An Interactive Shader for Natural Diffraction Gratings

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

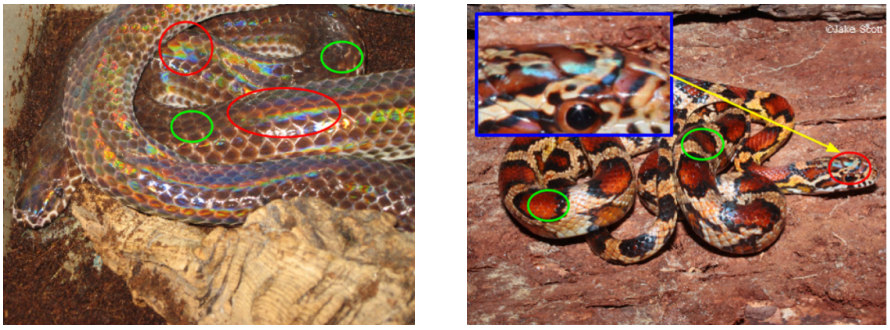
Contents

Chapter 1

Introduction

1.1 Motivation

As human beings, we visually perceive and experience our whole world in terms of colors, resulting from various physical phenomina involving interaction between light and matter. Particularly in nature, there are basically two main causes for color production. Firstly, due to pigmentation, which occurs since certain molecules in a biological structure selectively absorb or reflect specific wavelengths from an incident light source. And secondly because of structural colors which are the result of physical interaction of light with a nanostructure, exclusively relying on the structuring of the material and not any other property. A natural diffraction grating is a semitransparent layer of biological nano-structures which exhibits a certain degree of regularity to produce structural colors by diffracting an incident light source. One particular example for such biological color production are the colors we can see when having a closer look at the illuminated skin of snakes, as shown in figure 1.1.

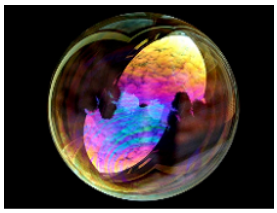


(a) Xenopeltis snake (b) Elaphe Guttata snake

Figure 1.1: Examples of pigmentation color (green circles) and structural color (red circles) on different snake species[1].

[1] image source of figure 1.1(*a*) `http://www.snakes-alive.co.uk/gallery_5.html` and figure 1.1(*b*) `http://www.the-livingrainforest.co.uk/living/view_price.php?id=464`

Some species like Xenopeltis express structural colors in form of iridescent patterns along their scales way strongerthan others like Elaphe species. The reason for this lies on the nanostructure of their skins. There are a vast amount of additional reasons for producing structural colors in nature, such as thin film interference, intra-cellular photonic crystals or diffraction gratings. More detailed examples are shown in figure 1.2.



(a) Thin Film Interference soap bubble

(b) Multilayer Interference on abdomen of beetle

(c) Photonic crystals in Wings of butterfly

(d) Scattering of light on butterfly wing

(e) Artificial Diffraction Grating on CD

(f) Natural Diffraction Grating on snake skin

Figure 1.2: Examples[2] for structural colors on the wings and the abdomen of insects, liquids, synthetic structures, and on scales on the skin of reptiles.

As far back as in the 17th century, Robert Hooke was able to relate the cause of structural colors to the the microstructure of a material. During his examinations of peacock feathers he found that the colors could be made disappear by wetting the feathers and further observed that the feathers are made of tiny ridges. Building on top of the latest knowledge about interference, Newton related structural colors with wave interference. Recently, in the field of computer graphics, many researchers have developed models to render structural colors, but most of the currently available models are not able to perform interactive rendering or are oversimplified and thus cannot model accurately the effect of diffraction.

[2]image source of figure:

- 1.2(a): `http://www.ualberta.ca/~pogosyan/teaching/PHYS_130/FALL_2010/lectures/lect33/lecture33.html`
- 1.2(b): `http://www.itp.uni-hannover.de/~zawischa/ITP/multibeam.html`
- 1.2(c): `http://upload.wikimedia.org/wikipedia/commons/a/a4/Parides_sesostris_MHNT_dos.jpg`
- 1.2(d): From paper [MT10], figure 6.
- 1.2(e): `http://cnx.org/content/m42496/latest/?collection=col11428/latest`
- 1.2(f): `http://www.snakes-alive.co.uk/gallery_5.html`

This thesis investigates this particular problem in detail and provides a solution for rendering structural colors due to diffraction on natural gratings.

1.2 Goals

The purpose of this thesis is firstly, to simulate physically accurate structural colors caused by the effect of diffraction on various biological structures and secondly implement this simulation as a renderer with interactive behaviour. We mainly focus on structural colors generated by natural diffraction gratings. In particular the approach presented in this thesis applies to surfaces with quasiperiodic structures at the nanometer scale which can be represented as height fields stored in gray-scale images.

Natural gratings like this are found on the scale of reptiles, wings of butterflies or the bodies of various insects but we restrict ourself and focus on snake skins. The data of our discrete valued height fields, which are representing the surface of a measured snake skin was acquired using atomic force microscopy (AFM)[3]. Figure 1.3 shows a measured height field of a Xenopeltis snake, which stored in a grayscale image. The surface of its skin is composed of many finger like structures. Locally, these fingers seem to be very regularly aligned (red box). However, globally, we observe that the alignment of the fingers is irregularly curved (indicated by green curves) along the whole surface. This kind of global irregularity is why it makes it hard to model the structural complexity of natural gratings[4].

