Diffraction Shader

Bachelorarbeit

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Michael Single

2014

Leiter der Arbeit:
Prof. Dr. Matthias Zwicker
Institut für Informatik und angewandte Mathematik

Abstract

Inhaltsverzeichnis

Kapitel 1

Theoretical Background

1.1 Definitions

1.1.1 Diffraction

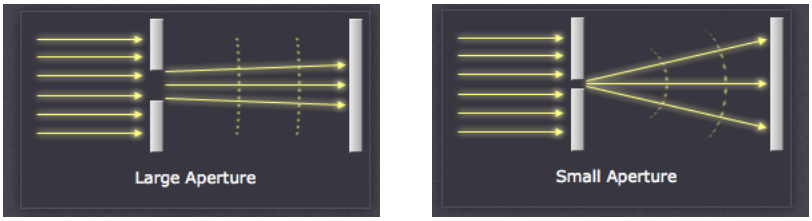
A wave is the spatial propagating variation or rather perturbation or also an oscillation of a location-and time-dependent physical quantity. In Mathematics, when talking about a wave, we are referring to the so called wavefunction $y(x,t) = Asin(wt - kx)$ which is a solution for the so called wave-equation. In general, this function depends on the location x and in time t. The maximal amplitude / magnitude of a wave is denoted by A. The phase of wave indicates in which part within its period with respect to a reference point in time / location the wave is located. Naturally occurring waves are uncommonly purely monochromatic waves, rather an overlapping of many waves with different wavelengths, which is denoted by interference. Interference is a phenomenon in which two waves superimpose to form a resultant wave of greater or lower amplitude. All portions of these corresponding wavelengths are forming the so called spectrum. For example sunlight which is a overlapping of many electromagnetic monochromatic waves of different wavelengths, ranging continuously from infrared across visible light to ultraviolet. Thereby, many effects may occur like interference - by overlapping two waves they either will amplify or cancel out partially or even totally each other in some location. Waves can experience in a different way a change in their form of appearance through reflexion, transmission, refraction or diffraction. In this thesis we are interested in particular in the bending of waves the so called effect of diffraction, which cannot be modeled using the standard ray theory of light.

When a wavefront is occluded partially by an obstacle, the wave is not only moving in the direction provided by the ray geometry, but also occur complex wave-movements outside of the geometric ray boundaries. This occurrence is called diffraction and occurs on the border of the wave's obstacle.

Generally, the effect of diffraction occurs on the border of obstacle whenever a propagating wave encounters that obstacle. But its effects are generally most pronounced for waves whose wavelength is roughly similar to the dimensions of the diffracting objects. Conversely, if wavelength is hardly similar, then there is almost no diffraction at all. This implies diffraction occurs mostly when the surface detail is highly anisotropic. If the obstructing object provides multiple,

closely spaced openings, a complex pattern of varying intensity can result. This is due to the superposition, or interference, of different parts of a wave that travels to the observer by different paths.

Example: Light rays passing through a small aperture will begin to diverge and interfere with one another. This becomes more significant as the size of the aperture decreases relative to the wavelength of light passing through, but occurs to some extent for any aperture or concentrated light source.



(a) Large Aperture (b) Small Aperture

Abbildung 1.1: Diffraction: Single Slit Example

Since the divergent rays now travel different distances, some move out of phase and begin to interfere with each other — adding in some places and partially or completely canceling out in others. This interference produces a diffraction pattern with peak intensities where the amplitude of the light waves add, and less light where they subtract. If one were to measure the intensity of light reaching each position on a line, the measurements would appear as bands similar to those shown below.

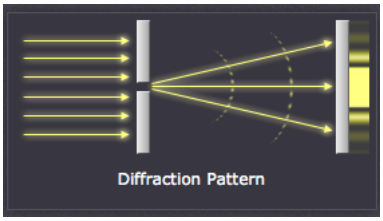


Abbildung 1.2: Diffraction Pattern

In general interference produces colorful effects due to the phase differences caused by a wave traversing thin media of different indices of refraction.

The effects of diffraction are often seen in everyday life. The most striking examples of diffraction are those that involve light; for example, the closely spaced tracks on a CD or DVD act as a diffraction grating to form the familiar rainbow pattern seen when looking at a disk

In optics, a diffraction grating is an optical component with a periodic structure, which splits and diffracts light into several beams travelling in different directions. The directions of these beams depend on the spacing of the grating and the wavelength of the light so that the grating acts as the dispersive element. The relationship between the grating spacing and the angles of the incident and diffracted beams of light is known as the grating equation. See chapter Evaluation Data Acquisition for further insight.

1.1.2 Radiometry

Light is fundamentally a propagation form of energy, so it is useful to define the SI unit of energy which is joule (J). To aid our intuition, let us describe radiometry in terms of collections of large numbers of photons. A photon can be considered as a quantum of light that has a position, direction of propagation and a wavelength λ measured in nanometers. A photon travels in a certain speed, denoted by $v = \frac{c}{n}$, that depends only on the refractive index n of the medium through which it propagates and the speed of light c. This allows us do define the frequency $f = \frac{c}{\lambda}$. The amount of energy q carried by a photon is given by the following relationship: $q = hf = \frac{hc}{\lambda}$ where h is the Plank's constant.

Spectral Energy
If there is a large collection of photons given, their total energy $Q = \sum_i q_i$ is the sum of energies of each photon q_i within the collection. But how is the energy distributed across wavelengths? One way in order to determine this distribution is to order all photons by their associated wavelength and then make a histogram from them. This is achieved by a discretization of their spectrum and combine all photons which will fall into the same interval, i.e. compute the sum for each interval from the energy of all their photons. By dividing such an interval by its length, denoted by Q_λ, we get a relatively scaled interval energy, which is called spectral energy and it is an intensive quantity. Intensive quantities can be thought of as density functions that tell the density of an extensive quantity at an infinitesimal point.

Power
Power is the estimated rate of energy production for light sources and is measured in the unit Watts, denoted by Q, which is another name for joules per second. Since power is a density over time, it is well defined even when energy production is varying over time. As with energy, we are really interested in the spectral power, measured in W/nm, denoted by Φ_λ

Irradiance
The term irradiance comes into place when we are interested in *how much light hits a given point*. In order to answer this question, we must make use of a density function. Figure 1.3 illustrates this idea. Let ΔA a finite area sensor that is smaller than the light field being measured. The spectral irradiance E is just the power per unit area $\frac{\Delta\Phi}{\Delta A}$ which is

$$E = \frac{\Delta q}{\Delta A \Delta t \Delta \lambda} \tag{1.1}$$

Thus the units of irradiance are $Jm^{-2}s^{-1}(nm)^{-1}$, the power per unit surface area.

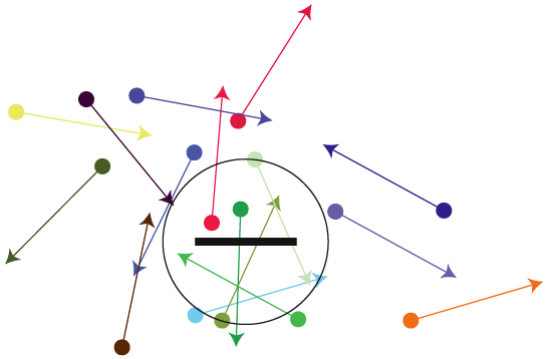


Abbildung 1.3: Irradiance is the summed up radiance over all directions

Radiance

Although irradiance tells us how much light is arriving at a point as illustrated in figure 1.3, it tells us little about the direction that light comes from. To measure something similar to what we see with our eyes we need to be able to associate the quantity *how much light with a specific direction.*

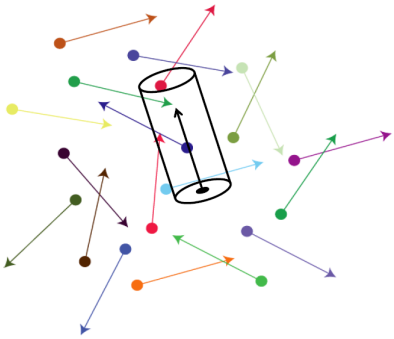


Abbildung 1.4: Radiance is photon ray density number of photons per area per solid angle - see figure 1.5

Radiance is a measure of the quantity of radiation that passes through or is emitted from a surface and falls within a given solid angle as shown in figure 1.5 in a specified direction as shown in figure 1.4. Think of it as the energy carried along a narrow beam of light. This means radiance characterizes the total emission of reflectance. It indicates how much of the power emitted by a reflecting surface will be received by an optical system looking at the surface from some given angle of view. Formally, this leads us to the following definition of radiance:

$$L(\omega) = \frac{d^2\Phi}{dAd\Omega cos(\theta)} \approx \frac{\Phi}{\Omega A cos(\theta)} \tag{1.2}$$

where L is the observed radiance in the unit energy per area per solid angle, which is $Wm^{-}2sr^{-}1$ in direction ω which has an angle θ between the surface normal and ω, Θ is the total flux or power emitted, θ is the angle between

the surface normal and the specified direction, A is the area of the surface and Ω is the solid angle in the unit steradian subtended by the observation or measurement.

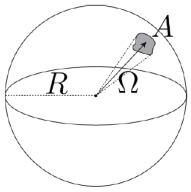


Abbildung 1.5: Solida angle is th earea of a surface patch on the unit sphere which is spanned by a set of directions. Its corresponding solid angle is equal $\Theta = \frac{A}{R^2}$.

It is useful to distinguish between radiance incident at a point on a surface and excitant from that point. Terms for these concepts sometimes used in the graphics literature are surface radiance L_r for the radiance *reflected* from a surface and field radiance L_i for the radiance *incident* at a surface.

BRDF
The bidirectional reflectance distribution function, in short BRDF, denoted as $f_r(\omega_i, \omega_r)$ is a four dimensional function that defines *how light is reflected at an opaque surface*. The function takes a negative directed incoming light direction, ω_i, and outgoing direction, ω_r as input argument. Both are defined with respect to the surface normal $\mathbf{n}$.Hence A BRDF returns the ratio of reflected radiance exiting along ω_r to the irradiance incident on the surface from direction ω_i, which is formally:

$$\begin{aligned} BRDF(\omega_i, \omega_r) &= f_r(\omega_i, \omega_r) \\ &= \frac{dL_r(\omega_r)}{dE_i(\omega_i)} \\ &= \frac{dL_r(\omega_r)}{L_i(\omega_i)cos(\theta_i)d\omega_i} \end{aligned} \tag{1.3}$$

L is the reflected radiance, E is the irradiance and θ_i is the angle between ω_i and the surface normal, $\mathbf{n}$. The index i indicates incident light, whereas the index r indicates reflected light.

Spectral Rendering
In Computer Graphics, spectral rendering is where a scene's light transport is modeled considering the whole span of wavelengths instead of R,G,B values (still relating on geometric optic, which ignore wave phase). The motivation is that real colors of the physical world are spectrum; trichromatic colors are only inherent to Human Visual System.

In Computer Graphics, when talking about Spectral Rendering, we are referring to use the whole span of wavelengths instead just using RGB values in order for rendering a scene's light transport. The motivation for using the

whole wavelength spectrum is due to the fact that trichromatic colors are only inherent to human visual system and therefore many phenomenons are poorly represented just using trichromy.

Colors

In order to see how crucial the role human vision plays, we only have to look the the definition of color: *Color is the aspect of visual perception by which an observer may distinguish differences between two structure-free fields of view of the same size and shape such as may be caused by differences in the spectral composition of the radiant energy concerned in the observation.* - Wyszechki and Siles, 2000 mentioned in Computer Graphics Fundamentals Book.

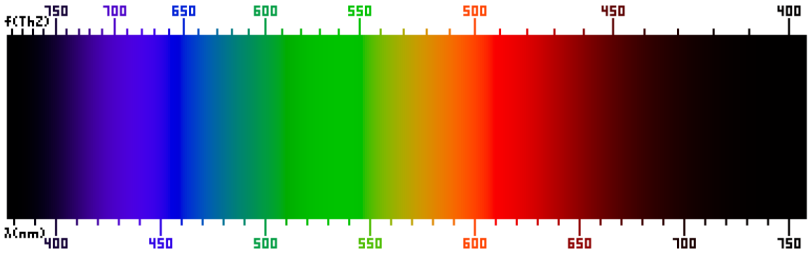


Abbildung 1.6: Frequency (top) and wavelenght (bottom) of colors of the visible light spectrum.

An eye consists of photoreceptor cells, cones and rods like illustrated in figure 1.7. Cones are responsible for color perception. The human eye has three kinds of cone cells, with spectral sensitivity peaks in short (between 420–440 nm), middle (between 530–540 nm), and long (between 560–580 nm) wavelengths. In principle, any color sensation in human color perception as shown in figure 1.6 can therefore be described by just three parameters, corresponding to levels of stimulus of the three types of cone cells.

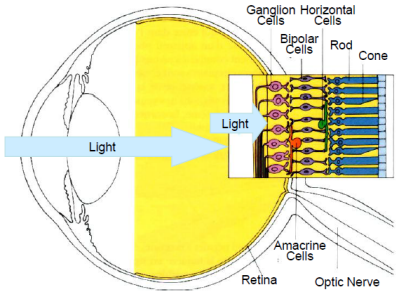


Abbildung 1.7: Schematic of photoreceptor cells, cones and rods, in human eye

A color space maps a range of physically produced colors from mixed light to an objective description of color sensations registered in the eye in terms of Tristimulus values.

Tristimulus values associated with a color space can be conceptualized as amounts of three primary colors in a tri-chromatic additive color model. In

some color spaces (including LMS and XYZ spaces), the primary colors used are not real colors, in the sense that they cannot be generated with any light spectrum.

Due to distribution of cones in human eye, the Tristimulus values depend on over's field of view. CIE defined a color mapping function called standard observer, to eliminate this variable represent an average human chromatic response.

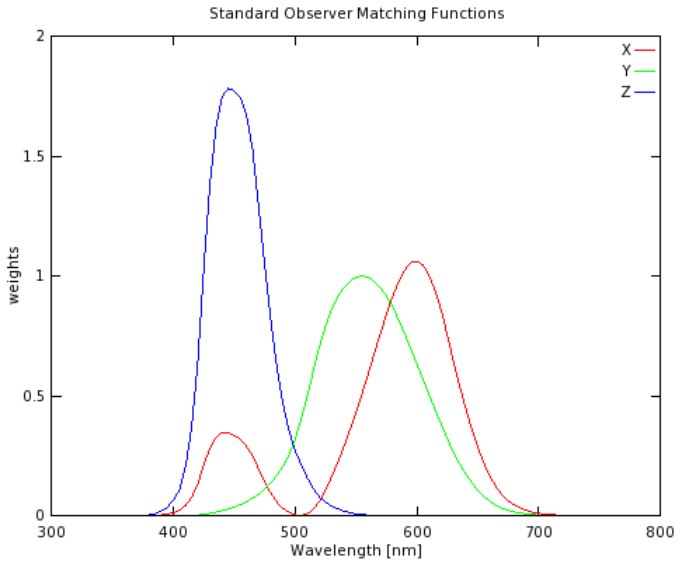


Abbildung 1.8: Plots of our color matching functions we used for rendering

The CIE's color matching functions $\overline{x}(\lambda)$, $\overline{y}(\lambda)$, $\overline{z}(\lambda)$ in figure 1.8 are the numerical description of the chromatic response of the observer. They can be thought of as the spectral sensitivity curves of three linear light detectors yielding the CIE Tristimulus values X, Y and Z.

