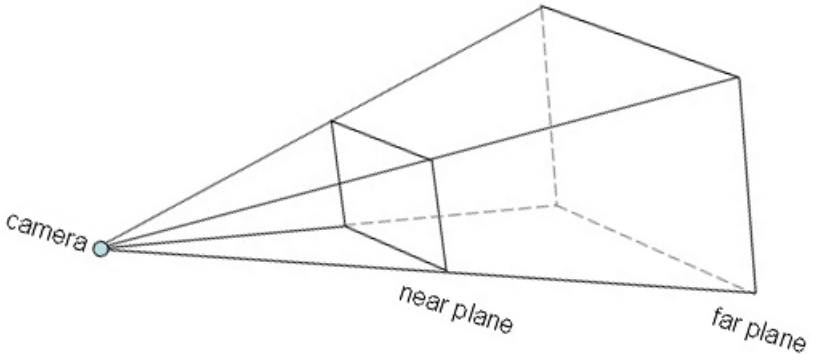

Figure 2.4: The camera near and far planes.

the near plane of the camera viewing frustum (Figure 2.4). Rays are then shot through the pixels of the image into the scene. When a ray is generated, the renderer determines whether an intersection with an object occurs and if so, which object intersects the ray closest to the camera. The object geometry is stored in an acceleration structure (by default a k-d tree) in order to speed up this intersection process. The intersection with an object results in a point to be shaded and some information about the geometry at this point (e.g. the surface normal). At the shading point, a new ray is generated called a shadow ray. The shadow ray starts from the point to be shaded and points towards the light source we are interested in. If the path between the object and the light source is unobstructed, it means the point on the surface can receive energy from the light source and thus the light source contributes to the shading of this point. However if the shadow ray intersects an object, the path is obstructed and the light will not contribute to the shading of the point; meaning the point will most likely be part of the shadow of an object.

Scene file

The scene file is a simple text file with the extension *.pbrt. PBRT defines the scene consisting of the camera, objects, materials and light sources in this scene file. The scene file also describes which renderer should be used, where the image should be rendered to and which integrator will be used to do the lighting simulation. The file consist of key-value pairs that describe all these properties. At run-time PBRT parses the key-value pairs and stores them in the corresponding object. For example, parameters for the camera in the scene file are parsed and then subsequently stored in the camera object. In Appendix A an entire scene file is shown.

Surface Integrators

The surface integrator is an abstraction used in PBRT that is responsible for calculating the scattered radiance from a surface. It is called an integrator because it evaluates the integral light transport equation of Kajiya [12] given by:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i. \tag{2.1}$$

Where, L_o is the outgoing radiance, L_e is the radiance emitted from the surface, L_i is the radiance incident to the surface and f describes the scattering properties of the surface. The

surface integrator is passed the geometry intersection information as an input which contains the location of the point on the surface that will be shaded. Using this information the surface integrator can compute the radiance L_i along the ray which is its output.

2.2.2 Radiometry

Radiometry concerns itself with the study of the transfer of radiant energy, aptly named radiative transfer. Radiometry was not originally based on the physics of light and thus omitted several properties of light including the effect of polarization. Instead, radiometry was a phenomenological study of the scattering behavior of light. It is based on geometric optics, meaning it only takes into account the macroscopic scattering properties of light. Geometric optics suffices for light simulation that describes light interacting with surfaces that are much larger than the wavelength of the light. This is sufficient to accurately model almost any type of surface reflection. The PBRT software has the same constraints and thus assumes a geometric optics model that ignores the wave nature of light. There are four basic radiometric quantities that are important in rendering: radiant flux, irradiance/radiant exitance, intensity and radiance.

Radiant Flux

Radiant flux (Φ) is the total amount of energy that passes through a surface or region of space per unit time. It is most commonly known as power and is measured in joules per second (J/s) or equivalently watts (W). This radiometric quantity is useful to describe the total emission from a light source. Consider a point light source that emits energy that passes through two imaginary spherical surfaces as shown in Figure 2.5. The amount of energy at any one point on the smaller sphere is higher than the amount of energy at any one point on the larger sphere; however the radiant flux is the same for both spheres. Although the amount of energy at any particular point is lower on the larger sphere, it also has a larger area. This means that the energy is simply partitioned over a larger area but the total amount of energy passing through the surface is the same as the energy passing through the smaller sphere.

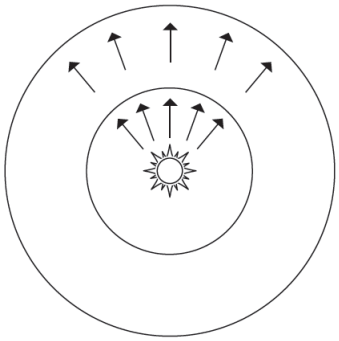

Figure 2.5: Radiant flux Φ passing through a region of space.

Irradiance and Radiant Exitance

Irradiance (E) is the flux per unit area *arriving* at a surface. Radiant Exitance (M) is the flux per unit area *leaving* a surface. These quantities are measured in units of watts per square meter (W/m^2). The irradiance can be expressed as:

$$E = \frac{d\Phi}{dA}, \tag{2.2}$$

where $d\Phi$ and dA are the differential flux and differential area respectively. They are both differential because both the distribution of flux and the surface area are not necessarily constant functions across the entire surface. To illustrate the irradiance quantity intuitively, we will determine the irradiance at the two imaginary spheres in Figure 2.5 as an example. In this case the irradiance of the smaller sphere is higher than that of the larger sphere since the area is smaller. For this specific case the irradiance can be expressed by:

$$E = \frac{\Phi}{4\pi r_2}, \tag{2.3}$$

where r is the radius of the sphere and Φ is the constant flux distribution of the point light. Equation 2.3 shows that if r is large, the resulting irradiance E will be small.

Intensity

The intensity (I) is defined as the flux per solid angle:

$$I = \frac{d\Phi}{d\omega}, \tag{2.4}$$

where $d\omega$ is the differential solid angle. A solid angle is simply the extension of an angle into 3D. In 2D an angle can be defined in a plane which is called a planar angle. A planar angle is the total angle subtended by some object with respect to a position. This is equivalent to considering a unit circle, projecting the object onto this unit circle and measuring the arc length of the circle covered by the projected object (Figure 2.6). The arc length s is equivalent to the

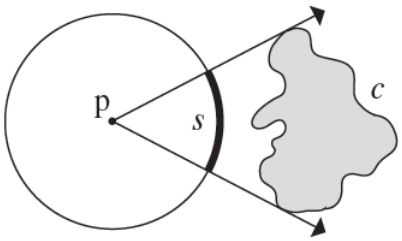

Figure 2.6: Projection of object onto the unit circle, resulting in the arc length s.

angle θ and is measured in terms of radians. A solid angle is simply the extension of this concept in three dimensions. In 3D a unit sphere is used and instead of measuring the arc length of the projected object, the projected area is measured (Figure 2.7).

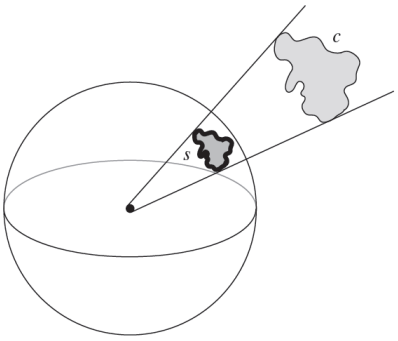

Figure 2.7: Projection of area onto the unit sphere, resulting in the solid angle s.

Radiance

Radiance (L) is the flux per unit area per unit solid angle:

$$L = \frac{d\Phi}{d\omega dA^{\perp}}. \tag{2.5}$$

Where, $d\Phi$ is the differential flux, $d\omega$ is the differential solid angle and $dA^{\perp}$ is the differential surface area projected on an imaginary surface perpendicular to $d\omega$ (see Figure 2.8). Radiance is an important quantity for a number of reasons. If we know the radiance then we can mathematically derive all other previously mentioned quantities. Another reason is that radiance remains constant along rays propagating through free space and thus is a desirable quantity to use in ray tracing.

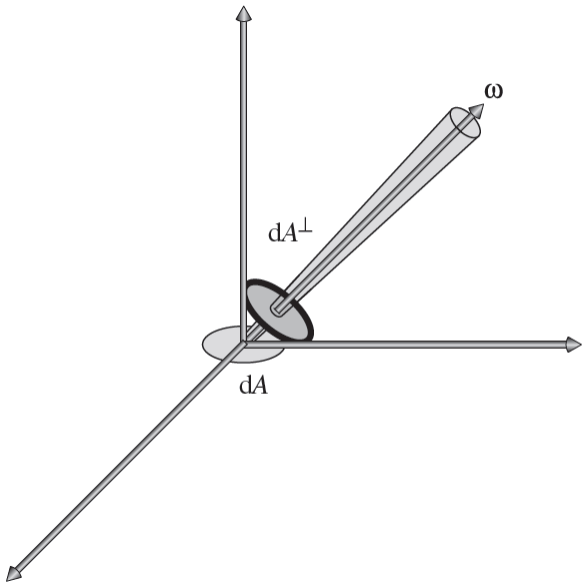

Figure 2.8: The definition of radiance.

2.2.3 BRDF

The BRDF or Bidirectional Reflectance Distribution Function describes the radiance leaving a surface towards an observer given some incident radiance arriving at a surface (Equation 2.7 and Figure 2.9).

$$dE(p, \omega_i) = L_i(p, \omega_i) \cos \theta_i d\omega_i. \tag{2.6}$$

$$f_r(p, \omega_o, \omega_i) = \frac{dL_o(p, \omega_o)}{dE(p, \omega_i)}. \tag{2.7}$$

The BRDF function $f_r(p, \omega_o, \omega_i)$ describes how much of the incident light along ω_i is scattered

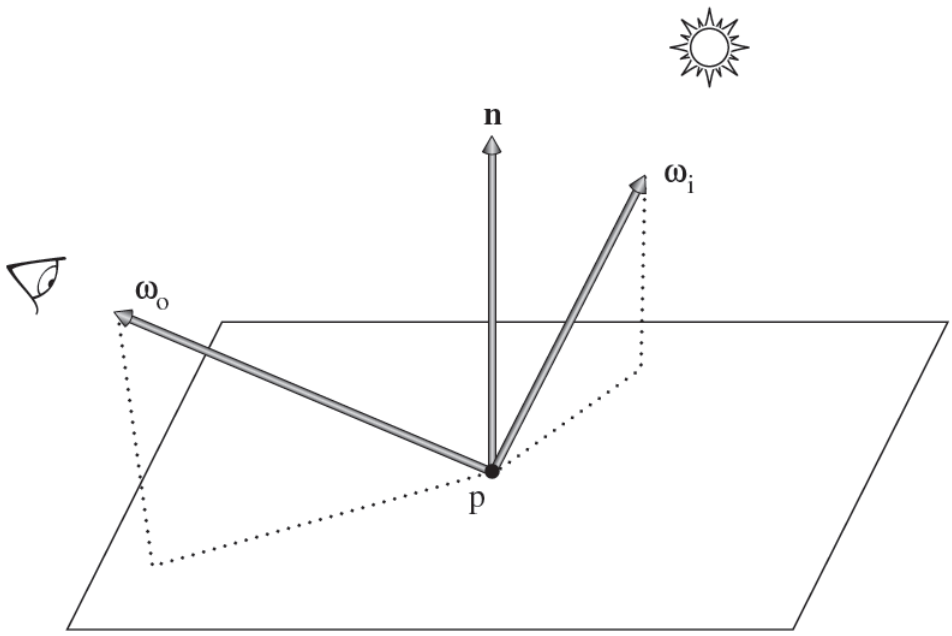

Figure 2.9: Light scattering in direction of observer.

from the surface towards ω_o. An important property is that the reflected differential radiance of a BRDF is proportional to the differential irradiance:

$$dL_o(p, \omega_o) \propto dE(p, \omega_i). \tag{2.8}$$

Physically based BRDFs have two other important properties. The first property is reciprocity:

$$f_r(p, \omega_i, \omega_o) = f_r(p, \omega_o, \omega_i), \tag{2.9}$$

meaning all pairs of directions ω_i and ω_o can be swapped and the resulting value of the BRDF will be the same; hence the term bidirectional. The second property is energy conservation:

$$\int_{H^2(n)} f_r(p, \omega_o, \omega') \cos \theta' d\omega' \leq 1, \tag{2.10}$$

meaning the total amount of energy of the reflected light should always be strictly less than or equal to the energy of the incident light. This energy conservation constraint applies to the whole hemisphere.

2.3 Wave Optics

Wave optics, also sometimes called physical optics, defers from geometric optics in that it is the branch of optics which concerns itself with wave based phenomena. Some of these phenomena include polarization, diffraction and interference.

2.3.1 Defining the wave

A wave is a disturbance or oscillation in a certain direction over a certain time. Waves are commonly defined mathematically as a sine function. A one-dimensional traveling wave is mathematically defined as:

$$D(x,t) = A\sin(\phi), \tag{2.11}$$

where $D(x,t)$ is the displacement of the wave at time t and spatial coordinate x, A is the amplitude of the wave and ϕ is the phase of the wave. Increasing the amplitude value means that the wave will be stretched out over the D-axis (see Figure 2.10). As a result, the minimum and maximum values of displacement increase. In order to understand what the phase of a wave

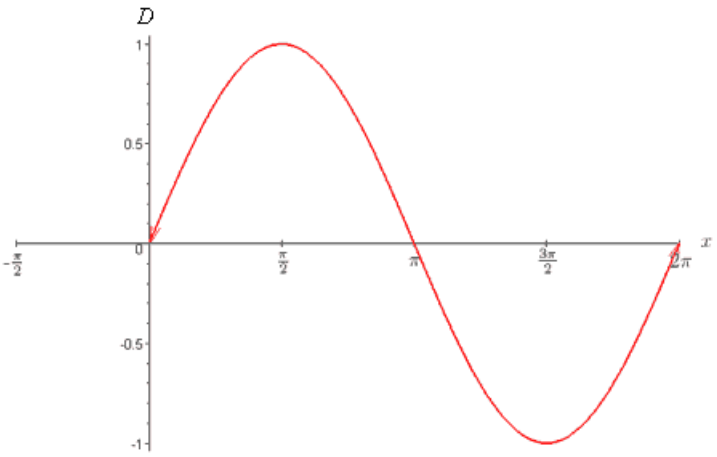

Figure 2.10: A graph of a sine function at t = 0.

does we can decompose ϕ into its individual variables:

$$\phi = kx - \omega t + \phi_0, \tag{2.12}$$

where k is the wave number, ω is the angular frequency and ϕ_0 is the initial phase constant. The wave number is the spatial frequency of the wave. It describes the number of waves that exist over a specified distance. In this case the wave number is angular, meaning it describes the number of waves that exist over a distance of 2π. We can also say that the wave number is the amount of cycles a wave with wavelength λ goes through over a distance of 2π. The wave number equation follows:

$$k = \frac{2\pi}{\lambda}. \tag{2.13}$$

If we take Figure 2.10 as an example, we can see that the wavelength is 2π thus the wave number for this particular wave will be $k = 1$; which is equivalent to one wave cycle over a distance of 2π radians. A sine function can not only be represented as a graph, we can also represent it as a

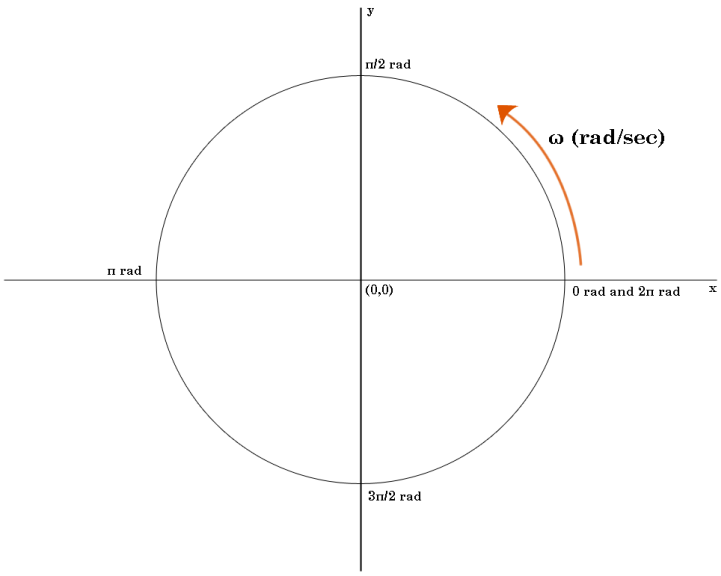

Figure 2.11: A unit circle can represent a sine function.

unit circle. The angular frequency ω. also sometimes called the angular speed, is the amount of radians we rotate around the unit circle per second. In other words, it determines the rotation speed around the unit circle. The angular frequency is related to the frequency and period of a wave by the following equation:

$$\omega = 2\pi f = \frac{2\pi}{T}. \tag{2.14}$$

The initial phase constant ϕ_0 is the initial phase offset of the wave (Figure 2.12). In other words, it controls where in the cycle the wave will initially start. In the case of the wave in Figure 2.10,

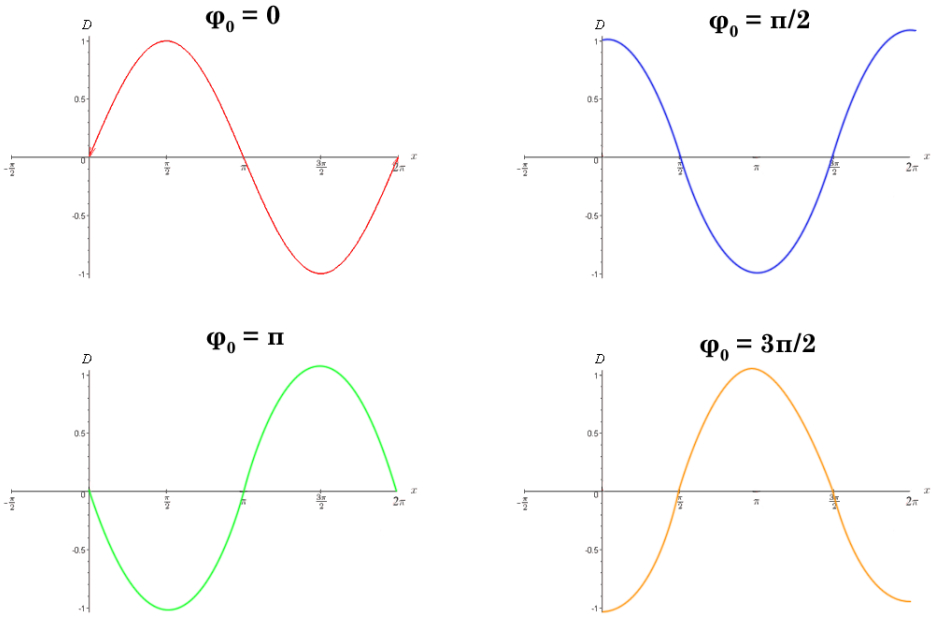

Figure 2.12: Influence of the initial phase constant on the wave.

the initial phase constant is $\phi_0 = 0$.

2.3.2 Light as a wave

Light is electromagnetic radiation for which the range of wavelengths from 380 nanometers to 740 nanometers can be perceived by the human eye; this is also called the visible light. A light wave is composed out of two components: an electric field and a magnetic field; both are perpendicular to the propagation direction and have a wavelength of λ. We perceive color

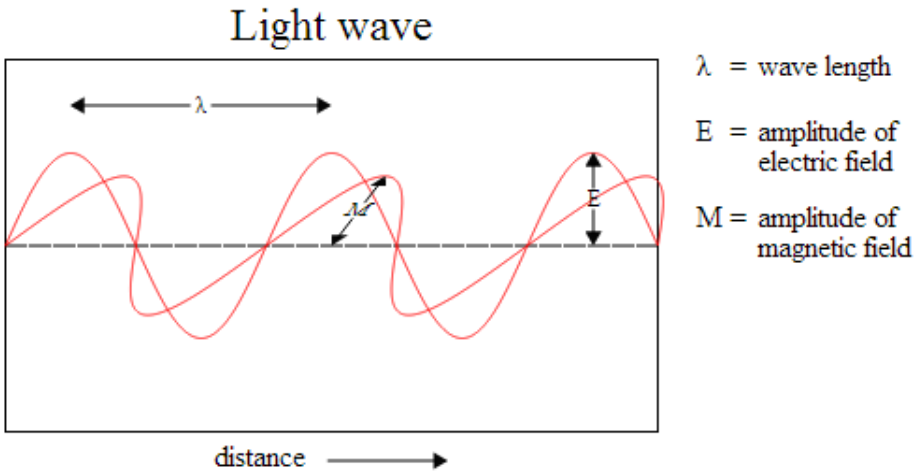

Figure 2.13: A propagating light wave.

due to the light reflecting off and transmitting through a surface towards our eyes. When light transmits into a medium with a different index of refraction n, the light ray will bend due to the index of refraction mismatch between the media. When the light wave crosses the boundary of two media, the wavelength of the light changes but the frequency of the wave remains the same. This is due to the propagation speed of the wave being different depending on the medium. The speed of the wave is dependent upon the index of refraction of the medium:

$$v = \frac{c}{n}, \tag{2.15}$$

where c is the speed of light in vacuum and n is the refractive index of the propagation medium. The speed of the wave changes, but so does its wavelength. Equation 2.16 shows how we can calculate the wavelength inside of a medium:

$$\lambda_{med} = \frac{\lambda_{vac}}{n}, \tag{2.16}$$

where λ_{med} is the wavelength inside of the medium and λ_{vac} is the wavelength in vacuum. We can model a traveling light wave in 3D space as a plane wave with the following equation:

$$D(\mathbf{r}, t) = A\sin(\mathbf{k} \cdot \mathbf{r} - \omega t + \phi_0), \tag{2.17}$$

where $\mathbf{r}$ is the position vector specifying a position in 3d space and $\mathbf{k}$ is the wave vector. The wave vector has a magnitude equal to the wave number and its direction can be defined as the direction of the traveling wave. We can also define a plane wave in the form of a complex number:

$$U(\mathbf{r}, t) = Ae^{i(\mathbf{k}\cdot\mathbf{r} - \omega t + \phi_0)}, \tag{2.18}$$

this is called a complex plane wave. This form allows for easier addition and subtraction rules as opposed to the trigonometric form of the plane wave.

2.3.3 Polarization

Light waves emanating from a light source are unpolarized, meaning they oscillate at right angles to the direction of propagation. We can create light waves however that oscillate in a specific direction or plane, this is called polarization. So called birefringent materials have an index of refraction that depends on the polarization and propagation direction of the light. Polarizing filters can be used on cameras to increase the contrast of the image and reduce the amount of reflection from non-metal objects. Thus polarization is an important property of light to consider. There are three kinds of polarization types:

- Linear Polarization
- Circular Polarization
- Elliptical Polarization

When we discuss polarization we only look at the x and y components of the electric field, because the magnetic field is equivalent.

Linear Polarization

Suppose we have an electric field split up into its x and y component. If the two components are in phase and we take the vector sum, we will get a vector for the electric field pointing in a single direction. If we look at the electric field vector over time in Figure 2.14 we can see it traces out a single line on the blue plane. This is the oscillation direction of the wave. The oscillation direction is dependent on the relative amplitudes of the two polarization components. In Figure

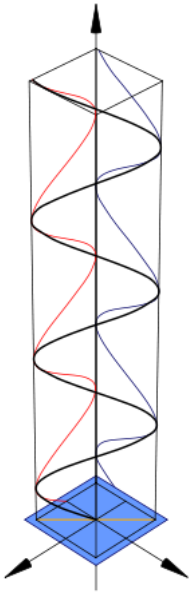

Figure 2.14: A linearly polarized wave.

2.15 a real world example of using a linear polarization filter is shown.

Figure 2.15: Left: unpolarized. Right: linearly polarized.

Circular Polarization

Suppose we have the two components of the electric field with exactly the same amplitude and the phase difference between the components is $\frac{\pi}{2}$ or 90 degrees. Meaning either the x component is $\frac{\pi}{2}$ ahead in its cycle compared to the y component or vice versa. The resulting electric field vector traces out a circle over time as shown in Figure 2.16. Depending upon which

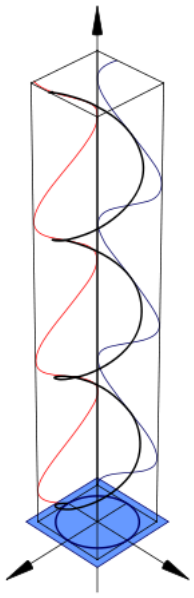

Figure 2.16: A circularly polarized wave.

component is $\frac{\pi}{2}$ ahead in its cycle, the field will rotate in either clockwise or counterclockwise directions. These are also called right-hand circular polarization and left-hand circular polarization respectively. In Figure 2.17 a real world example of using a circular polarization filter is

shown.

Figure 2.17: Left: unpolarized. Right: circularly polarized.

Elliptical Polarization

The elliptical polarization is a generalization of the other two types. When the components do not have the exact same amplitude or are not exactly $\frac{\pi}{2}$ out of phase, the electric field vector will trace out an ellipsoid shape (Figure 2.18).

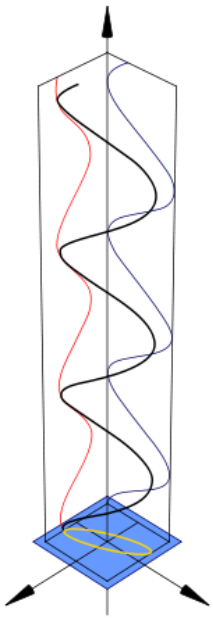

Figure 2.18: A elliptically polarized wave.