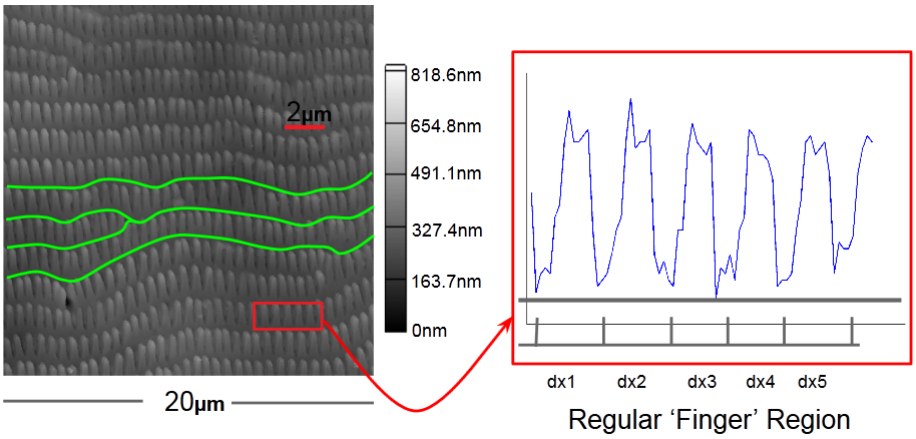


Figure 1.3: Height field of a Xenopeltis snake[5] skin taken by AFM and stored as a grayscale image. Locally, this natural grating consists of regularly aligned (red box) finger-like substructures, but globall we observe a curved alignment of these structures (green curves).

[3]All data is provided by the Laboratory of Artificial and Natural Evolition in Geneva. See their website:`www.lanevol.org`

[4]E.g. by relying on statistical methods, caputuring surface details by introducing an appropriate distribution function of the finger strucures.

The renderer discussed in this thesis is based on the pioneering work of J. Stam about diffraction shaders [Sta99] in which he formulated a BRDF modelling the effect of diffraction. Nevertheless we have to adapt his BRDF model since his model assumes, that a given surface of a grating can either be formulated by an analytical function, and therefore has a closed form solution or it is simple enough to be modelled effectively by relying on statistical methods. However, we are dealing with natural diffraction gratings, represented as explicitly formulated height fields, which unfortunately are neither known analytically nor do they fit into simple statistical models. This thesis thus proposes an extension of J. Stam's work for the complex case of explicitly defined, discrete and quasi-periodic height field structures.

In the following section a breif overview of previous work relevant and related to this work will be presented.

1.3 Previous work

The whole study of structural colors already started in the the 17th century, Since the first scientific descriptions on structural colors by Hooke in 1665 in his book Micrographia[R.H12]. Hooke investigated feathers of peacocks using one of the first microscopes from this time and found out that the colors on the feather were canceled out whenever a drop of water moistened the feather. He proposed the speculation that a layer of thin plates and air were responsible for reflecting the light and thus he related the structure of the feather to colors. In Newton's book Opticks[I.N14] he described that the colors of the peacock feather are related to the thinness of the transparent part of the feathers. Around 1800 T.Young related Structural colors as a result of wave interference by his two-slit experiment[6], published in the journal Philosophical Transactions of the Royal Society these times.

In the field of computer graphics J.Stam[Sta99] was the first one who was able to develop reflection models based on wave optics capturing the effect of diffraction. His model is a an approximation of far field diffraction[7] effects relying on the Kirchhof integral[8]. For a certain class of surfaces which can be modelled as a height field he provides an analytical solution of the BRDF model he proposes. The main idea of his model is the formulate a BRDF as the Fourier Transform applied on the given height field. However, the height field Stam is dealing with are either extremely regular or can be considered as a probabilistic superposition of bumps forming a periodic like structure relying on probabilistic distribution theory[9]. Both height field assumptions allow him to derive an analytical solution using statistical models. However, the height field we are dealing with are measured, complex, biological nano-structures and thus far apart from being globally very regular, as already demonstrated in figure 1.3. It is also not sufficient superimposing one particular finger (considering it as a bump) since

[5]This image was provided by the LANE lab in Geneva
[6]See `http://en.wikipedia.org/wiki/Double-slit_experiment`
[7]See `http://en.wikipedia.org/wiki/Fraunhofer_diffraction`
[8]See `http://en.wikipedia.org/wiki/Kirchhoff_integral_theorem`
[9]See `http://en.wikipedia.org/wiki/Probability_distribution`

it would not capture the complexity of the measured structure. Therefore, we cannot directly rely on his BRDF model when we want to perform interactive rendering since we do not have an analytical solution and hence are not able to solve the Fourier Transform of the given height field on the fly.

In 2012 Cuypers et all [CT12] proposed a wave based Bidirectional scattering distribution function (BSDF[10]) denoted as WBSDF. Using the rendering equation and Wigner Distribition Functions[11] (WDF) they related their WBSDF model to the incoming wavefront and hence, their model can be adapted such that it can be rendered by a Monte Carlo renderer. The advantage of their model over Stam's is that their models also captures near field diffraction effects. A disadvantage their model is computational expensive since the WDF of a two dimensional surface is a four dimensional function and therefore can hardly be used in order to perform interactive rendering.

The work of Linday an Agu [CT12] an approach in order to perform interactive rendering diffraction effect by precompute and store their BRDF model using spherical harmonics. Nonetheless, for complex natural gratings their BRDF may be insufficient accurate since their approach is using low order spherical harmonics.

1.4 Thesis Structure

The reminder of this thesis is organised as follows: Due to the fact that this thesis has a rather advanced mathematical complexity, chapter 2 introduces some important definitions about modelling light in computer graphics and some wave theory. These concept are required in order to be able to follow our later derivations. This is followed by a brief summary of J. Stam's Paper about diffraction shaders, since his BRDF formulation is the basis of our derivations.