The Tristimulus values for a color with a spectral power distribution $I(\lambda)$, are given in terms of the standard observer by:

$$\begin{aligned} X &= \int_{380}^{780} I(\lambda)\,\overline{x}(\lambda)\,d\lambda \\ Y &= \int_{380}^{780} I(\lambda)\,\overline{y}(\lambda)\,d\lambda \\ Z &= \int_{380}^{780} I(\lambda)\,\overline{z}(\lambda)\,d\lambda \end{aligned} \tag{1.4}$$

where λ, is the wavelength of the equivalent monochromatic light (measured in nanometers). It is not possible to build a display that corresponds to CIE XYZ, for this reasons it is necessary to design other color spaces, which are physical realizable, offers efficient encoding, are perceptual uniform and have an intuitive color specification. There are simple conversions between XYZ color space to any other color space defined, described as linear transformations.

1.1.3 Signal Processing Basics

A signal is a function that conveys information about the behavior or attributes of some phenomenon. In the physical world, any quantity exhibiting variation in time or variation in space (such as an image) is potentially a signal that might provide information on the status of a physical system, or convey a message between observers.

The Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

Fourier Transformation
The Fourier-Transform is a mathematical tool which allows to transform a given function or rather a given signal from defined over a time- (or spatial-) domain into its corresponding frequency-domain.

Let f an measurable function over $\mathbb{R}^n$. Then, the continuous Fourier Transformation(**FT**), denoted as $\mathcal{F}\{f\}$ of f, ignoring all constant factors in the formula, is defined as:

$$\mathcal{F}_{FT}\{f\}(w) = \int_{\mathbb{R}^n} f(x)e^{-iwt}dt \tag{1.5}$$

whereas its inverse transform is defined like the following which allows us to obtain back the original signal:

$$\mathcal{F}^{-1}_{FT}\{f\}(w) = \int_{\mathbb{R}} \mathcal{F}\{w\}e^{iwt}dt \tag{1.6}$$

Usual w is identified by the angular frequency which is equal $w = \frac{2\pi}{T} = 2\pi v_f$. In this connection, T is the period of the resulting spectrum and v_f is its corresponding frequency.

By using Fourier Analysis, which is the approach to approximate any function by sums of simpler trigonometric functions, we gain the so called Discrete Time Fourier Transform (in short **DTFT**). The DTFT operates on a discrete function. Usually, such an input function is often created by digitally sampling a continuous function. The DTFT itself is operation on a discretized signal on a continuous, periodic frequency domain and looks like the following:

$$\mathcal{F}_{DTFT}\{f\}(w) = \sum_{-\infty}^{\infty} f(x)e^{-iwk} \tag{1.7}$$

Note that the DTFT is not practically suitable for digital signal processing since there a signal can be measured only in a finite number of points. Thus, we can further discretize the frequency domain and will get then the Discrete Fourier Transformation (in short **DFT**) of the input signal:

$$\mathcal{F}_{DFT}\{f\}(w) = \sum_{n=0}^{N-1} f(x)e^{-iw_nk} \tag{1.8}$$

Where the angular frequency w_n is defined like the following $w_n = \frac{2\pi n}{N}$ and N is the number of samples within an equidistant period sampling.

Any continuous function $f(t)$ can be expressed as a series of sines and cosines. This representation is called the Fourier Series (denoted by FS) of $f(t)$.

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n cos(nt) + \sum_{n=1}^{\infty} b_n cos(nt) \tag{1.9}$$

where

$$\begin{aligned} a_0 &= \int_{-\pi}^{\pi} f(t)dt \\ a_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)cos(nt)dt \\ b_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)sin(nt)dt \end{aligned} \tag{1.10}$$

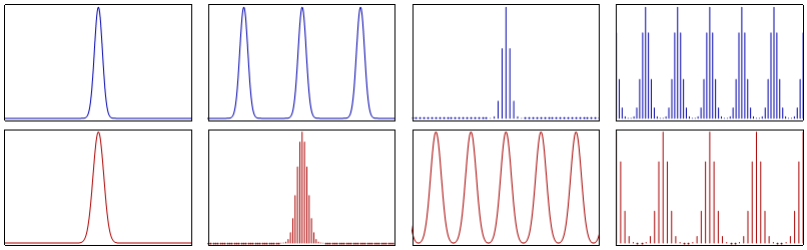


Abbildung 1.9: Relationship between the continuous Fourier transform and the discrete Fourier transform: Left column: A continuous function (top) and its Fourier transform 1.5 (bottom). Center-left column: Periodic summation of the original function (top). Fourier transform (bottom) is zero except at discrete points. The inverse transform is a sum of sinusoids called Fourier series 1.9. Center-right column: Original function is discretized (multiplied by a Dirac comb) (top). Its Fourier transform (bottom) is a periodic summation (DTFT) of the original transform. Right column: The DFT 1.8 (bottom) computes discrete samples of the continuous DTFT 1.7. The inverse DFT (top) is a periodic summation of the original samples.

Spetail signal $f(t)$ is	Operator	Transformed frequency signal $\hat{f}(\omega)$ is
continuous and periodic in t	FS 1.9	only discrete in ω
only continuous in t	FT 1.5	only continuous in ω
only discrete in t	DTFT 1.7	continuous and periodic in ω
discrete and periodic in t	DFT 1.8	discrete and periodic in ω

Tabelle 1.1: Fourier operator to apply for a given spatial input signal and the properties of its resulting output signal in frequency space

Convolution
The convolution $f * g$ of two functions f, g: $\mathbb{R}^n \to \mathbb{C}$ is defined as:

$$(f * g)(t) = \int_{\mathbb{R}^n} f(t)g(t-x)dx \tag{1.11}$$

Note that the Fourier transform of the convolution of two functions is the product of their Fourier transforms. This is equivalent to the fact that Convolution in spatial domain is equivalent to multiplication in frequency domain. Therefore, the inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms. Last an illustration of the relationships between the previous presented Fourier transformations and different given input signals. First an concrete example shown in Figure 1.9. Table 1.1 tells what Fourier transformation operator has to be applied to which kind of input signal and what properties its resulting Fourier transform will have.

Taylor Series
Taylor series is a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point.

The Taylor series $\mathcal{T}$ of a real or complex-valued function $f(x)$ that is infinitely differentiable at a real or complex number a is the power series:

$$\mathcal{T}(f;a)(x) = \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n \tag{1.12}$$

1.2 Thesis Basis: J.Stam's Paper about Diffraction Shader

In his paper about Diffraction Shader, J. Stam derives a BRDF which is modeling the effect of diffraction for various analytical anisotropic reflexion models relying on the so called scalar wave theory of diffraction for which a wave is assumed to be a complex valued scalar. It's noteworthy, that Stam's BRDF formulation does not take into account the polarization of the light. Fortunately, light sources like sunlight and light bulbs are unpolarized.

A further assumption in Stam's Paper is, the emanated waves from the source are stationary, which implies the wave is a superposition of independent monochromatic waves. This implies that each wave is associated to a definite wavelength lambda. However, sunlight once again fulfills this fact.

In our simulations we will always assume we have given a directional light source, i.e. sunlight. Hence, Stam's model can be used for our derivations.

For his derivations Stam uses the Kirchhoff integral (ADD REF TO WIKI), which is relating the reflected field to the incoming field. This equation is a formalization of Huygen's well-known principle that states that if one knows the wavefront at a given moment, the wave at a later time can be deduced by considering each point on the first wave as the source of a new disturbance. Mathematically speaking, once the field $\psi_1 = e^{ik\mathbf{x}\cdot\mathbf{ss}}$ on the surface is known, the field ψ_2 everywhere else away from the surface can be computed. More precisely, we want to compute the wave ψ_2 equal to the reflection of an incoming planar

monochromatic wave $\psi_1 = e^{ik\omega_i * x}$ traveling in the direction ω_i from a surface S to the light source. Formally, this can be written as:

$$\psi_2(\omega_i, \omega_r) = \frac{ike^{iKR}}{4\pi R}(F(-\omega_i - \omega_r) - (-\omega_i + \omega_r)) \cdot I_1(\omega_i, \omega_r) \tag{1.13}$$

with

$$I_1(\omega_i, \omega_r) = \int_S \hat{\mathbf{n}} e^{ik(-\omega_i - \omega_r)\cdot \mathbf{s}} d\mathbf{s} \tag{1.14}$$

In applied optics, when dealing with scattered waves, one does use differential scattering cross-section rather than defining a BRDF which has the following identity:

$$\sigma^0 = 4\pi \lim_{R\to\infty} R^2 \frac{\langle |\psi_2|^2 \rangle}{\langle |\psi_1|^2 \rangle} \tag{1.15}$$

where R is the distance from the center of the patch to the receiving point x_p, $\hat{\mathbf{n}}$ is the normal of the surface at s and the vectors:

The relationship between the BRDF and the scattering cross section can be shown to be equal to

$$BRDF = \frac{1}{4\pi}\frac{1}{A}\frac{\sigma^0}{cos(\theta_i)cos(\theta_r)} \tag{1.16}$$

where θ_i and θ_r are the angles of incident and reflected directions on the surface with the surface normal n. See 1.10.

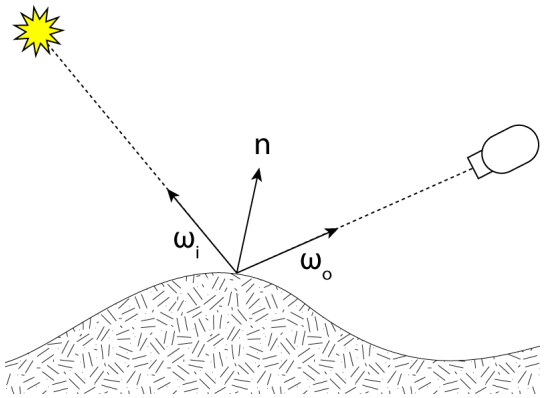


Abbildung 1.10: ω_i points toward the light source, ω_r points toward the camera, n is the surface normal

The components of vector resulting by the difference between these direction vectors: In order to simplify the calculations involved in his vectorized integral equations, Stam considers the components of vector

$$(u, v, w) = -\omega_i - \omega_r \tag{1.17}$$

explicitly and introduces the equation:

$$I(ku,kv) = \int_S \hat{\mathbf{n}} e^{ik(u,v,w)\cdot \mathbf{s}} d\mathbf{s} \tag{1.18}$$

which is a first simplification of 1.14. Note that the scalar w is the third component of 1.17 and can be written as $w = -(cos(\theta_i) + cos(\theta_r))$ using spherical coordinates. The scalar $k = \frac{2\pi}{\lambda}$ represent the wavenumber.

During his derivations, Stam provides a analytical representation for the Kirchhoff integral assuming that each surface point $s(x,y)$ can be parameterized by $(x,y,h(x,y))$ where h is the height at the position (x,y) on the given (x,y) surface plane. Using the tangent plane approximation for the parameterized surface and plugging it into 1.18 he will end up with:

$$\mathbf{I}(ku,kv) = \int\int (-h_x(x,y), -h_y(x,y), 1) e^{ikwh(x,y)} e^{ik(ux+vy)} dxdy \tag{1.19}$$

For further simplification Stam formulates auxillary function which depends on the provided height field:

$$p(x,y) = e^{iwkh(x,y)} \tag{1.20}$$

which will allow him to further simplify his equation 1.19 to:

$$\mathbf{I}(ku,kv) = \int\int \frac{1}{ikw}(-p_x, -p_y, ikwp) dxdy \tag{1.21}$$

where he used that $(-h_x(x,y), -h_y(x,y), 1)e^{kwh(x,y)}$ is equal to $\frac{(-p_x,-p_y,ikwp)}{ikw}$ using the definition of the partial derivatives applied to the function 1.20.

Let $P(x,y)$ denote the Fourier Transform (FT) of $p(x,y)$. Then, the differentiation with respect to x respectively to y in the Fourier domain is equivalent to a multiplication of the Fourier transform by $-iku$ or $-ikv$ respectively. This leads him to the following simplification for 1.19:

$$\mathbf{I}(ku,kv) = \frac{1}{w} P(ku,kv) \cdot (u,v,w) \tag{1.22}$$

Let us consider the term $g = (F(-\omega_i - \omega_r) - (-\omega_i + \omega_r))$, which is a scalar factor of 1.13. The dot product with g and $(-\omega_i - \omega_r)$ is equal $2F(1 + \omega_i \cdot \omega_r)$. Putting this finding and the identity 1.22 into 1.13 he will end up with:

$$\psi_2(\omega_i,\omega_r) = \frac{ike^{iKR}}{4\pi R} \frac{2F(1+\omega_i\cdot\omega_r)}{w} P(ku,kv) \tag{1.23}$$

By using the identity 1.16, this will lead us to his main finding:

$$BRDF_\lambda(\omega_i,\omega_r) = \frac{k^2F^2G}{4\pi^2Aw^2} \langle |P(ku,kv)|^2 \rangle \tag{1.24}$$

where G is the so called geometry term which is equal:

$$G = \frac{(1+\omega_i\cdot\omega_r)^2}{cos(\theta_i)cos(\theta_r)} \tag{1.25}$$

1.3 Derivations

1.3.1 BRDF formulation

Lets assume we have given an incoming light source with solid angle ω_i, θ_i is its angle of incidence, ω_r is the solid angle for the reflected light. Further let λ denote the wavelength and Ω is the hemisphere we of integration for the incoming light. Then, we are able to formulate a BRDF by using its definition 1.3:

$$\begin{aligned} f_r(\omega_i,\omega_r) &= \frac{dL_r(\omega_r)}{L_i(\omega_i)cos(\theta_i)d\omega_i} \\ \Rightarrow f_r(\omega_i,\omega_r)L_i(\omega_i)cos(\theta_i)d\omega_i &= dL_r(\omega_r) \\ \Rightarrow \int_\Omega f_r(\omega_i,\omega_r)L_i(\omega_i)cos(\theta_i)d\omega_i &= \int_\Omega dL_r(\omega_r) \\ \Rightarrow L_r(\omega_r) &= \int_\Omega f_r(\omega_i,\omega_r)L_i(\omega_i)cos(\theta_i)d\omega_i \end{aligned} \tag{1.26}$$

The last equation is the so called rendering equation. We assume that our incident light is a directional, unpolarized light source 2.6 like sunlight and therefore its radiance is given as

$$L_\lambda(\omega) = I(\lambda)\delta(\omega-\omega_i) \tag{1.27}$$

where $I(\lambda)$ is the intensity of the relative spectral power for the wavelength λ. Since all light rays are parallel, whenever we are provided by a directional light source and we can think of radiance as a measure of the light emitted from a particular surface location into a particular direction, above's radiance identity will follow immediately. By plugging the identity 1.27 into our current rendering equation 1.26, we will get:

$$\begin{aligned} L_\lambda(w_r) &= \int_\Omega BRDF_\lambda(\omega_i,\omega_r)L_\lambda(\omega_i)cos(\theta_i)d\omega_i \\ &= BRDF_\lambda(\omega_i,\omega_r)I(\lambda)cos(\theta_i) \end{aligned} \tag{1.28}$$

where $L_\lambda(\omega_i)$ is the incoming radiance and $L_\lambda(\omega_r)$ is the radiance reflected by the given surface. Note that the integral in equation 1.28 vanishes since $\delta(\omega-\omega_i)$ is only equal one if and only if $\omega=\omega_i$.

