Universität des Saarlandes

Rendering and Streaming of Bidirectional Texture Functions

Masterarbeit im Fach Informatik
Master's Thesis in Visual Computing
von / by

Oleksandr Sotnychenko

angefertigt unter der Leitung von / supervised by

Prof. Dr. Philipp Slusallek

betreut von / advised by

M. Sc. Kristian Sons

...

begutachtet von / reviewers

...

...

Saarbrücken, January 2015

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Sperrvermerk

Blocking Notice

Saarbrücken, January 2015 Oleksandr Sotnychenko

Abstract

Bidirectional Texture Functions (BTF) are 6-dimensional functions that depend on the spatial position on a surface, light and camera directions. Due to changes either of light or camera directions the appearance of real-world materials can severely change. BTF can represent such materials by capturing important material properties for a wide range of illumination changes.

In this work we present a technique to render BTFs at real-time within a web-browser using WebGL. The ever-growing number of mobile devices that use web-browsers imply constraints on the hardware capabilities. Thus, rendering has to be efficient to be possible on such devices and the fact that all data has to be transferred to the client - compression is inevitable. We employ a principal component analysis to compress the data, which allows for rendering of BTFs with interactive frame rates. To provide immediate feedback to the user we provide additionally streaming that allows to progressive enhance the rendering quality while still transferring the remaining data to the client.

Acknowledgements

I would like to express my sincere gratitude ...

To my family.

Contents

List of Figures

Chapter 1

Introduction

One of the main goals in computer graphics is realistic rendering. Even though computer graphics is constantly improving, we are still quite away from reality because material representation in a traditional way lack important realistic properties. A 2-D texture in conjunction with a shading model is a conventional way to represent material appearance in rendering. On the other side, real-world materials surfaces consist of surface meso-structures, i.e. intermediate in size local geometric details. Meso-structures are responsible for fine-scale shadows, self-occlusions, inter-reflection, subsurface scattering and specularities. Also, reflectance and the look of the real-world materials can drastically change when camera and light direction vary.

One of the possible solution to represent such material's attributes is to use sophisticated light functions, for instance a Bidirectional Texture Function (BTF). A BTF is a 6-dimensional function that depends on camera and light directions as well as on spatial texture coordinates. The BTF conceptually extends traditional 2-D texture by the dependence on light and camera directions. This function is usually acquired as a data-set of thousands images that cover discrete light and camera directions. Due to enormous size of such data direct rendering even on the modern hardware without any compression is impractical. Fortunately, there exist many techniques to deal with huge size of BTF, i.e. compression methods that were developed for BTF.

In this thesis, we introduce *WebGL*-based rendering for BTF over the Internet. Till now lots of emphasize were done for rendering 3-D objects in a web-browser using *WebGL*-based technique. Computer graphics in a web-browser is becoming popular for games and visualization. The reason of such popularity is that the user does not need to install any additional applications and WebGl-based rendering is cross-platform technology, which require only compatible browser. Therefore, it is very handy for the user.

Even though 3-D graphics for web-browsers is gaining popularity, BTF was rarely implemented for *WebGL* standard, due to its huge size and overall computational effort to render. This usually includes: decompression of BTF, computation and interpolation for camera and light directions. Such demanding computational effort may be time consuming, especially for low-budget devices such as mobile-devices, old laptops and PCs.

The other problem which arise when rendering BTF over the Internet is the transmission of the data. Before WebGl can start rendering on the client side, the transmission of the data must be finished beforehand. Even to transfer a compressed BTF data can be time consuming for the user. The compressed BTF can be around few megabytes of size, therefore it would it take some time to transfer the data. Consider the worst-case scenario, i.e. an average internet connection of a common user around 5 Mbit/s and a compressed BTF data size around 20 Mb. This will take around 30sec to download. Even though, it sounds as not a big of problem, but any user would like to see the result as quick as possible. One possible solution to this problem is a streaming of the data. With the streaming technology the user may be able to see the first preview of the 3-D object just in a few seconds. We introduce *Web-Sockets* technique, which is supported by most of the contemporary browsers. With *Web-Sockets* the full-duplex communication is available for the server and the client, which is faster than traditional HTTP-based methods. This promises fast and reliable solution for real-time performance in *WebGL*-based application.

In summary main contributions of this paper are:

- PCA compression on subsets of BTF
- improvement of PCA compression error by scaling the resulted parameters
- linear interpolation for camera and light directions using barycentric coordinates
- BTF streaming using *Web-Sockets* technology
- prioritized streaming order of BTF parameters, which let the progressive improvement of the rendering quality
- introduced ambient light term which depends on the light direction, which makes the object look less artificial
- implemented *WebGL*-based demo web-application and *Web-Sockets* server

1.1 Related Work

Lately 3D content started to gain popularity in context of web-based applications. New standards for 3D graphics in context of HTML domain are evolving, for instance such as XML3D [30]. That means that demand for realistic rendering in web-based context will grow. Such 3D web-based applications can be aimed for arbitrary websites. Any commercial website could represent their product as a 3D content for marketing purposes, so the future customers could examine the desired product fully in 3D. Thus, realistic rendering would be highly desirable to make the best impression on the customer.