2.3.4 Thin Film Interference

Thin film interference is an iridescent phenomenon that utilizes the wave nature of light. A thin film is a very small layer of only a couple of hundred nanometers thick through which the light waves propagate. An example of iridescence through thin film interference is a soap bubble. The thin film in this case is a layer of water with a thickness comparable to or smaller than the wavelength of the light (Figure 2.19). The water film is enclosed by air on both sides

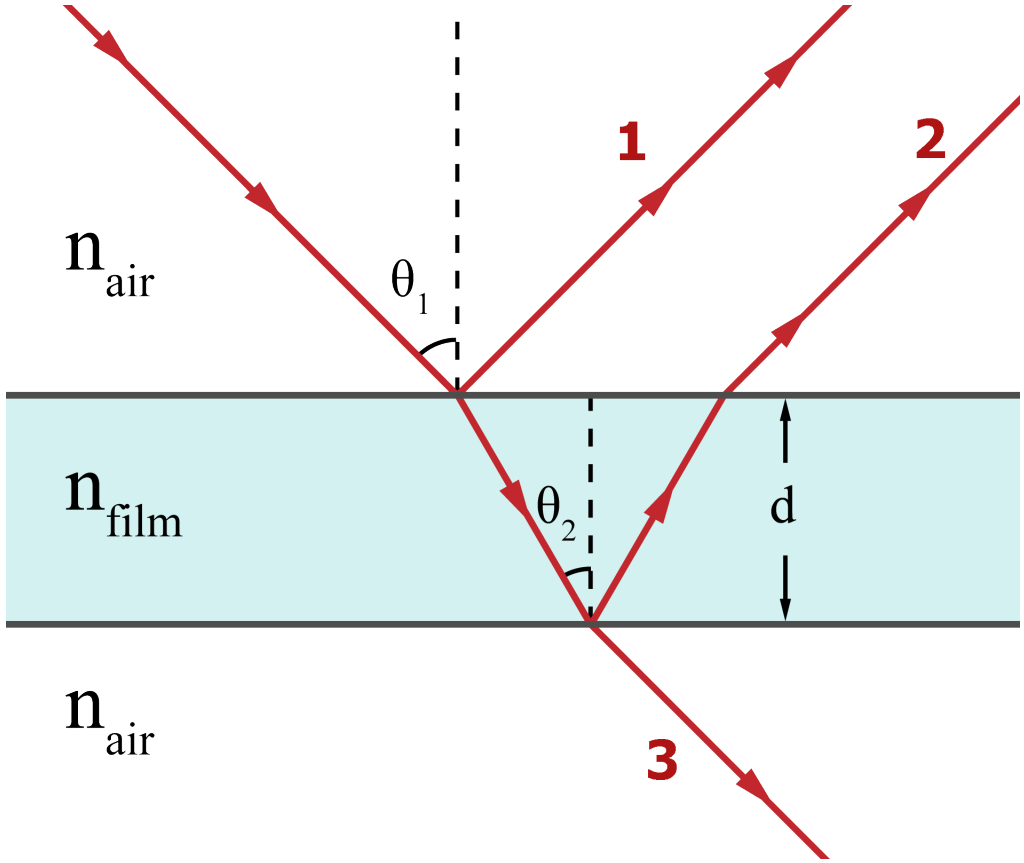

Figure 2.19: The water thin film of a soap bubble.

yielding two interfaces: air to water and water to air. In this case the light can propagate in three different ways:

1. Reflect from the top interface back into the top air medium.
2. Transmit through the top interface, reflecting off the lower interface and transmitting back into the top air medium.
3. Transmit through the top interface and transmit through the lower interface to the bottom air medium.

Light waves one and two in Figure 2.19 will interfere with each other due to the fact that the waves are traveling extremely close to each other (on a nanometer scale). This interference can either be constructive or destructive interference depending on the difference in phase between the waves.

Constructive Interference

Constructive interference occurs when the phase of both light waves is the same. Adding up two waves with the same phase results in a wave that is the sum of both light wave amplitudes (Figure 2.20 and Figure 2.21). This is most commonly called perfect constructive interference or interference maxima.

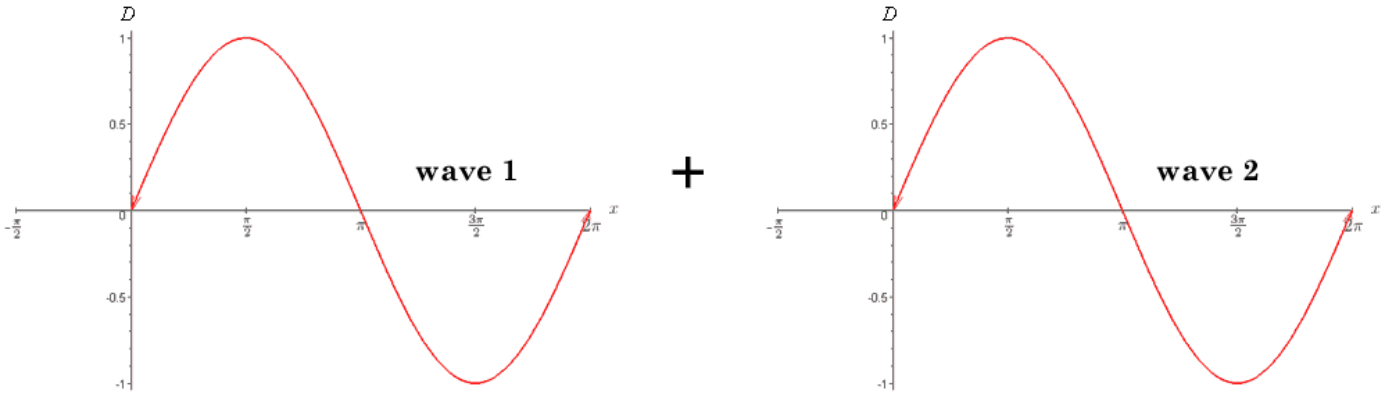

Figure 2.20: Summing up two waves with the same amplitude and phase.

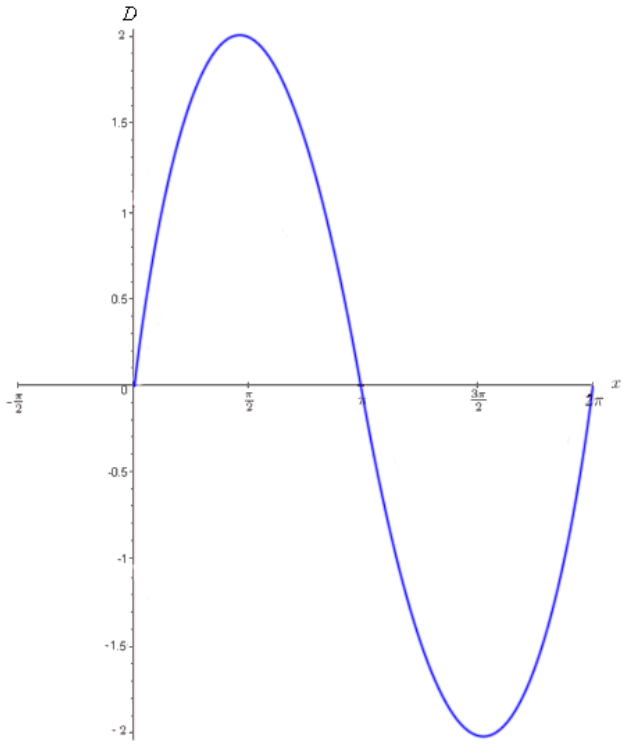

Figure 2.21: Perfect constructive interference by summing up two identical waves.

Destructive Interference

Destructive interference occurs when the light waves are out of phase with each other by π radians (Figure 2.22). Summing up two waves that are perfectly out of phase but have the same amplitude, results in a wave with amplitude zero (Figure 2.23). The amplitudes of the waves cancel each other out due to the difference in phase. This is most commonly called perfect destructive interference or interference minima.

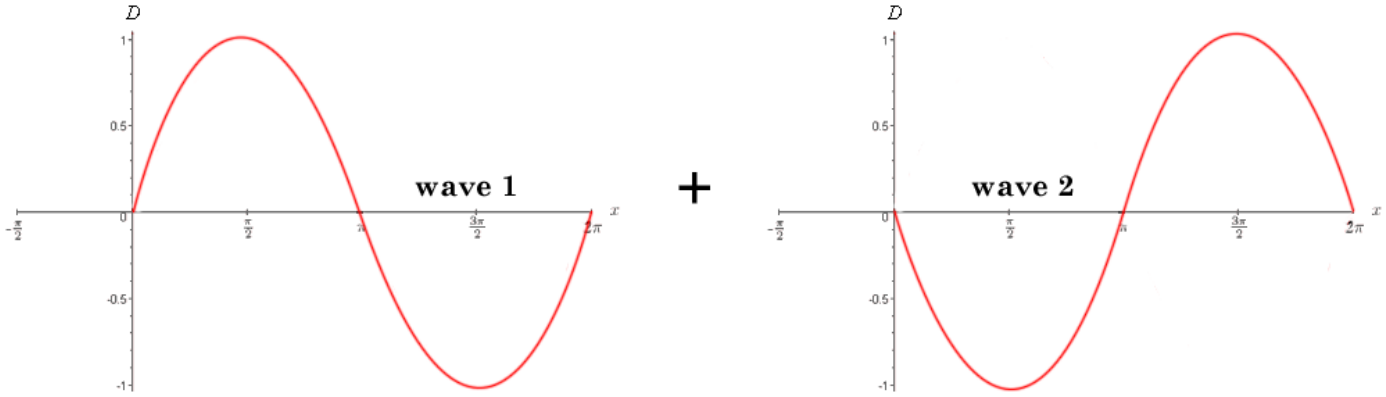

Figure 2.22: Summing up two waves with the same amplitude, but with a phase difference of π.

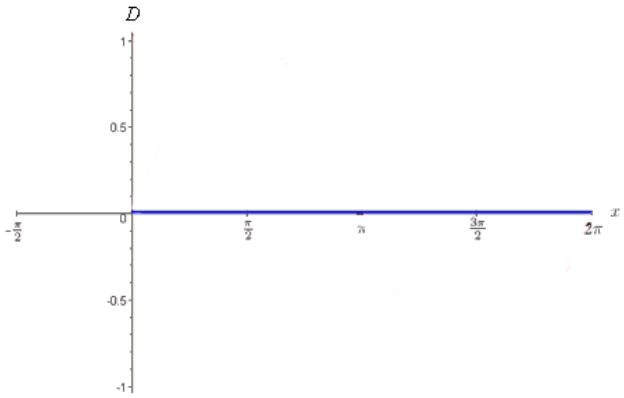

Figure 2.23: Perfect destructive interference.

Phase Difference

In the real world we do not necessarily always have either perfect destructive or perfect constructive interference; in most cases the iridescent object will have partial constructive interference or partial destructive interference. In Figure 2.24 two waves are shown that are slightly out of phase with each other. In this case the interference is neither perfectly constructive nor perfectly destructive. Whether the light waves constructively or destructively interfere with each other is very much dependent on the phase of the waves. More precisely, it is dependent on the relative difference in phase which is called the phase difference:

$$\Delta\phi = \phi_2 - \phi_1. \tag{2.19}$$

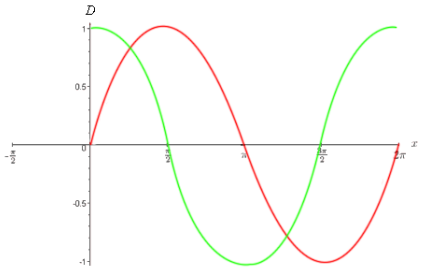

Figure 2.24: Two waves slightly out of phase relative to each other.

Recall the equation for the phase of a wave (Equation 2.12), we can then write Equation 2.19 as:

$$\Delta\phi = (kx_2 - \omega t + \phi_{0,2}) - (kx_1 - \omega t + \phi_{0,1}). \tag{2.20}$$

If it is assumed we are looking at the waves at a particular time, we can factor out the time variables and we can rewrite the equation to be:

$$\Delta\phi = k(x_2 - x_1) + (\phi_{0,2} - \phi_{0,1}). \tag{2.21}$$

Recall the equation for the wave number k (Equation 2.13), we can substitute k for the wave number equation giving us:

$$\Delta\phi = \frac{2\pi}{\lambda}(\Delta x) + (\Delta\phi_0). \tag{2.22}$$

Where Δx is the optical path difference and $\Delta\phi_0$ is the inherent phase difference.

Optical Path Difference

The optical path difference is the relative difference in distance between two waves. For exam-

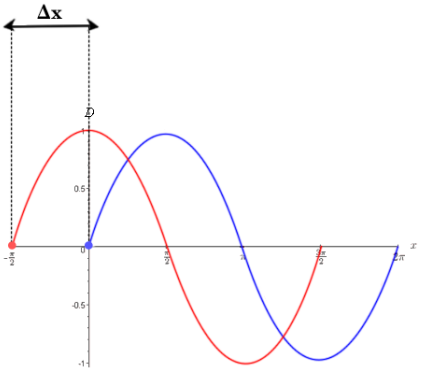

Figure 2.25: Two waves starting at different points on the x-axis.

ple, in Figure 2.25 the waves have different starting positions. Consequently, if we look at the distance traveled at any point x on the x-axis, the red wave will always have traveled a larger distance to get to point x. This means there is a difference in distance traveled and thus it is likely the red wave has taken a path different from the blue wave. In thin film interference, the optical path difference is defined to be the difference in distance traveled between interfering reflected waves. In Figure 2.26 the optical path difference is shown geometrically. This gives us

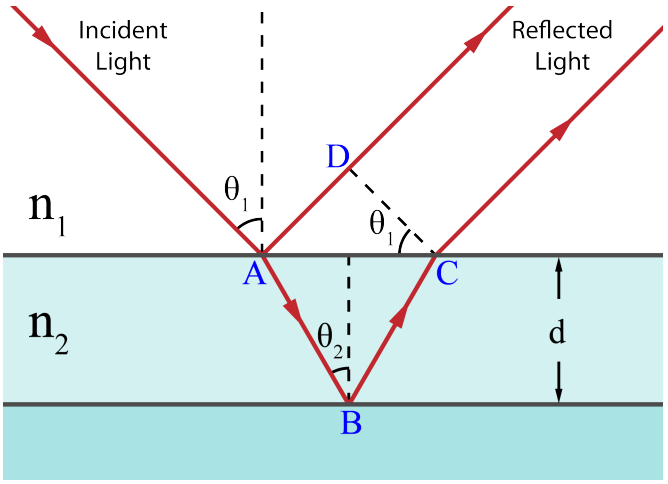

Figure 2.26: Reflected light waves interfering; both waves have taken a different optical path.

the following mathematical definition for the optical path difference:

$$\Delta x = \eta_2(\overline{AB} + \overline{BC}) - \eta_1(\overline{AD}). \tag{2.23}$$

Using trigonometry and Snell's Law this equation can be simplified to:

$$\Delta x = 2d\eta_2 \cos(\theta_2). \tag{2.24}$$

Equation 2.24 can be used to calculate the optical path difference in a thin film interference model.

Inherent Phase Difference

The inherent phase difference is the relative difference in phase between the two waves. For

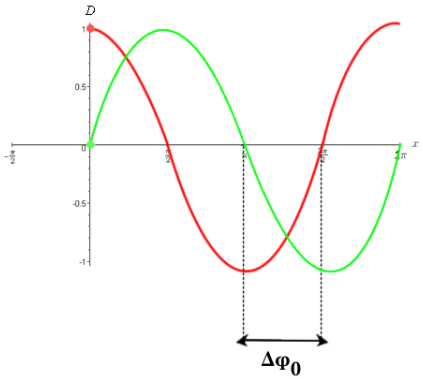

Figure 2.27: Two waves starting at the same point but with differing phase.

example, in Figure 2.27 the waves start at the same point but the initial phase of the waves differ. This gives us the following mathematical definition for the inherent phase difference:

$$\Delta\phi_0 = \phi_{0,2} - \phi_{0,1}. \tag{2.25}$$

Where $\phi_{0,1}$ and $\phi_{0,2}$ are the initial phase constants of the two waves.

2.4 Complex Numbers

The complex number is a mathematical concept that was introduced to solve problems that are not solvable using only real numbers. It is not possible to take the square root of a real-valued negative number. However if the real valued number is expressed as a complex number, we can calculate the square root of a negative number. A complex number is commonly defined as:

$$z = a + bi, \tag{2.26}$$

where a and b are real-valued numbers. The real number a in the complex number z is called the real part of the complex number. The real number b in the complex number z is called the imaginary part of the complex number. A complex number extends the real numbers by adding an imaginary unit i where:

$$i^2 = -1. \tag{2.27}$$

This is a property of the complex number that is used to solve the problem of taking square roots of negative numbers.

2.4.1 Complex Plane

A complex number is expressed as a position vector in a two-dimensional Cartesian coordinate system called the complex plane. The x-axis in this coordinate system is the real axis, while the y-axis is the imaginary axis. A complex number $z = a + bi$ plotted on the complex plane is said to be in Cartesian, or rectangular form. Additionally, another frequently used name for the complex plane is the rectangular coordinate system. In Figure 2.28 a complex number z is plotted in the rectangular coordinate system.

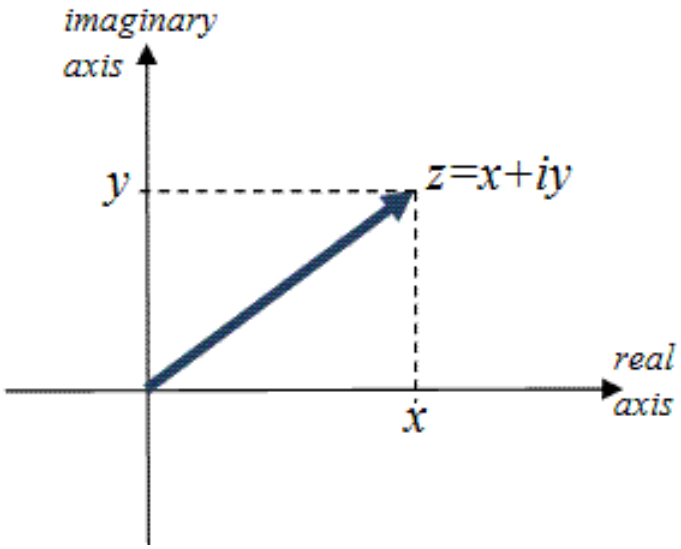

Figure 2.28: Complex number in complex plane

2.4.2 Polar Form

There is another way to define and plot a complex number. It is called the polar form and is plotted in the polar coordinate system. The coordinates in the polar coordinate system are called the modulus r and the argument φ.

Modulus

The modulus r is defined to be the length of the position vector in the complex plane. This can be calculated using Pythagorean theorem:

$$r = |z| = \sqrt{a^2 + b^2}. \tag{2.28}$$

Argument

The argument φ is defined to be the angle of the position vector with the real axis. The equation for the argument is:

$$\varphi = atan2(b, a) = 2 \arctan \frac{\sqrt{a^2 + b^2} - a}{b}. \tag{2.29}$$

The $atan2$ function is defined in most programming languages to find the arctangent of two parameters. By using the $atan2$ function we are calculating the angle between the position vector and the real-axis. For this reason, another name for the argument is the phase angle.

Polar Coordinate System

Given the coordinates r and φ, we can plot them in the polar coordinate system (Figure 2.29). Using these two polar coordinates we can determine the equivalent rectangular coordinates:

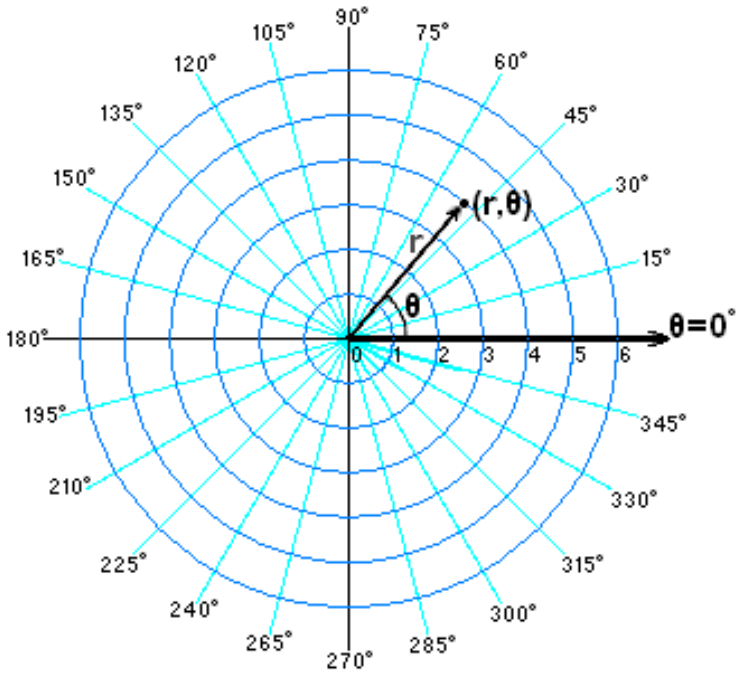

Figure 2.29: Complex number plotted in polar coordinates.

$$a = r\cos(\varphi), \tag{2.30}$$

$$b = r\sin(\varphi). \tag{2.31}$$

This gives us the following equation to convert a complex number from polar coordinates to rectangular coordinates:

$$z = r(\cos(\varphi) + i\sin(\varphi)). \tag{2.32}$$

Euler's Formula

Euler's Formula is a mathematical expression that relates trigonometric functions with the complex exponential function:

$$re^{i\varphi} = r(\cos(\varphi) + i\sin(\varphi)). \tag{2.33}$$

Where e is the base of the natural logarithm and φ and r are the argument and modulus of a complex number respectively. Euler's Formula is most commonly used to convert from polar coordinates to rectangular coordinates and vice versa. This specific formulation also simplifies some of the mathematical operations that can be done on complex numbers.

2.4.3 Complex Operations

A brief overview of the various complex operations is given along with their corresponding formulas.

Addition and Subtraction

Given a complex number z_1 and a complex number z_2 we can add them together:

$$z = z_1 + z_2 = (a_1 + a_2) + i(b_1 + b_2). \tag{2.34}$$

We can define subtraction in a similar way:

$$z = z_1 - z_2 = (a_1 - a_2) + i(b_1 - b_2). \tag{2.35}$$

Multiplication

Given complex numbers z_1 and z_2 complex multiplication can be defined as:

$$z = z_1 z_2 = (a_1 + ib_1)(a_2 + ib_2) = (a_1a_2 - b_1b_2) + i(a_1b_2 + b_1a_2) \tag{2.36}$$

Division

Dividing a complex number z_1 by z_2 can be accomplished by multiplying both numerator and denominator with the complex conjugate of the denominator. In other words, z_1 and z_2 both should be multiplied by the complex conjugate of z_2. The complex conjugate of a complex number z is defined as:

$$\overline{z} = \overline{a + ib} = a - ib. \tag{2.37}$$

The division of two complex numbers can then be defined as:

$$z = \frac{z_1}{z_2} = \frac{(a_1 + b_1 i)\overline{a_2 + b_2 i}}{(a_2 + b_2 i)\overline{a_2 + b_2 i}} = \frac{(a_1 + b_1 i)(a_2 - b_2 i)}{(a_2 + b_2 i)(a_2 - b_2 i)} = \frac{(a_1a_2 + b_1b_2) + i(b_1a_2 - a_1b_2)}{a_2^2 + b_2^2} \tag{2.38}$$

Square Root

Calculating the square root of a complex number is not a trivial task. Luckily, if the complex number is expressed in its polar coordinates r and φ we can use De Moivre's theorem. The theorem is used to raise complex numbers to the nth power:

$$[r(\cos(\varphi) + i\sin(\varphi))]^n = r^n\cos(n\varphi) + i\sin(n\varphi). \tag{2.39}$$

We know the following relationship between exponents and roots:

$$\sqrt[n]{x} = x^{1/n}. \tag{2.40}$$

Using this knowledge we can take the square root of a complex number z:

$$\sqrt{z} = z^{1/2} = [r(\cos(\varphi) + i\sin(\varphi))]^{1/2} = r^{1/2}[\cos(\frac{\varphi + k2\pi}{2}) + i\sin(\frac{\varphi + k2\pi}{2})]. \tag{2.41}$$

Where k is an integer value between zero and $n - 1$ where n is the power. Thus in this case k can be either zero or one since we are using a power of two. The value of k is used to determine which of the n roots is considered the solution. In the case of a square root, we have two solutions; hence k has two possible values. In most applications $k = 0$ will be used, which is called the principal branch.

Chapter 3

Multi-Layered Thin Film Interference

Multi-layered thin film interference simulates the light wave interference inside of a system of N-layered films taking into account multiple reflections in each layer. The main reference papers [8] and [9] by Hirayama et al. use a custom MFMR (Multilayer Film Multiple Reflection and Refraction) model based on wave optics to simulate this. Two different illumination models and algorithms are discussed in the papers. The first illumination model is for specular reflection and transmission of light, which we will call the smooth illumination model. The second illumination model is for specular and diffuse reflection of light, which we will call the LSRS (Locally Smooth Rough Surface) illumination model.

3.1 Smooth Illumination Model

The smooth illumination model for multi-layered thin film interference is proposed in [9]. The paper focuses on calculating the specular reflectance and transmittance of a multi-layered thin film system and proposes an illumination model to render an image from these values. The contribution of the paper is the inclusion of the complex refractive index in the calculations of the multi-layered thin film system. Complex index of refractions can be used to model metallic or semiconductor thin films due to the way it takes into account absorption and scattering inside of the material. This paper focuses strictly on the specular reflection and transmission of light; no diffuse reflection is taken into account.

3.1.1 Recurrence Equation

The total reflectance and transmittance contributions of a multi-layered thin film system are called the *composite reflectance* and *composite transmittance* respectively. In order to calculate the composite reflectance and transmittance, we need to calculate the *reflectivity* and *transmissivity* at every interface. We also need to calculate the absorption of light and the phase difference between waves in each layer (Figure 3.1). The result of the recurrence equations is a *composite reflectivity* and *composite transmissivity*. These are converted into reflectance and transmittance giving us the composite reflectance and composite transmittance as a result.