In chapter 3 we adapt Stam's BRDF model step-wise in a way that we will end up with a representation which can be implemented as an interactive diffraction renderer when using natural diffraction gratings. We also propose an alternative formulation, the so called PQ approach in this chapter and discuss its short-comings.

Chapter 4 addresses the practical part of this thesis, the implementation of our diffraction model, explaining all precomputation steps and how rendering is preformed in our reference framework for this thesis.

Chapter 5 gives some further insight about diffraction by explaining the topic about diffraction grating in depth. Furthermore, within this chapter we evaluates the qualitative validity of our BRDF model applied on different surface gratings by computing their reflectance and comparing the result to the grating equation under similar conditions.

[10] See `http://en.wikipedia.org/wiki/Bidirectional_scattering_distribution_function`
[11] See `http://en.wikipedia.org/wiki/Wigner_distribution_function`

Chapter 6 presents our rendered results, first the so called BRDF maps for all our gratings and shading approaches under various shading parameters and then the actual renderings on a snake skin. And finally chapter 7 contains the conclusion of this thesis discussing what has been achieved in this thesis and all the drawbacks of the proposed method. It also contains a note about some of my personal experiences during this thesis.

Chapter 2

Implementation

In computergraphics, we generally synthesize 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 a GPU hardware device, is to compute the colorization and illumination of the objects present 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. The DFT terms of the height field, required for rendering process in our framework is precomputed in Matlab. This is basically addressing all the required precomputations for the provided heigh-fields, referring to computation of the 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 preprocessing 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 and what we have to precompute.

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 times the imaginary number i since our taylor series approximation **??** 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: Pseudo code to generate Fourier terms