We are going to use Stam's main derivation (1.24) for the $BRDF(\omega_i,\omega_r)$ in 1.28 by applying the fact that the wavenumber is equal $k=\frac{2\pi}{\lambda}$:

$$\begin{aligned} BRDF(\omega_i,\omega_r) &= \frac{k^2F^2G}{4\pi^2Aw^2}\langle|P(ku,kv)|^2\rangle \\ &= \frac{k^2F^2(1+\omega_i\cdot\omega_r)^2}{cos(\theta_i)cos(\theta_r)4\pi^2Aw^2}\langle|P(ku,kv)|^2\rangle \\ &= \frac{4\pi^2F^2(1+\omega_i\cdot\omega_r)^2}{cos(\theta_i)cos(\theta_r)4\pi^2A\lambda^2w^2}\langle|P(ku,kv)|^2\rangle \\ &= \frac{F(w_i,w_r)^2(1+\omega_i\cdot\omega_r)^2}{cos(\theta_i)cos(\theta_r)A\lambda^2w^2}\langle|P(ku,kv)|^2\rangle \end{aligned} \tag{1.29}$$

Going back to the definition 1.17 of $(u,v,w) = -\omega_i - \omega_r$ and using spherical coordinates A.2, we get for w the following identity

$$\begin{aligned} w &= -\omega_i - \omega_r \\ &= -(\omega_i + \omega_r) \\ &= -\left(cos(\theta_i) + cos(\theta_r)\right) \end{aligned} \tag{1.30}$$

and therefore w^2 is equal $(cos(\theta_i) + cos(\theta_r))^2$ This new fact will allow us to get even further:

$$\begin{aligned} L_\lambda(\omega_r) &= \frac{F(\omega_i,\omega_r)^2(1+\omega_i\cdot\omega_r)^2}{A\lambda^2 cos(\theta_i)cos(\theta_r)(cos(\theta_i)+cos(\theta_r))^2}\left\langle\left|P_{cont}\left(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda}\right)\right|^2\right\rangle cos(\theta_i)I(\lambda) \\ &= I(\lambda)\frac{F(\omega_i,\omega_r)^2(1+\omega_i\cdot\omega_r)^2}{\lambda^2 A(cos(\theta_i)+cos(\theta_r))^2 cos(\theta_r)}\left\langle\left|P_{cont}\left(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda}\right)\right|^2\right\rangle \\ &= I(\lambda)\frac{F(\omega_i,\omega_r)^2(1+\omega_i\cdot\omega_r)^2}{\lambda^2 A(cos(\theta_i)+cos(\theta_r))^2 cos(\theta_r)}\left\langle\left|T_0^2 P_{dtft}\left(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda}\right)\right|^2\right\rangle \end{aligned} \tag{1.31}$$

Where P_{cont} denotes the continuous inverse Fourier-Transform 1.6 for the Taylor-Series 1.12 of our height field representing the nano-scaled surface structure, i.e. $P(k,l) = \mathcal{F}^{-1}\{p\}(k,l)$ and P_{dtft} is the inverse Discrete Time Fourier Transform 1.7 of $p(x,y) = e^{ikwh(x,y)}$. Furthermore T_0 the sampling distance for the discretization of $p(x,y)$ assuming equal and uniform sampling in both dimensions x and y.

1.3.2 Relative BRDF

In this section we are going to explain how to scale our BRDF formulation such that all of its possible output values are mapped into the range $[0,1]$. Such a relative BRDF formulation will ease our life for later rendering purposes since usually color values are within the range $[0,1]$, too. Furthermore, this will allow us to properly blend the resulting illumination caused by diffraction with a texture map.

Let us examine what $L_\lambda(\omega_r)$ will be for $\omega_r = \omega_0 := (0,0,*)$ i.e. specular reflection case, denoted as $L_\lambda^{spec}(\omega_0)$. When we know the expression for $L_\lambda^{spec}(\omega_0)$ we would be able to compute the relative reflected radiance for our problem 1.31 by simply taking the fraction between $L_\lambda(\omega_r)$ and $L_\lambda^{spec}(\omega_0)$ which is denoted by:

$$\rho_\lambda(\omega_i,\omega_r) = \frac{L_\lambda(\omega_r)}{L_\lambda^{spec}(\omega_0)} \tag{1.32}$$

But first, let us derive the following expression:

$$
\begin{aligned}
L_\lambda^{spec}(\omega_0) &= I(\lambda)\frac{F(\omega_0,\omega_0)^2(1+\begin{pmatrix}0\\0\\1\end{pmatrix}\cdot\begin{pmatrix}0\\0\\1\end{pmatrix})^2}{\lambda^2 A(cos(0)+cos(0))^2 cos(0)}\langle\left|T_0^2 P_{dtft}(0,0)\right|^2\rangle \\
&= I(\lambda)\frac{F(\omega_0,\omega_0)^2(1+1)^2}{\lambda^2 A(1+1)^2 1}\left|T_0^2 N_{sample}\right|^2 \\
&= I(\lambda)\frac{F(\omega_0,\omega_0)^2}{\lambda^2 A}\left|T_0^2 N_{sample}\right|^2
\end{aligned} \tag{1.33}
$$

Where $N_{samples}$ is the number of samples of the DTFT 1.7. Thus, we can plug our last derived expression 1.33 into the definition for the relative reflectance radiance 1.32 in the direction w_r and will get:

$$
\begin{aligned}
\rho_\lambda(\omega_i,\omega_r) &= \frac{L_\lambda(\omega_r)}{L_\lambda^{spec}(\omega_0)} \\
&= \frac{I(\lambda)\frac{F(\omega_i,\omega_r)^2(1+\omega_i\cdot\omega_r)^2}{\lambda^2 A(cos(\theta_i)+cos(\theta_r))^2 cos(\theta_r)}\langle\left|T_0^2 P_{dtft}(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda})\right|^2\rangle}{I(\lambda)\frac{F(\omega_0,\omega_0)^2}{\lambda^2 A}\left|T_0^2 N_{sample}\right|^2} \\
&= \frac{F^2(\omega_i,\omega_r)(1+\omega_i\cdot\omega_r)^2}{F^2(\omega_0,\omega_0)(cos(\theta_i)+cos(\theta_r))^2 cos(\theta_r)}\langle\left|\frac{P_{dtft}(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda})}{N_{samples}}\right|^2\rangle
\end{aligned} \tag{1.34}
$$

For simplification and a better overview, let us introduce the following expression, the so called gain-factor:

$$
C(\omega_i,\omega_r) = \frac{F^2(\omega_i,\omega_r)(1+\omega_i\cdot\omega_r)^2}{F^2(\omega_0,\omega_0)(cos(\theta_i)+cos(\theta_r))^2 cos(\theta_r) N_{samples}^2} \tag{1.35}
$$

Using this substitute, we will end up with the following expression for the relative reflectance radiance from equation 1.34:

$$
\rho_\lambda(\omega_i,\omega_r) = C(\omega_i,\omega_r)\langle\left|P_{dtft}(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda})\right|^2\rangle \tag{1.36}
$$

Using the previous definition for the relative reflectance radiance 1.32:

$$
\rho_\lambda(\omega_i,\omega_r) = \frac{L_\lambda(\omega_r)}{L_\lambda^{spec}(\omega_0)} \tag{1.37}
$$

Which we can rearrange to the expression:

$$
L_\lambda(\omega_r) = \rho_\lambda(\omega_i,\omega_r) L_\lambda^{spec}(\omega_0) \tag{1.38}
$$

Let us choose $L_\lambda^{spec}(w_0) = S(\lambda)$ such that is has the same profile as the relative spectral power distribution of CIE Standard Illuminant $D65$ discussed in 2.4.3. Furthermore, when integrating over λ for a specular surface, we should

get CIE_{XYZ} values corresponding to the white point for $D65$. The corresponding tristimulus values using CIE colormatching functions 1.4 for the CIE_{XYZ} values look like:

$$\begin{aligned} X &= \int_\lambda L_\lambda(\omega_r)\overline{x}(\lambda)d\lambda \\ Y &= \int_\lambda L_\lambda(\omega_r)\overline{y}(\lambda)d\lambda \\ Z &= \int_\lambda L_\lambda(\omega_r)\overline{z}(\lambda)d\lambda \end{aligned} \tag{1.39}$$

where $\overline{x}$, $\overline{y}$, $\overline{z}$ are the color matching functions. Using our last finding 1.38 for $L_\lambda(\omega_r)$ with the definition for the tristimulus values 1.39, we can actually derive an expression for computing the colors for our initial BRDF formula 1.26. Without any loss of generality it satisfies to derive an explicit expression for just one tristimulus term, for example X. Since The other have a similar formulation, except the we have to replace all X with Y or Z respectively. Therefore, we get:

$$\begin{aligned} X &= \int_\lambda L_\lambda(\omega_r)\overline{x}(\lambda)d\lambda \\ &= \int_\lambda \rho_\lambda(\omega_i,\omega_r)L_\lambda^{spec}(\omega_0)\overline{x}(\lambda)d\lambda \\ &= \int_\lambda \rho_\lambda(\omega_i,\omega_r)S(\lambda)\overline{x}(\lambda)d\lambda \\ &= \int_\lambda C(\omega_i,\omega_r)\langle\left|P_{dtft}(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda})\right|^2\rangle S(\lambda)\overline{x}(\lambda)d\lambda \\ &= C(\omega_i,\omega_r)\int_\lambda\langle\left|P_{dtft}(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda})\right|^2\rangle S(\lambda)\overline{x}(\lambda)d\lambda \\ &= C(\omega_i,\omega_r)\int_\lambda\langle\left|P_{dtft}(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda})\right|^2\rangle S_x(\lambda)d\lambda \end{aligned} \tag{1.40}$$

Where we used the definition $S_x(\lambda)\overline{x}(\lambda)$ in the last step.

1.3.3 Taylor approximation for BRDF

In this section, we will deliver an approximation for the inverse Fourier Transformation of Stam's auxiliary function $p(x,y)$. This derivation will rely on the definition of Taylor Series expansion 1.12. Further, we will provide an error bound for our approximation approach for a given number of iterations. Last, we will extend our current BRDF formula by the findings derived within this section.

Given $p(x,y) = e^{ikwh(x,y)}$ form Stam's Paper 1.2 where $h(x,y)$ is a given height field. Let be y real or even complex value, and lets consider the power series for the the exponential function

$$e^t = 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + ... = \sum_{n=0}^{\infty}\frac{t^n}{n!} \tag{1.41}$$

Let us define

$$t = t(x, y) = ikwh(x, y) \tag{1.42}$$

where i is the imaginary number. For simplification, let us denote $h(x, y)$ as h. Then it follows by our previous stated identities:

$$\begin{aligned} e^t &= 1 + (ikwh) + \frac{1}{2!}(ikwh)^2 + \frac{1}{3!}(ikwh)^3 + ... \\ &= \sum_{n=0}^{\infty} \frac{(ikwh)^n}{n!}. \end{aligned} \tag{1.43}$$

Hence it holds $p(x, y) = \sum_{n=0}^{\infty} \frac{(ikwh(x,y))^n}{n!}$. Let us now compute the Fourier Transformation of p(x,y) form above:

$$\begin{aligned} \mathcal{F}\{p\}(u, v) &= \mathcal{F}\left\{ \sum_{n=0}^{\infty} \frac{(ikwh)^n}{n!}. \right\}(u, v) \\ &=^{\mathcal{F}\, lin\, Operator} \sum_{n=0}^{\infty} \mathcal{F}\left\{ \frac{(ikwh)^n}{n!} \right\}(u, v) \\ &= \sum_{n=0}^{\infty} \frac{(ikw)^n}{n!} \mathcal{F}\{h^n\}(u, v) \end{aligned} \tag{1.44}$$

Therefore it follows: $P(\alpha, \beta) = \sum_{n=0}^{\infty} \frac{(ikw)^n}{n!} \mathcal{F}\{h^n\}(\alpha, \beta)$ for which $\mathcal{F}_{FT}\{h^n\}(u, v)$. Next we are going to look for an $N \in \mathbb{N}$ such that

$$\sum_{n=0}^{N} \frac{(ikwh)^n}{n!} \mathcal{F}\{h^n\}(\alpha, \beta) \approx P(\alpha, \beta) \tag{1.45}$$

is a good approximation. But first the following two facts have to be proven:

1. Show that there exist such an $N \in \mathbb{N}$s.t the approximation holds true.
2. Find a value for B s.t. this approximation is below a certain error bound, for example machine precision ϵ.

Proof Sketch of 1.

By the **ratio test** (see [**1**]) It is possible to show that the series $\sum_{n=0}^{N} \frac{(ikwh)^n}{n!} \mathcal{F}\{h^n\}(\alpha, \beta)$ converges absolutely:

Proof: Consider $\sum_{k=0}^{\infty} \frac{y^n}{n!}$ where $a_k = \frac{y^k}{k!}$. By applying the definition of the ratio test for this series it follows:

$$\forall y : limsup_{k\to\infty} |\frac{a_{k+1}}{a_k}| = limsup_{k\to\infty} \frac{y}{k+1} = 0 \tag{1.46}$$

Thus this series converges absolutely, no matter what value we will pick for y.

Part 2: Find such an N
Let $f(x) = e^x$. We can formulate its Taylor-Series, stated above. Let $P_n(x)$denote the n-th Taylor polynom,

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k \tag{1.47}$$

where a is our developing point (here a is equal zero).

We can define the error of the n-th Taylor polynom to be $E_n(x) = f(x) - P_n(x)$. the error of the n-th Taylor polynom is difference between the value of the function and the Taylor polynomial This directly implies $|E_n(x)| = |f(x) - P_n(x)|$. By using the Lagrangian Error Bound it follows:

$$|E_n(x)| \leq \frac{M}{(n+1)!}|x-a|^{n+1} \tag{1.48}$$

with $a = 0$, where **M** is some value satisfying $|f^{(n+1)}(x)| \leq M$ on the interval $I = [a, x]$. Since we are interested in an upper bound of the error and since **a** is known, we can reformulate the interval as $I = [0, x_{max}]$, where

$$x_{max} = \|i\|k_{max}w_{max}h_{max} \tag{1.49}$$

We are interested in computing an error bound for $e^{ikwh(x,y)}$. Assuming the following parameters and facts used within Stam's Paper:

- Height of bump: 0.15micro meters
- Width of a bump: 0.5micro meters
- Length of a bump: 1micro meters
- $k = \frac{2\pi}{\lambda}$ is the wavenumber, $\lambda \in [\lambda_{min,}\lambda_{max}]$ and thus $k_{max} = \frac{2\pi}{\lambda_{min}}$. Since $(u, v, w) = -\omega_i - \omega_r$ and both are unit direction vectors, each component can have a value in range [-2, 2].
- for simplification, assume$[\lambda_{min,}\lambda_{max}] = [400nm, 700nm]$.