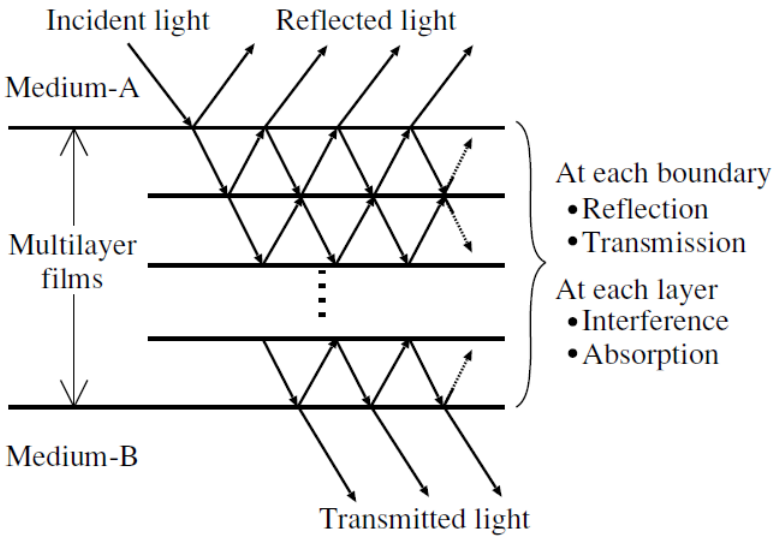

Figure 3.1: The common scenario for multi-layered thin film interference.

Reflectivity and Transmissivity

It is important to note that reflectivity and reflectance or transmissivity and transmittance are not the same thing. Reflectivity describes the ratio of incident electromagnetic wave amplitudes versus reflected electromagnetic wave amplitudes. Similarly, transmissivity describes the ratio of incident electromagnetic wave amplitudes versus transmitted electromagnetic wave amplitudes. Both are dependent on the amplitude of the light wave. In contrast, reflectance and transmittance describe ratios of energies of electromagnetic waves. The distinction between these quantities is important.

Composite Reflectivity and Composite Transmissivity

The recurrence equation is a recursive method to iteratively calculate the composite reflectivity and transmissivity for a N-layered film system. The iteration starts at the bottom layer of the film system (medium B) and ends at the top layer (medium A). At every layer L_j, a composite reflectivity γ_j and composite transmissivity τ_j is calculated using the recurrence equation. It describes the total reflectivity and transmissivity of the thin film system in between layer L_j and layer L_{N+1}. In other words, it describes the combined reflectivity and transmissivity of layer j through $N+1$. In Figure 3.2 the problem of calculating the composite reflectivity and composite transmissivity is shown schematically. Note that in Figure 3.2 the medium B layer (where light transmits) is pictured at the top since the recurrence equation is done in order from the last layer to the first. The reflectivity r_j and transmissivity t_j is evaluated at every interface B_j. Furthermore, inside every layer L_j the phase difference φ_j is calculated. Given the reflectivity, transmissivity and phase difference we can define the recurrence equations at the jth layer as:

$$\gamma_{N-j+1} = \frac{r_j + \gamma_{N-j} e^{2i\varphi_j}}{1 + r_j \gamma_{N-j} e^{2i\varphi_j}}, \tag{3.1}$$

$$\tau_{N-j+1} = \frac{t_j \tau_{N-j} e^{i\varphi_j}}{1 + r_j \gamma_{N-j} e^{2i\varphi_j}}. \tag{3.2}$$

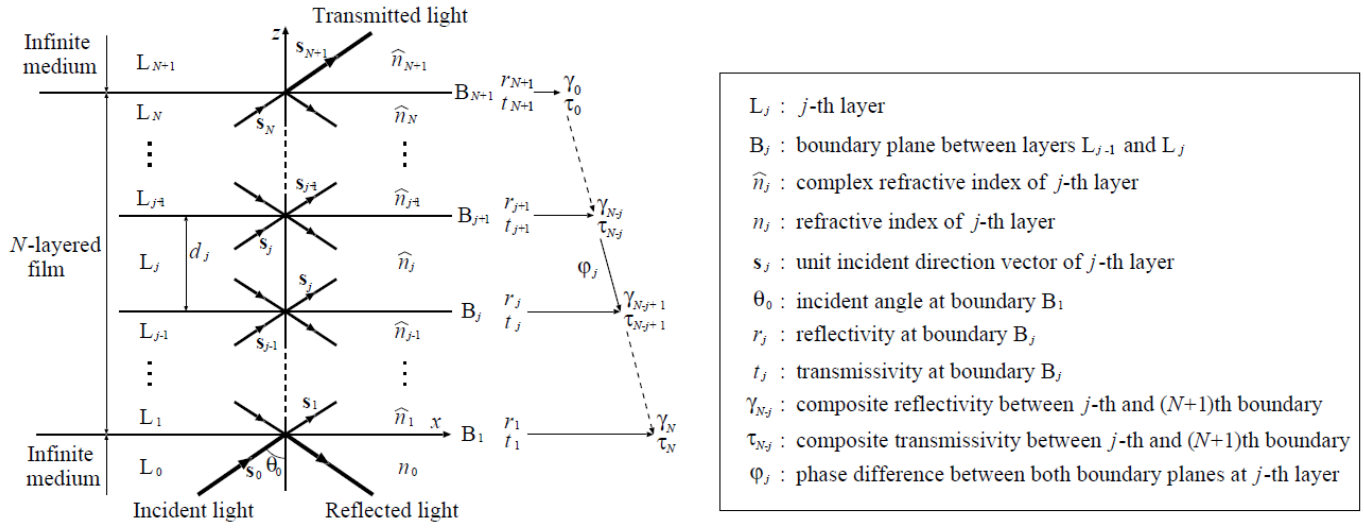

Figure 3.2: Calculating composite reflectivity and transmissivity.

Where γ_{N-j+1} is the composite reflectivity at layer L_j, τ_{N-j+1} is the composite transmissivity at layer L_j, γ_{N-j} is the previous composite reflectivity and τ_{N-j} is the previous composite transmissivity. Note that the previous composite reflectivity and transmissivity are equivalent to saying they are the composite reflectivity and transmissivity of layer $L_j + 1$ (the previous layer). However, at the first iteration we do not have a previous composite reflectivity/transmissivity. Thus for the first iteration, when we are at boundary $B_N + 1$, we can use the following seed equations:

$$\gamma_0 = r_{N+1}, \tag{3.3}$$

$$\tau_0 = t_{N+1}. \tag{3.4}$$

Where r_{N+1} and t_{N+1} are the reflectivity and transmissivity at boundary B_{N+1}. After the first iteration, we use Equations 3.1 and 3.2 to calculate the composite reflectivity and transmissivity iteratively. Note that γ, τ, r and t all consist of a parallel polarization component ($\gamma_\parallel$, $\tau_\parallel$, $r_\parallel$, $t_\parallel$) and a perpendicular polarization component ($\gamma_\perp$, $\tau_\perp$, $r_\perp$, $t_\perp$). A complete derivation of the recurrence equation can be found in the appendix of the main reference paper [8].

Phase Difference

The phase difference φ is responsible for the iridescent colors appearing on objects. The paper calculates the optical path difference of the waves in order to determine how the light waves interfere. For the jth layer, the phase difference is expressed by:

$$\varphi_j = \frac{2\pi}{\lambda} \hat{n}_j d_j s_{z,j}, \tag{3.5}$$

where λ is the wavelength of light in a vacuum, $\hat{n}_j$ is a complex index of refraction ($n + ik$), d_j is the thickness of the jth layer and $s_{z,j}$ is the cosine of the light angle in the jth layer. The unit direction vector s_j describes the direction of the light in the jth layer, which can be derived

from Snell's Law. The x, y and z components of the unit direction vector s_j are defined by:

$$s_{x,j} = \frac{n_0 \sin(\theta_0)}{\widehat{n}_j}, s_{y,j} = 0, s_{z,j} = \sqrt{1 - s_{x,j}^2}, \tag{3.6}$$

where θ_0 is the incident angle of the light at medium A, n_0 is the real valued index of refraction of medium A and $\widehat{n}_j$ is the complex index of refraction of the jth layer. The z-component $s_{z,j}$ represents the cosine angle of the refracted light ray.

Fresnel Equations

Reflectivity and transmissivity can be expressed in its polarized components $r_\parallel$, $r_\perp$, $t_\parallel$ and $t_\perp$. These components can be calculated by using the corresponding Fresnel equations. For the jth interface the Fresnel equations are given by:

$$r_{\parallel j} = \frac{\widehat{n}_j s_{z,j-1} - \widehat{n}_{j-1} s_{z,j}}{\widehat{n}_j s_{z,j-1} + \widehat{n}_{j-1} s_{z,j}}, \tag{3.7}$$

$$r_{\perp j} = \frac{\widehat{n}_{j-1} s_{z,j-1} - \widehat{n}_j s_{z,j}}{\widehat{n}_{j-1} s_{z,j-1} + \widehat{n}_j s_{z,j}}, \tag{3.8}$$

$$t_{\parallel j} = \frac{2\widehat{n}_{j-1} s_{z,j-1}}{\widehat{n}_j s_{z,j-1} + \widehat{n}_{j-1} s_{z,j}}, \tag{3.9}$$

$$t_{\perp j} = \frac{2\widehat{n}_{j-1} s_{z,j-1}}{\widehat{n}_{j-1} s_{z,j-1} + \widehat{n}_j s_{z,j}}. \tag{3.10}$$

Converting Amplitudes into Energy

A relation exists between reflectivity/transmissivity and reflectance/transmittance due to the conservation of energy given by:

$$r^2 + t^2 \frac{n_t \cos(\theta_t)}{n_i \cos(\theta_i)} = 1 \tag{3.11}$$

This relationship can be used in order to calculate the reflectance and transmittance values. In order to convert reflectivity into reflectance we can take the absolute squared value of both polarization components and then take the average. Given the composite reflectivity γ_N we can convert this to the composite reflectance by:

$$k_r = \frac{|\gamma_{\parallel,N}|^2 + |\gamma_{\perp,N}|^2}{2}. \tag{3.12}$$

Where $\gamma_{\parallel,N}$ and $\gamma_{\perp,N}$ are the polarization components of the composite reflectivity and k_r is the resulting composite reflectance value. Squaring the value of a wave amplitude means converting it to flux per unit area, also known as intensity. Since we have two reflectivity components, we take the average value which results in a single reflectance value. The conversion from transmissivity to transmittance can be defined in a similar way. However, Equation 3.11 suggests there is an extra mathematical term involved in this conversion. The transmissivity to transmittance conversion is defined by:

$$k_t = \frac{\widehat{n}_{N+1} s_{z,N+1}}{n_0 s_{z,0}} \left[\frac{|\tau_{\parallel,N}|^2 + |\tau_{\perp,N}|^2}{2} \right], \tag{3.13}$$

where $\tau_{\parallel,N}$ and $\tau_{\perp,N}$ are the polarization components of the composite transmissivity and k_t is the resulting composite transmittance value. The first term in Equation 3.13 describes the area of the refracted light beam. This area will be smaller if medium A has a higher index of refraction than medium B. When $\widehat{n}_{N+1}$ is not a real valued refractive index we know the composite transmittance will be zero. This is because medium B is an infinitely thick layer and if it has a complex valued refractive index, it means it absorbs light. Absorption of light inside of an infinitely thick layer means we will never see the transmitted light and thus the composite transmittance will be zero.

3.1.2 The Algorithm

Figure 3.3 shows the situation of incident light upon a multi layer thin film system. The algo-

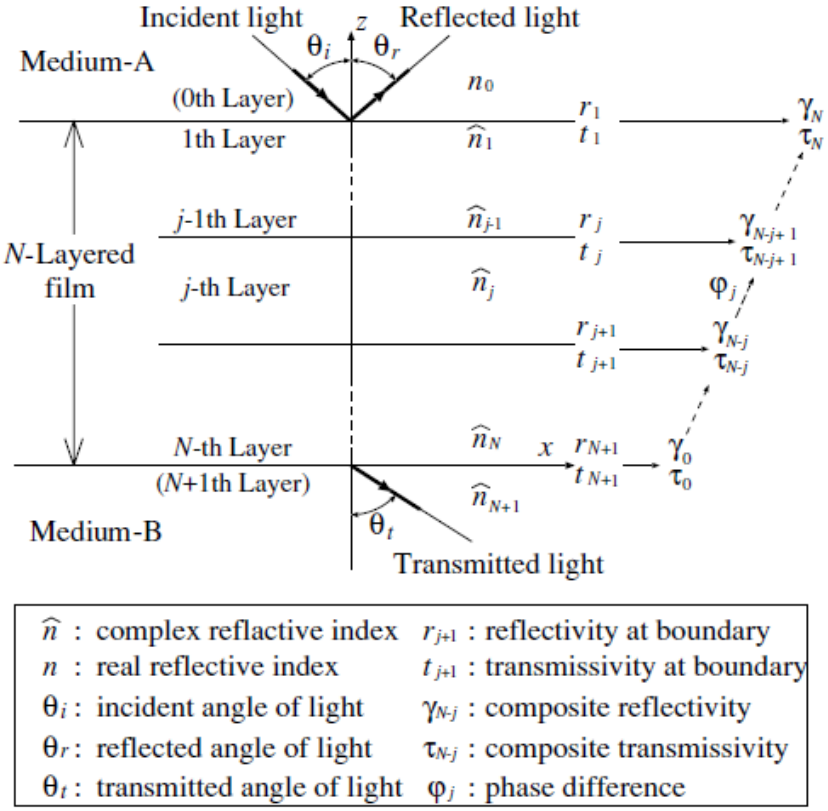

Figure 3.3: Overview of the multilayer film system.

rithm revolves around computing the composite reflectance and transmittance for every incident light angle and wavelength in a preprocessing stage. Both incident angle and wavelength are sampled discreetly. The composite reflectance and transmittance are obtained by recursively calculating the composite reflectivity and transmissivity from the bottom to the top layer using the recurrence equations. The final composite reflectivity and transmissivity are then converted into composite reflectance and transmittance using Equations 3.12 and 3.13. The composite reflectance and transmittance is stored in a 2-dimensional table with incident angle θ_i and wavelength λ as keys to the composite values. During the ray tracing stage, θ_i is evaluated and λ is known. Based on these values, the composite reflectance k_r and composite

transmittance k_t are either retrieved from the table or linearly interpolated between values if the specific angle and wavelength cannot be found in the table. In order to obtain the intensity along the ray, the following illumination model can be used:

$$I_e(\lambda) = k_r(\theta_i, \lambda)I_r(\lambda) + k_t(\theta_i, \lambda)I_t(\lambda). \tag{3.14}$$

Where I_r is the intensity of the reflected light, I_t is the intensity of the transmitted light and I_e is the intensity along the ray observed by the viewer.

3.2 LSRS Illumination Model

The locally smooth rough surface (LSRS) illumination model is proposed in [8]. The paper is very similar in concept to [9], though with some key differences. The paper proposes a method for calculating diffuse and specular reflection contributions of the multi layer thin film system by considering medium B to be a rough surface consisting of micro facets. The paper uses the He-Torrance-Sillion-Greenberg BRDF [7] in order to describe the specular and diffuse reflectivity of a rough surface. Boundaries between thin film layers are assumed to be smooth.

3.2.1 Recurrence Equation

The recurrence equations in this illumination model calculate composite specular reflectivity and composite diffuse reflectivity, denoted by γ_s and γ_d, respectively. Similar to the smooth illumination model, the first iteration we cannot use the recurrence equations since we do not have some previous composite reflectivity. This model also uses a seed equation for the first iteration. However, this particular seed equation takes into account the attenuated reflectivity due to surface roughness. Figure 3.4 shows an overview of a thin film system with a rough surface.

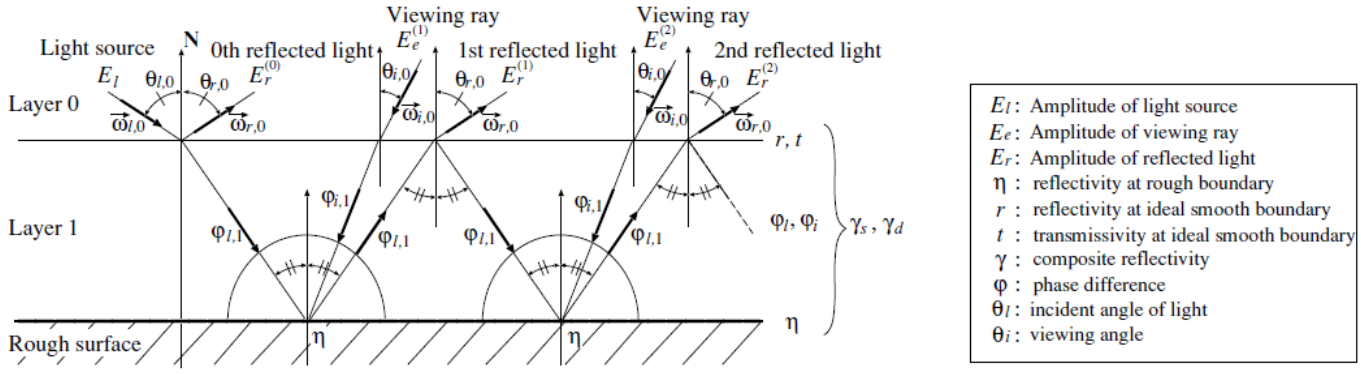

Figure 3.4: Overview of a thin film system with medium B as a rough surface.

Specular Reflectivity

The specular reflectivity recurrence equation is similar to the one described for the smooth illumination model. It is given by:

$$\gamma_{s,N-j+1} = \frac{r_j + \gamma_{s,N-j}e^{2i\varphi_j}}{1 + r_j\gamma_{s,N-j}e^{2i\varphi_j}}, \tag{3.15}$$

where $\gamma_{s,N-j+1}$ is the composite specular reflectivity, $\gamma_{s,N-j}$ is the previous composite specular reflectivity, r_j is the reflectivity between the jth and $j-1$th layers and φ is the phase difference. The recurrence equation is calculated for both parallel and perpendicular polarizations of light, where r_j is either $r_{\parallel,j}$ or $r_{\perp,j}$. Since we do not have a previous composite reflectivity at the first iteration, we will need a seed equation which is given by:

$$\gamma_{s,0} = \eta(\overrightarrow{\omega}_{i,N}, \overrightarrow{\omega}_{i,N}, \lambda). \tag{3.16}$$

Where $\gamma_{s,0}$ is the initial composite specular reflectivity, η is the BRDF function for specular reflectivity taking into account surface roughness described in [7], $\overrightarrow{\omega}_{i,N}$ is the incident direction denoted by its spherical coordinate angles (θ,ϕ) and λ is the wavelength of light. The BRDF function assumes 3D direction vectors as input for the incident and reflected directions of the light respectively. This is why $\overrightarrow{\omega}_{i,N}$ is composed of two angles which can describe the 3D direction in spherical coordinates. Also note that $\overrightarrow{\omega}_{i,N}$ is used for both the incident and reflected direction since $\theta_i = \theta_r$ and $\phi_i = \phi_r$ when we have a specular reflection.

Diffuse Reflectivity

The initial composite diffuse reflectivity can be expressed by the seed equation:

$$\gamma_{d,0} = \eta(\overrightarrow{\omega}_{l,N}, \overrightarrow{\omega}_{i,N}, \lambda). \tag{3.17}$$

Where $\gamma_{d,0}$ is the initial composite diffuse reflectivity, η is the BRDF function describing the diffuse reflectivity from a rough surface, $\overrightarrow{\omega}_{l,N}$ is the light direction, $\overrightarrow{\omega}_{i,N}$ is the incident direction and λ is the wavelength of light. Note that both the light and viewing directions are used in η. This is because, unlike specular light, diffuse light comes from all possible directions on the hemisphere. Thus the BRDF has to take into account how much a specific light direction contributes to the diffuse appearance of an object. The recurrence equation for composite diffuse reflectivity can be expressed as:

$$\gamma_{d,N-j+1} = \frac{\Gamma_j}{1 + r_j(\theta_{l,j})\gamma_{s,N-j}(\theta_{l,j})e^{2i\varphi_{l,j}}}. \tag{3.18}$$

Where $\gamma_{d,N-j+1}$ is the composite diffuse reflectivity, $r_j(\theta_{l,j})$ is the reflectivity given the light angle, $\gamma_{s,N-j}(\theta_{l,j})$ is the previous composite specular reflectivity given the light angle and $\varphi_{l,j}$ is the phase difference given the light angle. The reflectivity, phase difference and composite specular reflectivity in Equation 3.18 are not the same as the ones we have previously discussed. Normally, the angle we use as input to these functions is θ_i which is the angle of the incident viewing ray. This is fine for specular reflection since it reflects in the perfect mirror direction ($\theta_i = \theta_r$). However diffuse light reflects in all kinds of directions and the accumulation of light from all these directions gives us the diffuse appearance of the object. Thus in order to accurately simulate the diffuse lighting, the reflectivity, phase difference and composite specular reflectivity all have to be calculated with respect to the direction of the light. The equation for Γ_j is given by:

$$\Gamma_j = \frac{\widehat{n}_j s_{z,j}}{\widehat{n}_{j-1} s_{z,j-1}} t_j(\theta_{i,j-1}) t_j(\theta_{l,j-1}) \gamma_{d,N-j} e^{i[\varphi_{l,j} + \varphi_{i,j}]}. \tag{3.19}$$

Where $s_{z,j}$ and $s_{z,j-1}$ are unit direction vectors with respect to the light direction, $t_j(\theta_{i,j-1})$ is the specular transmissivity, $t_j(\theta_{l,j-1})$ is the diffuse transmissivity, $\varphi_{i,j}$ is the phase difference with respect to the viewing direction and $\varphi_{l,j}$ is the phase difference with respect to the light direction (Figure 3.4).

3.2.2 The Algorithm

Composite specular and diffuse reflectivities are calculated for every sampled viewing angle, light angle and wavelength in a preprocessing stage. If we denote the number of samples for viewing angles by N, the number of samples for light angles by M and the number of samples for wavelengths by K, the calculations for composite specular and composite diffuse reflectivity need to be done $N \times M \times K$ times. The composite specular and diffuse reflectivity can then be converted into composite specular and diffuse reflectance by using Equation 3.12. The composite specular and diffuse reflectance are stored in a 3-dimensional table with viewing angle θ_i, light angle θ_l and wavelength λ as keys to the composite values. During the ray tracing stage, the incident angle, light angle and wavelength are evaluated and used to obtain the composite specular and diffuse reflectance from the table. Because the angles and wavelength are sampled discreetly, it can happen that the exact incident angle, light angle or wavelength is not present in the table. Linear interpolation is used to interpolate several composite specular and diffuse reflectance values from the table that are close to the evaluated angles and wavelength. In order to obtain the intensity along the ray, the following illumination model can be used:

$$I_e(\lambda) = k_r(\theta_i, \lambda) I_r(\lambda) + \sum_{m=1}^{M} k_d(\overrightarrow{\omega}_l, \overrightarrow{\omega}_i, \lambda) \cos(\theta_l) L_m(\lambda). \tag{3.20}$$

Where I_e is the intensity along the viewing ray, I_r is the intensity of the reflected light, k_r is a composite specular reflectance, k_d is a composite diffuse reflectance and L_m is the intensity of light source m. The number of light sources in the scene is denoted by M. The first term describes the global illumination or specular reflection. The second term describes the local illumination or the diffuse reflection. The diffuse reflection is a local illumination component, because light comes from all possible directions and accumulates at the point we are shading. Specular reflection is a global illumination component because it takes into account the inter-reflections between objects. Figure 3.5 shows the global and local illumination components schematically.

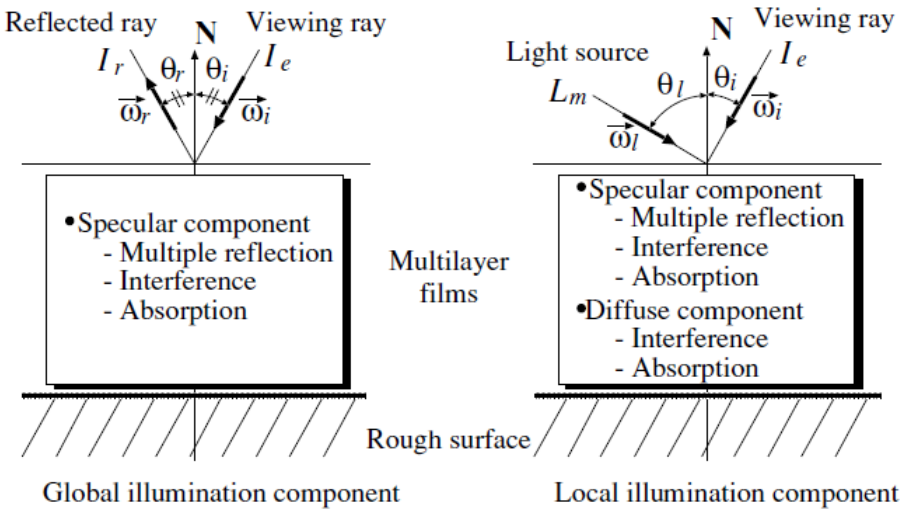

Figure 3.5: The global (specular) and local (diffuse) illumination components.

3.3 CPU Implementation

We have implemented both algorithms on the CPU using C++ inside of the ray tracer PBRT. The multi-layered thin film interference simulation is done in the *SmoothIridescenceIntegrator* and *RoughIridescenceIntegrator* surface integrators for smooth and LSRS illumination models, respectively. The iridescence surface integrators have two tasks: simulation and rendering. Simulation is done in a preprocessing stage of the surface integrator and rendering is done at run-time.