INPUT *heightfieldImg, maxH, dH, termCnt*
OUTPUT *DFT terms stored in Files*

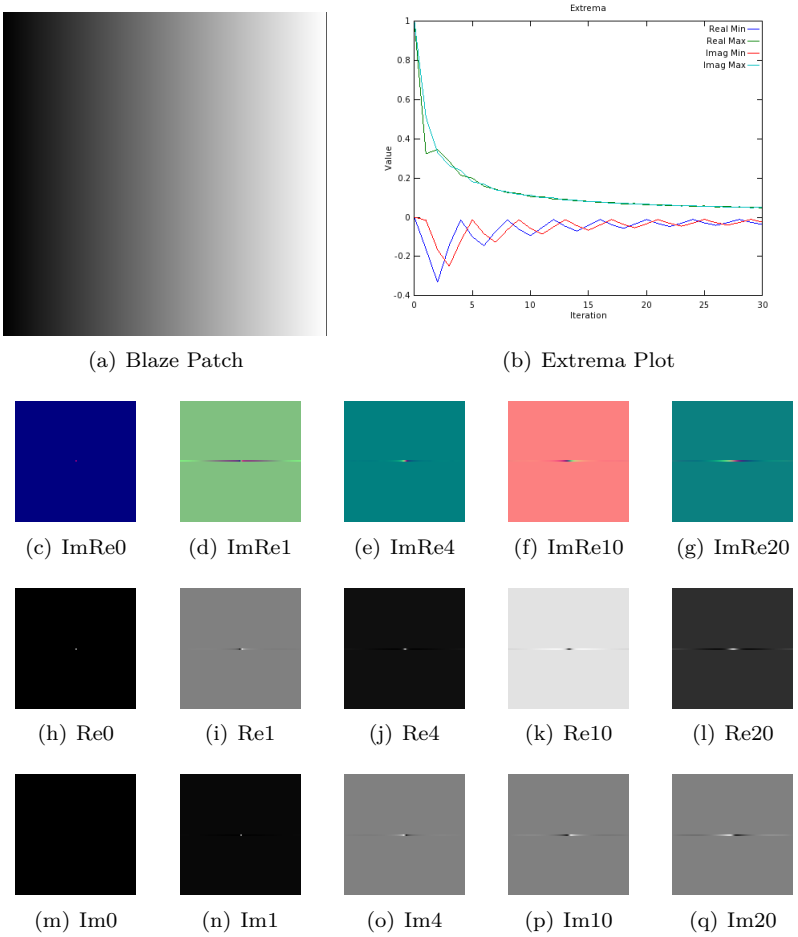
```
% maxH:     A floating-point number specifying
%           the value of maximum height of the
%           height-field in MICRONS, where the
%           minimum-height is zero.
%
% dH:       A floating-point number specifying
%           the resolution (pixel-size) of the
%           'discrete' height-field in MICRONS.
%           It must be 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(heightfieldImg, maxH, dh, termCnt)
dH = dh*1E-6;
% load patch into heightfieldImg
patchImg = heightfieldImg.*maxH;
% rotate patchImg by 90 degrees
for t = 0 : termCnt
  patchFFT = power(1j*patchImg, t);
  fftTerm{t+1} = fftshift(ifft2(patchFFT));

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

  % rotate imOut by -90 degrees
  % 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 simplifies the computation of lookup coordinates during rendering.



(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

Figure 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.

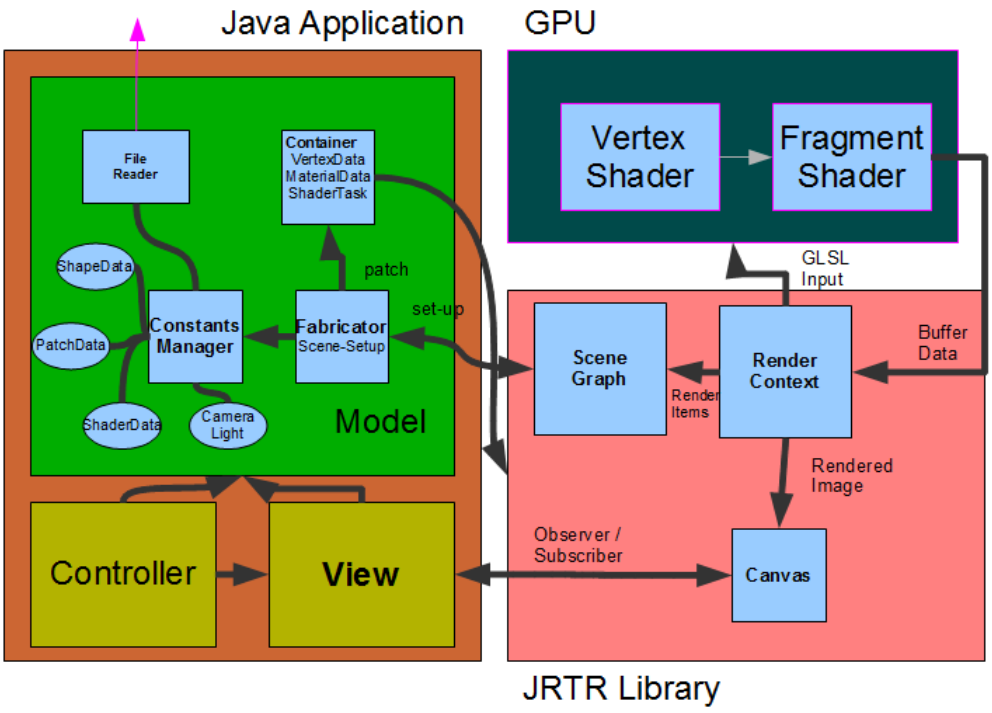


Figure 2.2: Renderer Architecture

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.

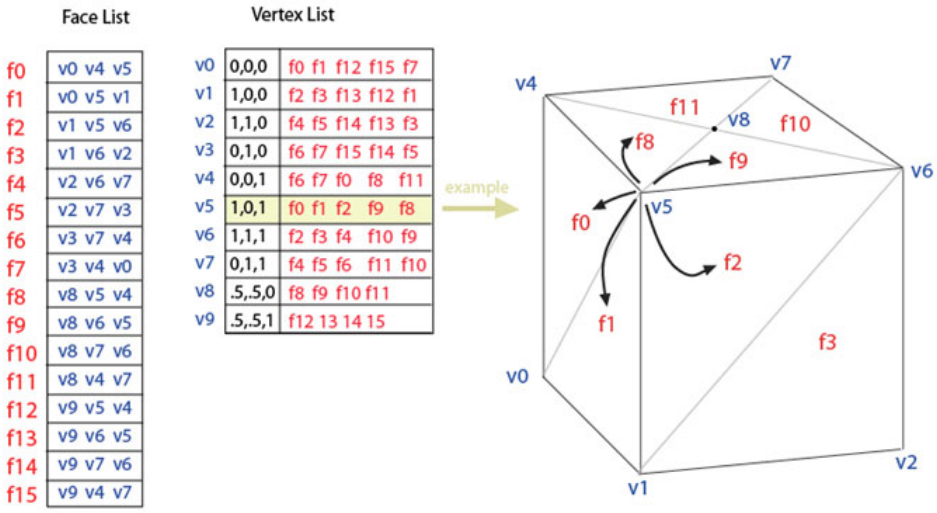


Figure 2.3: A wireframe mesh represents an object as a set of faces and a set of vertices.

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.

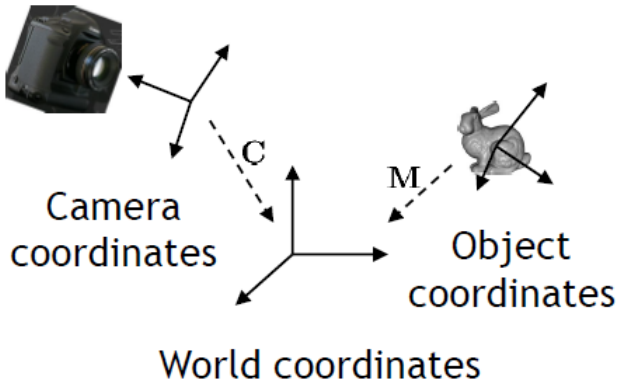


Figure 2.4: Camera coordinate system where its origin defines the center of projection of camera

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

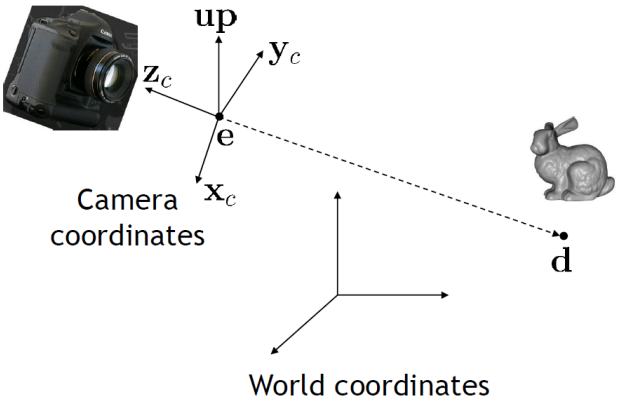


Figure 2.5: Illustration of involved components in order to construct the camera matrix. We introduce some helper vectors $z_c = \frac{e-d}{||e-d||}$, $x_c = \frac{up \times z_c}{||up \times z_c||}$ and $z_c \times x_c$ for the actual construction of the camera matrix

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 **??**. Those direction vectors are transformed onto the tangent space, a local coordinate system spanned by a vertex's normal, tangent and binormal vector. Have a look at the appendix *D.4* for further information and insight about the tangent space. The algorithm 2.2 stated below shows our vertex shader.

Algorithm 2.2 Vertex diffraction shader pseudo code

INPUT: $N, T, Shape, lightDir$
OUTPUT: $StructuralColoronFragment$
Foreach $Vertex\, v \in Shape$ **do**
$vec3\ N = normalize(modelM * vec4(normal, 0.0).xyz)$
$vec3\ T = normalize(modelM * vec4(tangent, 0.0).xyz)$
$vec3\ B = normalize(cross(N, T))$
$vec3\ Pos = ((cop_w - position)).xyz$
$lightDir = normalize(lightDir)$
$l = projectVectorOnTo(lightDir, TangentSpace)$
$p = projectVectorOnTo(Pos, TangentSpace)$
$normalize(l); normalize(p)$
$p_{per} = P \cdot C^{-1} \cdot M \cdot p_{obj}$
end for

As already mentioned in section **??**, our light source is a directional light source (See figure 2.6).

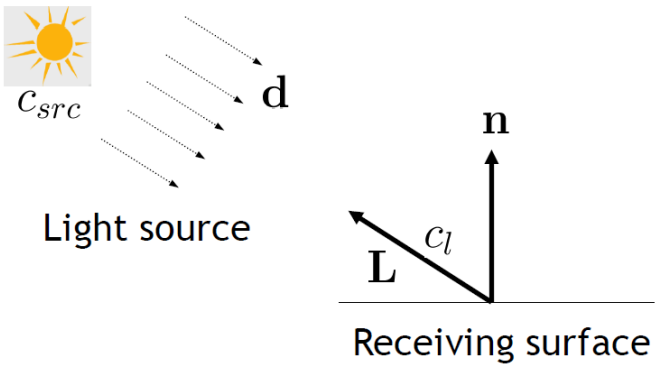


Figure 2.6: For a directional light source all light rays are in parallel.

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 direction of light and the viewing direction. Our shader performs a numerical integration for our final derived expression in equation **??** 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 and this implies that the (u, v) space is sampeled non-linearly.

The Gaussian window approach derived in section **??** is performed for each discrete λ value using a window large enough to span $4\sigma_f$ in both dimensions. For precomputing 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 all the wavelengths in the visible spectrum, i.e. from 380nm to 780nm.

Algorithm 2.3 Fragment diffraction shader pseudo code

INPUT: *Precomputed DFT Terms, Scene Geometry*
OUTPUT: *StructuralColoronFragment*