Until now previous works that used BTF compression for realistic rendering were primarily intended for offline application, i.e. standalone application. However, such approach may be not suitable in context of web-based application, due to the data transfer between the server and the client, which can delay the rendering for certain time. Schwartz *et. al.* [28] presented a work in which compressed BTF data is streamed from the server to the client. The streaming over the Internet is done by means of HTTP streaming, i.e. the web-application requests the data in small chunks. With each new chunk of the BTF data the rendering quality of the 3D object is progressively enhanced. We on the other hand, deploy *Web-Sockets* technology, which can improve application performance compared to HTPP streaming, by reducing the network latency and an option to stream the data in a binary form.

Up-to-date existing BTF compression methods are reviewed by Haindl *et. al.* [12, 10]. Depending on the intended application, a trade-off has to be made between a rendering quality and a compression rate. Our compression method is related to PCA RF (Principal Component Analysis Reflectance Field) method, which was introduced by Sattler *et. al.* [22]. This method allows for real-time performance with realistic rendering quality. Sattler perform PCA on each of n camera directions separately. We on the other hand, perform PCA on k neighbour camera directions at once. In Haindl's review [12] PCA based methods achieve better or at least not worse reconstruction quality results than most others methods. Compression rates of PCA methods are mild compared to other methods, however those methods which produce better compression rates have worse quality than PCA based methods. To overcome the problem of mild compression rates in web-based application, we introduce streaming with *Web-Sockets*, which allows to start rendering just in a few seconds. During the streaming the rendering quality improves.

1.2 Outline

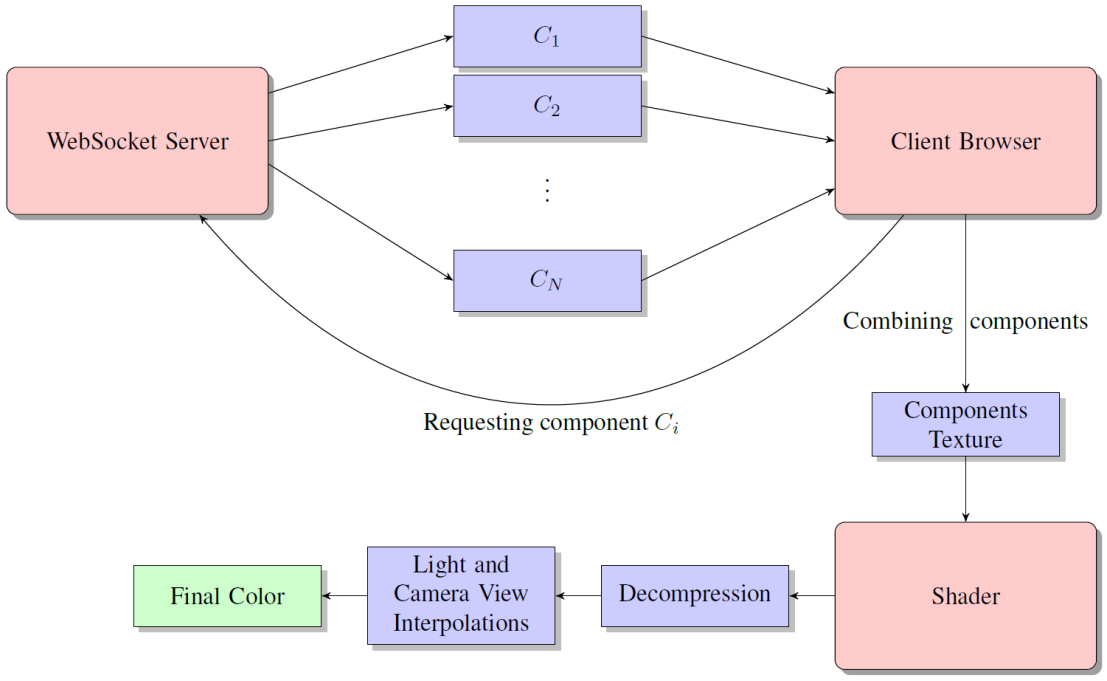

FIGURE 1.1: **Model Overview**

Chapter 2

A Brief Review of Scattering Functions

In this chapter we will review a hierarchy of scattering functions. Scattering functions describe how incoming and outgoing directions of the light are related for a surface at which light-material interaction occurs [8]. Such functions are possible to measure for a given object. After obtaining the data, scattering functions can provide all the necessary information to render the material appearance. Due to diversity of material properties, different functions were introduced. In order to chose suitable scattering function for capturing certain scattering effects for a particular type of material, it is important to be aware of the hierarchy of scattering functions.

2.1 Light-Material Interaction

Before defining any scattering functions of light the basics light-material interaction. Generally speaking, when light hits a material's surface, a sophisticated light-matter process happens. Such process depends on physical properties of the material as well as on physical properties of light[35]. For instance, an opaque surface such as wool will reflect light differently than a smooth surface with high specularities such as metal.