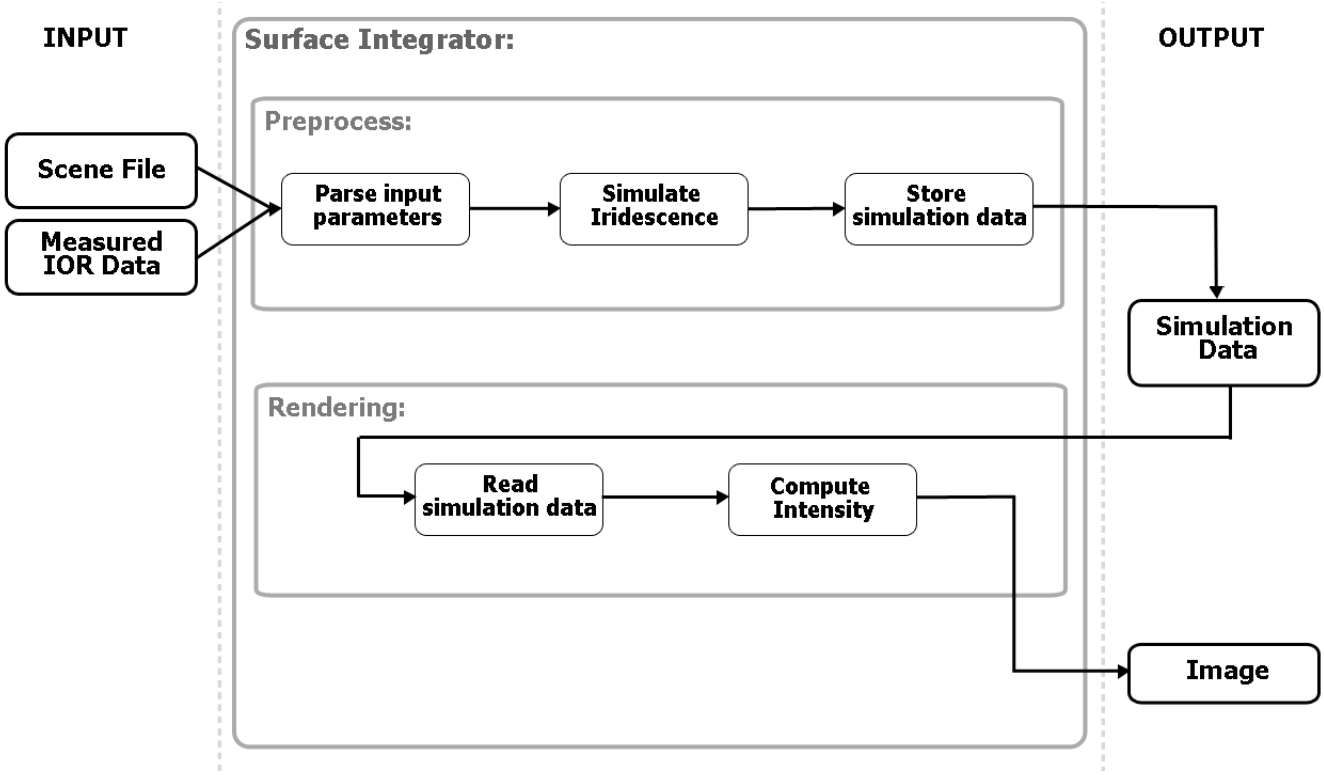

Figure 3.6: Overview of the implementation.

3.3.1 Implementation Input

The input to our implementation is a scene file and optionally measured data for refractive indices (Figure 3.6). The scene file describes the objects, light sources, materials, renderer, integrator and camera type that will be used by PBRT. We have added the parameters of the multi-layered thin film system to this scene file. The thin film parameters are parsed and stored in the iridescence surface integrators. The full list of parameters for the integrators is discussed in section 3.3.3 and section 3.3.4. The scene file stores a string value for every layer of the thin film system which denotes the complex index of refraction of that layer. There are two possible ways of defining the material for a layer: a complex index of refraction constant over all wavelengths or measured complex index of refraction data. A constant complex index of refraction can be defined as:

```
1 "string bottomIOR" ["1.0;0.0"];
```

Listing 3.1: Index of refraction of air, constant over all wavelengths.

where the semicolon is the delimiter separating the real and imaginary parts of the refractive index. Measured complex index of refraction data can be defined as:

```
"string bottomIOR"  ["f:goldPalik.ior"];
```

Listing 3.2: Index of refraction of gold, spectrally varying.

where "f:" denotes that we are parsing measured refractive index data from a file, in this case "goldPalik.ior". Measured complex index of refraction data is most commonly used for metals since both n and k can vary greatly between different wavelengths for these kinds of materials. Figure 3.7 shows a plot of the complex refractive index of gold, where the dashed line is the index of refraction n and the solid line is the absorption coefficient k. We have created an index

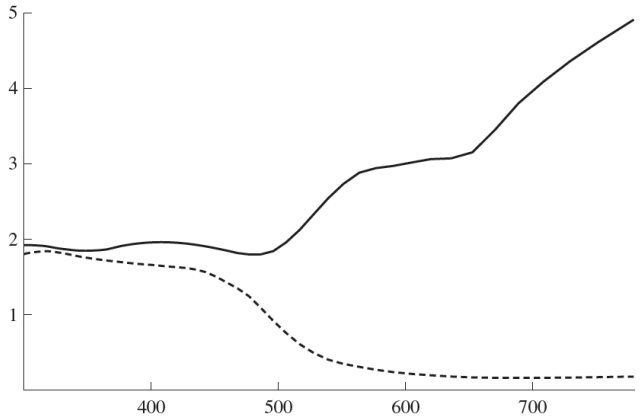

Figure 3.7: The complex refractive index of gold is wavelength dependent.

of refraction parser that takes string values in the format that is shown in Listing 3.1 and 3.2. Based on the format, the parser determines whether the refractive index is constant or varying spectrally. When the refractive index is a constant value, the same complex index of refraction is stored for every sampled wavelength. However, when we are dealing with measured data, we have to find the complex refractive indices in the file that correspond to the sampled wavelengths. Oftentimes the wavelengths we would like to sample do not necessarily correspond to the wavelengths in the measured file data. If this is the case, wavelengths that are close to the sampled wavelength are obtained from the measured data instead of the exact wavelength we are looking for. We then linearly interpolate the complex index of refractions values at these wavelengths to get a complex refractive index that corresponds with the sampled wavelength.

3.3.2 Implementation Output

The output to our implementation is the simulation data at the end of the simulation stage and an image at the end of the rendering stage. The simulation data contains the composite reflectance and/or composite transmittance values. Based on the simulation (smooth or LSRS illumination model) this data is stored differently. For the smooth illumination model the data is stored per sampled viewing angle and sampled wavelengths of light. For the LSRS illumination model the data is stored per sampled viewing angle, per sampled light angle and sampled wavelengths of light. The simulation data is written to a *.csv file at the end of the preprocessing stage (Figure 3.8). The simulation data is used in the rendering stage as a look up table. Values

Incident angle	Wavelength	Composite Reflectance	Composite Transmittance
1	400	0,0334109	0,966589
1	410	0,0110914	0,988908
1	420	0,000374166	0,999626
1	430	0,00420771	0,995792
1	440	0,0193369	0,980663
1	450	0,03932	0,96068
1	460	0,0579601	0,94204
1	470	0,0711054	0,928895
1	480	0,0768683	0,923132
1	490	0,0751423	0,924858
1	500	0,0670375	0,932963
1	510	0,054432	0,945568
1	520	0,039616	0,960384
1	530	0,0249591	0,975041
1	540	0,0125804	0,98742
1	550	0,00405254	0,995948
1	560	0,000210634	0,999789
1	570	0,00111792	0,998882
1	580	0,00618789	0,993812
1	590	0,0144078	0,985592
1	600	0,0245845	0,975415
1	610	0,0355497	0,96445
1	620	0,0462942	0,953706
1	630	0,0560309	0,943969
1	640	0,0642063	0,935794
1	650	0,0704793	0,92952
1	660	0,0746873	0,925313
1	670	0,0768082	0,923192
1	680	0,0769274	0,923073
1	690	0,0752091	0,924791
2	400	0,0331476	0,966853
2	410	0,0109059	0,989094
2	420	0,000338664	0,999662
2	430	0,00432415	0,995676
2	440	0,0195498	0,98045
2	450	0,0395548	0,960445
2	460	0,0581544	0,941846

Figure 3.8: Simulation data for a soap bubble in the smooth illumination model.

for the composite reflectance and/or composite transmittance are obtained from the data and linearly interpolated if necessary. The illumination model is then used to compute the intensity

Figure 3.9: Soap bubbles rendered with the simulation data of Figure 3.8.

along the viewing ray for every pixel, which results in a rendered image (Figure 3.9).

3.3.3 SmoothIridescenceIntegrator

The *SmoothIridescenceIntegrator* class is responsible for simulating and rendering iridescent objects with the smooth illumination model.

Surface Integrator Parameters

The parameters for the surface integrator are defined in the scene file (Listing 3.3).

```
SurfaceIntegrator "smoothIridescence"
  "string tableName"       ["SoapForest.csv"]
  "integer angleSamples"     [80]

  # medium top
  "float iorTop"       [1.0]

  # thin film layer parameters
  "string layerIORs"       ["1.33;0.0"]
  "float thicknessLayers"    [635]

  # bottom medium
  "string bottomIOR"       ["1.0;0.0"]
```

Listing 3.3: SmoothIridescenceIntegrator scene file parameters.

The surface integrator type *smoothIridescence* denotes that the *SmoothIridescenceIntegrator* surface integrator should be used in this scene. This integrator has a number of parameters associated with it that are shown in Listing 3.3. The *tableName* parameter specifies the filename for the file that will store the simulation data. The *angleSamples* parameter describes the number of samples that should be taken for the viewing angle in the simulation. The *iorTop* parameter is the real-valued refractive index of medium A in the thin film system. The *layerIORs* parameter describes the complex index of refractions for the thin film layers. The *thicknessLayers* parameter describes the thickness of the thin film layers in nanometers. The *layerIORs* and the *thicknessLayers* parameters are arrays of values, thus if we wanted to specify a thin film system with two layers instead of one we would write the following:

```
"string layerIORs"       ["1.33;0.0" "1.5;0.0"]
"float thicknessLayers"    [635 348]
```

Listing 3.4: Parameters for two layer thin film system.

Note, that for every complex index of refraction we add a corresponding thin film thickness should be specified. Lastly, the *bottomIOR* parameter is the complex refractive index of medium B in the thin film system.

Simulation

The thin film interference simulation is done in the *Preprocess* function of the *SmoothIridescenceIntegrator* class. The resulting output at the end of this function is a file containing the composite reflectance and composite transmittance data for every viewing angle and wavelength. Pseudo code for the surface integrator preprocessing stage is provided in Algorithm 1.

Algorithm 1 SmoothIridescenceIntegrator Simulation

for $i \leftarrow 0$ to $angleSamples$ **do**
    $incidentAngle \leftarrow ComputeAngleSample(i)$

    for $j \leftarrow 0$ to $wavelengthSamples$ **do**
        $wavelength \leftarrow ComputeWavelengthSample(j)$

        Retrieve complex index of refractions $seedNi$ and $seedNt$ at the last interface
        $seedCosi \leftarrow ComputeUnitDirection(incidentAngle, seedNi).z$
        $seedCost \leftarrow ComputeUnitDirection(incidentAngle, seedNt).z$
        $compositeRefl \leftarrow ComputeReflectivity(seedNi, seedNt, seedCosi, seedCost)$
        $compositeTrans \leftarrow ComputeTransmissivity(seedNi, seedNt, seedCosi, seedCost)$

        for $k \leftarrow numLayers - 1$ to 0 **do**
            Retrieve complex index of refractions ni and nt at the current interface
            Retrieve thin film layer thickness d of the kth layer
            $cosi \leftarrow ComputeUnitDirection(incidentAngle, ni).z$
            $cost \leftarrow ComputeUnitDirection(incidentAngle, nt).z$
            $refl \leftarrow ComputeReflectivity(ni, nt, cosi, cost)$
            $trans \leftarrow ComputeTransmissivity(ni, nt, cosi, cost)$
            $phaseDiff \leftarrow ComputePhaseDifference(d, wavelength, cost)$
            $compositeRefl \leftarrow DoReflRecurrence(refl, 2 * phaseDiff)$
            $compositeTrans \leftarrow DoTransRecurrence(trans, phaseDiff)$
        end for

        $reflectance \leftarrow ComputeReflectance(compositeRefl)$
        $transmittance \leftarrow ComputeTransmittance(compositeTrans)$
        $WriteToFile(reflectance, transmittance)$
    end for
end for

Sampling The *ComputeAngleSample* and *ComputeWavelengthSample* functions describe the calculation of the current angle and wavelength samples respectively. For example, calculating the current angle can be done using the pseudo code in Listing 3.5.

```
function ComputeAngleSample(int sample)
{
  // Calculate incident angle in degrees
  int incidentAngleDegrees = (90 / angleSamples) * (sample + 1);

  // Convert to incident angle in radians
  float incidentAngleRadians = DegreesToRadians(incidentAngleDegrees);
}
```

Listing 3.5: Calculating the current angle from the sample index.

Where *angleSamples* is the number of viewing angle samples and *DegreesToRadians* is a function that takes an angle in degrees as input and returns the angle in radians.

Light direction The *ComputeUnitDirection* function calculates the refracted direction inside a layer based on the layer's index of refraction and the angle at the top of the thin film system (*incidentAngle*). We use this function to calculate the incident (*seedCosi/cosi*) and refracted (*seedCost/cost*) cosine angles at boundaries between layers. Equation 3.6 can be used to calculate the incident and refracted directions. The z- component of the incident and refracted directions is the cosine of the incident and refracted angle respectively.

Light Quantities The *ComputeReflectivity* and *ComputeTransmissivity* functions calculate the reflectivity and transmissivity respectively at an interface. The Fresnel Equations 3.7 through 3.10 are used to calculate these quantities. The *ComputePhaseDifference* function calculates the phase difference in the kth layer using Equation 3.5. Equations 3.1 and 3.2 describe the recurrence equations which are named *DoReflRecurrence* and *DoTransRecurrence* respectively in Algorithm 1. The composite reflectance and composite transmittance is calculated using the *ComputeReflectance* and *ComputeTransmittance* functions. We can use Equations 3.12 and 3.13 to calculate the conversion from reflectivity to reflectance and transmissivity to transmittance. After the conversion, the reflectance and transmittance value are written into a *.csv file and the algorithm starts the next iteration.

Rendering

In the rendering stage, rays are shot from the camera into the scene. When a ray intersects

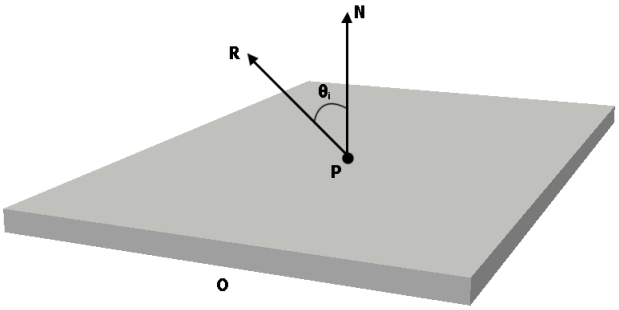

Figure 3.10: Rendering situation for the smooth illumination model.

an object O at a point P, the viewing angle θ_i can be determined. The angle θ_i is defined as the dot product between the normalized ray vector R pointing away from the surface and the normalized surface normal N:

$$\theta_i = R \cdot N. \tag{3.21}$$

We can now retrieve the composite reflectance and composite transmittance from the simulation data table. For every wavelength λ and given θ_i, use λ and θ_i as keys in the table to search for the composites. It is possible that an exact match for either θ_i or λ cannot be found. In this case, linear interpolation is employed to calculate suitable composite reflectance and transmittance values. Once the composite reflectance and transmittance for every sampled wavelength is obtained, Equation 3.14 can be used to calculate the intensity along the camera ray.

3.3.4 RoughIridescenceIntegrator

The *RoughIridescenceIntegrator* class is responsible for simulating and rendering iridescent objects with the LSRS illumination model.

Surface Integrator Parameters

The parameters for the surface integrator are defined in the scene file (Listing 3.6).

```
SurfaceIntegrator "roughIridescence"
  # General properties
  "float maxDepth"          [5]
  "string tableName"          ["RoughIri.csv"]
  "integer incidentAngleSamples"  [80]
  "integer lightAngleSamples"    [80]

  # Surface properties
  "integer D_Iterations"        [5]
  "float rmsRoughness"        [25]
  "float autocorrelation"       [300]

  # Medium top
  "float iorTop"            [1.0]

  # Thin film layer parameters
  "integer numberOfLayers"      [1]

  "string layerIORs"          ["1.33;0.0"]
  "float thicknessLayers"       [348]

  # Medium bottom
  "string bottomIOR"          ["f:silicon.ior"]
```

Listing 3.6: RoughIridescenceIntegrator scene file parameters.

Where *maxDepth* is a parameter for global illumination specifying how many times a light ray can bounce before being terminated. The *tableName* parameter denotes the filename for the simulation data. The *incidentAngleSamples* parameter determines how many viewing angle samples are taken in the simulation. The *lightAngleSamples* parameter specifies how many samples are taken for the light angles. The *D_iterations* parameter is used to limit an infinite summation that is used in the calculation of the diffuse reflectivity due to a rough surface [7]. The *rmsRoughness* parameter specifies the height distribution of the rough surface in nanometers. The *autocorrelation* parameter specifies the horizontal distance between peaks in nanometers. The *iorTop* parameter specifies the real valued index of refraction of medium A. The *numberOfLayers* parameter specifies how many thin film layers should be used in the simulation. When this value is zero, it means we only want to render the base material i.e. medium B. The *layerIORs* parameter is the array of complex index of refractions of the thin film layers. The *thicknessLayers* parameter is an array of thickness values for every thin film layer specified in nanometers. The *bottomIOR* parameter is the complex index of refraction of medium B.

Simulation

Pseudo code for the surface integrator preprocessing stage is provided in Algorithm 2. The algorithm shares some similarities with Algorithm 1 since it also computes the composite specular reflectance. The difference is that we need to calculate the composite diffuse reflectance in this algorithm instead of the composite specular transmittance. A notable difference between specular and diffuse light is that not all light reflects into a perfect mirror direction, rather it can reflect in all possible directions on the hemisphere. Light bouncing around in between the micro facets of a rough surface result in the diffuse appearance of an object. This means that the reflected angle for diffuse light is in nearly all cases not the same as the incident light angle. For

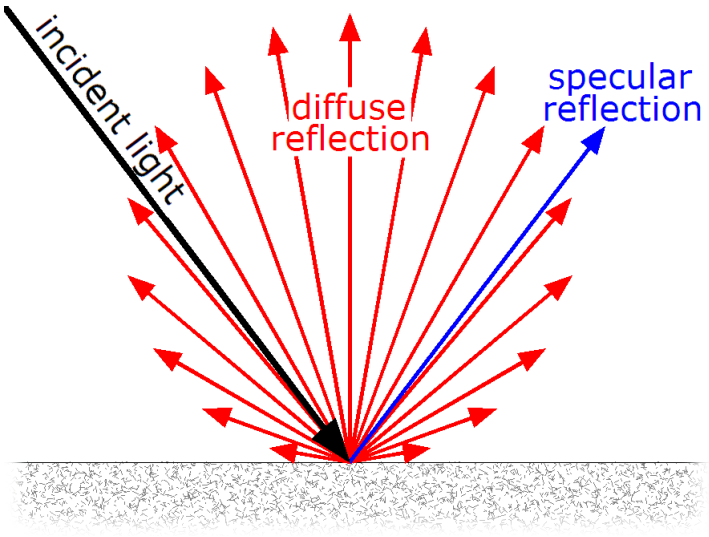

Figure 3.11: The difference in reflection behavior for specular and diffuse lighting.

our simulation this means we need to not only take into account the incident (viewing) angles but also the angles of the light with respect to the surface normal. The *ComputeSeedSpecRefl* and *ComputeSeedDiffRefl* functions calculate the specular and diffuse reflectivity due to a rough surface using the HTSG BRDF [7] (Equations 3.16 and 3.17). The *SpecReflRecur* and *DiffReflRecur* functions are the recurrence equations for the specular and diffuse component respectively. These recurrence equations are given by Equations 3.15 and 3.18. The other functions are exactly the same as in Algorithm 1 and thus will not be described here.

Algorithm 2 RoughIridescenceIntegrator Simulation

for $i \leftarrow 0$ to $viewAngleSamples$ **do**
    $viewAngle \leftarrow ComputeAngleSample(i)$

    for $j \leftarrow 0$ to $lightAngleSamples$ **do**
        $lightAngle \leftarrow ComputeAngleSample(j)$

        for $k \leftarrow 0$ to $wavelengthSamples$ **do**
            $wavelength \leftarrow ComputeWavelengthSample(k)$

            Retrieve complex index of refraction $lastIOR$ of the last thin film layer
            $incViewDir \leftarrow ComputeUnitDirection(viewAngle, lastIOR)$
            $incLightDir \leftarrow ComputeUnitDirection(lightAngle, lastIOR)$
            $compSpecRefl \leftarrow ComputeSeedSpecRefl(incViewDir, incViewDir, wavelength)$
            $compDiffRefl \leftarrow ComputeSeedDiffRefl(incLightDir, incViewDir, wavelength)$

            for $n \leftarrow numLayers - 1$ to 0 **do**
                Retrieve complex index of refractions ni and nt at the current interface
                Retrieve thin film layer thickness d of the nth layer

                $cosiView \leftarrow ComputeUnitDirection(viewAngle, ni).z$
                $costView \leftarrow ComputeUnitDirection(viewAngle, nt).z$
                $reflView \leftarrow ComputeReflectivity(ni, nt, cosiView, costView)$
                $transView \leftarrow ComputeTransmissivity(ni, nt, cosiView, costView)$
                $phaseView \leftarrow ComputePhaseDifference(d, wavelength, costView)$

                $cosiLight \leftarrow ComputeUnitDirection(lightAngle, ni).z$
                $costLight \leftarrow ComputeUnitDirection(lightAngle, nt).z$
                $reflLight \leftarrow ComputeReflectivity(ni, nt, cosiLight, costLight)$
                $transLight \leftarrow ComputeTransmissivity(ni, nt, cosiLight, costLight)$
                $phaseLight \leftarrow ComputePhaseDifference(d, wavelength, costLight)$
                $phaseTotal \leftarrow phaseView + phaseLight$

                $compSpecRefl \leftarrow SpecReflRecur(reflView, 2 * phaseView)$
                $compDiffRefl \leftarrow DiffReflRecur(transView, reflLight, transLight, phaseTotal)$
            end for

            $specularReflectance \leftarrow ComputeReflectance(compSpecRefl)$
            $diffuseReflectance \leftarrow ComputeReflectance(compDiffRefl)$
            $WriteToFile(specularReflectance, diffuseReflectance)$

        end for
    end for
end for

Rendering

In the rendering stage, rays are shot from the camera into the scene. When a ray intersects an object O at a point P, the viewing angle θ_i and light angle θ_l can be determined. The angle θ_i can be calculated using Equation 3.21. In order to calculate θ_l we need to know the angle between the light direction and the surface normal. This can be defined as the dot product between the normalized light vector L pointing towards the light source and the normalized surface normal N:

$$\theta_l = L \cdot N. \tag{3.22}$$

Given θ_i, θ_l and the wavelength λ we can access the composite specular and diffuse reflectance from the simulation data table. Linear interpolation between composite reflectance values is employed if no exact match is found in the table for either θ_i, θ_l or λ. Once the composite specular and diffuse reflectance are retrieved for every wavelength, the illumination model (Equation 3.20) is used to calculate the intensity along the camera ray.

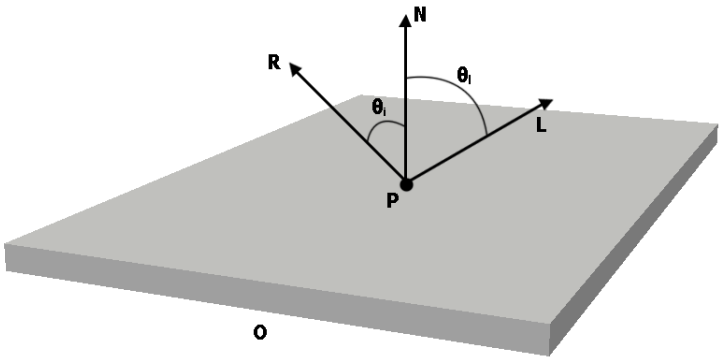

Figure 3.12: Rendering situation for the LSRS illumination model.

3.4 Constraints

The calculation of light reflecting off and transmitting through a multi-layered thin film system has a few preconditions that are mentioned in the papers. The following constraints on the multilayer film system apply:

1. Boundaries between layers are smooth.
2. Light reflects in the direction of mirror reflection at layer boundaries.
3. Light transmits in the direction obeying Snell's Law between the boundaries of layers.
4. The layer media are homogeneous and isotropic.
5. The top and bottom mediums extend to infinity.
6. The top medium has a real valued refractive index meaning no absorption.

Each boundary, or interface, is smooth because the authors assume constant layer thickness; though every layer can have a different thickness. The second and third constraints are based upon the first constraint. Since we have these perfectly smooth interfaces, light will reflect in mirror directions. Furthermore, transmission of light with Snell's Law is only true for smooth interfaces. The index of refraction of a layer medium is assumed to be a single (spectrally varying and complex valued) refractive index which is the fourth constraint. The fifth constraint is based on the fact that we know how thick the thin film mediums are, but we do not know how thick the media incident to the thin films are. As an example, medium A is in nearly all cases air and air has no specifically defined thickness. Furthermore, medium B is usually some object where the interference phenomenon is occurring on. We do not have knowledge about how thick that material is, since we are only considering the effect medium B has on the iridescence. Furthermore, both medium A and medium B are more often than not orders of magnitude larger than the wavelength of light which is in stark contrast to the thin film thicknesses. This means we can assume both media to be infinitely thick relative to the thin films. This leads us to the sixth constraint: medium A is infinitely thick thus it should not have absorption. The incident light propagates through medium A first, if it were infinitely thick the light would never reach the first interface because all of the energy would be absorbed instead.