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** $(anchorX + winW))$ **do**
15: **for** $(j = (anchorY - winW)$ **TO** $(anchorY + winW))$ **do**
16: $dist = distVecFromOriginTo(i, j)$
17: $pos = localLookUp(i, j, n)$
18: $fftVal = rescaledFourierValueAt(pos)$
19: $fftVal *= gaussWeightOf(dist)$
20: $F_{fft} += fftVal$
21: **end for**
22: **end for**
23: $P += taylorScaleF * F_{fft}$
24: **end for**
25: $xyzPixelColor += dot(vec3(|P|^2), xyzWeights)$
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:
This loop performs uniform sampling along wavelength-space. $ColorWeights(\lambda)$ computes the color weight for the current wavelength λ by linear 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:
This loop performs the Taylor series approximation using T terms. 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 inner most loop, the convolution of the gaussian window with the DFT of the patch is performed. $gaussWeightOf(dist)$ computes the weights in equation (**??**) 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 $C(\omega_i, \omega_r)$ **??** is multiplied by the current computed pixel color like formulated in **??**. The gain factor contains the geometric term $refeq : geometricterm$ and the Fresnel term F. W approximate F by the Schlick approximation $D.2$, 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 **??**.

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:

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

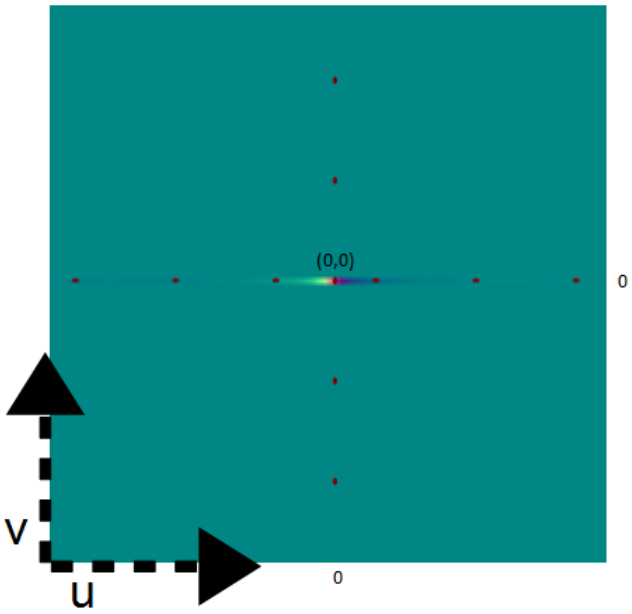


Figure 2.7: (u, v) lookup image

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 **??**.

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 **??**. 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 **??**. Depending on the interpolation method, there are two possible variants we can think of as described in **??**. 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 **??**.

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 **??**. 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.

Appendix A

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.

A.1 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:

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

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

$$\mathcal{F}_{FT}^{-1}\{f\}(w) = \int_{\mathbb{R}} \mathcal{F}\{w\}e^{iwt}dt \tag{A.2}$$

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:

$$\mathcal{F}_{DTFT}\{f\}(w) = \sum_{-\infty}^{\infty} f(x)e^{-iwk} \tag{A.3}$$

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:

$$\mathcal{F}_{DFT}\{f\}(w) = \sum_{n=0}^{N-1} f(x)e^{-iw_n k} \tag{A.4}$$

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 $f(t)$.

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n cos(nt) + \sum_{n=1}^{\infty} b_n cos(nt) \tag{A.5}$$

where

$$\begin{aligned} a_0 &= \int_{-\pi}^{\pi} f(t)dt \\ a_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)cos(nt)dt \\ b_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)sin(nt)dt \end{aligned} \tag{A.6}$$

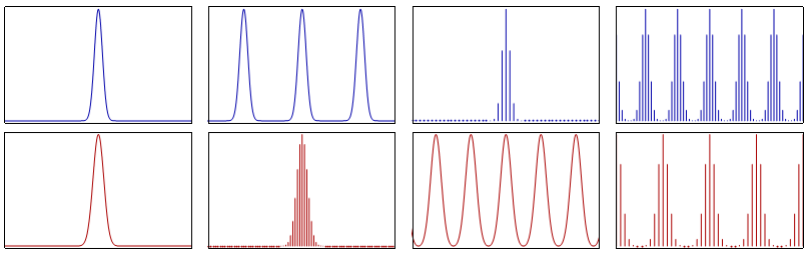


Figure A.1: Relationship[1] between the continuous Fourier transform and the discrete Fourier transform: Left column: A continuous function (top) and its Fourier transform $A.1$ (bottom). Center-left column: Periodic summation of the original function (top). Fourier transform (bottom) is zero except at discrete points. The inverse transform is a sum of sinusoids called Fourier series $A.5$. 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 $A.4$ (bottom) computes discrete samples of the continuous DTFT $A.3$. 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 $A.5$	only discrete in ω
only continuous in t	FT $A.1$	only continuous in ω
only discrete in t	DTFT $A.3$	continuous and periodic in ω
discrete and periodic in t	DFT $A.4$	discrete and periodic in ω