We get:

$$\begin{aligned} x_{max} &= \|i\| * k_{max} * w_{max} * h_{max} \\ &= k_{max} * w_{max} * h_{max} \\ &= 2 * (\frac{2\pi}{4 * 10^{-7}m}) * 1.5 * 10^{-7} \\ &= 1.5\pi \end{aligned} \tag{1.50}$$

and it follows for our interval $I = [0, 1.5\pi]$.

Next we are going to find the value for M. Since the exponential function is monotonically growing (on the interval I) and the derivative of the **exp** function is the exponential function itself, we can find such an M:

$$\begin{aligned} M &= e^{x_{max}} \\ &= exp(1.5\pi) \end{aligned}$$

and $|f^{(n+1)}(x)| \leq M$ holds. With

$$\begin{aligned} |E_n(x_{max})| &\leq \frac{M}{(n+1)!}|x_{max} - a|^{n+1} \\ &= \frac{exp(1.5\pi) * (1.5\pi)^{n+1}}{(n+1)!} \end{aligned} \tag{1.51}$$

we now can find a value of n for a given bound, i.e. we can find an value of $N \in \mathbb{N}$ s.t. $\frac{exp(1.5\pi)*(1.5\pi)^{N+1}}{(N+1)!} \leq \epsilon$. With Octave/Matlab we can see:

- if N=20 then $\epsilon \approx 2.9950 * 10^{-4}$
- if N=25 then $\epsilon \approx 8.8150 * 10^{-8}$
- if N=30 then $\epsilon \approx 1.0050 * 10^{-11}$

With this approach we have that $\sum_{n=0}^{25} \frac{(ikwh)^n}{n!} \mathcal{F}\{h^n\}(\alpha, \beta)$ is an approximation of $P(u,v)$ with error $\epsilon \approx 8.8150 * 10^{-8}$. This means we can precompute 25 Fourier Transformations in order to approximate P(u,v) having an error $\epsilon \approx 8.8150 * 10^{-8}$.

Using now our approximation for $P_{dtft} = \mathcal{F}^{-1}\{p\}(u,v)$ for the tristimulus value X, we will get:

$$\begin{aligned} X &= C(w_i, w_r) \int_\lambda \langle \left| P_{dtft}(\frac{2\pi u}{\lambda}, \frac{2\pi v}{\lambda}) \right|^2 \rangle S_x(\lambda) d\lambda \\ &= C(w_i, w_r) \int_\lambda \left| \sum_{n=0}^{N} \frac{(wk)^n}{n!} \mathcal{F}^{-1}\{i^n h^n\}(\frac{2\pi u}{\lambda}, \frac{2\pi v}{\lambda}) \right|^2 S_x(\lambda) d\lambda \end{aligned} \tag{1.52}$$

1.3.4 Sampling: Gaussian Window

Practically, we cannot compute the DTFT 1.7 numerically due to finite computer arithmetic, since w is a continuous function for the DTFT. The DFT 1.8 of a discrete height field patch is equivalent to the DTFT of an infinitely periodic function consisting of replicas of the same discrete patch. By windowing with a window function that is zero outside the central replica, the convolution of either the DFT or the DTFT of height field with the Fourier Transform of the window becomes equivalent.

Let $window_g$ denote the gaussian window with $4\sigma_s$ μm where $\sigma_f = \frac{1}{2\pi\sigma_s}$ let us further substitute $\mathbf{t}(\mathbf{x}, \mathbf{y}) = i^n h(x,y)^n$

$$\mathcal{F}^{-1}_{dtft}\{\mathbf{t}\}(u,v) = \mathcal{F}^{-1}_{fft}\{\mathbf{t}\}(u,v) window_g(\sigma_f) \tag{1.53}$$

Therefore we can deduce the following expression from this:

$$\begin{aligned}
\mathcal{F}^{-1}_{dtft}\{\mathbf{t}\}(u,v) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F^{-1}_{fft}\{\mathbf{t}\}(w_u,w_v)\phi(u-w_u,v-w_v)dw_u dw_v \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\sum_i\sum_j F^{-1}_{fft}\{\mathbf{t}\}(w_u,w_v) \\
&\quad \delta(w_u-w_i,w_v-w_j)\phi(u-w_u,v-w_v)dw_u dw_v \\
&= \sum_i\sum_j\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F^{-1}_{fft}\{\mathbf{t}\}(w_u,w_v) \\
&\quad \delta(w_u-w_i,w_v-w_j)\phi(u-w_u,v-w_v)dw_u dw_v \\
&= \sum_i\sum_j F^{-1}_{fft}\{\mathbf{t}\}(w_u,w_v)\phi(u-w_u,v-w_v) \qquad (1.54)
\end{aligned}$$

where

$$\phi(x,y) = \pi e^{-\frac{x^2+y^2}{2\sigma_f^2}} \qquad (1.55)$$

1.3.5 Final Expression

As the last step of our series of derivations, we plug all our findings together to one big equation in order to compute the color for each pixel on our mesh in the CIE_{XYZ} colorspace. For any given heigh-field $h(x,y)$ representing a small patch of a nano structure of a surface and the direction vectors w_s and w_r from figure 1.10 the resulting color caused by the effect of diffraction can be computed like: Let

$$P_\lambda(u,v) = F^{-1}_{fft}\{i^n h^n\}(\frac{2\pi u}{\lambda},\frac{2\pi v}{\lambda}) \qquad (1.56)$$

Then our final expression using our previous derivations will look like:

$$\begin{aligned}
\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} &= C(\omega_i,\omega_r)\int_\lambda \sum_{n=0}^{N}\frac{(wk)^n}{n!}\sum_{(r,s)\in\mathcal{N}_1(u,v)}|P_\lambda(u-w_r,v-w_s)|^2 \\
&\quad \phi(u-w_r,v-w_s)\begin{pmatrix} S_x(\lambda) \\ S_y(\lambda) \\ S_z(\lambda)\end{pmatrix} d\lambda
\end{aligned} \qquad (1.57)$$

where $\phi(x,y) = \pi e^{-\frac{x^2+y^2}{2\sigma_f^2}}$ is the Gaussian window 1.3.4.

1.4 Alternative Approach

1.4.1 PQ factors

In this section we are presenting an alternative approach to the previous Gaussian window approach 1.3.4 in order to solve the issue working with $DTFT$ instead the DFT. We assume, that a given surface S is covered by a number of replicas

of a provided representative surface patch f. In a simplified, one dimensional scenario, mathematically speaking, f is assumed to be a periodic function, i.e. $\forall x \in \mathbb{R} : f(x) = f(x + nT)$, where T is its period and $n \in \mathbb{N}_0$. Thus, the surfaces can be written formally as:

$$S(x) = \sum_{n=0}^{N} f(x + nT) \tag{1.58}$$

What we are looking for is an identity for the inverse Fourier transform of our surface S, required in order to simplify the (X, Y, Z) colors from 1.52:

$$\begin{aligned} \mathcal{F}^{-1}\{S\}(w) &= \int f(x)e^{iwx}dx \\ &= \int_{-\infty}^{\infty} \sum_{n=0}^{N} f(x + nT)e^{iwx}dx \\ &= \sum_{n=0}^{N} \int_{-\infty}^{\infty} f(x + nT)e^{iwx}dx \end{aligned} \tag{1.59}$$

Next, apply the following substitution $x + nT = y$ which will lead us to:

$$\begin{aligned} x &= y - nT \\ dx &= dy \end{aligned} \tag{1.60}$$

Plugging this substitution back into equation 1.59 we will get:

$$\begin{aligned} \mathcal{F}^{-1}\{S\}(w) &= \sum_{n=0}^{N} \int_{-\infty}^{\infty} f(x + nT)e^{iwx}dx \\ &= \sum_{n=0}^{N} \int_{-\infty}^{\infty} f(y)e^{iw(y-nT)}dy \\ &= \sum_{n=0}^{N} e^{-iwnT} \int_{-\infty}^{\infty} f(y)e^{iwy}dy \\ &= \sum_{n=0}^{N} e^{-iwnT} \mathcal{F}^{-1}\{f\}(w) \\ &= \mathcal{F}^{-1}\{f\}(w) \sum_{n=0}^{N} e^{-iwnT} \end{aligned} \tag{1.61}$$

We used the fact that the exponential term e^{-iwnT} is a constant factor when integrating along dy and the identity for the inverse Fourier transform of the function f. Next, let us examine the series $\sum_{n=0}^{N} e^{-iwnT}$ closer:

$$\begin{aligned} \sum_{n=0}^{N} e^{-uwnT} &= \sum_{n=0}^{N} (e^{-uwT})^n \\ &= \frac{1 - e^{iwT(N+1)}}{1 - e^{-iwT}} \end{aligned} \tag{1.62}$$

We recognize the geometric series identity for the left-hand-side of equation 1.62. Since our series is bounded, we can simplify the right-hand-side of equation 1.62.

Note that e^{-ix} is a complex number. Every complex number can be written in its polar form, i.e.

$$e^{-ix} = cos(x) + isin(x) \tag{1.63}$$

Using the following trigonometric identities

$$\begin{aligned} cos(-x) &= cos(x) \\ sin(-x) &= -sin(x) \end{aligned} \tag{1.64}$$

combined with 1.63 we can simplify the series 1.62 even further to:

$$\frac{1 - e^{iwT(N+1)}}{1 - e^{-iwT}} = \frac{1 - cos(wT(N+1)) + isin(wT(N+1))}{1 - cos(wT) + isin(wT)} \tag{1.65}$$

Equation 1.65 is still a complex number, denoted as $(p + iq)$. Generally, every complex number can be written as a fraction of two complex numbers. This implies that the complex number $(p + iq)$ can be written as $(p + iq) = \frac{(a+ib)}{(c+id)}$ for any $(a + ib), (c + id) \neq 0$. Let us use the following substitutions:

$$\begin{aligned} a &:= 1 - cos(wT(N+1)) & b &= sin(wT(N+1)) \\ c &= 1 - cos(wT) & d &= sin(wT) \end{aligned} \tag{1.66}$$

Hence, using 1.66, it follows

$$\frac{1 - e^{iwT(N+1)}}{1 - e^{-iwT}} = \frac{(a + ib)}{(c + id)} \tag{1.67}$$

By rearranging the terms, it follows $(a + ib) = (c + id)(p + iq)$ and by multiplying its right hand-side out we get the following system of equations:

$$\begin{aligned} (cp - dq) &= a \\ (dp + cq) &= b \end{aligned} \tag{1.68}$$

After multiplying the first equation of 1.68 by c and the second by d and then adding them together, we get using the law of distributivity new identities for p and q:

$$\begin{aligned} p &= \frac{(ac + bd)}{c^2 + d^2} \\ q &= \frac{(bc + ad)}{c^2 + d^2} \end{aligned} \tag{1.69}$$

Using some trigonometric identities and putting our substitution from 1.66 for a, b, c, d back into the current representation 1.69 of p and q we will get:

$$p = \frac{1}{2} + \frac{1}{2}\left(\frac{cos(wTN) - cos(wT(N+1))}{1 - cos(wT)}\right)$$
$$q = \frac{sin(wT(N+1)) - sin(wTN) - sin(wT)}{2(1 - cos(wT))} \tag{1.70}$$

Since we have seen, that $\sum_{n=0}^{N} e^{-uwnT}$ is a complex number and can be written as $(p + iq)$, we now know an explicit expression for p and q. Therefore, the one dimensional inverse Fourier transform of S is equal:

$$\begin{aligned}\mathcal{F}^{-1}\{S\}(w) &= \mathcal{F}^{-1}\{f\}(w) \sum_{n=0}^{N} e^{-iwnT} \\ &= (p + iq)\mathcal{F}^{-1}\{f\}(w)\end{aligned} \tag{1.71}$$

Now lets consider our actual problem description. Given a patch of a nano-scaled sureface snake shed represented as a two dimensional heightfield $h(x, y)$. We once again assume that this provided patch is representing the whole surface S of our geometry by some number of replicas of itself. Therefore, $S(x, y) = \sum_{n=0}^{N} h(x + nT_1, y + mT_2)$, assuming the given height field has the dimensions T_1 by T_2. In order to derive an identity for the two dimensional inverse Fourier transformation of S we can similarly proceed like we did to derive equation 1.71.

$$\begin{aligned}\mathcal{F}^{-1}\{S\}(w_1, w_2) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} h(x_1 + n_1T_1, x_2 + n_2T_2)e^{iw(x_1+x_2)}dx_1dx_2 \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} h(y_1, y_2)e^{iw((y_1-n_1T_1)+(y_2+n_2T_2))}dx_1dx_2 \\ &= \sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(y_1, y_2)e^{iw(y_1+y_2)}e^{-iw(n_1T_1+n_2T_2)}dy_1dy_2 \\ &= \sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} e^{-iw(n_1T_1+n_2T_2)} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} Box(y_1, y_2)e^{iw(y_1+y_2)}dy_1dy_2 \\ &= \left(\sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} e^{-iw(n_1T_1+n_2T_2)}\right)\mathcal{F}^{-1}\{h\}(w_1, w_2) \\ &= \left(\sum_{n_2=0}^{N_1} e^{-iwn_1T_1}\right)\left(\sum_{n_2=0}^{N_2} e^{-iwn_2T_2}\right)\mathcal{F}^{-1}\{h\}(w_1, w_2) \\ &= (p_1 + iq_1)(p_2 + iq_2)\mathcal{F}^{-1}\{h\}(w_1, w_2) \\ &= ((p_1p_2 - q_1q_2) + i(p_1p_2 + q_1q_2))\mathcal{F}^{-1}\{h\}(w_1, w_2) \\ &= (p + iq)\mathcal{F}^{-1}_{DTFT}\{h\}(w_1, w_2)\end{aligned} \tag{1.72}$$

Where we have defined

$$\begin{aligned}p &:= (p_1p_2 - q_1q_2) \\ q &:= (p_1p_2 + q_1q_2)\end{aligned} \tag{1.73}$$

For the identity of equation 1.72 we made use of Green's integration rule which allowed us to split the double integral to the product of two single integrations. Also, we used the definition of the 2-dimensional inverse Fourier transform of the height field function. We applied a similar substitution like we did in 1.60, but this time twice, once for x_1 and once for x_2 separately. The last step in equation 1.72, substituting with p and q in equation 1.73 will be useful later in the implementation. The insight should be, that the product of two complex numbers is again a complex number. We will have to compute the absolute value of $\mathcal{F}^{-1}\{S\}(w_1, w_2)$ which will then be equal $(p^2+q^2)^{\frac{1}{2}} \left|\mathcal{F}^{-1}\{h\}(w_1, w_2)\right|$

1.4.2 Interpolation

In 1.4.1 we have derived an alternative approach when we are working with a periodic signal instead using the gaussian window approach from sec *sec* : *gaussianwindow*. Its main finding 1.72 that we can just integrate over one of its period instead iterating over the whole domain. Nevertheless, this main finding is using the inverse DTFT. Since we are using

We are interested in recovering an original analog signal $x(t)$ from its samples $x[t] =$

Therfore, for a given sequence of real numbers $x[n]$, representing a digital signal, its correspond continuous function is:

$$x(t) = \sum_{n=-\infty}^{\infty} x[n] sinc\left(\frac{t-nT}{T}\right) \tag{1.74}$$

which has the Fourier transformation $X(f)$ whose non-zero values are confined to the region $|f| \leq \frac{1}{2T} = B$. When $x[n]$ represents time samples at interval T of a continuous function, then the quantity $f_s = \frac{1}{T}$ is known as its sample rate and $\frac{f_s}{2}$ denotes the Nyquist frequency. The sampling Theorem states that when a function has a Bandlimit B less than the Nyquist frequency, then $x(t)$ is a perfect reconstruction of the original function.