Chapter 4

Real-time Multi-Layered Thin Film Interference

Our method is based on the smooth illumination model algorithm proposed by Hirayama et al. [9]. However, the method proposed by Hirayama et al. cannot simulate objects that have a spatially varying thin film thickness. Furthermore, their algorithm is not suitable for real-time rendering of iridescent objects. We propose a method that can simulate iridescent objects with arbitrary spatially varying thin film thickness in real-time on the GPU.

4.1 OpenGL Renderer

A custom physically-based real-time OpenGL renderer was created in order to have complete control over the rendering pipeline. It is written in C++ with OpenGL version 3.x in mind. The renderer supports the following features:

1. Environment Mapping.
2. Scene Files.
3. Measured IOR Data Files.
4. First Person Camera.
5. Scene Graph.
6. GLSL Shaders.
7. HDR Lighting.
8. Textures.
9. Mesh Loading.
10. Gamma Correction.

Environment Mapping Cube mapping is the supported environment mapping technique in the renderer. A unit cube is placed at the origin of the world and is transformed using a modified camera matrix. This camera matrix strictly contains the camera rotation. The idea here is that if we rotate the camera so does the cube map. However, we always want to be in the center of the cube map hence why the modified matrix does not contain translation and scale components. The cube is rendered to the screen with depth testing turned off so that it is rendered at infinity and does not obstruct the view of any other object we will render later. When the rendering is complete, the depth testing is turned on again. A vertex and pixel shader are used to render the cube map texture onto the unit cube. The cube map texture data contains the six individual faces of the cube map. We can sample these faces using 3D texture coordinates. Essentially, these coordinates are vectors in 3D space pointing towards a particular face of the cube map. In the vertex shader the cube is transformed using the modified camera matrix. The position of the vertex is set as the 3D texture coordinate and passed to the pixel shader. Data that is passed from the vertex shader to the pixel shader is automatically interpolated over the surface. In Figure 4.1 we show an example. The red line denotes the vector pointing from the

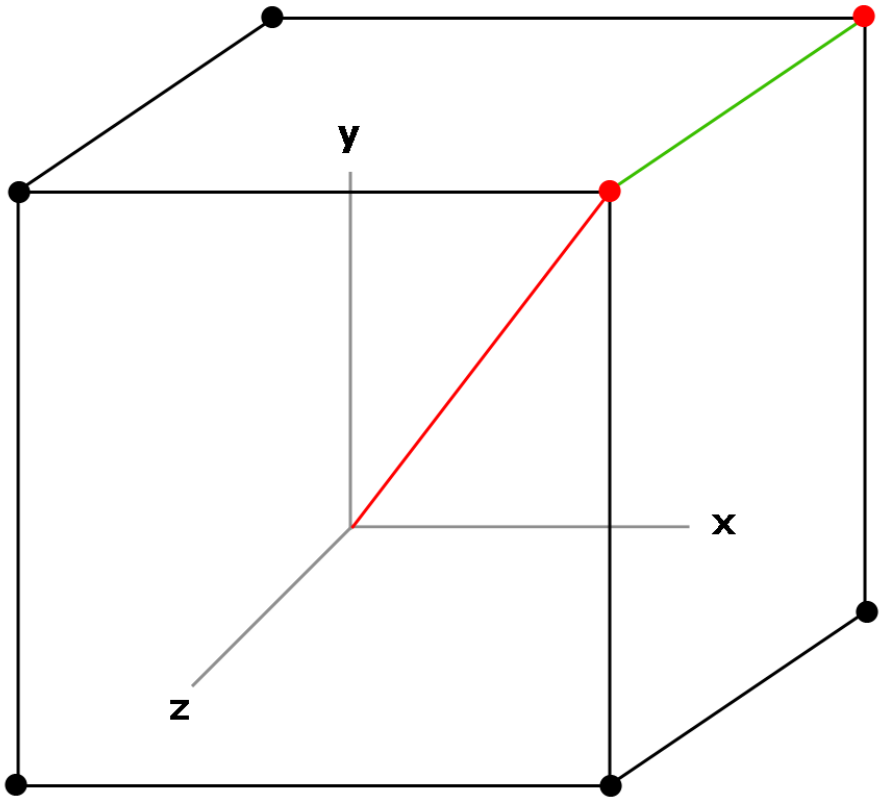

Figure 4.1: Determining the 3D texture coordinate.

origin to the vertex we are evaluating and thus is our 3D texture coordinate. If we are evaluating a pixel on the green edge of the cube, the vector will be interpolated between the red vertices. The interpolation gives us the correct 3D texture coordinate for sampling the cube map. The 3D texture coordinate is given to a texture sampler along with the cube map texture. The output of the sampler is a color, sampled from the cube map texture, which we can render to the screen.

Scene Files Scene files are in *.XML format and contain the mesh descriptions, shader programs, textures and object instances. The mesh XML node specifies a file from which we will load the geometry data. A shader program XML node specifies a vertex shader and a pixel shader that can be referenced as a pair by a shader program name. Texture XML nodes specify the texture file name and contain a boolean value that determines whether the texture is in linear or non-linear color space. Objects are instanced as objects through nodes, not to be confused with the aforementioned XML nodes. A node contains references to a mesh, a name identifier and a shader program. Every node also has basic transformation components like translation, rotation and scale.

Measured IOR Data Files Measured complex index of refraction data files can be parsed by the renderer. It uses a modified version of the parser that we created for the CPU algorithm implementations in PBRT. This allows us to accurately simulate metallic objects in real-time on the GPU (Figure 4.2).

Figure 4.2: Head rendered with a measured gold material.

First Person Camera The first person camera can be controlled using the WASD keys to move forward, backwards, left and right respectively. The Q and E keys move the camera up and down. Additionally, by holding down the left mouse button and dragging the mouse, the camera can be rotated. By holding down the Shift key the mouse will translate in smaller increments allowing for more precise control over the translation of the camera. By holding down ALT, left mouse button and dragging the mouse the camera can be rotated around the z-axis.

Scene Graph A scene graph is used in the renderer to be able to transform and render objects in the world more easily. Every object in the scene is attached to a node. A node contains information about the object geometry, shader programs and textures. A node can also have child nodes. This makes it possible to do complex object transformations, by making use of the inherent parent child relationship between the nodes.

GLSL Shaders The renderer uses the GLSL shading language for all of its shaders. We have written a number of common GLSL shaders that are included with the renderer. The included shaders are: Lambertian diffuse, Fresnel reflection using cube maps, Fresnel transmission using cube maps, Blinn-Phong shading and lastly, a shader that draws the cube map. Figure 4.3 shows some rendering results using the Lambertian diffuse and Blinn-Phong shaders.

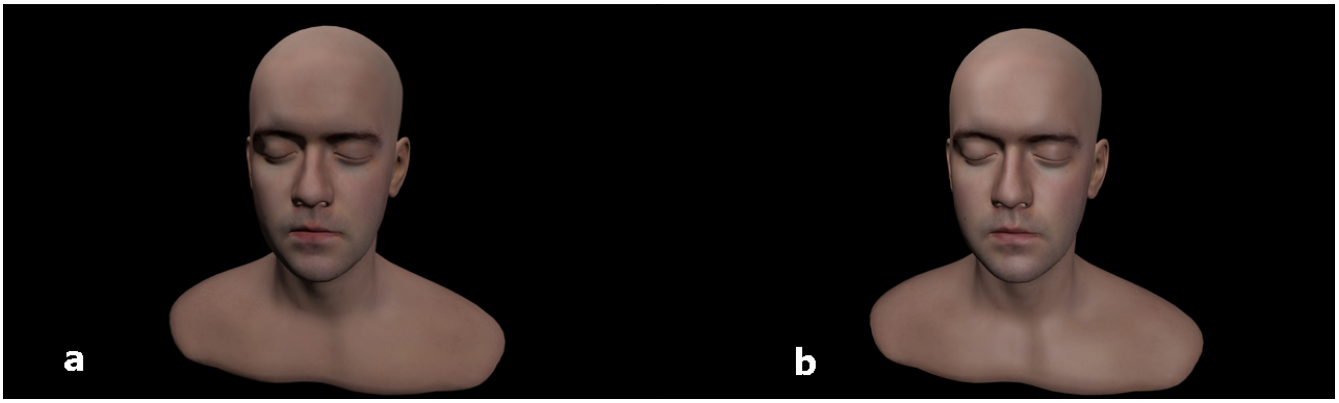

Figure 4.3: a) Lambertian diffuse. b) Blinn-Phong.

HDR Lighting Typically rendering is done in LDR space or Low Dynamic Range space. LDR lighting values are restricted to the normalized RGB value interval [0,1]. A monitor only supports color values in this LDR interval. Using LDR lighting, details in very dark or very bright regions of the image are not necessarily preserved. Thus all lighting calculations that are done can greatly benefit from a wider range of possible lighting values to prevent this loss of detail from happening. This is called HDR lighting or High Dynamic Range lighting. With HDR lighting, light computations are not necessarily restricted to the [0,1] interval. By doing the lighting calculations in HDR space, the details in the image are preserved even when we convert it back into LDR space for display on a monitor.

Figure 4.4: HDR/LDR lighting comparison of the game Age of Empires.

Textures Texture loading for common formats such as *.jpg, *.tga and *.dds is supported. A texture can be automatically bound to an object by specifying it in an object instance node in the scene file.

Mesh Loading A mesh loading library by the name of Assimp is used to load in custom geometry in various file formats. Additionally, geometry can be defined in an *.xml file containing the vertex positions, normals, texture coordinates and vertex colors using the format in Appendix B.

Gamma Correction Preserving linearity in rendering calculations is of utmost importance in physically-based rendering. We can use gamma correction, also sometimes called tone mapping, in order to preserve linearity throughout the entire rendering pipeline. Calculations in a linear-space are especially important for lighting. However, most monitors follow a gamma curve of approximately 2.2 for its output (Figure 4.5). This means that the RGB color we use in

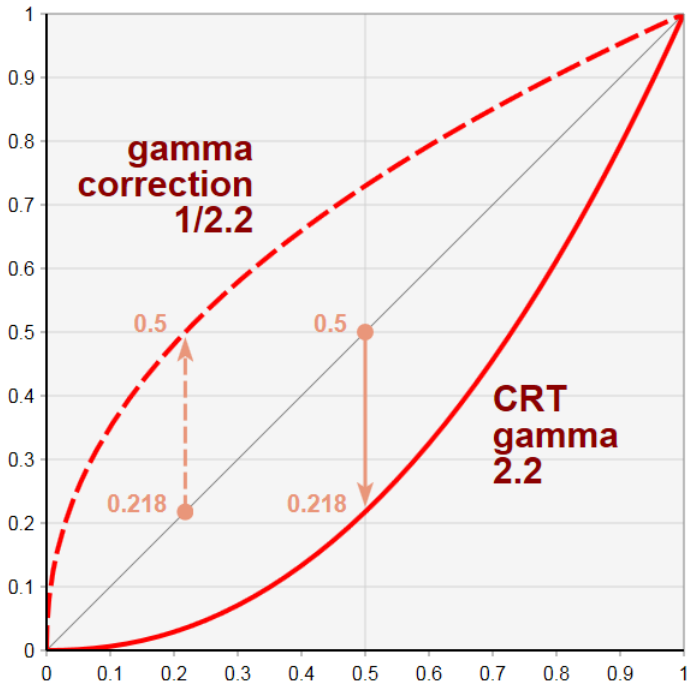

Figure 4.5: From top to bottom: gamma = 0.45, gamma = 1.0, gamma = 2.2.

calculations does not necessarily represent what we will expect to see displayed on the monitor. An example: consider we want the RGB value in the middle of pure black (RGB = 0) and pure white (RGB = 1) which is gray (RGB = 0.5). We would expect this gray color to have the same RGB value on the monitor as it did in our rendering output. However, a value of RGB = 0.218 is displayed on the monitor instead. The monitor has applied its gamma curve to the rendering output before displaying it on the screen:

$$DisplayColor = RenderColor^{gamma}. \tag{4.1}$$

This converted our color from linear-space into a non-linear color space resulting in a darker color. In order to preserve linearity, we have to output values that will remain linear after the

monitor gamma curve is applied. We can apply the inverse gamma curve (Equation 4.2) to our image before the monitor gamma curve is applied (Equation 4.1) in order to display the image linearly on the monitor:

$$CorrectedColor = RenderColor^{\frac{1}{gamma}}. \tag{4.2}$$

This is commonly called gamma correction or tone mapping.

4.2 Spatially Varying Thickness

Spatially varying thin film thickness (SVT) aims to solve one of the problems of the original multi-layered thin film interference model, namely the constant thin film thickness for every layer. Instead, SVT allows for the thin film thickness to vary over the surface of an object. The SVT equation can be defined as:

$$d(x) = d_{avg} + d_{rel}\Delta_{rel}(x), \tag{4.3}$$

where d_{avg} is a user defined average thickness of the thin film, d_{rel} is a user defined relative thickness value and $\Delta_{rel}(x)$ a function varying over the surface describing the relative change in thin film thickness. $\Delta_{rel}(x)$ can be any type of function as long as the values are distributed over the interval [-1, 1]. For example, a Perlin noise function would be suitable to describe the relative change in thickness. Intuitively, this equation raises or lowers the average thin film thickness by values in the interval $[-d_{rel}, d_{rel}]$ depending on the position x on the surface. In

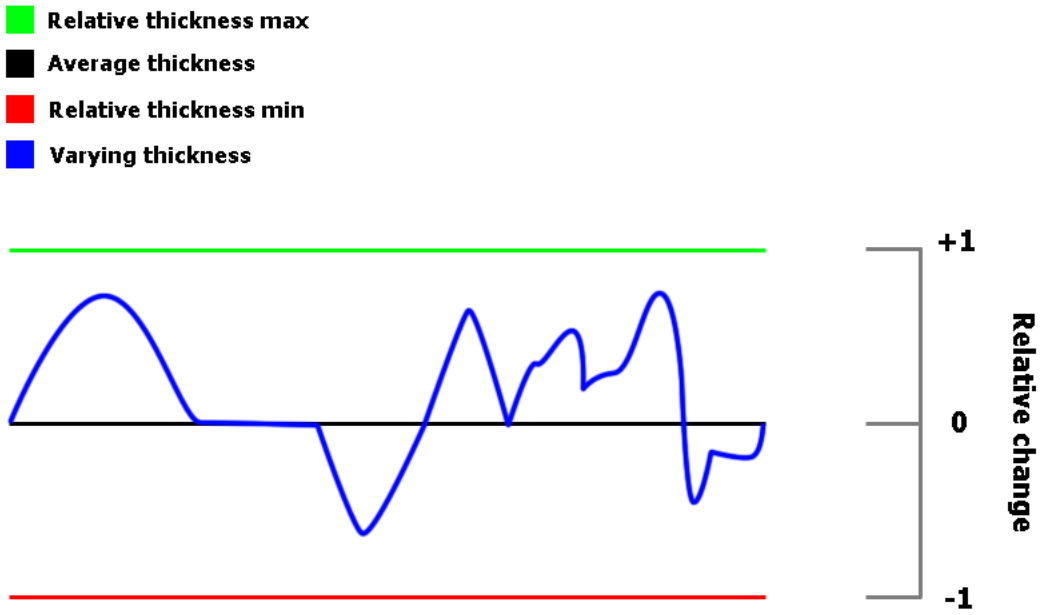

Figure 4.6: Spatially varying thickness.

Figure 4.6 the SVT parameters and functions are shown schematically. We can see that the relative thickness parameter d_{rel} controls the maximum and minimum allowed variation of the thin film thickness. Furthermore, we can see that the relative change function $\Delta_{rel}(x)$ directly influences how the thin film thickness will vary based on position. If $\Delta_{rel}(x)$ is zero, we see that we get the average thickness as the thin film thickness value for position x.

4.3 GPU Implementation

4.3.1 System Overview

An overview of the OpenGL renderer is shown in Figure 4.7. The renderer performs specific tasks on the CPU and the GPU.

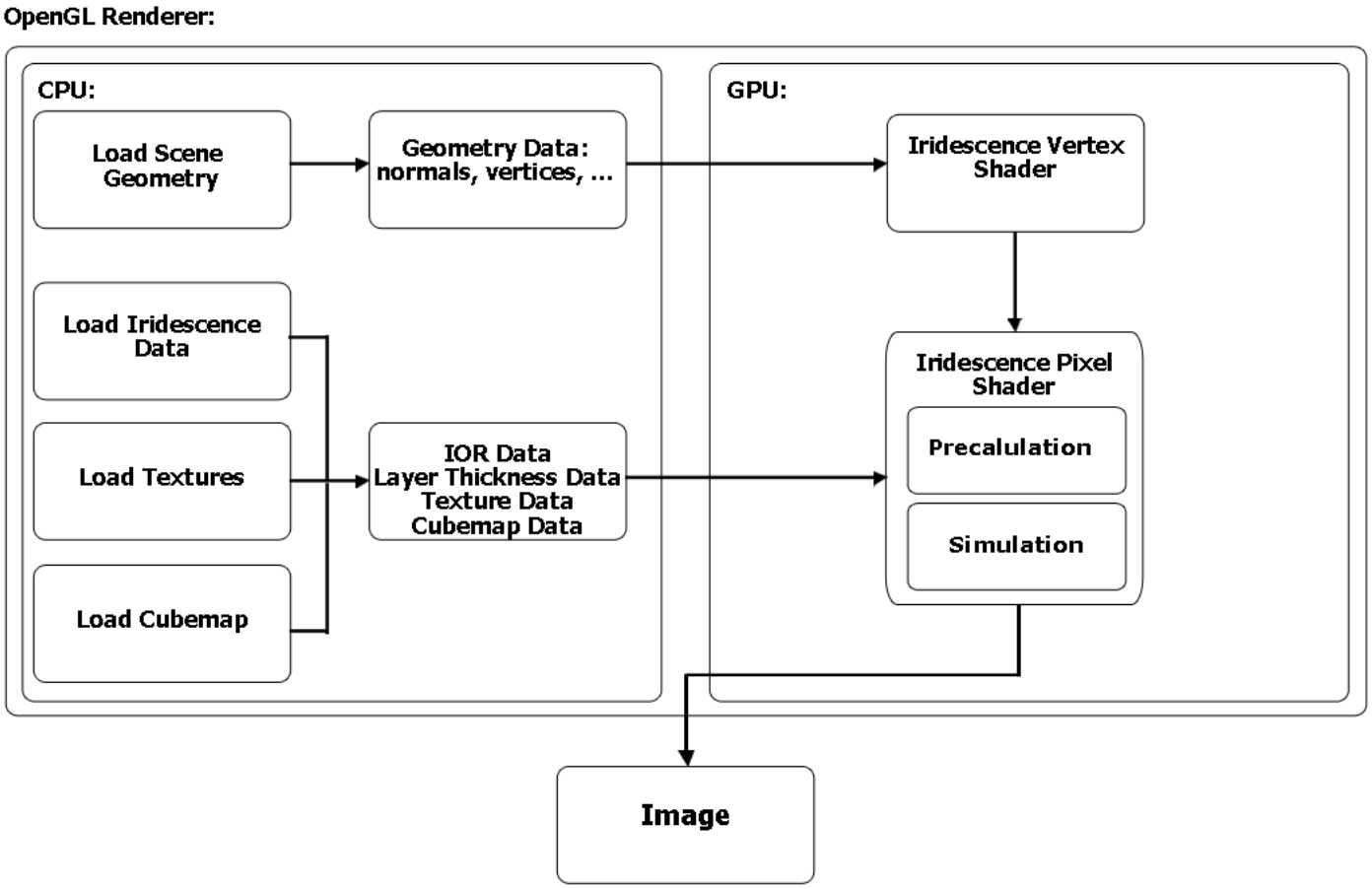

Figure 4.7: Overview of the OpenGL Renderer Pipeline.

Scene Descriptions

The CPU is relegated to performing tasks like loading data, assigning values to constant parameters and traversing the scene graph. All of these CPU tasks are done in a so called scene description *.cpp file. The scene description file is an abstraction in the renderer, responsible for loading a specific scene file from the hard drive and parsing the geometry, texture and shader information. A scene description is also used to setup parameters or functions unique to that scene. For example, we do not always need to have a cube map, because the scene could be indoors. Thus in that specific scene, we do not need to do the calculations for rendering a cube map. The scene description code for that particular scene will reflect this by not including the cube map loading and rendering code. This will make rendering specific scenes more efficient because we exclude unnecessary data and parameters. The parameters for thin film systems and SVT can also be defined and stored in the scene description code.

Sending Data

Once all the necessary data (geometry, textures, etc.) is loaded it can be send to the GPU. Transferring data between the CPU and the GPU is a costly operation. Because our data does not change over time, we send the data to the GPU at the initialization of the scene. At all subsequent rendering calls the GPU simply retrieves the data from its local memory. The vertex shader and pixel shader use the data to render the object.

4.3.2 Texture Based SVT

Texture based spatially varying thin film thickness is the application of the SVT theory (Equation 4.3) on the GPU. The average thickness d_{avg} is the thin film thickness of the layer. The relative thickness d_{rel} is a single constant thickness value that is send to the GPU alongside the other thin film simulation data. We use a texture with gray-scale colors to represent the relative change function $\Delta_{rel}(x)$. The first reason is that textures are efficient data structures for storing large amounts of data on the GPU. The second reason is that we can associate pixels of the texture with positions on the object through so called texture coordinates. The texture coordinates define how the pixels of the texture are mapped to the surface of the object. Thus every pixel of the texture can represent a relative change value for a specific position x on the surface of the object.

Authoring Pipeline

The creation of an SVT texture is straightforward. The object should first be projected to a

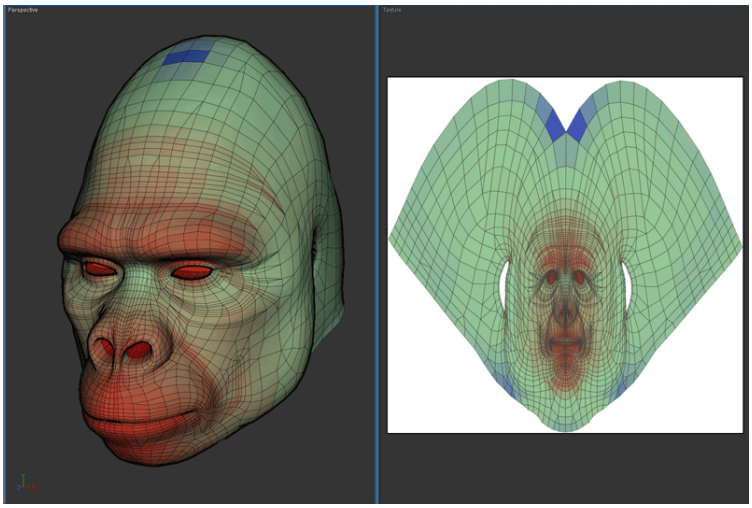

Figure 4.8: Left: A 3D object. Right: The unwrapped object.

texture. The most common way of doing this is to use UV unwrapping (Figure 4.8) in one of the several available modeling packages (3D Studio Max, Maya, Blender, etc.). The resulting UV texture should be saved. The object should be exported with texture coordinates. These coordinates provide the relation between a pixel of the texture and the surface of the object. Lastly, the texture should be painted using gray-scale colors where black denotes a relative change value of minus one and white denotes a relative change value of one.

Value Range Transformation

Color values are commonly in the interval [0, 255]. Color values in rendering typically are normalized to the range [0, 1]. When loading a color texture into OpenGL this normalization is done automatically. However, Equation 4.3 expects relative change values distributed over the interval [-1, 1]. We use the following transformation to relate color value α ([0, 1]) to relative change value Δ_{rel} ([-1, 1]):

$$\Delta_{rel} = \left(\frac{\alpha}{0.5}\right) - 1. \tag{4.4}$$

4.3.3 Iridescence Algorithm Overview

Inputs and Output

The output of the iridescence pixel shader is simply the pixel color value of the multi-layered thin film interference simulation. The iridescence pixel shader has several inputs. The texture coordinate, world space vertex position and world space normal are outputs of our iridescence vertex shader and thus they are inputs to the iridescence pixel shader. Furthermore, the camera world space position is send separately to the pixel shader. This camera position can be used in conjunction with the vertex position in world space to determine the direction vector incident to the surface. A cube map sampler is send to the pixel shader in order to sample color values from the cube map texture. These cube map color values represent the light source intensity. This is commonly called an image-based lighting technique. Three parameters pertaining to wavelengths are inputs to the pixel shader: the start wavelength, the end wavelength and the number of wavelength samples. The start and end wavelength parameters specify the wavelength sampling interval. The material parameters of the thin film system are also inputs. The material parameters are the average thin film thickness and the index of refraction of ever layer. Lastly, CIE XYZ matching curves data is send as an input. We will need these spectral response curves to convert our spectral reflectance and transmittance into colors.

CIE XYZ color space

The output of the simulation is reflectance and transmittance for N sampled wavelengths which is called spectral reflectance and transmittance respectively. However, for rendering purposes we would like to display this data on a monitor as RGB color values. By integrating the spectral

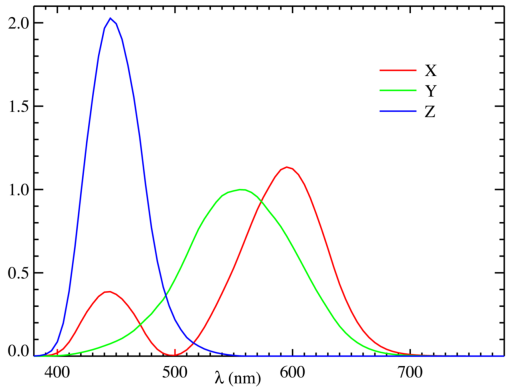

Figure 4.9: The CIE XYZ matching curves.

reflectance and transmittance over the CIE XYZ matching curves we obtain the XYZ color values for spectral reflectance and transmittance. The conversion from XYZ to RGB can be trivially computed using a conversion matrix.