Table A.1: Fourier operator to apply for a given spatial input signal and the properties of its resulting output signal in frequency space

A.2 Convolution

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

$$(f * g)(t) = \int_{\mathbb{R}^n} f(t)g(t-x)dx \tag{A.7}$$

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 $A.1$. Table $A.1$ tells what Fourier transformation operator has to be applied to which kind of input signal and what properties its resulting Fourier transform will have.

A.3 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:

$$\mathcal{T}(f;a)(x) = \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n \tag{A.8}$$

[1] image of illustration has been taken from wikipedia

Appendix B

Summary of Stam's Derivations

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[1], 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:

$$\psi_2(\omega_i, \omega_r) = \frac{ike^{iKR}}{4\pi R}(F(-\omega_i - \omega_r) - (-\omega_i + \omega_r)) \cdot I_1(\omega_i, \omega_r) \tag{B.1}$$

with

$$I_1(\omega_i, \omega_r) = \int_S \hat{\mathbf{n}} e^{ik(-\omega_i - \omega_r)\cdot\mathbf{s}} d\mathbf{s} \tag{B.2}$$

[1] See `http://en.wikipedia.org/wiki/Kirchhoff_integral_theorem` for further information.

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:

$$\sigma^0 = 4\pi \lim_{R\to\infty} R^2 \frac{\langle|\psi_2|^2\rangle}{\langle|\psi_1|^2\rangle} \tag{B.3}$$

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

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

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

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

$$(u, v, w) = -\omega_i - \omega_r \tag{B.5}$$

explicitly and introduces the equation:

$$I(ku, kv) = \int_S \hat{\mathbf{n}} e^{ik(u,v,w)\cdot \mathbf{s}} d\mathbf{s} \tag{B.6}$$

which is a first simplification of *B.2*. Note that the scalar w is the third component of **??** 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 *B.6* he will end up with:

$$\mathbf{I}(ku, kv) = \int\int (-h_x(x, y), -h_y(x, y), 1) e^{ikwh(x,y)} e^{ik(ux+vy)} dxdy \tag{B.7}$$

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

$$p(x, y) = e^{iwkh(x,y)} \tag{B.8}$$

which will allow him to further simplify his equation *B.7* to:

$$\mathbf{I}(ku, kv) = \int\int \frac{1}{ikw}(-p_x, -p_y, ikwp) dxdy \tag{B.9}$$

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 **??**.

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 *B.7*:

$$\mathbf{I}(ku,kv) = \frac{1}{w}P(ku,kv)\cdot(u,v,w) \tag{B.10}$$

Let us consider the term $g = (F(-\omega_i-\omega_r)-(-\omega_i+\omega_r))$, which is a scalar factor of *B.1*. 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 *B.10* into *B.1* he will end up with:

$$\psi_2(\omega_i,\omega_r) = \frac{ike^{iKR}}{4\pi R}\frac{2F(1+\omega_i\cdot\omega_r)}{w}P(ku,kv) \tag{B.11}$$

By using the identity *B.4*, this will lead us to his main finding:

$$BRDF_\lambda(\omega_i,\omega_r) = \frac{k^2F^2G}{4\pi^2Aw^2}\langle|P(ku,kv)|^2\rangle \tag{B.12}$$

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

$$G = \frac{(1+\omega_i\cdot\omega_r)^2}{cos(\theta_i)cos(\theta_r)} \tag{B.13}$$

Appendix C

Summary of Stam's Derivations

C.1 Taylor Series Approximation

For an $N \in \mathbb{N}$ such that

$$\sum_{n=0}^{N} \frac{(ikwh)^n}{n!} \mathcal{F}\{h^n\}(\alpha, \beta) \approx P(\alpha, \beta) \tag{C.1}$$

we have to prove:

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.

C.1.1 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:

$$\forall y : limsup_{k\to\infty} |\frac{a_{k+1}}{a_k}| = limsup_{k\to\infty} \frac{y}{k+1} = 0 \tag{C.2}$$

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

C.1.2 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,

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k \tag{C.3}$$

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:

$$|E_n(x)| \leq \frac{M}{(n+1)!}|x-a|^{n+1} \tag{C.4}$$