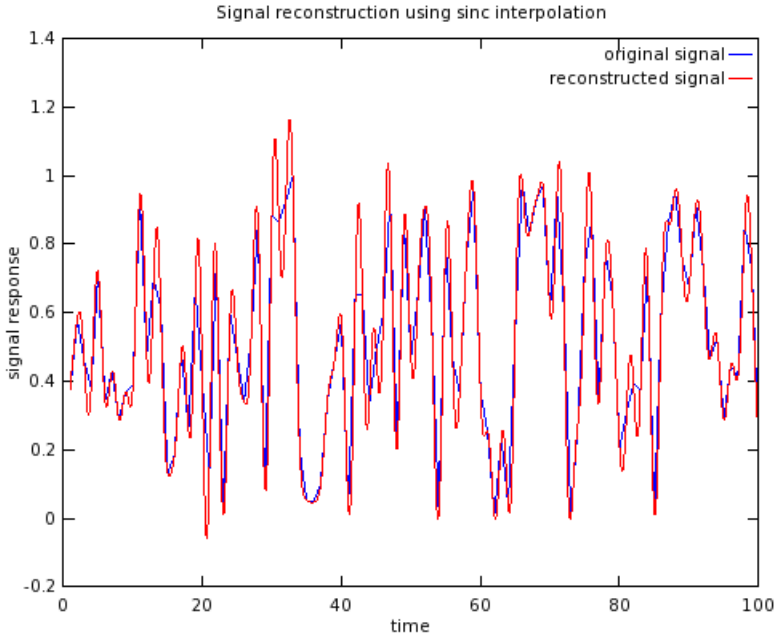


Abbildung 1.11: Comparission between a given random one dimensional input signal $s(t)$ and its sinc interpolation $\hat{s}(t)$. Notice that for the interpolation there were $N = 100$ samples from the original signal provided.

Kapitel 2

Implementation

In computergraphics, we are interested in synthesizing 2d images from a given scene containing our 3d geometries by using so called shader programs. This process is denoted as rendering. The purpose of shader programs, which are executed directly on the GPU hardware device, is to compute the colorization and illumination of the objects living in our scene. All these computations happen in several stages and depend on the provided scene-input parameters like the camera, light sources, objects material constants and the desired rendering effect one is interested in to model. The shader stages are implemented sequentially as small little programs, the so called vertex-, geometry- and fragment-shaders. Those stages are applied within the rendering pipeline sequentially.

Our shaders which we use are written in a high-level language called GLSL, the OpengGL Shading Language. The decision for using OpenGL has been made since my underlying framework, which is responsible for the precomputation of all scene date, is based on another framework, written in Java using JOGL in oder to communicate with the GPU and is also responsible to precompute all the relevant scene data. This framework, the so called jrtr framework, has been developed as an exercise during the class computer graphics held by M. Zwicker which I attended in autumn 2012. The framework itself has been used and extended during this thesis quite a lot. All necessary input data required within our java framework in order to perform the shading is precomputed in Matlab. This is basically addressing all the required precomputations for the provided heigh-fields, referring to computation of the inverse two dimensional Fourier transformations which are further explained within this chapter. The Matlab scripts themself rely on the provided snake nano-scaled sheds images, taken by AFM.

It's noteworthy that all the vertices and their associated data are processed within the vertex-shader, whereas the fragement shader's responsibility is to perform pixelwise rendering, using the input from the vertex shader. Just remember, fragements are determined by a triple of vertices. hence each pixel has assigned a trilinear interpolated value of all input parameters of its spanning vertices. Usually, all necessary transformations are applied vertex-wise, considering the vertex-shader as the precomputation stage for the later rendering within the rendering pipeline, in the fragment-shader. In the geometry shader, new vertices around a considered vertex can be created. this is useful for debugging - displaying normals graphically for example.

In this part of thesis we are going to explain how we render our BRDF formulation derived in the last section in practice. All the necessary computations in order to simulate the effect of diffraction are performed within a fragment shader. This implies that we are modeling pixelwise the effect of diffraction and hence the overall rendering quality and runtime complexity depends on rendering window's resolution.

By the end of this chapter we will have seen how our render works, what we have to precompute and how our shaders work.

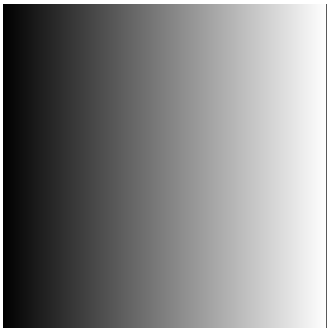
2.1 Precomputations in Matlab

Our first task is to precompute the inverse two dimensional discrete Fourier transformations for a given snake shed patch of interest taken by AFM. For that purpose we have written a small Matlab script conceptialized algorithmically in 2.1. Matlab is a interpreted scripting language which offers a huge collection of mathematical and numerically fast and stable algorithms. Our Matlab script reads a given image, which is representing a nano-scaled height field, and computes its inverse two dimensional DFT by using Matlab's internal inverse fast Fourier Transformation function, denoted by $ifft2$. Note that we only require one color channel of the input image since the input image is representing an height field, encoded by just one color. Basically, we are interested in computing the $ifft2$ for different powers of the input image since our taylor series approximation 1.3.3 relies on this. Keep in mind that taking the Fourier transformation of an arbitrary function will result in a complex valued output which implies that we will get a complex value for each pixel of our input image. Therefore, for each input image we get as many output images, representing the two dimensional inverse Fourier Transformation, as the minimal amount of taylor terms required for a well-enough approximation. In order to store our output images, we have to use two color channels instead just one like it was for the given input image. Some rendered images are shown in figure 2.1. Instead storing the computed values in plain images, we store our results in binary files. This allows us to have much higher precision for the output values and also it does not waste color channels. In our script every pixel value is normalized by its corresponding Fourier image extrema values such that is it lays in the range $[0, 1]$. Therefore, we have to remember store four scaling factors for each output image as well. Those are the real and imaginary minimum and maximum values. Later, using linear interpolation within the shader, we will get back the rescaled image's original pixel values.

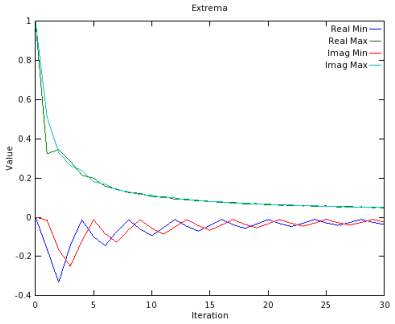
Algorithm 2.1 Precomputation: Fourier images

```
% maxH:     A floating-point number specifying
%           the value of maximum height of the
%           height-field in MICRONS, where the
%           minimum-height is zero.
%
% dH:       A floating-point number specifying
%           the resolution (pixel-size) of the
%           'discrete' height-field in MICRONS.
%           It must less than 0.1 MICRONS to
%           ensure proper response for
%           visible-range of light spectrum.
%
% termCnt: An integer specifying the number of
%           Taylor series terms to use.

function []= ComputeFFTImages(maxH, dh, termCnt)
dH = dh*1E-6;
% load patch into patchImg
patchImg = patchImg.*maxH;
% perform imrotate(patchImg, angle)
for t = 0 : termCnt
  patchFFT = power(1j*patchImg, t);
  fftTerm{t+1} = fftshift(ifft2(patchFFT));

  imOut(:,:,1)  = real(fftTerm{t+1});
  imOut(:,:,2)  = imag(fftTerm{t+1});
  imOut(:,:,3)  = 0.5;

  % perform imrotate(imOut, -angle)
  % find real and imaginary extrema of
  % write imOut, extrema, dH, into files.
end
```

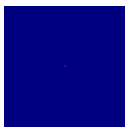
The command fftshift rearranges the output of the ifft2 by moving the zero frequency component to the centre of the image. This is useful for visualizing a Fourier Transform with zero frequency components in the middle of the spectrum.



(a) Blaze Patch



(b) Extrema Plot



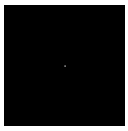
(c) ImRe0

(d) ImRe1

(e) ImRe4

(f) ImRe10

(g) ImRe20



(h) Re0

(i) Re1

(j) Re4

(k) Re10

(l) Re20

(m) Im0

(n) Im1

(o) Im4

(p) Im10

(q) Im20

Abbildung 2.1: Blaze

In figure 2.1 we see some Fourier images rendered by our Matlab script for the blaze grating $2.1(a)$ used as input image. For example figure $2.1(e)$ represents the two dimensional Fourier transform of the input image to the power of four times the imgaginary number i stored in a RGB image. Note that the red color channel $2.1(j)$ contains the real- and the green color channel $2.1(j)$ the imaginary part of the Fourier transform. The plot in figure $2.1(b)$ shows the development of blaze grating's extreme values for different powers.

2.2 Java Renderer

In autumn 2012, during the semester I have attented the class computer graphics held by M. Zwicker where we have developed a real time renderer program written in java. The architecture of the program is divided into two parts: a rendering engine, the so called jrtr (java real time renderer) and an application program. Figure 2.2 outlines the architecture of our renderer.

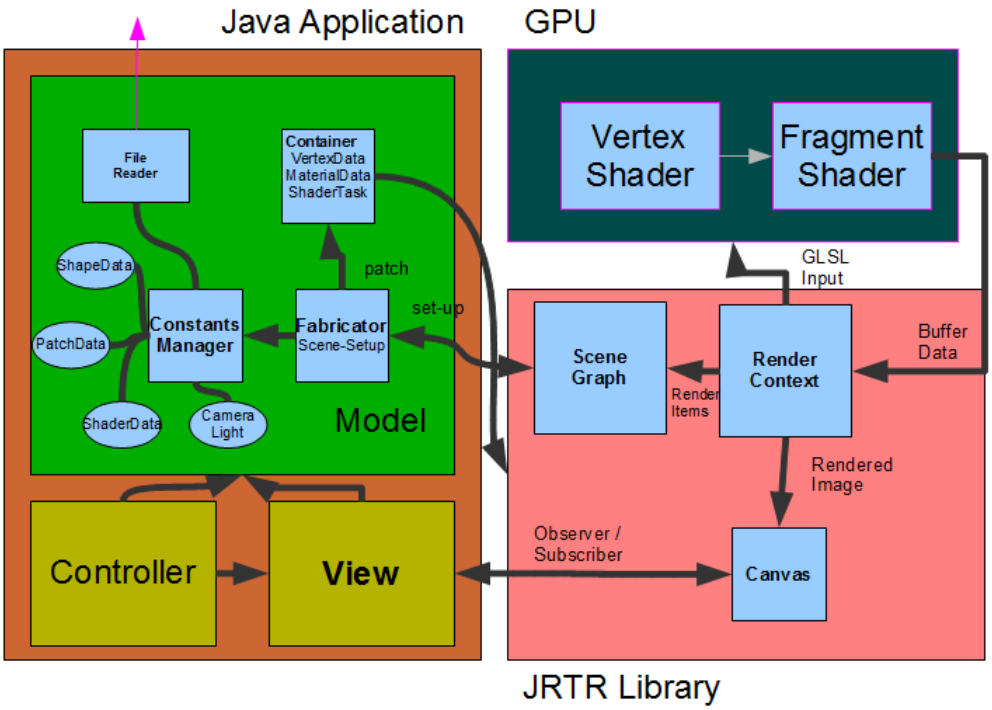


Abbildung 2.2: Renderer Architecture

The application program relies on the MVC (Model-View-Controller) architecture pattern. The View just represents a canvas in which the rendered images are shown. The Controller implements the event listener functionalities in order manipulate the rendered shape within the canvas. The Model of our application program consists of a Fabricator, a file reader and a constants manager. The main purpose of a Fabricator is to set up a rendering scene by accessing a constant manager containing many predefined scene constants. A scene consists of a camera, a light source, a frustum, shapes and their associated material constants. Such materials contain a shape's texture, associated Fourier images $reffig : matlabBlazeFourierImages$ for a given height field and other height field constants such as the maximal height of a bump. A shapes is a geometrical object defined by a wireframe mesh as shown in figure 2.3.

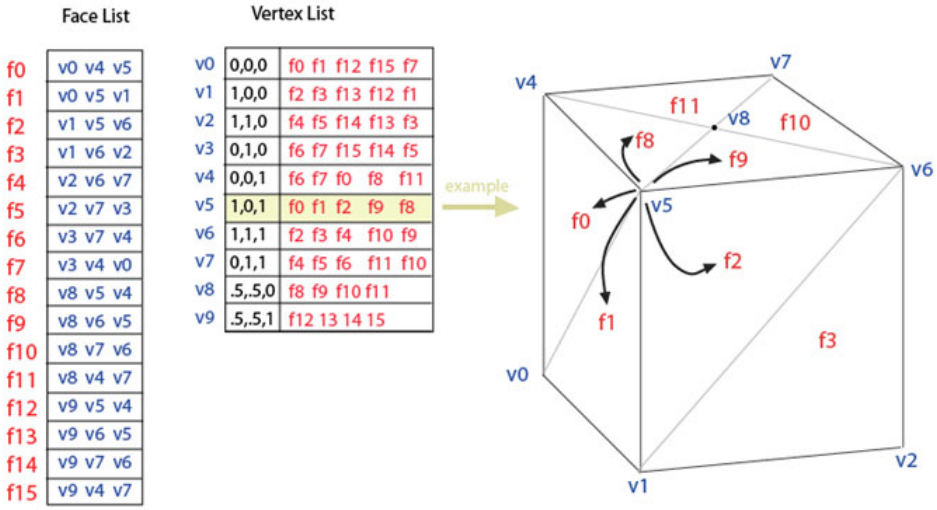


Abbildung 2.3: A wireframe mesh represents an object as a set of faces and a set of vertices.

Such a mesh is a special data structure consisting of vertices, each stored as a triple of xyz positions in an float array and triangles, each defined by a triple of vertex-indices which form a fragment each stored in an integer array. It is also possible to assign additional geometry data like a color, normals and texture coordinates associated to each vertex. The whole scene is stored within container data-structures, defined and managed within jrtr. In our case we rely on a scene graph, which contains all geometries and their transformations in a tree like structured hierarchy. The geometries are stored within an container, including all vertex attributes and the material constants. The jrtr rendering engine uses a low-level API, called OpenGL in order to communicate with the graphics processor unit (GPU) where the actual shading happens. Within jrtr's render context object, the whole resource-management for the rendering pipeline takes place. This means all required low-level buffers are allocated, flushed and assigned by the scene data attributes. The GPU's rendering pipeline will use those buffers for its shading process. Its first stage is the vertex shader 2.3.1 followed by the fragment shader 2.3.2. The jrtr framework also offers the possibility to assign arbitrary shaders.