Precalculation

The pseudo code in Algorithm 3 describes the calculation of several quantities ahead of time that are not dependent on wavelength. Note, that this precalculation is *not* a preprocessing stage, it is merely a way to reduce the number of calculations that needs to be performed. The precalculation stage is done for every pixel for every frame just like the simulation stage of the algorithm. The precalculation stage starts by sampling the SVT texture and retrieving the color

Algorithm 3 Iridescence GPU Simulation (precalculation stage)

1: $svt_{color} \leftarrow SampleSvtTexture(texturecoordinate)$
2: $\Delta_{rel} \leftarrow TransformValue(svt_{color})$
3:
4: **for** $n \leftarrow 1$ to $numLayers + 1$ **do** ▷ Precalculation:
5: Retrieve index of refractions n_i and n_t at interface between ith and $i-1$th layers
6: Retrieve layer thickness d_{avg} of ith layer
7: Calculate incident and transmitted directions $s_{z_{Inc}}$ and $s_{z_{Trans}}$ at interface
8:
9: $d \leftarrow d_{avg} + (\Delta_{rel} d_{rel})$ ▷ (Equation 4.3)
10: $phaseDiffPart \leftarrow s_{z_{Trans}} n_t d$ ▷ Partial phase difference calculation
11: $reflectivity \leftarrow CalculateReflectivity(n_i, n_t, s_{z_{Inc}}, s_{z_{Trans}})$ ▷ (Equations 3.7, 3.8)
12: $transmissivity \leftarrow CalculateTransmissivity(n_i, n_t, s_{z_{Inc}}, s_{z_{Trans}})$ ▷ (Equations 3.9, 3.10)
13:
14: Store $reflectivity$ in $reflectivities$ data structure
15: Store $transmissivity$ in $transmissivities$ data structure
16: Store $phaseDiffPart$ in $phaseDiffParts$ data structure
17: **end for**

value at the specified texture coordinate. The color value is transformed into the value range we use to describe relative change (Equation 4.4). The $numLayers$ variable is the number of layers the thin film system is composed of. If we have a thin film system with N layers then we have $N + 1$ interfaces or boundaries between layers. Thus $numLayers + 1$ is equivalent to the number of interfaces of the thin film system. We loop over all these interfaces and retrieve the index of refractions of the upper and lower layer at the current interface. The average thin film thickness of the lower layer is also retrieved from its data structure. Next, the cosine of the angle of incidence and transmission are calculated using Equation 3.6. These cosine angles are necessary in order to calculate the reflectivity and transmissivity using the Fresnel Equations. The thin film thickness due to the spatially varying nature of the material is calculated in the precalculation stage because the thin film thickness is not a wavelength dependent quantity. The phase difference is only partially precalculated, because it is dependent on the wavelength of light. Lastly, the reflectivity, transmissivity and partial phase difference are stored in data structures so we can access them later.

Simulation

Algorithm 4 Iridescence GPU Simulation (simulation stage)

18: **for** $i \leftarrow 0$ to $numWavelengths$ **do** ▷ Simulation:
19: $wavelength \leftarrow SampleWavelength(i)$
20: $compositeRefl \leftarrow reflectivities[numLayers]$ ▷ Initial composite reflectivity
21: $compositeTrans \leftarrow transmissivities[numLayers]$ ▷ Initial composite transmissivity
22: **for** $j \leftarrow numLayers - 1$ to 0 **do**
23: $reflectivity \leftarrow reflectivities[j]$
24: $transmissivity \leftarrow transmissivities[j]$
25: $\varphi_j \leftarrow \left(\frac{2\pi}{wavelength}\right) phaseDiffParts[j]$ ▷ (Equation 3.5)
26:
27: $prevCompRefl \leftarrow PolarToRectangular(compositeRefl, 2\varphi_j)$ ▷ (Equations 2.30, 2.31)
28: $prevCompTrans \leftarrow PolarToRectangular(compositeTrans, \varphi_j)$
29: $compositeRefl \leftarrow DoReflRecurrence()$ ▷ (Equation 3.1)
30: $compositeTrans \leftarrow DoTransRecurrence()$ ▷ (Equation 3.2)
31: **end for**
32: $reflectance \leftarrow CalculateReflectance()$ ▷ (Equation 3.12)
33: $transmittance \leftarrow CalculateTransmittance()$ ▷ (Equation 3.13)
34: Multiply reflectance and transmittance with CIE XYZ Curves
35: Accumulate result into $xyzRefl$ and $xyzTrans$ respectively
36: **end for**
37: ▷ Illumination Model/Rendering
38: $finalReflectColorXYZ \leftarrow xyzRefl \times reflectColorXYZ$
39: $finalRefractColorXYZ \leftarrow xyzTrans \times refractColorXYZ$
40: Convert XYZ colors to RGB values
41: $pixelColor \leftarrow finalReflectColorRGB + finalRefractColorRGB$

The pseudo code in Algorithm 4 describes the calculation of the composite reflectance and composite transmittance. We loop over all sampled wavelengths and calculate the current wavelength sample λ accordingly. The initial composite reflectivity and transmissivity are retrieved from the data structures that store the reflectivity and transmissivity for every layer (Algorithm 3). Next, we loop over all layers starting from the medium B layer and ending at the medium A layer. In every iteration, we retrieve the reflectivity and transmissivity of the jth interface. We also calculate the wavelength dependent part of the phase difference equation. The previous composite reflectivity $\gamma_{N-j}e^{2i\varphi_j}$ and previous composite transmissivity $\tau_{N-j}e^{i\varphi_j}$ are in polar form, where the reflectivity/transmissivity is the modulus and the phase difference is the argument. The function $PolarToRectangular$ converts these polar coordinates into values in the rectangular coordinate system. Once we know the reflectivity, transmissivity, previous composite reflectivity and previous composite transmissivity we can use these values in the recurrence equation to get a new value for the composite reflectivity and transmissivity. The final composite reflectivity and transmissivity for wavelength λ are converted into composite reflectance and composite transmittance using the appropriate equations. The composite reflectance and composite transmittance for wavelength λ are then used in the integration for the CIE XYZ color values $xyzRefl$ and $xyzTrans$. The XYZ color values can be used once the integration of reflectance and transmittance has taken place for every sampled wavelength. The

$reflectColorXYZ$ and $refractColorXYZ$ are the RGB colors sampled from the cube map in the reflection and refraction directions, converted from RGB to XYZ color space. The intensities sampled from the cube map and the reflectance/transmittance from the simulation need to be in the same color space to be able to multiply their values. We can now use the smooth illumination model (Equation 3.14) to determine the final pixel color.

4.4 Constraints

In order to make the algorithmic changes, some assumptions have been made. With these assumptions we are able to achieve significant performance increases compared to the previous algorithms described in the main reference papers [8] and [9]. The following assumptions were made:

1. Sparse wavelength sampling.
2. Media are strictly dielectric.
3. Media are wavelength invariant.

The sparse wavelength sampling is a constraint that does not necessarily need to be enforced. It is however highly recommended in order to improve the overall performance of the algorithm significantly. The key is to choose the number of wavelength samples in such a way that the decrease in visual quality is negligible while keeping the wavelength sample count small. The assumption is made that all materials in the thin film system should be dielectric. By assuming strictly dielectric materials, we do not need to represent quantities like reflectivity/transmissivity using complex numbers. Instead, we can use real-valued numbers for all but one quantity (composite reflectivity/transmissivity). This results in improved performance because we do not need to use the more expensive complex number operations for basic arithmetic. The final assumption states that all materials in the thin film system should not be dependent on wavelength. This gives us the ability to change the algorithm and add the precalculation stage because reflectivity and transmissivity will not be dependent on wavelength anymore. Doing this precalculation stage increases performance by reducing the number of calculations that need to be done. An example: given a thin film system with 2 thin film layers, which means we have 3 interfaces in total. The number of wavelength samples in this example is set to 30. In the previous algorithm, this would mean the reflectivity would be calculated $3 \times 30 = 90$ times. However, because the media are now wavelength invariant the reflectivity is only calculated 3 times in our algorithm. Furthermore, if the number of wavelength samples is increased in our algorithm, the number of reflectivity calculations does not change. The same example can be used to argue the decrease in necessary transmissivity calculations with our algorithm.

Chapter 5

Experimentation and Evaluation

5.1 Quantitative Analysis

5.1.1 Simulation Validation

We validate the original multi-layered thin film interference method, proposed by Hirayama et al., in order to find out if this method is based on real world thin film interference effects. The PBRT implementation of the smooth illumination model is used in this experiment. We will be comparing it to data obtained from Filmetrics. Filmetrics is a company providing thin film measurement systems that can measure index of refraction, absorption coefficient, thin film thickness and much more of real world thin film samples. On their website they provide a calculator that computes the spectral reflectance due to thin film interference. The input is a configuration of several thin film layers and a top and bottom medium. The output is a graph plotting the spectral reflectance of the thin film system. The reflectance calculator is actual software that is used in their thin film measurement systems in order to do retrieve certain properties like index of refraction or absorption coefficient from the measured data. This is the reason why we can use this calculator in order to validate the simulation.

Hypothesis: The authors of the original paper do not provide any proofs for the physical basis of the algorithm. We want to know if our method is physically plausible. If we can find out if the original method is physically plausible, this means that our method is physically plausible as well because the core algorithm remains the same. We hypothesize that the original method proposed by Hirayama et al. is a physically plausible rendering method to simulate thin film interference effects.

The Experiment

We will use the Filmetrics spectral reflectance calculator to obtain spectral reflectance plots of several different thin film systems. We will simulate these thin film systems in our PBRT simulation and construct the spectral reflectance plots from the simulation data. A comparison between the reflectance calculator plots and PBRT simulation plots will be made. The amount of wavelengths we will sample in the PBRT simulation is 30 wavelengths between 400 and 700 nanometers. This means we have a data point in the spectral reflectance graph every 10 nanometers. We will use the same data point spacing and wavelength sampling interval for the graphs obtained from the Filmetrics spectral reflectance calculator. The spectral reflectance

data of the PBRT simulation is taken from an incident angle of one degree with respect to the normal. The Filmetrics spectral reflectance calculator data is calculated for normal incidence.

Results

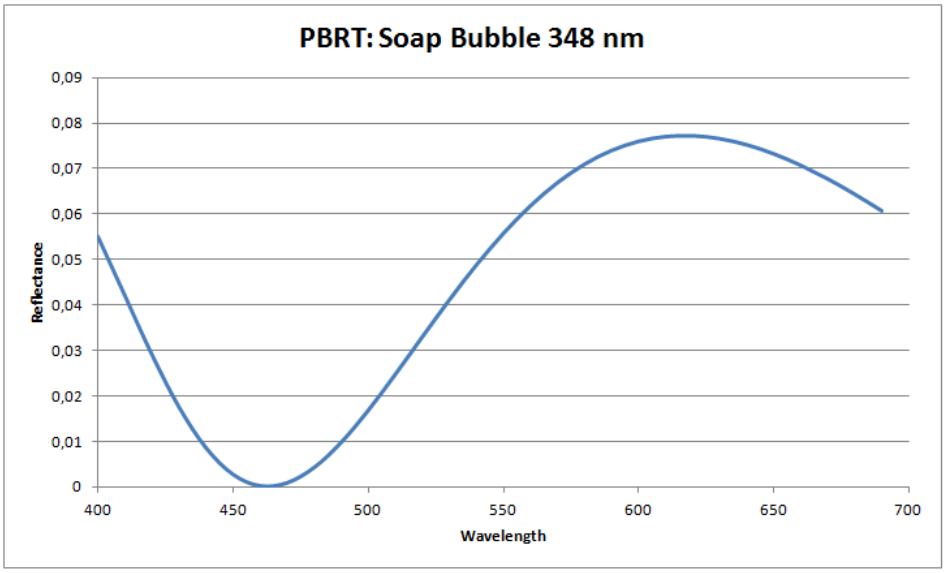

Figure 5.1: PBRT simulation data of soap bubble with a 348 nm water thin film.

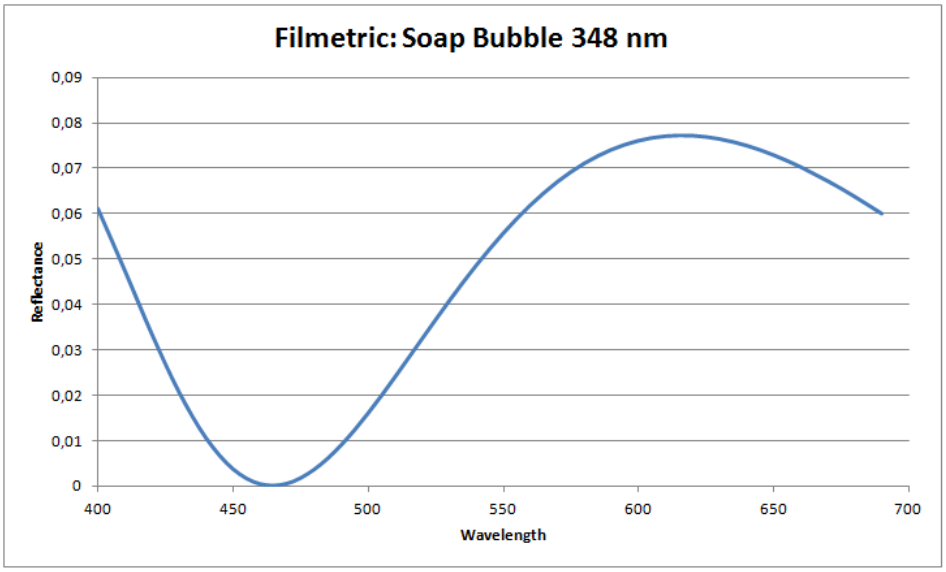

Figure 5.2: Filmetrics data of soap bubble with a 348 nm water thin film.

Soap Bubble Figures 5.1 and 5.2 show the spectral reflectance graphs of a soap bubble. The thin film system is composed of a top medium of air ($n = 1.0$), a thin film of water ($n = 1.33$) with a thickness of 348 nanometers and a bottom medium of air. We can see that the two plots are nearly identical in terms of spectral reflectance distribution. However, we see that around 400 nanometers the reflectance is higher in the Filmetrics plot. When the thickness of

the water thin film changes, so does the spectral reflectance distribution. In Figures 5.3 and 5.4 the spectral reflectance graphs for a soap bubble with a 635 nanometers thick water thin film is plotted. The thin film system for the soap bubble is still the same; only the thin film thickness has been changed. In these plots we observe again that the spectral reflectance distributions

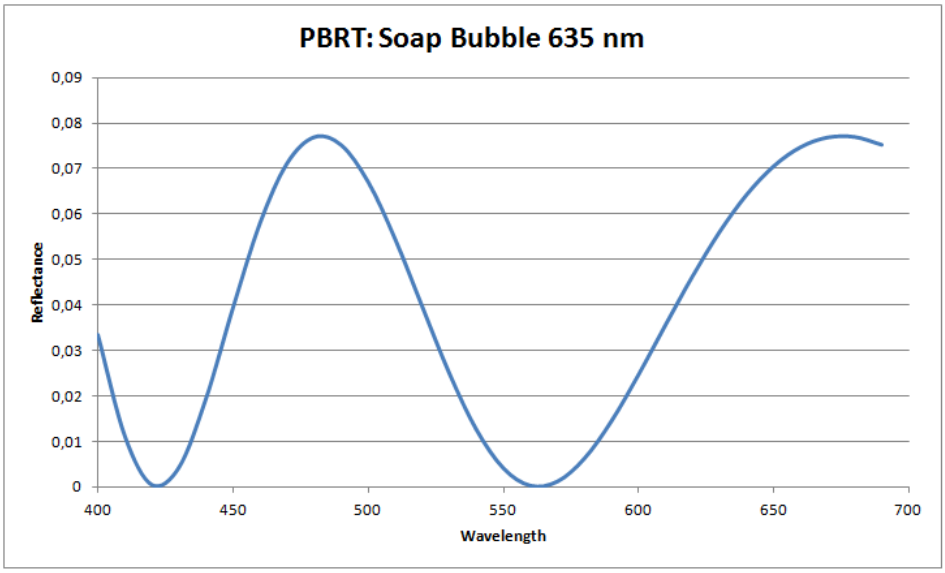

Figure 5.3: PBRT simulation data of soap bubble with a 635 nm water thin film.

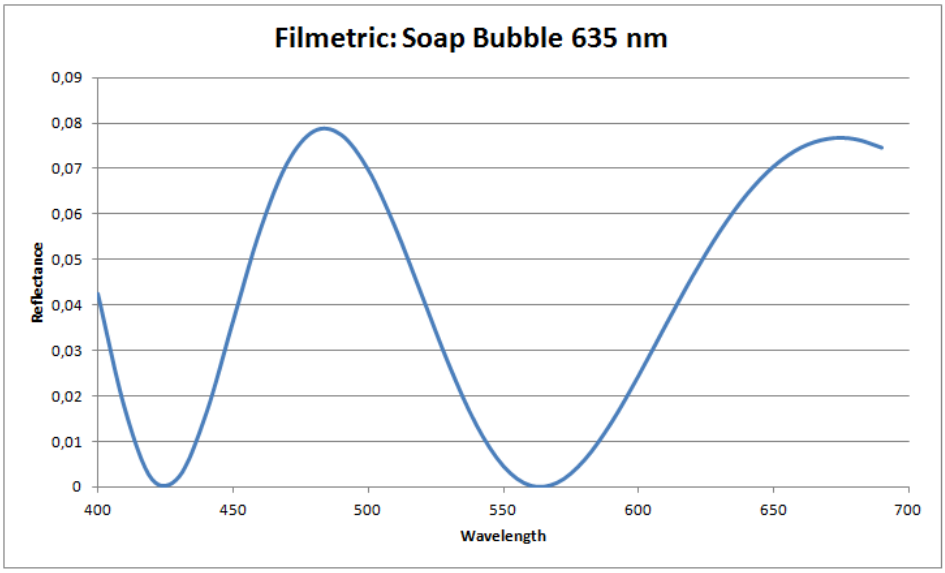

Figure 5.4: Filmetrics data of soap bubble with a 635 nm water thin film.

are nearly the same with the exception being around 400 nanometers. Figures 5.1 through 5.4 show that in terms of the simulation of dielectric single-layered thin film systems the error is negligibly small and the simulation data fits well to the Filmetrics data. We have proven that the simulation remain physically plausible with single-layered thin film systems.

Oil Slick We will now proof that the simulation remains physically plausible when we simulate multi-layered thin film systems. The thin film system is based on an iridescent oil slick, which typically are composed of gasoline and water thin films. The top medium is air ($n = 1.0$), the first thin film layer is gasoline ($n = 1.4$) with a thickness of 216 nanometers, the second thin film layer is water ($n = 1.33$) with a thin film thickness of 220 nanometers and the bottom medium is asphalt ($n = 1.635$). Figures 5.5 and 5.6 show the spectral reflectance graphs for the iridescent oil slick. A clear difference in reflectance can be seen in the 450 to 500 nanometer

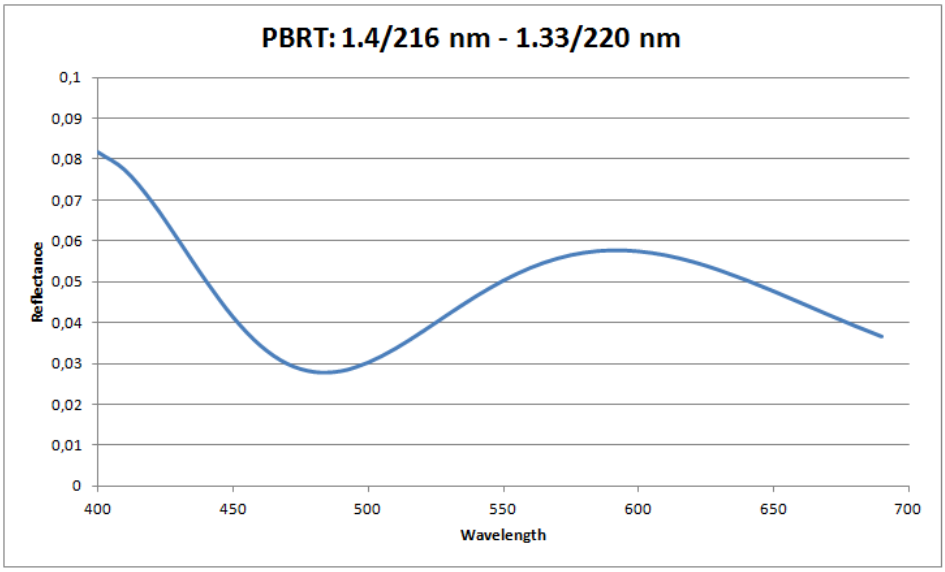

Figure 5.5: PBRT simulation data of an iridescent oil slick.

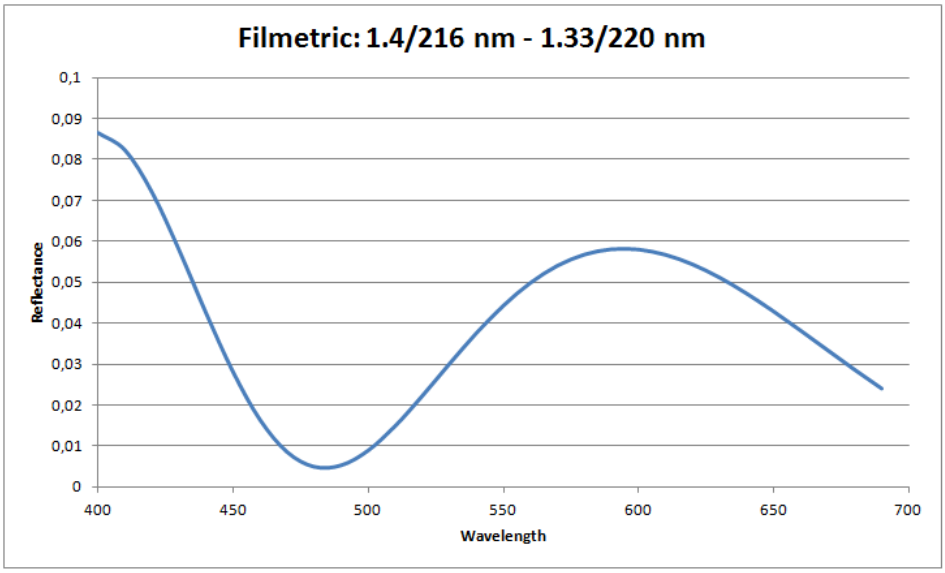

Figure 5.6: Filmetrics data of an iridescent oil slick.

range. The simulation has a higher reflectance in that range compared to the Filmetrics data. The most likely cause for this is that the simulation uses single-precision floating points for values. It is very likely that the spectral reflectance calculator uses the more accurate but slower

double-precision floating points. However, the overall shape of the spectral reflectance distribution aside from the 450 to 500 nanometer range remains similar to the Filmetrics data. Furthermore, the spectral reflectance in both graphs is low since it never goes above a reflectance of 0.09. This means that the difference between the plots is also relatively small. We will be using the simulations in graphics rendering. Thus small differences in reflectance values compared to the real world will not be significant since it will most likely not be discernible in the rendered image. We have proven that the simulation remain physically plausible with multi-layered thin film systems.

Metallic Thin Films This experiment will test whether or not a semiconductor/metallic thin film can be used in a thin film system in the simulation. Metals generally have very high reflectance, thus by doing this experiment we can see if the algorithm remains energy conservative. Energy conservation is one of the most important properties of physically based rendering algorithms. The thin film system we will use is composed of a top medium of air ($n = 1.0$), a thin film of silicon with a thickness of 108 nanometers and the bottom medium is glass ($n = 1.5$). The silicon is described by measured data and thus the index of refraction and absorption coefficient vary over the wavelength spectrum. In Figures 5.7 and 5.8 we see

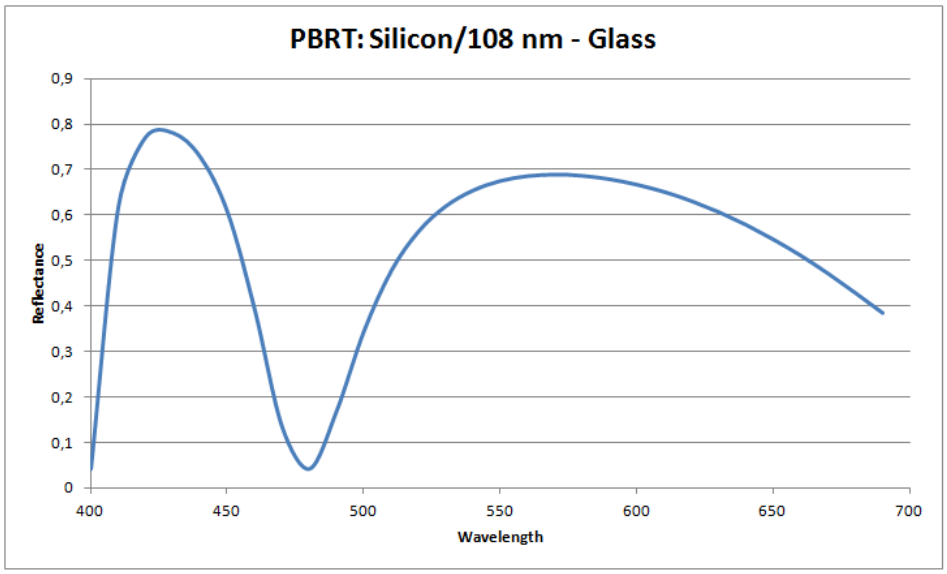

Figure 5.7: PBRT simulation data of a silicon thin film on glass.

the spectral reflectance distribution graphs of a silicon thin film on glass. We see that even in this case the error between the graphs is small. The error is most noticeable at the 400 to 500 nanometer range. There are many databases that contain measured complex refractive index data. Furthermore, differences in silicon samples or measurement equipment will also affect the measurements. Thus the most likely cause for the discrepancies in spectral reflectance are because the measured silicon data for the simulation and the spectral reflectance calculator are different. The overall distribution of the spectral reflectance however seems to conform well to the Filmetrics data. We have proven that the simulation can give physically plausible results even for thin films with complex index of refractions.