with $a = 0$, where $\mathbf{M}$ 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 $\mathbf{a}$ is known, we can reformulate the interval as $I = [0, x_{max}]$, where

$$x_{max} = \|i\| k_{max} w_{max} h_{max} \tag{C.5}$$

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:

$$\begin{aligned} x_{max} &= \|i\| * k_{max} * w_{max} * h_{max} \\ &= k_{max} * w_{max} * h_{max} \\ &= 2 * (\frac{2\pi}{4 * 10^{-7} m}) * 1.5 * 10^{-7} \\ &= 1.5\pi \end{aligned} \tag{C.6}$$

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

$$\begin{aligned} |E_n(x_{max})| &\leq \frac{M}{(n+1)!}|x_{max} - a|^{n+1} \\ &= \frac{exp(1.5\pi) * (1.5\pi)^{n+1}}{(n+1)!} \end{aligned} \tag{C.7}$$

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 $\epsilon \approx 8.8150 * 10^{-8}$.

C.2 PQ approach

C.2.1 One dimensional case

Since our series is bounded, we can simplify the right-hand-side of equation **??**.

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

$$e^{-ix} = cos(x) + isin(x) \tag{C.8}$$

Using the following trigonometric identities

$$\begin{aligned} cos(-x) &= cos(x) \\ sin(-x) &= -sin(x) \end{aligned} \tag{C.9}$$

combined with C.8 we can simplify the series **??** even further to:

$$\frac{1 - e^{iwT(N+1)}}{1 - e^{-iwT}} = \frac{1 - cos(wT(N+1)) + isin(wT(N+1))}{1 - cos(wT) + isin(wT)} \tag{C.10}$$

Equation C.10 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:

$$\begin{aligned} a &:= 1 - cos(wT(N+1)) & b &= sin(wT(N+1)) \\ c &= 1 - cos(wT) & d &= sin(wT) \end{aligned} \tag{C.11}$$

Hence, using C.11, it follows

$$\frac{1 - e^{iwT(N+1)}}{1 - e^{-iwT}} = \frac{(a + ib)}{(c + id)} \tag{C.12}$$

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:

$$\begin{aligned}(cp - dq) &= a \\ (dp + cq) &= b\end{aligned} \tag{C.13}$$

After multiplying the first equation of C.13 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:

$$\begin{aligned}p &= \frac{(ac + bd)}{c^2 + d^2} \\ q &= \frac{(bc + ad)}{c^2 + d^2}\end{aligned} \tag{C.14}$$

Using some trigonometric identities and putting our substitution from C.11 for a, b, c, d back into the current representation C.14 of p and q we will get:

$$\begin{aligned}p &= \frac{1}{2} + \frac{1}{2}\left(\frac{cos(wTN) - cos(wT(N+1))}{1 - cos(wT)}\right) \\ q &= \frac{sin(wT(N+1)) - sin(wTN) - sin(wT)}{2(1 - cos(wT))}\end{aligned} \tag{C.15}$$

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:

$$\begin{aligned}\mathcal{F}^{-1}\{S\}(w) &= \mathcal{F}^{-1}\{f\}(w) \sum_{n=0}^{N} e^{-iwnT} \\ &= (p + iq)\mathcal{F}^{-1}\{f\}(w)\end{aligned} \tag{C.16}$$

C.2.2 Two dimensional case

$$\begin{aligned}
\mathcal{F}^{-1}\{S\}(w_1,w_2) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} h(x_1+n_1T_1,x_2+n_2T_2)e^{iw(x_1+x_2)}dx_1dx_2 \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} h(y_1,y_2)e^{iw((y_1-n_1T_1)+(y_2+n_2T_2))}dx_1dx_2 \\
&= \sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(y_1,y_2)e^{iw(y_1+y_2)}e^{-iw(n_1T_1+n_2T_2)}dy_1dy_2 \\
&= \sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} e^{-iw(n_1T_1+n_2T_2)}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} Box(y_1,y_2)e^{iw(y_1+y_2)}dy_1dy_2 \\
&= \left(\sum_{n_2=0}^{N_1}\sum_{n_2=0}^{N_2} e^{-iw(n_1T_1+n_2T_2)}\right)\mathcal{F}^{-1}\{h\}(w_1,w_2) \\
&= \left(\sum_{n_2=0}^{N_1} e^{-iwn_1T_1}\right)\left(\sum_{n_2=0}^{N_2} e^{-iwn_2T_2}\right)\mathcal{F}^{-1}\{h\}(w_1,w_2) \\
&= (p_1+iq_1)(p_2+iq_2)\mathcal{F}^{-1}\{h\}(w_1,w_2) \\
&= ((p_1p_2-q_1q_2)+i(p_1p_2+q_1q_2))\mathcal{F}^{-1}\{h\}(w_1,w_2) \\
&= (p+iq)\mathcal{F}_{DTFT}\{h\}(w_1,w_2)
\end{aligned} \tag{C.17}$$

Where we have defined

$$\begin{aligned}
p &:= (p_1p_2-q_1q_2) \\
q &:= (p_1p_2+q_1q_2)
\end{aligned} \tag{C.18}$$

Appendix D

Appendix

D.1 The 3rd component w

From the definition **??** of $(u, v, w) = -\omega_i - \omega_r$ and using spherical coordinates *D*.3, we get for w the following identity:

$$\begin{aligned} w &= -\omega_i - \omega_r \\ &= -(\omega_i + \omega_r) \\ &= -\left(cos(\theta_i) + cos(\theta_r)\right) \end{aligned} \tag{D.1}$$

and therefore w^2 is equal $(cos(\theta_i) + cos(\theta_r))^2$.

D.2 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:

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5 \tag{D.2}$$

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.

D.3 Spherical Coordinates

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

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

D.4 Tangent Space

The concept of tangentspace-transformation of tangent space is used in order to convert a point between world and tangent space. GLSL fragment shaders require normals and other vertex primitives declared at each pixel point, which mean that we have one normal vector at each texel and the normal vector axis will vary for every texel.

Think of it as a bumpy suface defined on a flat plane. If those normals were declared in the world space coordinate system, we would have to rotate these normals every time the model is rotated, even when just for a small amount. Since the lights, cameras and other objects are usually defined in world space coordinate system, and therefore, when they are involved in an calculation within the fragment shader, we would to have to rotate them as well for every pixel. This would involve almost countless many object to world matrix transformations meed to take place at the pixel level. Therefore, instead doing so, we transform all vertex primitives into tangent space within the vertex shader.

To make this point clear an example: Even we would rotate the cube in figure *D*.1, the tangent space axis will remain aligned with respect to the face. Which practically speaking, will save us from performing many space transformations applied pixel-wise within the fragment shader and instead allows us to perform us the tangenspace transformation of every involved vertex primitive in the vertex-shader.

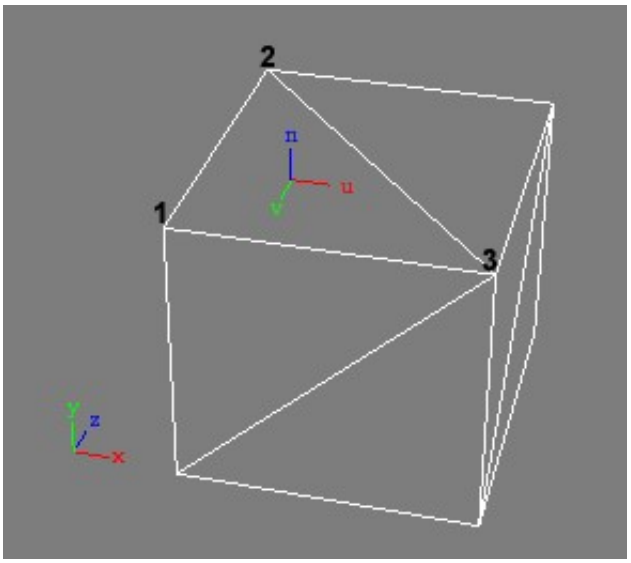


Figure D.1: Cube in world space (x, y, z) showing the tangen space (u, v, n) of its face $(2, 1, 3)$

List of Tables

List of Figures

List of Algorithms

Bibliography

[Bar07] Bartsch, Hans-Jochen: *Taschenbuch Mathematischer Formeln*. 21th edition. HASNER, 2007. – ISBN 978–3–8348–1232–2

[CT12] Cuypers T., et a.: Reflectance Model for Diffraction. In: *ACM Trans. Graph. 31, 5* (2012), September

[DSD14] D. S. Dhillon, et a.: Interactive Diffraction from Biological Nanostructures. In: *EUROGRAPHICS 2014/ M. Paulin and C. Dachsbacher* (2014), January

[For11] Forster, Otto: *Analysis 3*. 6th edition. VIEWEG+TEUBNER, 2011. – ISBN 978–3–8348–1232–2

[I.N14] I.Newton: *Opticks, reprinted*. CreateSpace Independent Publishing Platform, 2014. – ISBN 978–1499151312

[JG04] Juan Guardado, NVIDIA: Simulating Diffraction. In: *GPU Gems* (2004). `https://developer.nvidia.com/content/gpu-gems-chapter-8-simulating-diffraction`

[LM95] Leonard Mandel, Emil W.: *Optical Coherence and Quantum Optics*. Cambridge University Press, 1995. – ISBN 978–0521417112

[MT10] Matin T.R., et a.: Correlating Nanostructures with Function: Structurral Colors on the Wings of a Malaysian Bee. (2010), August

[PAT09] Paul A. Tipler, Gene M.: *Physik für Wissenschaftler und Ingenieure*. 6th edition. Spektrum Verlag, 2009. – ISBN 978–3–8274–1945–3

[PS09] P. Shirley, S. M.: *Fundamentals of Computer Graphics*. 3rd edition. A K Peters, Ltd, 2009. – ISBN 978–1–56881–469–8

[R.H12] R.Hooke: *Micrographia, reprinted*. CreateSpace Independent Publishing Platform, 2012. – ISBN 978–1470079031

[RW11] R. Wright, et a.: *OpenGL SuperBible*. 5th edition. Addison-Wesley, 2011. – ISBN 978–0–32–171261–5

[Sta99] Stam, J.: Diffraction Shaders. In: *SIGGRPAH 99 Conference Proceedings* (1999), August

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