2.3 GLSL Diffraction Shader

2.3.1 Vertex Shader

The Vertex shader is the first shading stage within our rendering pipeline and responsible for computing all necessary per vertex data. Usually, within a vertex shader each vertex position is transformed into a projective space:

$$p_{projective} = P \cdot C^{-1} \cdot M \cdot p_{obj} \tag{2.1}$$

Where M is a transformation from the local object space to the reference

coordinate system, called world space, C^{-1} camera matrix C and P the projection matrix. The camera matrix defines transformation from camera to world coordinates as shown in figure 2.4.

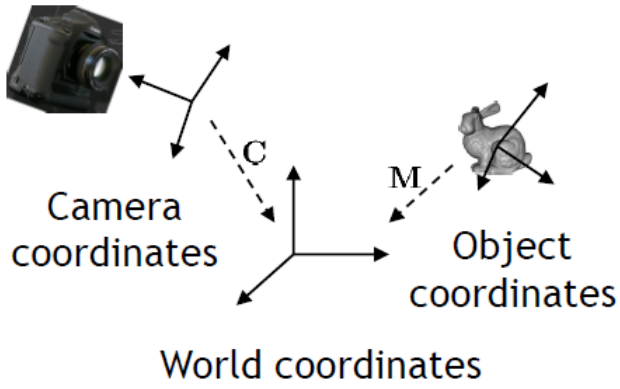


Abbildung 2.4: Camera coordinate system where its origin defines the center of projection of camera

The camera matrix is constructed from its center of projection e, the position the cameras looks at d and up vector denoted by up given in world coordinates like illustrated in figure 2.5

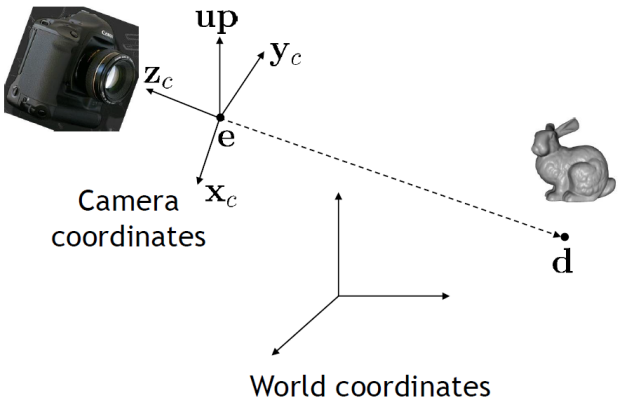


Abbildung 2.5: Illustration of involved components in order to construct the camera matrix. We introduce some helper vectors $z_c = \frac{e-d}{||e-d||}$, $x_c = \frac{up \times z_c}{||up \times z_c||}$ and $z_c \times x_c$ for the actual construction of the camera matrix

The mathematical representation of the camera matrix, using the helper vectors introduced in figure 2.5, looks like:

$$C = \begin{bmatrix} x_c & y_c & z_c & e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

All vertex shader output will be used within the fragment shader 2.3.2. In our vertex shader we also compute for every vertex in our current geometry the

direction vectors ω_i and ω_r described like in figure 1.10. Those direction vectors are transformed onto the tangent space, a local coordinate system spanned by a vertex's normal, tangent and binormal vector. The algorithm 2.2 stated below shows our vertex shader.

Algorithm 2.2 Vertex diffraction shader

foreach *Vertex* $v \in$ *Shape* **do**
$vec3N = normalize(modelM * vec4(normal, 0.0)).xyz$
$vec3T = normalize(modelM * vec4(tangent, 0.0)).xyz$
$vec3B = normalize(cross(N, T))$
$vec3Pos = ((cop_w - position)).xyz$
$vec4lightDir = (directionArray[0])$
$lightDir = normalize(lightDir)$
$l = projectVectorOnTo(lightDir, TangentSpace)$
$p = projectVectorOnTo(Pos, TangentSpace)$
$normalize(l); normalize(p)$
$p_{per} = P \cdot C^{-1} \cdot M \cdot p_{obj}$
end for

As already mentioned within our derivations 1.3.1, our light source is a directional light source 2.6.

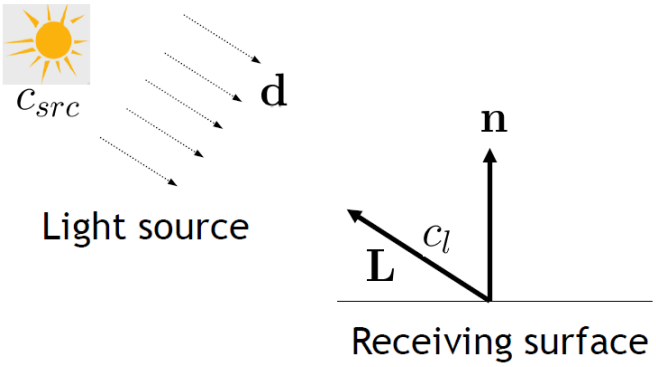


Abbildung 2.6: For a directional light source all light rays are in parallel.

2.3.2 Fragment Shader

The purpose of a fragment shader is to render per fragment. A fragment is spanned by three vertices of a given mesh. For each pixel within a fragment in the fragment shader, the output of from its spanning vertices computed in the vertex shaders 2.2 is trilinearly interpolated depending on the pixel's position within the fragment. Furthermore, there can be additional input be assigned which is not directly interpolated from the output of vertex shader programs. In our fragment shader 2.3 this will be: all the references to the image buffers, containing the Fourier images computed in Matlab 2.1, the number steps for the taylor approximation (in our shader 30), the minimal and maximal wavelength, scaling factors, a reference to a lookup table containing the CIE_{XYZ} color weights. Basically the whole computation within our fragment shader relies on

the the direction of light and the viewing direction. Our shader performs a numerical integration for our final derived expression 1.57 using the trapezoidal-rule with uniform discretization of the wavelength spectrum at $5nm$ step sizes. This implies we are compressing sampled frequencies to the region near to the origin of their frequency domain due to the fact we are dividing the (u, v) by the wavelength. The Gaussian window approach derived in 1.3.4 is performed for each discrete λ value using a window large enough to span $4\sigma_f$ in both dimensions. For computing DFT tables we generally use nanostructure height fields that span at least $65\mu m^2$ and are sampled with resolution of at least 100nm. This ensures that the spectral response encompasses al the wavelengths in the visible spectrum, i.e. from 380nm to 780nm. For natural structures in nano-scale, most of their spectral energy lies at lower spatial frequencies which maps closer to region $(u, v) = (0, 0)$ than higher frequencies. This is why we have chosen to sample (u, v) space non-linearly.

Algorithm 2.3 Fragment diffraction shader

1: **foreach** *Pixel* $p \in$ *Fragment* **do**
2: **INIT** $BRDF_{XYZ}, BRDF_{RGB}$ **TO** $vec4(0.0)$
3: $(u, v, w) = -\omega_i - \omega_r$
4: **for** $(\lambda = \lambda_{min};\ \lambda \leq \lambda_{max};\ \lambda = \lambda + \lambda_{step})$ **do**
5: $xyzWeights = ColorWeights(\lambda)$
6: $lookupCoord = lookupCoord(u, v, \lambda)$
7: **INIT** P **TO** $vec2(0.0)$
8: $k = \frac{2\pi}{\lambda}$
9: **for** $(n = 0$ **TO** $T)$ **do**
10: $taylorScaleF = \frac{(kw)^n}{n!}$
11: **INIT** F_{fft} **TO** $vec2(0.0)$
12: $anchorX = int(floor(center.x + lookupCoord.x * fftImWidth)$
13: $anchorY = int(floor(center.y + lookupCoord.y * fftImHeight)$
14: **for** $(i = (anchorX - winW)$ **TO** $(anchorX + winW + 1))$ **do**
15: **for** $(j = (anchorY - winW)$ **TO** $(anchorY + winW + 1))$ **do**
16: $dist = distVecFromOriginTo(i, j)$
17: $pos = localLookUp(i, j, n)$
18: $fftVal = rescaledFourierValueAt(pos)$
19: $fftVal \mathrel{*}= gaussWeightOf(dist)$
20: $F_{fft} \mathrel{+}= fftVal$
21: **end for**
22: **end for**
23: $P \mathrel{+}= taylorScaleF * F_{fft}$
24: **end for**
25: $xyzPixelColor \mathrel{+}= dot(vec3(|P|^2), xyzWeights)$
26: **end for**
27: $BRDF_{XYZ} = xyzPixelColor * C(\omega_i, \omega_r) * shadowF$
28: $BRDF_{RGB}.xyz = D_{65} * M_{XYZ-RGB} * BRDF_{XYZ}.xyz$
29: $BRDF_{RGB} = gammaCorrect(BRDF_{RGB})$
30: **end for**

From line 4 to 26:
Within this loop happens the uniform sampling along wavelength-space. $ColorWeights(\lambda)$ computes the color weight for the current wavelength λ by bilinear interpolation between the color weight for $\lceil\lambda\rceil$ and $\lfloor\lambda\rfloor$ which are stored in a external weights-table (assuming this table contains wavelengths in 1nm steps). At line 6: $lookupCoord(u, v, \lambda)$ the coordinates for the texture lookup are computed - See 2.5. Line 25 sums up the diffraction color contribution for the current wavelength in iteration λ.

From line 9 to 24:
Within this loop happens the Taylor series approximation until a predefined upper bound, denoted by T, has been reached. Basically, the spectral response is approximated for our current (u, v, λ). Furthermore, neighborhood boundaries for the gaussian-window sampling are computed, denoted as anchorX and anchorY.

From line 14 to 22:
In this most inner loop, the convolution of the gaussian window with the inverse FFT of the patch is pixel-wise performed. $gaussWeightOf(dist)$ computes the weights (1.55) from the distance between the current pixel's coordinates and the current neighbor's position in texture space. Local lookup coordinates for the current fourier coefficient $fftVal$ value are computed at line 17 and computed like described in 2.7. The actual texture lookup is performed at line 18 using those local coordinates. Inside $rescaledFourierValueAt$ the values $fftVal$ is rescaled by its extrema, i,e. $(fftVal * Max + Min)$ is computed, since $fftVal$ is normalized 2.1. The current $fftVal$ values in iteration is scaled by the current gaussian weight and then summed to the final neighborhood FFT contribution at line 20.

After line 26:
At line 27 the gain factor $C(\omega_i, \omega_r)$ 1.35 is multiplied by the current computed pixel color like formulated in 1.36. The gain factor contains the geometric term $refeq : geometricterm$ and the Fresnel term F. W approximate F by the Schlick approximation $A.1$, using an reactive index at 1.5 since this is close to the measured value from snake sheds. Our BRDF values are scaled by s shadowing function as described in (SEE REFERENCES - PAPER), since most of the grooves in the snake skin nano-structures would from a V-cavity along the plane for a wave front with their top-edges at almost the same height.

Last, we transform our colors from the CIE_XYZ colorspace to the CIE_RGB space using the CIE Standard Illuminant D65, followed by a gamma correction. See 2.4.3 for further insight.

2.4 Technical details

2.4.1 Texture lookup

In a GLSL shader the texture coordinates are normalized which means that the size of the texture maps to the coordinates on the range $[0, 1]$ in each dimension.

By convention the the bottom left corner of an image has the coordinates $(0,0)$, whereas the top right corner has the value $(1,1)$ assigned.

Given a nano-scaled surface patch P with a resolution A by A microns stored as an N by N pixel image I. Then one pixel in any direction corresponds to $dH = \frac{A}{N}\mu m$. In Matlab we compute a series of n output images $\{I_{out_1}, ..., I_{out_n}\}$ from I, which we will use for the lookup in our shader - See figure 2.7. For the lookup we use scaled and shifted (u,v) coordinates from 1.17.

Since the zero frequency component of output images was shifted towards the centre of each image, we have to shift u, v to the center of the current N by N pixel image by a bias b. Mathematically, the bias is a constant value is computed the following:

$$b = (N\%2 == 0) \quad ? \quad \frac{N}{2} : \frac{N-1}{2} \tag{2.3}$$

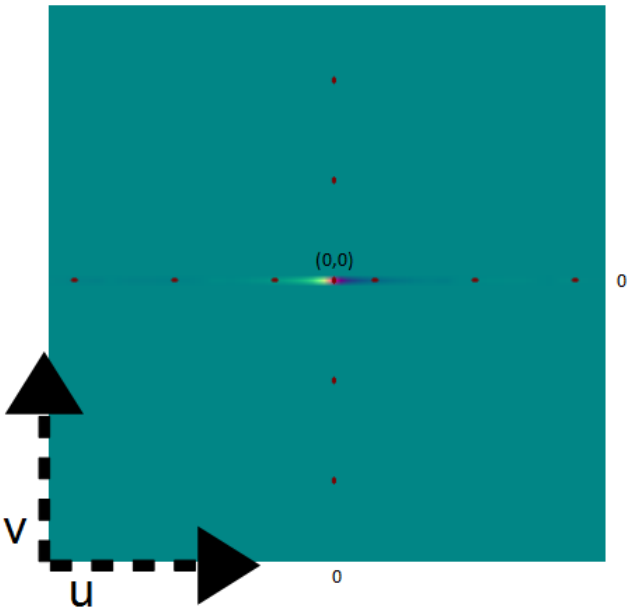


Abbildung 2.7: (u,v) lookup image

For the scaling we have to think a little further: lets consider a T periodic signal in time, i.e. $x(t) = x(t+nT)$ for any integer n. After applying the DFT, we have its discrete spectrum $X[n]$ with frequency interval $w0 = 2pi/T$ and time interval $t0$. Let $k = \frac{2\pi}{\lambda}$ denote the wavenumber for the current wavelength λ. Then the signal is both periodic with time period T and discrete with time interval t_0 then its spectrum should be both discrete with frequency interval w_0 and periodic with frequency period $\Omega = \frac{2\pi}{t_0}$. This gives us the idea how to discretize the spectrum: Let us consider our Patch P assuming it is distributed as a periodic function on our surface. Then, its frequency interval along the x direction is $w_0 = \frac{2\pi}{T} = \frac{2\pi}{N*dH}$. Thus only wave numbers that are integer multiples of w_0 after a multiplication with u must be considered, i.e. ku is integer multiple of w_0. Hence the lookup for the u-direction will look like:

$$\frac{ku}{w_0} = \frac{kuNdH}{2\pi} \tag{2.4}$$

$$= \frac{uNdH}{\lambda} \tag{2.5}$$

Using those findings 2.3, 2.5, the final (u, v) texture lookup-coorinates for the current wavelength λ in iteration, will then look like:

$$(u_{lookup}, v_{lookup}) = \left(\frac{uNdH}{\lambda} + b, \frac{vNdH}{\lambda} + b \right) \tag{2.6}$$

Note for the windowing approach we are visiting a one pixel neighborhood for each pixel p. This is like a base change with (u_{lookup}, v_{lookup}) as new coordinate system origin. The lookup coordinates for the neighbor-pixel (i, j) are:

$$(u_{lookup}, v_{lookup}) = (i, j) - (u_{lookup}, v_{lookup}) \tag{2.7}$$

2.4.2 Texture Blending

The final rendered color for each pixel is a weighted average of different color components, such as the diffraction color, the texture color and the diffuse color. In our shader the diffraction color is weighted by a constant $w_{diffuse}$. the texture color is once scales by a binary weight determined by the absolute value of the Fresnel Term F and once by $1 - w_{diffuse}$.