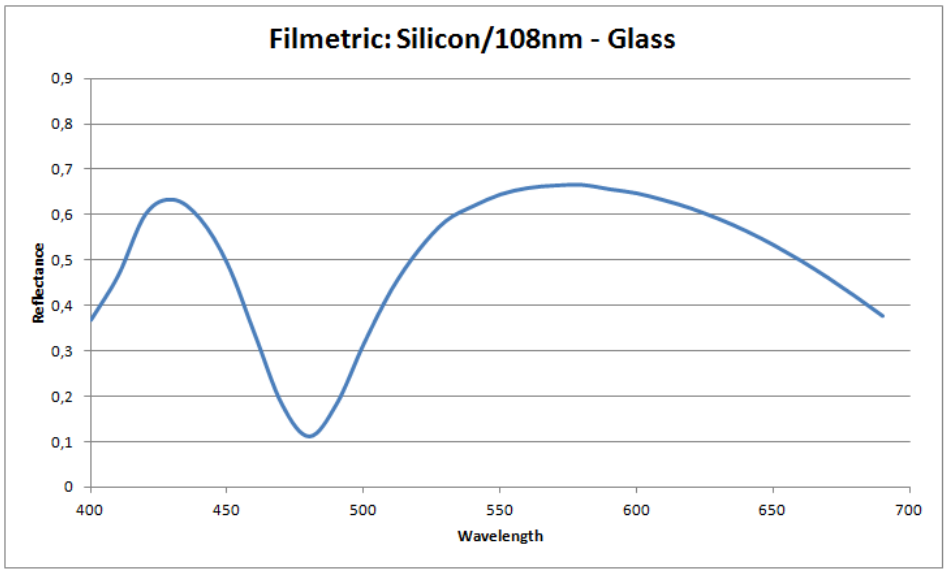

Figure 5.8: Filmetrics data of a silicon thin film on glass.

5.1.2 Performance Optimizations

In order to get the algorithm to run smooth on the GPU, we had to do a couple of optimizations. The two most significant optimizations are based on the assumptions that the index of refractions are wavelength invariant and that we only assume dielectric materials in the thin film system. We can use the wavelength invariance assumption in a N-layered thin film system to only calculate the reflectivity and transmissivity $N + 1$ times. This is opposed to the original method, where we would have to calculate reflectivity and transmissivity $N \times M$ times, where M denotes the number of sampled wavelengths. The dielectric materials assumption makes it so that nearly all equations simplify to real-valued numbers, e.g. we do not need to represent reflectivity, transmissivity or index of refractions using complex numbers.

Hypothesis: We would like to compare the average rendering times in milliseconds between the original method, proposed by Hirayama et al., and our method. Our method is composed of two optimizations that may or may not increase the overall performance of the algorithm. We want to measure the significant increase (or decrease) in performance these optimizations will give to the algorithm. We hypothesize that complex numbers are the biggest bottleneck in achieving significant real-time performance of the algorithm.

The Experiment

We show the effect our two optimizations have on the overall performance of the algorithm and compare them to the original method. The original method was implemented on the CPU in the ray tracer PBRT. However, comparing the PBRT implementation with GPU implementations would not be fair. Generally rendering can be performed significantly faster on the GPU depending on the algorithm. Furthermore, PBRT is an off-line ray tracer which focuses more on accuracy than speed. Thus we have implemented the original method as a shader on the GPU in our renderer for the purpose of comparing the performance gains (or losses) of our algorithmic changes. No changes were made to the theory or the algorithm of the original method.

Scene The scene contains a moving camera orbiting around an object coated with a thin film system. The thin film system we use is based on an iridescent oil slick and has index of refractions $n = 1.4$ and $n = 1.33$ for the gasoline and water thin film layers respectively. The thin film thicknesses are set to 216 nanometers for the gasoline thin film and 230 nanometers for the water thin film. The top and bottom media are air ($n = 1.0$) and asphalt ($n = 1.635$) respectively. The object we are rendering is a surface of approximately 8000 polygons generated with Perlin noise. Figure 5.9 shows the wireframe of the object. We set the number of wavelengths samples to 5, which is an empirically defined value for the minimum number of wavelength samples while preserving the visual quality of the output (see section 5.2.2). The measurements are performed at a screen resolution of 1080p (1920x1080 pixels). The iridescent object occupies about 1/4th of the screen in terms of pixels, meaning the iridescent shaders need to be executed for approximately 500000 pixels. Therefore, the test reflects the upper bound of the algorithm because in most practical cases these iridescent objects will just be a small part of a large scene.

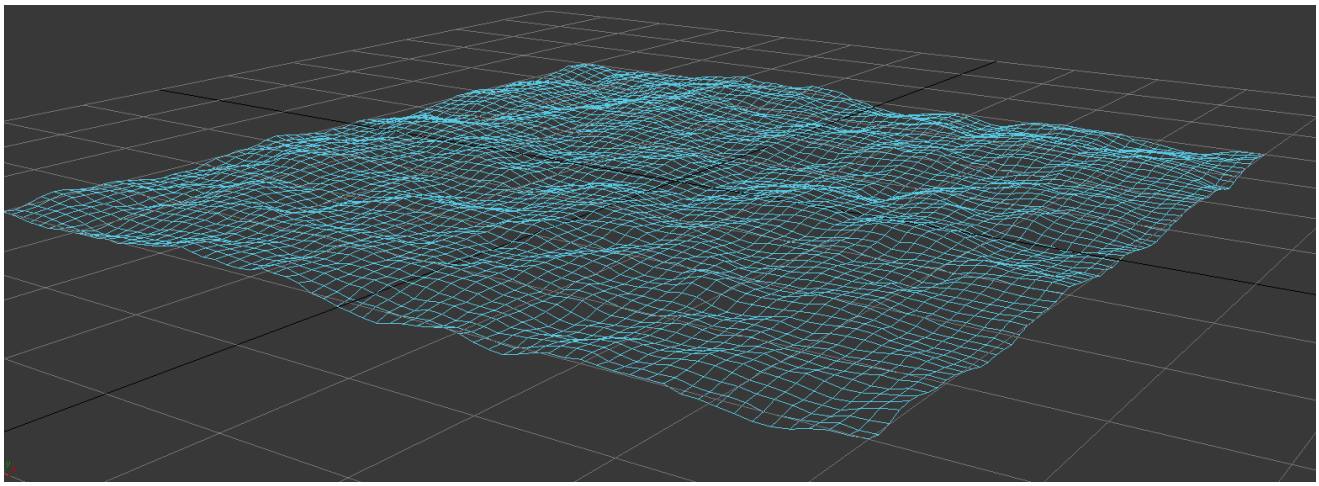

Figure 5.9: Wireframe of experiment object.

Measurement A profiler called gDEBugger (version 5.8.1) is used in the experiments to measure the time it takes to render a frame in milliseconds. We measure over a fixed period of time of approximately a minute for every algorithm. The data is compiled and an average, minimum and maximum frame time in milliseconds is the result for every algorithm. Furthermore, we graph the measurements over time to show the stability of the algorithms.

Algorithms We have 4 different algorithms we will perform the experiment on. We have the original method proposed by Hirayama et al. implemented on the GPU, which we will call the *naive* algorithm. We have the algorithm where we assume refractive indices to be wavelength invariant, which we will call the *precalculation* algorithm. We have the algorithm where we assume strictly dielectric materials, which we will call the *real-valued numbers* algorithm. Lastly, we have the final version of our algorithm where we have combined both optimizations, which we will call the *optimized* algorithm. These are simply names we will use throughout the experiment to differentiate the different methods and versions of the algorithm.

Hardware The specifications of the test system are provided in Table 5.1. We have used a desktop PC that is generally considered to be an average gaming PC as of the time of writing.

	Desktop
CPU:	Intel Core i7 920 (2.67 GHz)
GPU:	ATI Radeon HD5850
RAM:	4 GB DDR3

Table 5.1: Hardware specifications of test system.

Results

In Table 5.2 the results of the performance measurements are shown. By assuming wavelength

Algorithm	Average	Minimum	Maximum
Naive:	9.85	9	10
Precalculation:	6.92	6	7
Real-valued numbers:	3.71	3	5
Optimized:	4	4	4

Table 5.2: Rendering times in milliseconds.

invariant refractive indices in the *precalculation* algorithm we have increased the performance of the algorithm by approximately 30% over the *naive* algorithm. In the *real-valued numbers* algorithm we found an increase in performance of approximately 62% over the *naive* algorithm. Lastly, by combining both optimizations in our *optimized* algorithm we have increased the overall performance of the algorithm by approximately 59% over the *naive* algorithm. The experiment results clearly show that the removal of complex numbers from the equations gives the largest increase in rendering speed of the algorithm. We actually see a very small decrease in performance (approximately 3%) in our *optimized* algorithm compared to the *real-valued numbers* algorithm. This is most likely caused by optimizations the GLSL shading language compiler cannot do in the *optimized* algorithm which it can in the *real-valued numbers* algorithm. The main difference between the *optimized* and *real-valued numbers* algorithms is that the former has an extra iterative part where the precalculation is done. This suggests that compiler optimizations are heavily dependent on code structure since the only significant difference between the algorithms is a slightly different algorithm structure. While the *real-valued numbers* algorithm might have a slightly faster average rendering time compared to the *optimized* algorithm, we see something interesting when we look at the rendering speed over time. In Figure 5.10 we can see that the *optimized* algorithm (purple line) remains constant over time while the frame time of all the other algorithms fluctuates. Thus the *optimized* algorithm is a more stable and reliable algorithm in terms of performance. We have proven that in our case the removal of complex numbers from the algorithm gave us the most significant performance increase.

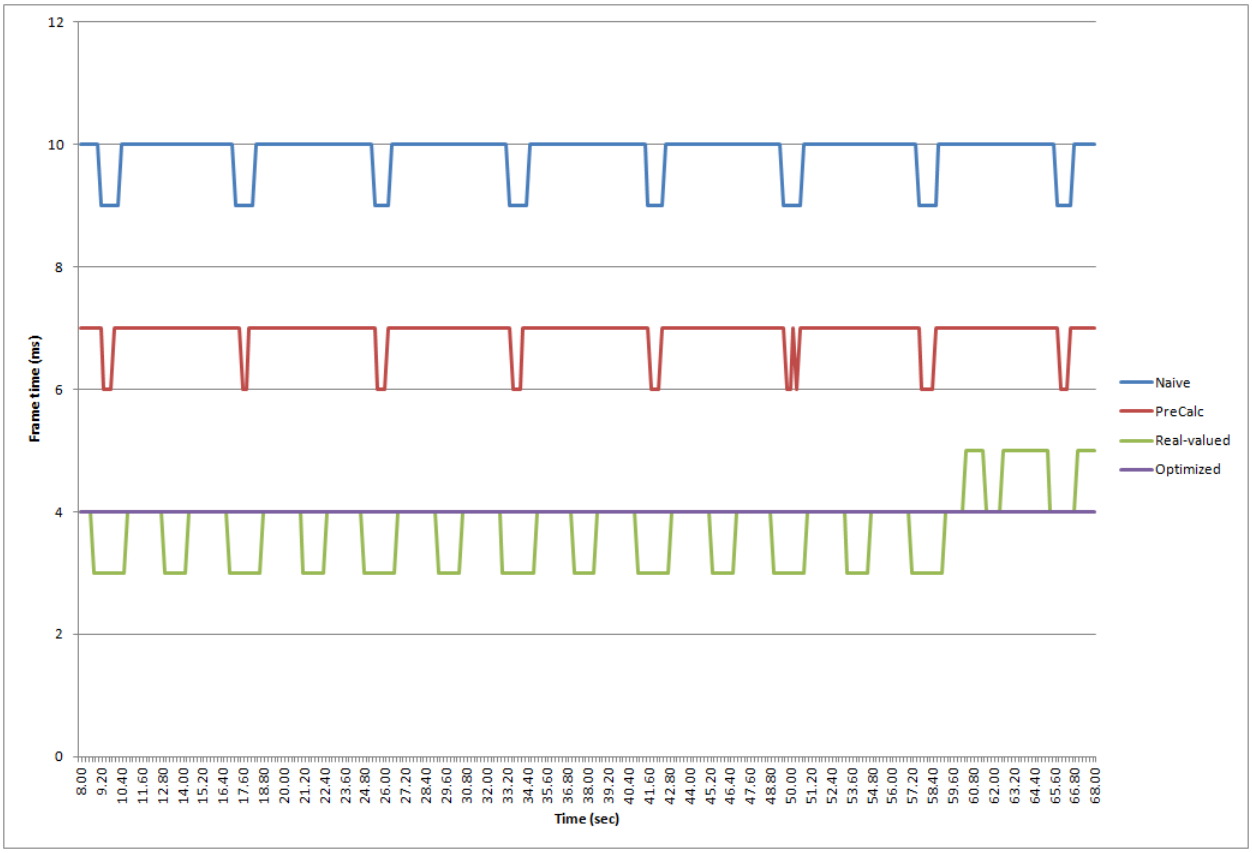

Figure 5.10: The performance of the algorithms (in milliseconds) over time.

5.2 Qualitative Analysis

5.2.1 Soap Bubble Simulation Validation

We have quantitatively validated the CPU implementation of the original method proposed by Hirayama et al. However, since we are concerned with rendering iridescent objects, we would also like to validate the simulation qualitatively. We will look at how the thickness of the water thin film influences the colors appearing on a soap bubble. To test the validity of the simulation, we will compare this to a chart showing the colors of a soap bubble film at several different thin film thicknesses. The chart, shown in Figure 5.11, was created by the German scientist Björn Böttcher of the Dresden University of Technology.

Hypothesis: In order to quantitatively validate our multi-layered thin film interference algorithm, we will compare the rendered output to the chart in Figure 5.11. We hypothesize that the colors appearing on the surface of the simulated soap bubble object will conform to the results shown in the chart.

The Experiment

In this experiment we will use our method implemented on the GPU to simulate a soap bubble. The thin film system we are using has a single thin film layer of water ($n = 1.33$). The top and bottom medium are both air ($n = 1.0$). During the experiment we will change the thickness of

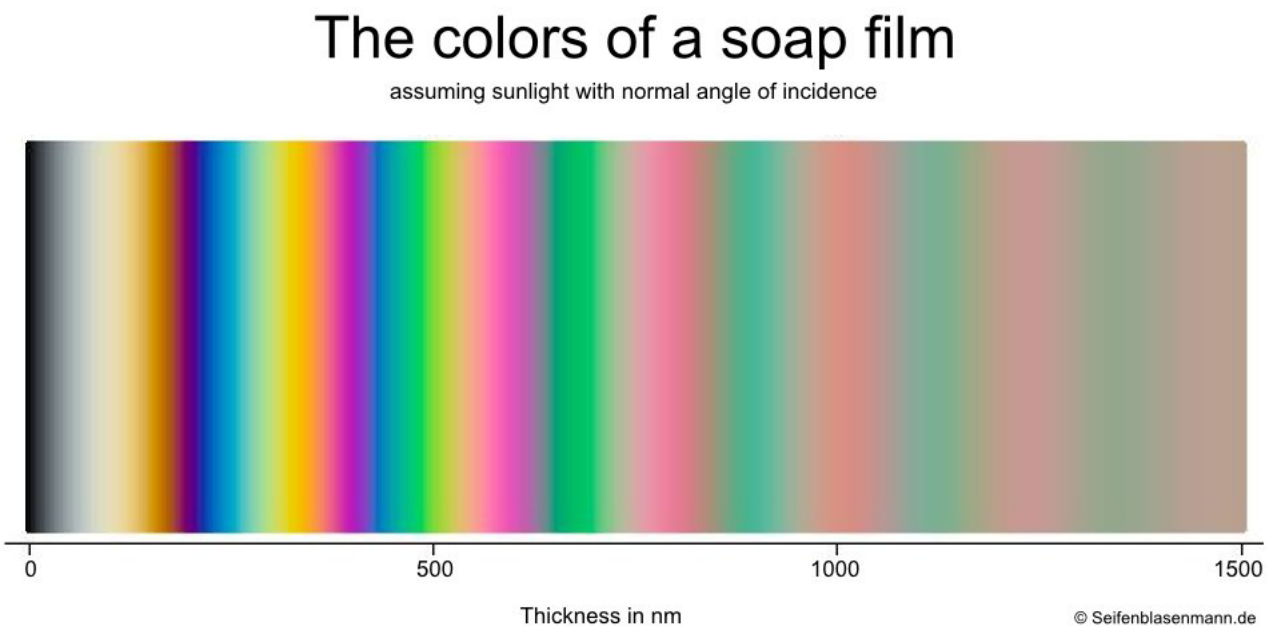

Figure 5.11: A chart showing the thin thickness influence on the color of the object.

the water thin film layer and show the visual output for each thickness. These results are then compared to the chart of soap bubble colors (Figure 5.11). We will take 30 wavelength samples for this experiment in order to obtain accurate iridescent colors in the rendered images.

Results

Figures 5.12 and 5.13 show the results of our soap bubble simulation. Our simulation results compare favorably to the soap bubble data obtained at a normal angle of incidence even though our simulation takes into account all possible angles of incidence. We can clearly see that the overall appearance of the bubble is dim at a low thin film thickness (50 nanometers) which is shown in the chart as a gray color. By increasing the thickness to 100 nanometers, we see the reflections become more apparent; this is reflected in the chart by the shift from gray to white color. For even higher thin film thicknesses, the chart shows a shift from white to yellow followed by orange. You can clearly see the same shifts in color in our simulation at 150 nanometers and 250 nanometers thicknesses respectively. At approximately 250 nanometers in the chart we see a shift to purple followed by a sudden shift to dark blue. Similarly, in our simulation at 250 nanometers we can clearly see the purple color appearing on the object and when increasing the thin film thickness to 300 nanometers we start to see the dark blue color emerge. Lastly, at the end of the color chart starting around a thickness of 1000 nanometers we find something unusual. Green and pink colors alternate in the chart and become increasingly dull when the thin film thickness gets larger. In our simulation you can see the same effect occurring. At a thickness of 2000 nanometers in our simulation the green and pink colors are still there but they have become so dull that it is hard to perceive them as these colors. We have qualitatively evaluated our simulation and have proven that it compares favorably to experimentally obtained qualitative data.

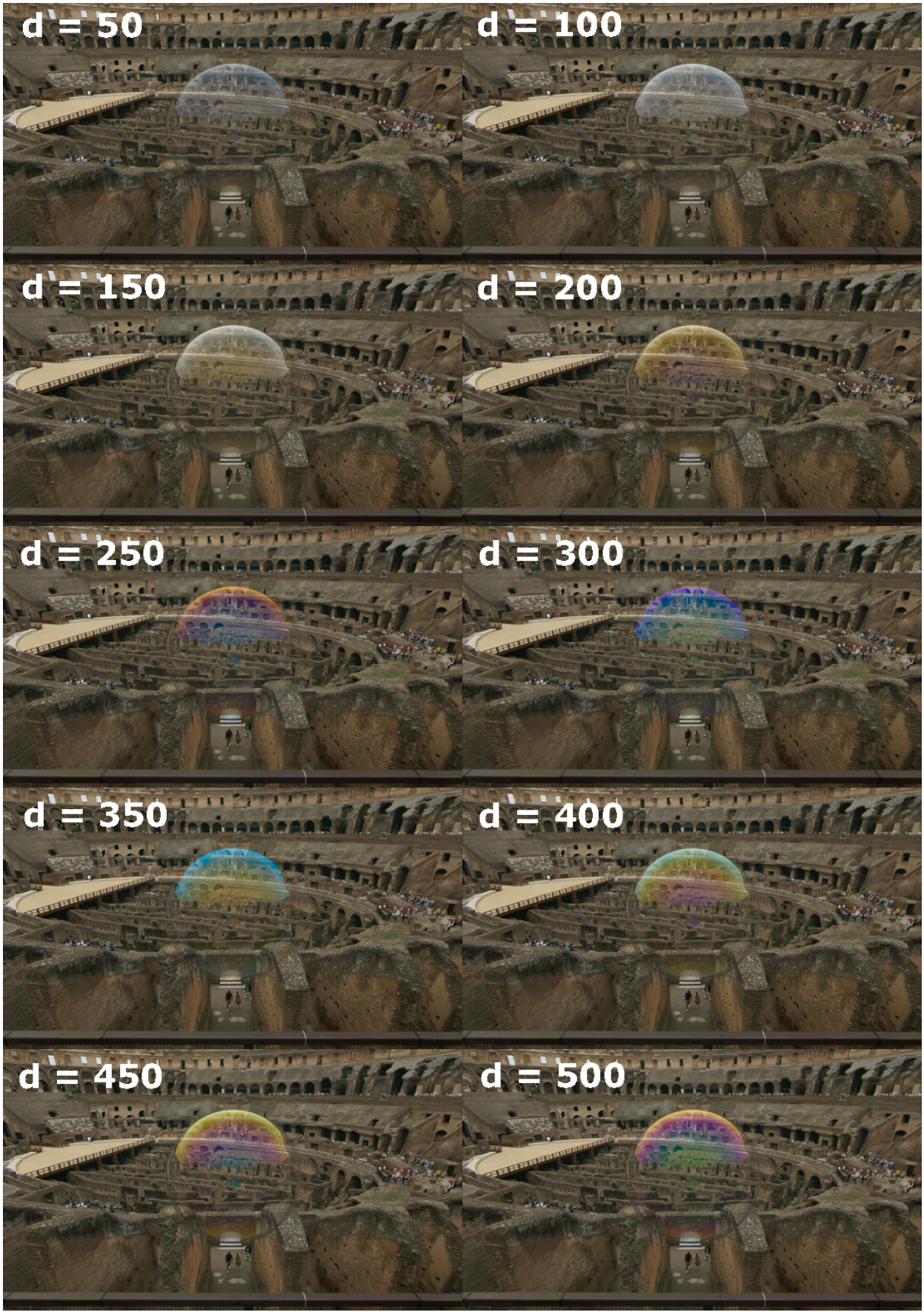

Figure 5.12: Varying the thin film thickness of a soap bubble.

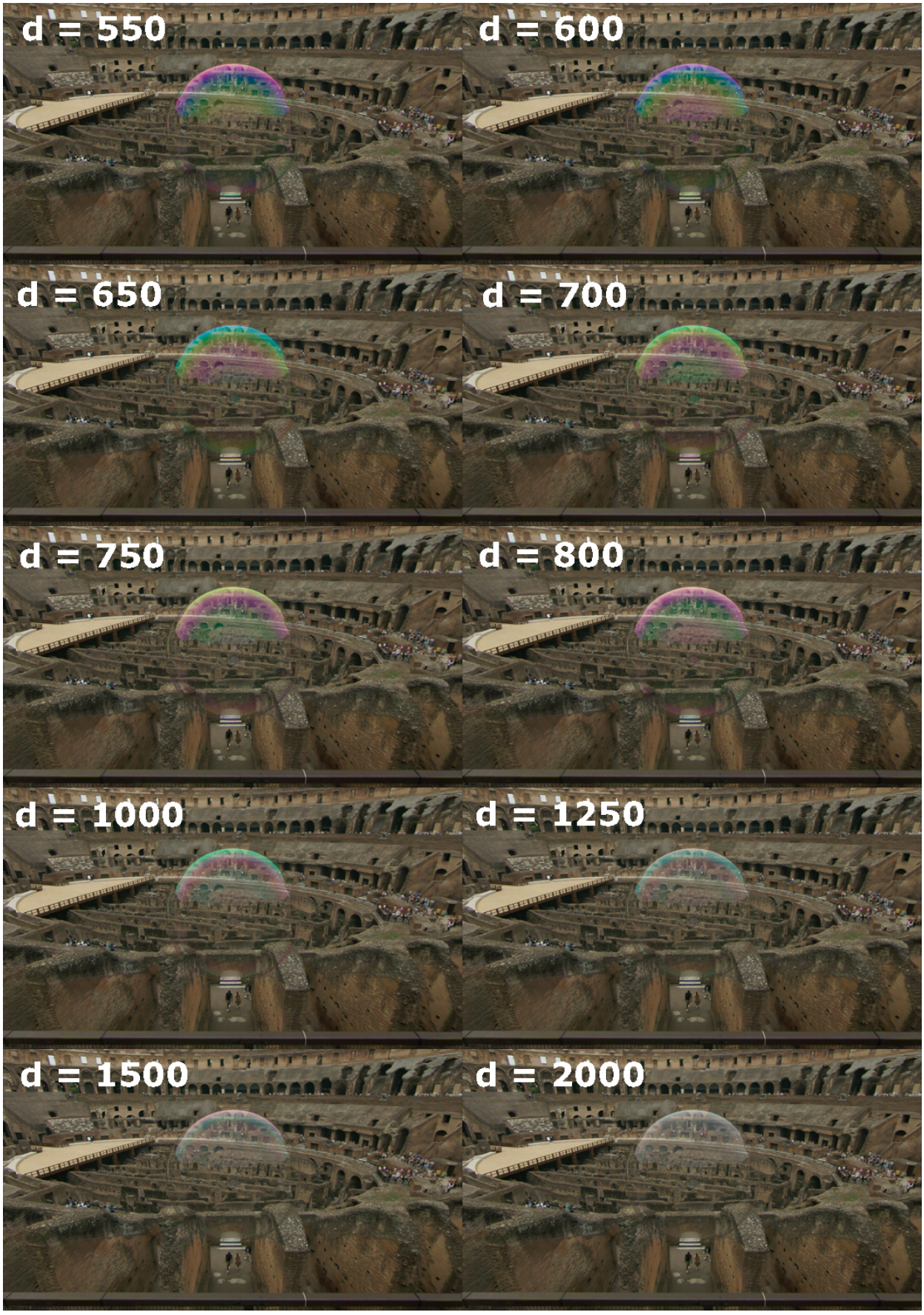

Figure 5.13: Varying the thin film thickness of a soap bubble.

5.2.2 Sparse Wavelength Sampling

In order to optimize the algorithm even further, we would like to limit the number of wavelength samples taken. The number of wavelength samples dictates how many times the algorithm runs.

Hypothesis: By limiting the number of wavelength samples, the performance of the algorithm will increase while the overall image quality will decrease. We want to find a lower bound for the number of wavelength samples while keeping the visual quality high. We hypothesize that the number of wavelength samples taken greatly affects the quality of the visual output of the simulation.

The Experiment

We will perform the experiment empirically using our method implemented on the GPU with SVT on. The relative change function we will use is a Perlin noise texture shown in Figure 5.14. The relative thickness parameter is set to 95 nanometers. Wavelengths will be sampled between the 400 and 700 nanometer interval. Every time we change the number of wavelength samples, we will show the resulting visual output. In the end, based on the results, we will define the minimum number of wavelength samples that should be taken to have good performance while preserving the quality of the visuals.

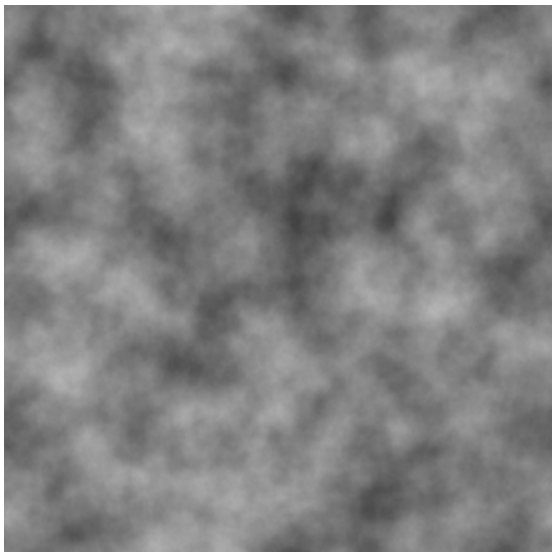

Figure 5.14: The relative change function we use throughout the experiment.

Results

In Figure 5.15 the results of the sparse wavelength sampling experiment are shown. We can see that number of wavelength samples only seems to affect the quality significantly when less than 10 wavelength samples are used. We have empirically defined 5 wavelength samples to give the best quality versus performance trade off. It is nearly identical in visual quality when we compare it to the output with 10 wavelength samples. However, 5 samples is the

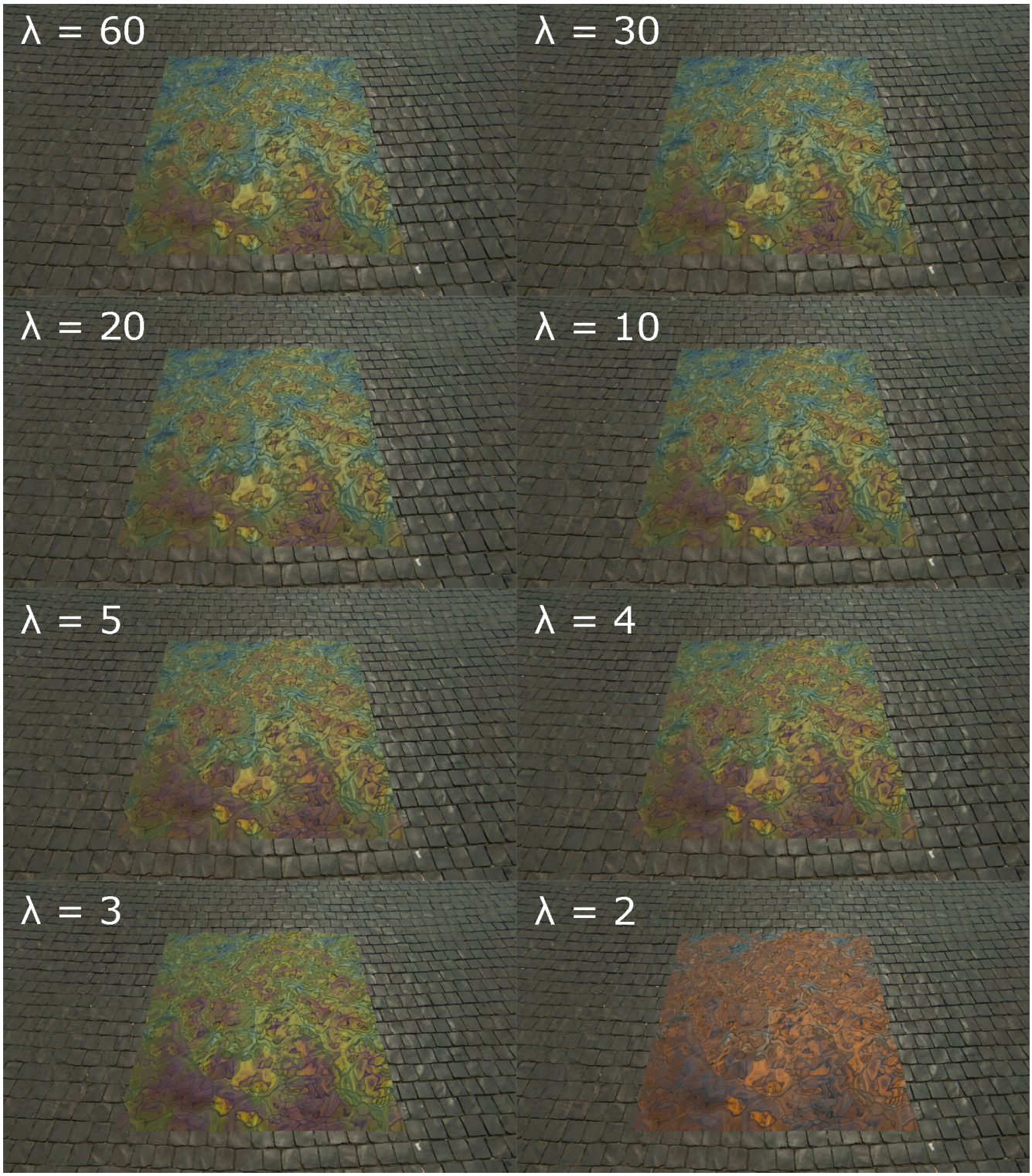

Figure 5.15: Sparse wavelength sampling results.

absolute minimum for convincing iridescent colors. We can clearly see that the quality degrades significantly when we choose values lower than 5.

5.2.3 Method Comparisons

We will show the significance of the spatially varying thin film thickness (SVT) by doing a qualitative comparison between the original method proposed by Hirayama et al., our method and real world iridescent imagery.

Hypothesis: The spatially varying thin film thickness allows for a more faithful rendering of the iridescent effects perceived in the real world. We hypothesize that the output of our method using SVT resembles real world iridescent objects more faithfully than the original method which does not have SVT.

The Experiment

We will use our method implemented on the GPU in this experiment. A visual comparison will be made between our method without SVT, our method with SVT on and real life iridescent objects. Our method without SVT on has the exact same visual output as the smooth illumination model of the original method proposed by Hirayama et al. Thus for convenience, we will use the GPU method without SVT on instead of the PBRT implementation of the original method. All renders are made using 30 wavelength samples taken in the range of 400 to 700 nanometers. For every result we will show the thin film system and the SVT parameters that are used.

Results

Single-layered thin film systems In Figure 5.16 we show a soap bubble where the water thin film is 480 nanometers thick. However, this is only the case in the original method since our

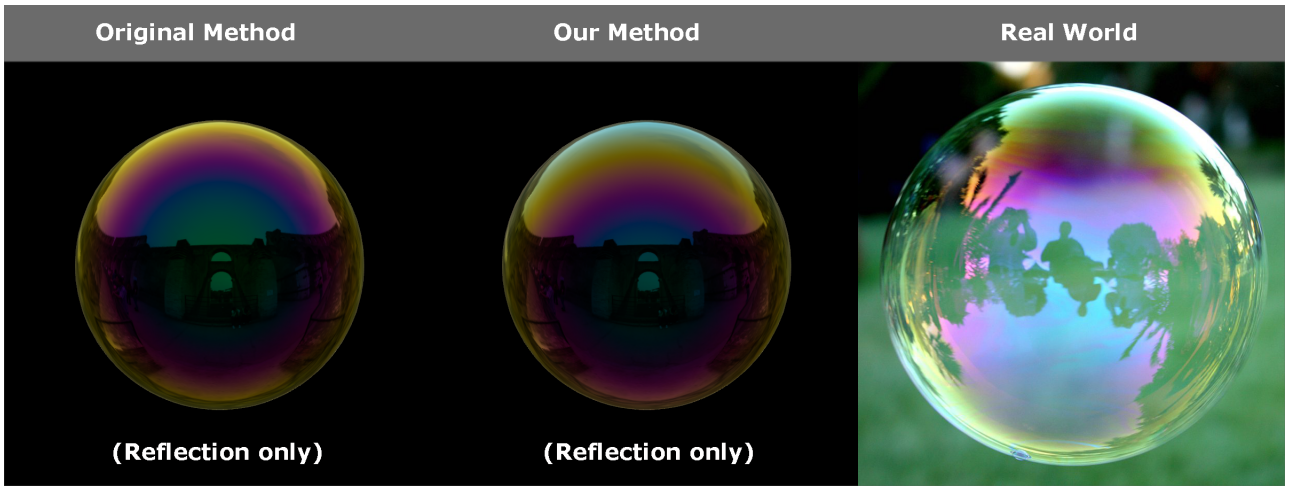

Figure 5.16: A 480 nm thick soap bubble simulation compared to a real world image.

method using SVT does not have a constant thin film thickness. The relative thickness has been set to 100 nanometers. For the relative change function we have created a gradient texture that is shown in Figure 5.17. For illustration purposes, we only show the reflection components of

both methods because it will be easier to distinguish the colors appearing on the object without the transmission component. Soap bubbles are generally thinner at the top and thicker at the

Figure 5.17: The relative change function for the soap bubble in Figure 5.16.

bottom. This is due to gravity pulling the water thin film down and thus more water is present at the bottom of the bubble. This gravitational effect is called drainage. The same kind of thickness distribution is reflected in the gradient because it gradually goes from pure black (relatively thin) to pure white (relatively thick). We associate the top and bottom of the gradient with the top and bottom of the soap bubble respectively using the texture coordinates that are generated. By using SVT in our method, we have simulated the effect gravity has on the thin film thickness of the bubble. The original method cannot simulate this effect and we can clearly see a qualitative difference in the output of both methods. Our method is significantly more realistic in this case and compares well with the real life image. In Figure 5.18 we show a soap

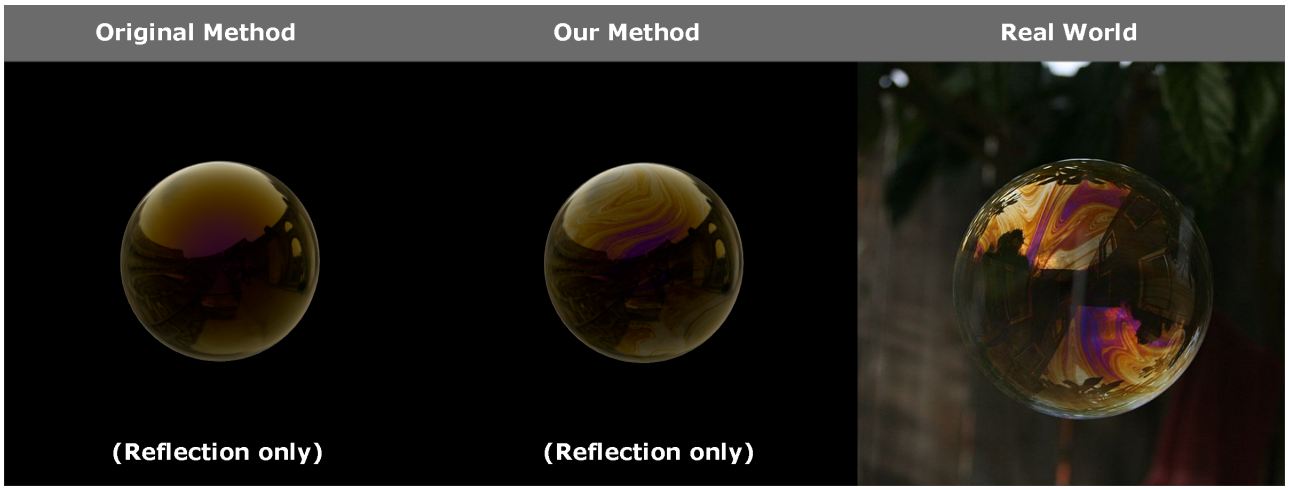

Figure 5.18: A 190 nm thick soap bubble simulation compared to a real world image.

bubble with a water thin film of 190 nanometers thick. The relative thickness has been set to 60 nanometers. For the relative change function we use a gray scale image of an oil slick in order to

simulate the oil like appearance that is created on the surface of the soap bubble. We can clearly see the difference between a constant thickness (the original method) and a spatially varying thickness (our method) when we compare it to the real life image. Our method compares more favorably to the real life image since they both show the instability of the soap bubble due to the water moving around in the soap bubble.

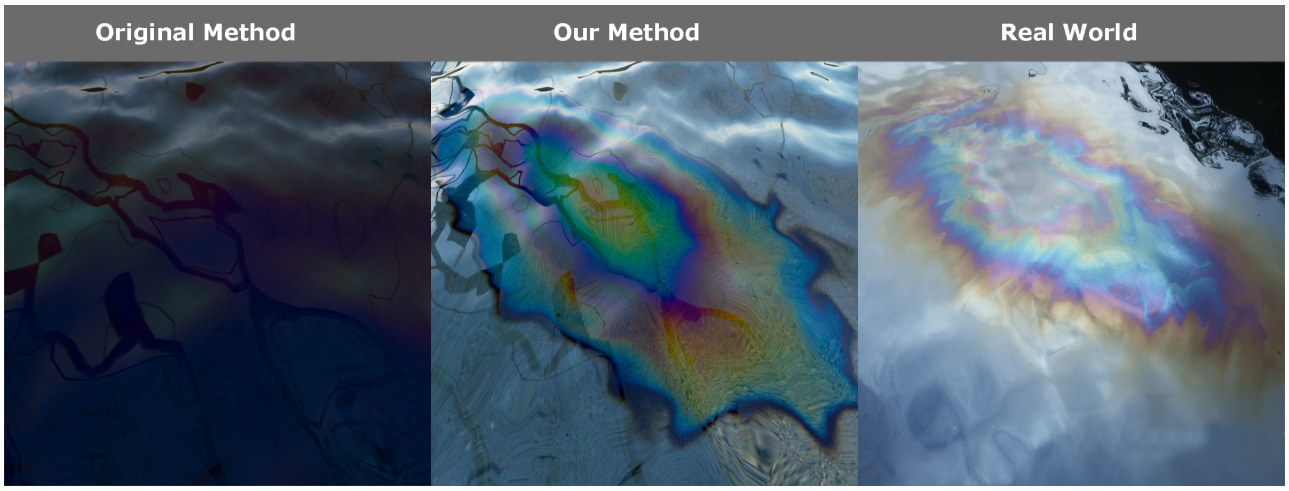

Figure 5.19: An oil spill simulation compared to the real world.

Multi-layered thin film system In Figure 5.19 a simulation of an oil spill is shown. The thin film system we use is air ($n = 1.0$) for the top medium, gasoline ($n = 1.4$) for the first thin film layer, water ($n = 1.33$) for the second thin film layer and asphalt ($n = 1.635$) for the bottom medium. The gasoline and water thin films are both 350 nanometers thick. The relative thickness has also been set to 350 nanometers. For the relative change function we use a radial gradient to represent the gradual change of thickness. We elongate this radial gradient to create an elliptical shape. Lastly, we modified the ellipsoid shape of the gradient in Photoshop using the smudge tool to make it resemble the shape of the real life oil spill. The relative change function is shown in Figure 5.20. In our method we can clearly see an oil slick appearing on

Figure 5.20: The relative change function of the oil spill in Figure 5.19.

the watery surface with the same colors and in the same order as in the real life image. Our

method however has blue and dark blue colors appearing at the outer edges of the oil spill. This can most likely be fixed by choosing a more narrow range of gray-scale colors for the radial gradient. Using SVT we also show that regions without iridescent effects can be simulated. The surface regions around the oil spill (shown in black in the SVT function) are regions where the thin film thickness in this case is zero. This means that we can create localized iridescent effects using SVT. In the real world we can see that these iridescent oil spills are also very much localized and do not necessarily spread out over the entire surface. Our method compares favorably to the real world image because we can use SVT to make the iridescence simulation render non-uniform and localized iridescent effects. The original method due to the uniform thin film thickness cannot simulate a (localized) oil spill. We have proven that our method more faithfully reproduces the effects of iridescence compared to the original method.

Chapter 6

Conclusion

We have explained some of the basic concepts used throughout the thesis including: ray tracing, complex numbers, thin film interference and wave optics. Furthermore, the theory of the main reference papers [8] and [9] has been discussed as well as how we have implemented both algorithms inside of the ray tracer PBRT. The GPU implementation of our method and the improvements we have made in the algorithm have also been discussed. A brief overview was given of our physically based OpenGL renderer, which was created to accurately visualize the iridescent effects. Our proposed method is a real-time physically based algorithm on the GPU for rendering iridescent objects. We have extended the theory of the original method by considering the fact that the thin film thickness in real life is not necessarily constant across the entire surface of an object. With SVT, we have proposed an intuitive, artist friendly way to model the spatially varying nature of thin film thickness across a surface. An implementation of the SVT theory was shown where we used a texture with gray scale values to represent the relative change function. The original method proposed by Hirayama et al. has been validated both qualitatively and quantitatively to ensure the method was physically plausible. Our method is thus also physically plausible, since the main ideas for simulating the thin film interference remain the same. We have achieved significant performance increases compared to the original method which makes our method useful for games and other interactive applications. However, in order to achieve this performance we have made two important assumptions: materials are wavelength invariant and they are strictly dielectric.

Future Work Research should be done in order to find ways to efficiently simulate semiconductor/metallic thin films without affecting the overall performance of the algorithm significantly. Precalculating the reflectivity and transmissivity on the CPU once and sending it to the GPU could be a viable way to increase the overall performance of the algorithm. By calculating it once on the CPU we avoid having to recalculate these quantities every frame for every pixel which would reduce the rendering time. Measurements should be done to determine how significant the performance would increase due to this change. The SVT concept could also be extended by considering a temporal dimension to simulate the effects of moving thin films. Lastly, this method could be extended to support simulation of all kinds of iridescent objects. In this thesis we have focused our efforts on simulating iridescent objects using thin film interference. However, objects like Compact-Disks or butterfly wings are due to diffraction. Our method could be extended by including the interference of light due to diffraction in order to increase the robustness of the algorithm.

Appendices

Appendix A

Scene File Example

Scene File

```
Film "image"
  "integer xresolution" [600] "integer yresolution" [1200]
"string filename" "buddha.exr"

Sampler "lowdiscrepancy" "integer pixelsamples" [8]
PixelFilter"box"

LookAt 0 .2 .2    0 .11 0   0 1 0
Scale -1 1 1
Camera "perspective" "float fov" [36]

WorldBegin
AttributeBegin
   AreaLightSource "area" "color L" [ 1 1 1 ] "integer nsamples" [16]
   Translate 0 2 0
   Rotate 90 1 0 0
AttributeEnd

AttributeBegin
    LightSource "infinite" "integer nsamples" [16] "color L" [1 1 1]
        "string mapname" ["textures/doge2_latlong.exr"]
AttributeEnd

AttributeBegin
  Material "matte" "color Kd" [.4 .42 .48]
  Shape "trianglemesh" "point P" [ -1 0 -1 1 0 -1 1 0 1 -1 0 1 ]
    "integer indices" [ 0 1 2 2 3 0]
AttributeEnd

Material "metal"  "float roughness" [.01]
    "spectrum eta" "spds/metals/Cu_palik.eta.spd"
    "spectrum k" "spds/metals/Cu_palik.k.spd"

Include "geometry/happy.pbrt"
WorldEnd
```

Listing A.1: Scene file for a metallic Buddha figurine.

Result

Figure A.1: The result of running the scene file in PBRT.

Appendix B

Mesh File Example

```
<mesh>
  <attribute index="0" type="float" size="3" >
        0.5 0 -0.5  0.5 0 0.5
        -0.5 0 0.5  -0.5 0 -0.5
        0.5 0 -0.5  0.5 0 0.5
        -0.5 0 0.5  -0.5 0 -0.5
  </attribute>
  <attribute index="2" type="float" size="3" >
        0 1 0 0 1 0
        0 1 0 0 1 0
        0 -1 0  0 -1 0
        0 -1 0  0 -1 0
  </attribute>
  <attribute index="5" type="float" size="2" >
        1 0 1 1
        0 1 0 0
        1 0 1 1
        0 1 0 0
  </attribute>
  <vao name="flat" > <source attrib="0" /> </vao>
  <vao name="lit" >
    <source attrib="0" />
    <source attrib="2" />
  </vao>
  <vao name="lit-tex" >
    <source attrib="0" />
    <source attrib="2" />
    <source attrib="5" />
  </vao>
  <indices cmd="triangles" type="ushort" >
        0 1 2
        2 3 0
        4 6 5
        6 4 7
  </indices>
</mesh>
```

Listing B.1: The geometry data for a unit plane.

Bibliography

[1] COOK, R. L., AND TORRANCE, K. E. A reflectance model for computer graphics. *ACM Trans. Graph. 1*, 1 (Jan. 1982), 7–24.

[2] CUYPERS, T., OH, S., HABER, T., BEKAERT, P., AND RASKAR, R. Ray-based reflectance model for diffraction.

[3] DIAS, M. L. Ray tracing interference color: Visualizing newton's rings. *IEEE Computer Graphics and Applications 14*, 3 (May 1994), 17–20.

[4] DURIKOVIC, R., AND KIMURA, R. Gpu rendering of the thin film on paints with full spectrum. In *Information Visualization, 2006. IV 2006. Tenth International Conference on* (2006), pp. 751–756.

[5] GLASSNER, A. Andrew glassner's notebook: Soap bubbles part 2. *Computer Graphics and Applications, IEEE 20*, 6 (2000), 99–109.

[6] GONDEK, J. S., MEYER, G. W., AND NEWMAN, J. G. Wavelength dependent reflectance functions. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (1994), SIGGRAPH '94, pp. 213–220.

[7] HE, X. D., TORRANCE, K. E., SILLION, F. X., AND GREENBERG, D. P. A comprehensive physical model for light reflection. *SIGGRAPH Computer Graphics 25*, 4 (July 1991), 175–186.

[8] HIRAYAMA, H., KANEDA, K., YAMASHITA, H., AND MONDEN, Y. An accurate illumination model for objects coated with multilayer films. *Computers and Graphics 25*, 3 (2001), 391 – 400.

[9] HIRAYAMA, H., KANEDA, K., YAMASHITA, H., YAMAJI, Y., AND MONDEN, Y. Visualization of optical phenomena caused by multilayer films with complex refractive indices. In *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications* (1999), PG '99.

[10] IWASAKI, K., MATSUZAWA, K., AND NISHITA, T. Real-time rendering of soap bubbles taking into account light interference. In *Proceedings of the Computer Graphics International* (2004), CGI '04, pp. 344–348.

[11] IWASAKI, K., UEDA, K., OMOYA, Y., AND TAKAGI, S. Real-time rendering of objects coated with multilayer thin films.

[12] KAJIYA, J. T. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), SIGGRAPH '86, ACM, pp. 143–150.

[13] NAGATA, N., DOBASHI, T., MANABE, Y., USAMI, T., AND INOKUCHI, S. Modeling and visualization for a pearl-quality evaluation simulator. *Visualization and Computer Graphics, IEEE Transactions on 3*, 4 (1997), 307–315.

[14] NAKAMAE, E., KANEDA, K., OKAMOTO, T., AND NISHITA, T. A lighting model aiming at drive simulators. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), SIGGRAPH '90, ACM, pp. 395–404.

[15] NAYAR, S., IKEUCHI, K., AND KANADE, T. Surface reflection: physical and geometrical perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 13*, 7 (1991), 611–634.

[16] PHARR, M., AND HUMPHREYS, G. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[17] PRUSTEN, M. Photo-real rendering of bioluminescence and iridescence in creatures from the abyss, 2008.

[18] SMITS, B., AND MEYER, G. Newtons colors: Simulating interference phenomena in realistic image synthesis. In *Photorealism in Computer Graphics*, Eurographic Seminars. Springer Berlin Heidelberg, 1992, pp. 185–194.

[19] STAM, J. Diffraction shaders. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), SIGGRAPH '99, pp. 101–110.

[20] STRATTON, J., ANSSARI, N., RODRIGUES, C., SUNG, I.-J., OBEID, N., CHANG, L., LIU, G., AND HWU, W. Optimization and architecture effects on gpu computing workload performance. In *Innovative Parallel Computing (InPar), 2012* (2012), pp. 1–10.

[21] SUN, Y., FRACCHIA, F. D., DREW, M. S., AND CALVERT, T. W. Rendering iridescent colors of optical disks. In *IN* (2000), Eurographics/ACM, pp. 341–352.

[22] WEIDLICH, A., AND WILKIE, A. Rendering the effect of labradoescence. In *Proceedings of Graphics Interface 2009* (2009), GI '09, pp. 79–85.

[23] WYMAN, C., SLOAN, P., AND SHIRLEY, P. Simple analytic approximations to the CIE XYZ color matching functions. *Journal of Computer Graphics Techniques (JCGT) 2*, 2 (July 2013), 1–11.