Algorithm 2.4 Texture Blending

$\alpha = (abs(F) > 1)?1 : 0$
$c_{out} = (1 - w_{diffuse}) * c_{diffraction} + (1 - \alpha) * c_{texture} + w_{diffuse} * c_{texture})$

2.4.3 Color Transformation

In our shader we access a table which contains precomputed CIE's color matching functions values from $\lambda_{min} = 380nm$ to $\lambda_{max} = 780nm$ in $5nm$ steps. Such a function value table can be found at downloaded at cvrl.ioo.ucl.ac.uk for example. We compute the (X, Y, Z) CIE_{XYZ} color values as described in 1.1.2.

We can transform the color values into CIE_{RGB} by performing the following linear transformation:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = M \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.8}$$

where one possible transformation is:

$$M = \begin{bmatrix} 0.41847 & -0.15866 & -0.082835 \\ -0.091169 & 0.25243 & 0.015708 \\ 0.00092090 & -0.0025498 & 0.17860 \end{bmatrix} \tag{2.9}$$

There are some other color space transformation. The shader uses the CIE Standard Illuminant D65 which is intended to represent average daylight. Using D65 the whole colorspace transformation will look like:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = M \cdot \begin{bmatrix} X \cdot D65.x \\ Y \cdot D65.y \\ Z \cdot D65.z \end{bmatrix} \tag{2.10}$$

Last we perfrom gamma correction on each pixel's (R, G, B) value. Gamma correction is a non linear transformation which controls the overall brightness of an image.

2.5 Discussion

The fragment shader algorithm described in 2.3 performs the gaussian window approach by sampling over the whole wavelength spectrum in uniform step sizes. This algorithm is valid but also slow since we iterate for each pixel over the whole lambda spectrum. Furthermore, for any pixel, we iterate over its 1 neighborhood. Considering the loop for the taylor approximation as well, we will have a run-time complexity of $O(\#spectrtumIter \cdot \#taylorIter \cdot neighborhoodRadius^2)$. Hence, Instead sampling over the whole wavelength spectrum, we could instead integrate over just a few required lambdas which are elicited like the following: Lets consider (u, v, w) defined as 1.17. Let d be the spacing between two slits of a grating. For any $L(\lambda) \neq 0$ it follows $\lambda_n^u = \frac{du}{n}$ and $\lambda_n^v = \frac{du}{n}$. For $n = 0$ there it follows $(u, v) = (0, 0)$. If $u, v > 0$

$$N_{min}^u = \frac{du}{\lambda_{max}} \leq n_u \leq \frac{du}{\lambda_{min}} = N_{min}^u$$
$$N_{min}^v = \frac{dv}{\lambda_{max}} \leq n_v \leq \frac{dv}{\lambda_{min}} = N_{min}^v$$

If $u, v < 0$

$$N_{min}^u = \frac{du}{\lambda_{min}} \leq n_u \leq \frac{du}{\lambda_{min}} = N_{max}^u$$
$$N_{min}^v = \frac{dv}{\lambda_{min}} \leq n_v \leq \frac{dv}{\lambda_{min}} = N_{max}^v$$

By transforming those equation to $(\lambda_{min}^u, \lambda_{min}^u)$, $(\lambda_{min}^v, \lambda_{min}^v)$ respectively for any (u, v, w) for each pixel we can reduce the total number of required iterations in our shader.

Another variant is the PQ approach described in chapter 2 1.4.1. Depending on the interpolation method, there are two possible variants we can think of as described in 1.4.2. Either we try to interpolate linearly or use sinc interpolation. The first variant does not require to iterate over a pixel's neighborhood, it is also faster than the gaussian window approach. One could think of a combination of those tho optimization approaches. Keep in mind, both of these approaches are further approximation. The quality of the rendered images will suffer using those two approaches. The second variant, using the sinc function interpolation is well understood in the field of signal processing and will give us reliable

results. The drawback of this approach is that we again have to iterate over a neighborhood within the fragment shader which will slow down the whole shading. The following algorithm describes the modification of the fragment shader 2.3 in oder to use sinc interpolation for the pq approach 1.4.1.

Algorithm 2.5 Sinc interpolation for pq approach

foreach *Pixel* $p \in$ *Image* I **do**
  $w_p = \sum_{(i,j)\in\mathcal{N}_1(p)} sinc(\Delta_{p,(i,j)} \cdot \pi + \epsilon) \cdot I(i,j)$
  $c_p = w_p \cdot (p^2 + q^2)^{\frac{1}{2}}$
  $render(c_p)$
end for

In a fragment shader we compute for each pixel p in the current fragment its reconstructed function value $f(p)$ stores in w_p. w_p is the reconstructed signal value at $f(p)$ by the sinc function as described in 1.4.2. We calculate the distance $\Delta_{p,(i,j)}$ between the current pixel p and each of its neighbor pixels $(i,j) \in \mathcal{N}_1(p)$ in its one-neighborhood. Multiplying this distance by π gives us the an angle used for the sinc function interpilation. We add a small integer ϵ in order to avoid division by zeros side-effects.

Kapitel 3

Evaluation Data Acquisition

3.1 Data Acquisition

For measurement on the true surface topography of snake sheds, samples are stuck on glass plates using double face tape, the animal was pushed up below a hollowed plate letting the skin emerging from the top of the plate using an atomic force microscope (AFM). An AFM is a microscope that uses a tiny probe mounted on a cantilever to scan the surface of an object. The probe is extremely close tobut does not touch the surface. As the probe traverses the surface, attractive and repulsive forces arising between it and the atoms on the surface induce forces on the probe that bend the cantilever. The amount of bending is measured and recorded, providing a map of the atoms on the surface. Atomic force microscopes is a very high-resolution type of scanning probe microscopy, with demonstrated resolution on the order of fractions of a nanometer, more than 1000 times better than the optical diffraction limit.

3.2 Diffraction Gratings

An idealised grating like in figure 3.2 is made of a very large number of parallel, evenly spaced slits in an opaque sheet. Typically, it would have about 10,000 slits. In order to cause diffraction, the spacing between slits must be wider than the wavelength of the incoming light beam. Each slit in the grating acts as a quasi point light source from which light propagates in all directions. Figure 3.1 illustrates this behaviour for a monochromatic light source passing through a grating and shows that the the outgoing angle will be different from the incident angle. Hence, the diffracted light 1.1.1 is composed of the sum of interfering wave components emanating from each slit in the grating.

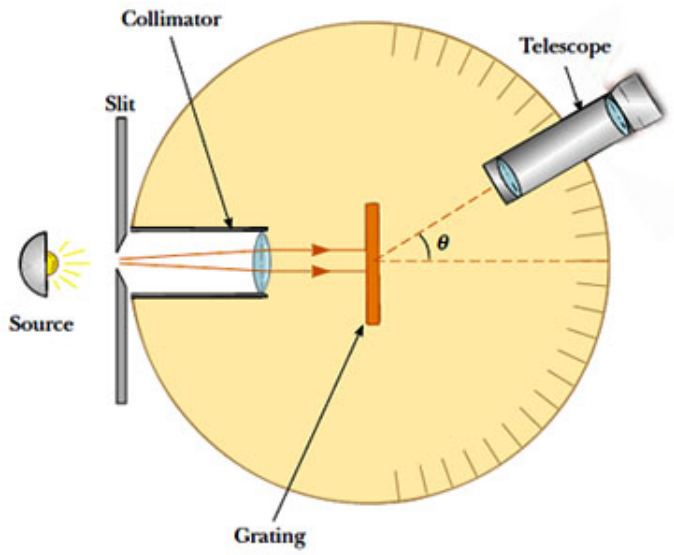


Abbildung 3.1: Spectometer: When a beam of monochromatic light passes through a grating placed in a spectrometer, images of the sources can be seen through the telescope at different angles.

Suppose monochromatic light is directed at the grating parallel to its axis as shown in figure 3.1. Let the distance between successive slits be equal d.

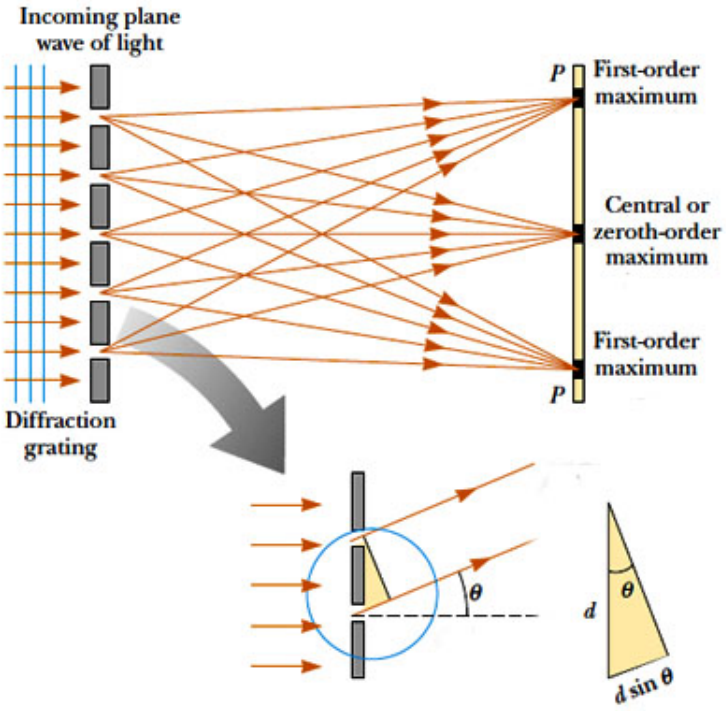


Abbildung 3.2: Light directed to parallel to grating:

The diffraction pattern on the screen is the result of the combined effects of diffraction and interference. Each slit causes diffraction, and the diffracted beams in turn interfere with one another to produce the pattern. The path difference between waves from any two adjacent slits can be found by dropping a perpendicular line between the parallel waves. By geometry, this path difference is $dsin(\theta)$. If the path difference equals one wavelength or some integral multiple of a wavelength, waves from all slits will be in phase and a bright line will be observed at that point. Therefore, the condition for maxima in the interference pattern at the angle θ is:

$$dsin(\theta) = m\lambda \tag{3.1}$$

where $m \in \mathbb{N}_0$ is the order of diffraction.

Because d is very small for a diffraction grating, a beam of monochromatic light passing through a diffraction grating is splitted into very narrow bright fringes at large angles θ.

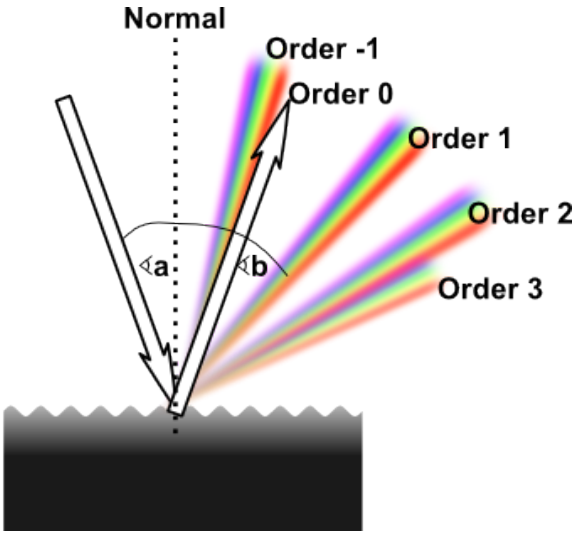


Abbildung 3.3: Different Orders of diffraction

When a narrow beam of white light is directed at a diffraction grating along its axis, instead of a monochromatic bright fringe, a set of colored spectra are observed on both sides of the central white band as shown in figure 3.3.

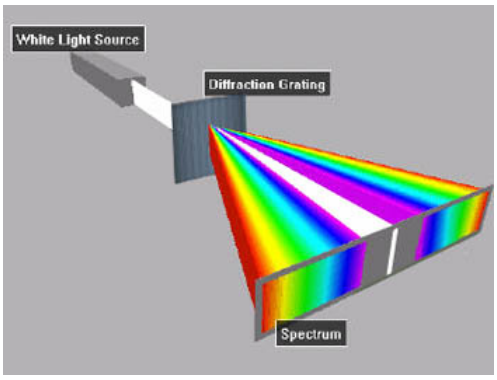


Abbildung 3.4: White Light beam causes coloured diffraction spectra

Since the angle θ increases with wavelength λ, red light, which has the longest wavelength, is diffracted through the largest angle. Similarly violet light has the shortest wavelength and is therefore diffracted the least. This relationship between angle and wavelength is illustrated in figure 3.4. Thus, white light is split into its component colors from violet to red light. The spectrum is repeated in the different orders of diffraction, emphasizing certain colors differently, depending on their order of diffraction like shown in figure 3.3. Note that only the zero order spectrum is pure white. Figure 3.5 shows the relative intensity resulting when a beam of light hits a diffraction grating for different number of periods. From the graph we recognise that the more slits a grating has, the sharper more slopes the function of intensity gets. This is similar like saying that, the more periods a grating has, the sharper the diffracted color spectrum

gets like shown in figure 3.6.

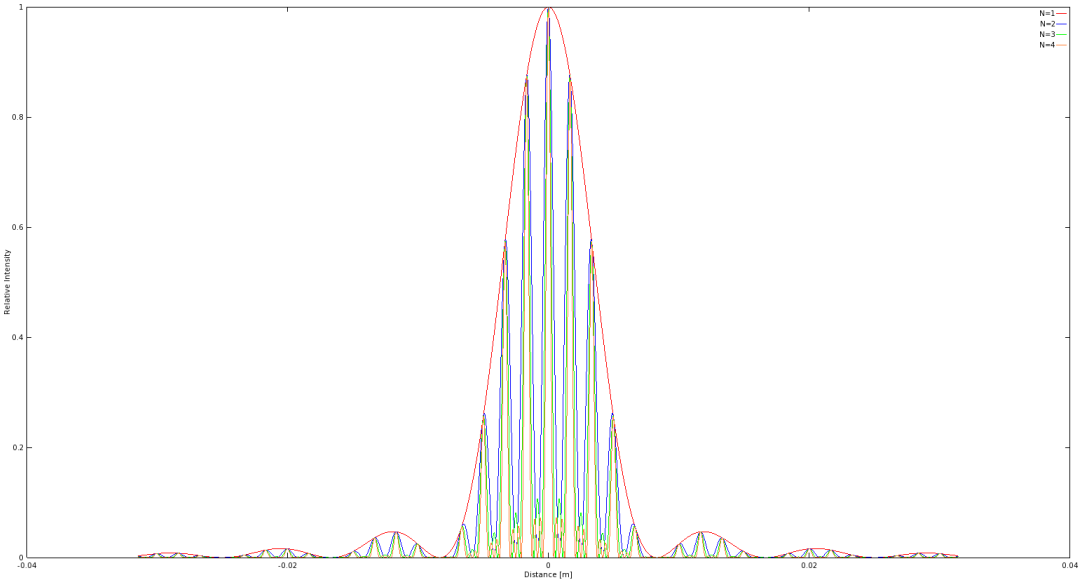
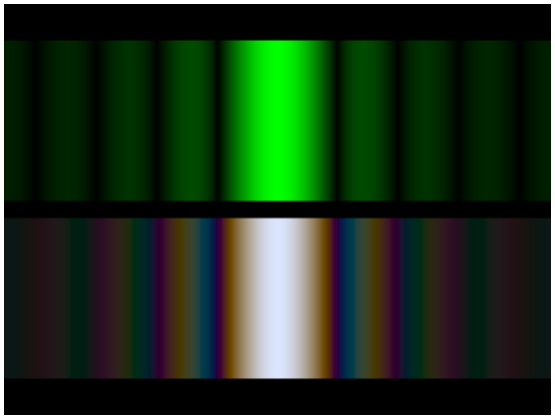
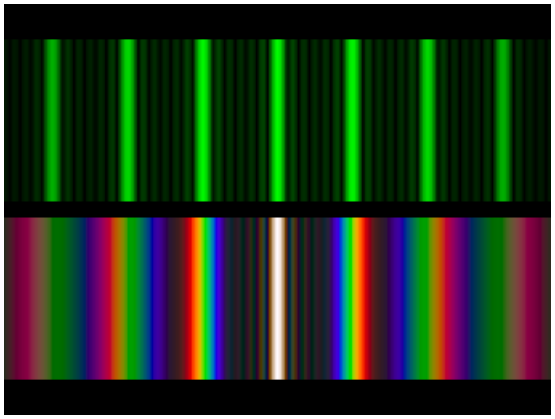


Abbildung 3.5: Relative intensitiies of a diffracted beam of light at wavelength $\lambda = 500nm$ on a grating for different number of periods N width slit width of 30 microns and slit seperation of 0.15 mm each. The viewer is 0.5m apart from the grating.



(a) one slit



(b) seven slits

Abbildung 3.6: Difference of diffraction pattern between a monochromatic (top) and a white (bottom) light spectra for different number of slits.

3.3 Evaluation

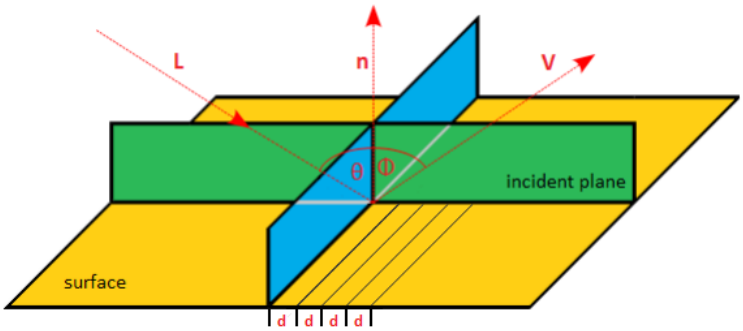


Abbildung 3.7: Experimental setup for evaluation: A light beam with direction L hits the surface, representing a grating pattern with periodicity d, at the incident plane relative to the surface normal n at angle θ and emerges at angle ϕ with direction V.

The physical reliability of our BRDF models has been verified by applying those on various patches which are a synthetic blazed grating, an Elaphe and a Xenopeltis snake shed sample patch. We compared the resulting response against the response resulting by the grating equation, which models the relationship between the grating spacing and the angles of the incident and diffracted beams of light. Figure 3.7 illustrates the geometrical setup for our evaluation approach: A monochromatic beam of light with wavelength λ hits a surface with periodicity d at an angle θ relative to the normal n along its incident plane. The beam emerges from the surface at the angle ϕ.

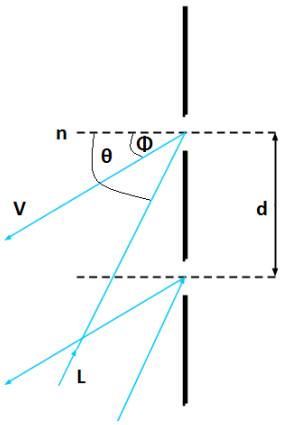


Abbildung 3.8: Reflecting grating: When the incident light direction is not parallel to its axis at the grating, there is another $sin(\phi)$ involved. See also the grating equation 3.2.

The maximum in intensity is given by the grating equation derived from the

equation 3.1 following figure 3.8:

$$sin(\theta) = sin(\phi) + \frac{m\lambda}{d} \tag{3.2}$$

In our evaluation we are interested in the first order diffraction, i.e. m equals one which. We further assume that the incident light direction ω_i is given. In contrast the direction of the reflected wave ω_r is not given. In Mathematics, a three dimensional direction vector is fully defined by two two angles, i.e. it can be represented by spherical coordinates with radius $r = 1$. By convention, we denote those two vectors by θ and ϕ like in figure 3.7. Hence, θ_i, ϕ_i and ϕ_r are given constants whereas θ_i is a free parameter for our evaluation simulation. Therefore, we are going to compare the maxima for peak viewing angles corresponding to each wavelength using data produced by our method against the maxima resulting by the grating equation 3.2.

3.3.1 Precomputation

For evaluation purposes we have implemented our brdf models in java. We once again use our geometrical setup as illustrated in figure 1.10 where θ_i, ϕ_i and ϕ_r are provided as input values and θ_i is a free parameter. Within our evaluation we have set them to $\theta_i = 75$ $\phi_i = 0$ $\phi_r = 180$ degrees. The wavelength space Λ and the range Θ of our free parameter θ_i are discretized in equidistant steps whereas their step sizes are given as input arguments for our Java program:

$$\Lambda = \{\lambda | \lambda = \lambda_{min} + k \cdot \lambda_{step}, \quad k \in \{0, .., C-1\}\} \tag{3.3}$$

where $\lambda_{step} = \frac{\lambda_{max} - \lambda_{min}}{C-1}$ and C is the discretisation level of the lambda space. We similarly discretise the angle space by predefining an minimal and maximal angle boundary and $ceil(angMax - angMin)/angInc$ is the number of angles. Our Java BRDF model implementations are applied on the grid $[\Lambda, \Theta]$ and will store their spectral response in a matrix

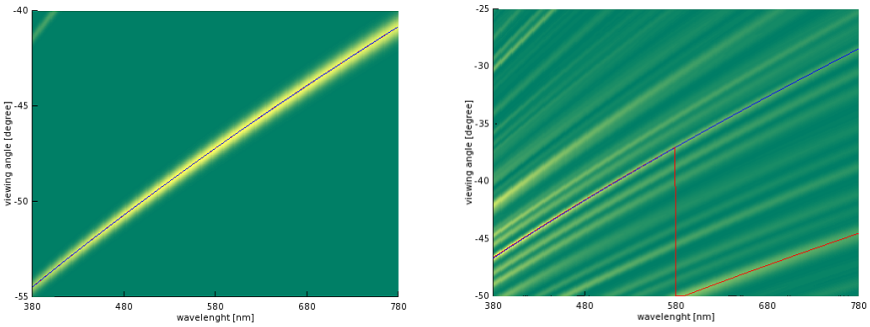
$$R = \{response(\lambda_i, \theta_r^j) | i \in Index(\Lambda), \quad j \in Index(\Theta)\} \tag{3.4}$$

We will plot this matrix and compare its graph against the grating equation for similar condition like in stated in algorithm 3.1.

Algorithm 3.1 Vertex diffraction shader

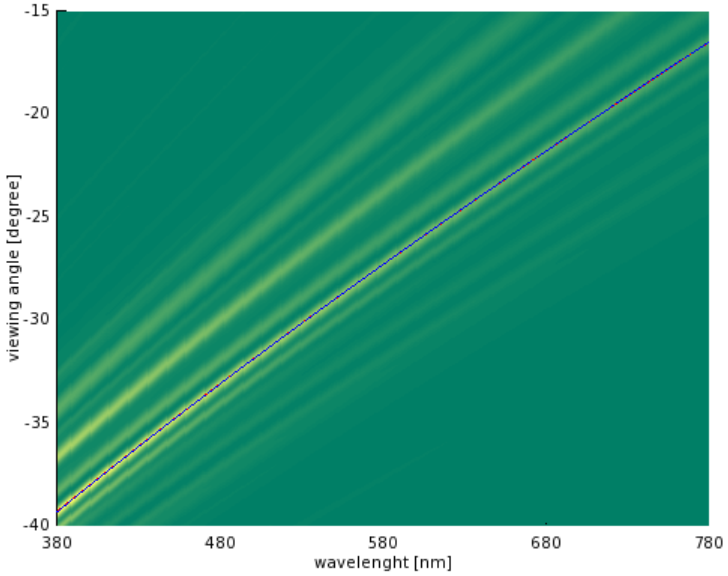
load matrix R 3.4
$\lambda_{count} = |\Lambda|$
$\lambda_{inc} = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{count}}$
$\lambda = \lambda_{min} + \lambda_{inc} \cdot (-1 + [1 : \lambda_{count}])$
$[maxCmaxI] = max(R)$
$viewAngForMax = angMin + angInc \cdot (maxI - 1)$
$thetaV = asin\left(\frac{\lambda}{d} - \sin\left(\frac{\theta_I \pi}{180}\right)\right) \cdot \frac{180}{\pi}$
$plot(\lambda, viewAngForMax)$ ▷ graph resulting by our brdf model
$plot(lambda, thetaV)$ ▷ graph resulting by grating equation

3.3.2 Data evaluation



(a) Blaze grating

(b) Xenopeltis



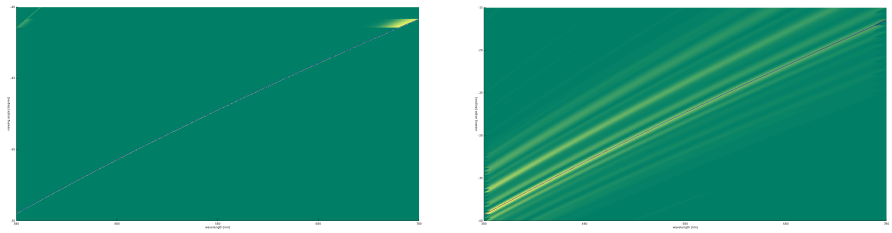
(c) Elaphe

Abbildung 3.9: Reflectance obtained by using the shading approach described in algorithm 2.3 simulating a BRDF which models the effect of diffraction at different viewing angles over the spectrum of visible light.

Figure 3.9 contains our evaluations graphs of algorithm 2.3 for different idealized periodic structure using an illumination angle $\theta_i = 75$ degrees. The graphs 3.9(c) and 3.9(c) in figure 3.9 show reflectance. Higher values are in yellow and lower values in green. For each of the graphs we determine the viewing angles with peak reflectance for various wavelengths and then plot this peak viewing angles against their wavelength as solid red curves. The blue curve represents diffraction angles for an idealized periodic structure with a certain periodicity d according to the grating equation 3.2. The corresponding periodicity is estimated using the precomputed response data. The red and blue curve are closely

overlapping in our figures 3.9(a) and 3.9(c). For blaze and Elaphe there is only diffraction along only along one direction perceivable. Since blazed grating is synthetic we use its exact periodicity to plot the blue curve instead of estimating it. Xenopeltis evaluated just along the direction for the finger like structures. For Xenopeltis it is interesting to see that the red curve for the peak viewing angle toggles between two ridges corresponding to two different periodicities. this happens because there are multiple sub regions of the nanostructure with slightly different orientations and periodicity. Each sub region carves out a different yellowish ridge. depending on the viewing agnle, reflectance due to one such subregion can be higher than from the others.

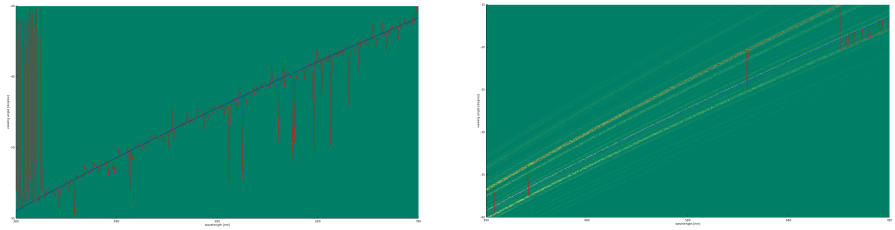
Figure **??** shows the evaluation plots for the (N_{min}, N_{max}) approach 2.5 which integrates over a reduced wavelength spectrum. This optimization is mentioned within the discussion section of the implementation chapter as a further optimization in order to enhance the overall run-time complexity. As we can see from the plots the graphs again closely match the grating equation graphs. Nevertheless we observe little variations: discuss them



(a) Blaze grating (b) Elaphe

Abbildung 3.10: Reflectance obtained using $N_{min}N_{max}$ optimization apporach

Last let us consider the graphs for the PQ approach 2.5 in figure **??**. for blaze grating, differences, but also similarities.



(a) Blaze grating (b) Elaphe

Abbildung 3.11: Reflectance obtained using PQ optimization apporach

subvariants: sample whole lambda space, just a few lambdas, pq approach.

Anhang A

Appendix

A.1 Schlick's approximation

The Fresnel's equations describe the reflection and transmission of electromagnetic waves at an interface. That is, they give the reflection and transmission coefficients for waves parallel and perpendicular to the plane of incidence. Schlick's approximation is a formula for approximating the contribution of the Fresnel term where the specular reflection coefficient R can be approximated by:

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5 \tag{A.1}$$

and

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

where θ is the angle between the viewing direction and the half-angle direction, which is halfway between the incident light direction and the viewing direction, hence $\cos\theta = (H \cdot V)$. And n_1, n_2 are the indices of refraction of the two medias at the interface and R_0 is the reflection coefficient for light incoming parallel to the normal (i.e., the value of the Fresnel term when $\theta = 0$ or minimal reflection). In computer graphics, one of the interfaces is usually air, meaning that n_1 very well can be approximated as 1.

A.2 Spherical Coordinates

$$\forall \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3 : \exists r \in [0, \infty) \exists \phi \in [0, 2\pi] \exists \theta \in [0, \pi] \text{ s.t.}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} rsin(\theta)cos(\phi) \\ rsin(\theta)sin(\phi) \\ rcos(\theta) \end{pmatrix}$$

Tabellenverzeichnis

Abbildungsverzeichnis

List of Algorithms

Literaturverzeichnis

[Doe00] Doe, John: *Title*. Publisher, 0000. – ISBN 0000000000

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: .....

Matrikelnummer: .....

Studiengang: ……

Bachelor ☐ Master ☐ Dissertation ☐

Titel der Arbeit: .....

.....

.....

LeiterIn der Arbeit: .....

.....

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

.....
Ort/Datum

.....
Unterschrift