When light makes a contact with a material, three types of interactions may occur: light *reflection*, light *absorption* and light *transmittance*. Light *reflection* is the change in direction of light at an interface between two different media so that light returns into the medium from which it originated. Light *absorption* is a process when a light is being taken up by a material and transformed into internal energy of the material, for instance thermal energy. When a material is transparent, light *transmittance* can occur.

It means, that the light travels through the material and exits on the opposite side of the object. Figure 2.1 demonstrates these 3 types of interactions.

Because light is a form of energy, conservation of energy says that [35]

incident light at a surface = light reflected + light absorbed + light transmitted

2.2 General Scattering Function

To define the general scattering function(GSF) imagine the light-wave hitting the surface at time t_i and position x_i and with wavelength λ_i[22]. With a given local coordinate system at a surface point, the incoming direction of light can be defined as (θ_i, ϕ_i). Light travels inside the material and exits the surface at position x_o and time t_o, with possibly changed wavelength λ_o in the outgoing direction (θ_o, ϕ_o). Figure 2.1 illustrates the process.

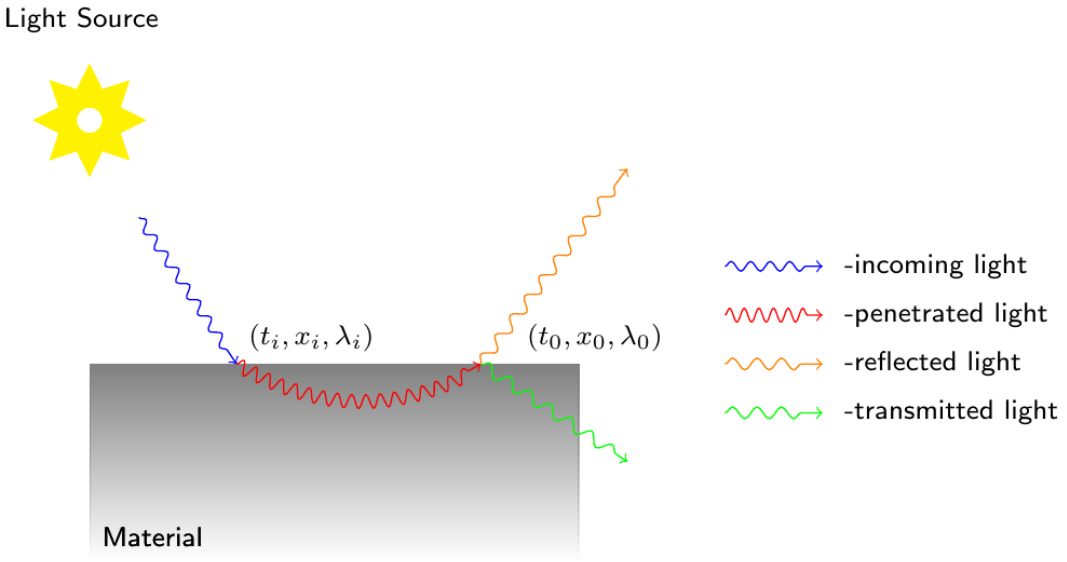

FIGURE 2.1: **Example of Light-Material interaction.**

According to the description we get a GSF

$$GSF(t_i, t_o, x_i, x_{0,}\theta_i, \phi_i, \theta_o, \phi_o, \lambda_i, \lambda_o)$$

in which spatial positions $x_{i,o}$ are 2-D variables. This function describes light interaction for each surface point for any incoming light and outgoing direction at certain time. Such function compromise 12 parameters. Also, note that we neglected light transmittance, which would even further complicate the function.

2.3 Bidirectional Scattering-Surface Reflectance Distribution Function

Since the measurement, modeling and rendering of a 12-D GSF function is currently not practical, additional assumptions have to be made to simplify the function. Usually such assumption are made [22]:

- light interaction takes zero time ($t_i = t_o$)
- wavelength is separated into the three color bands red, green and blue ($\lambda_{r,g,b}$)
- interaction does not change wavelength ($\lambda_i = \lambda_0$)

After mentioned assumptions we get a 8-D bidirectional scattering-surface reflectance distribution function (BSSRDF)

$$BSSRDF(x_i, x_o\theta_i, \phi_i\theta_o, \phi_o)$$

BSSRDF describes various light interactions for heterogeneous both translucent and opaque materials. That is why BSSRDF can be used for rendering materials such as skin, marble, milk and other objects which do not look realistic without subsurface scattering. Subsurface scattering is a process when light penetrates an object at an incident point, travels inside the object and exists at a different point of the object.

2.4 Bidirectional Texture Function

If we simplify further and assume that

- light entering a material exits at the same point $x_i = x_o$, while internal subsurface scattering is still present

we will get a 6-D bidirectional texture function (BTF).

Subsurface scattering, self-occlusion, self-shadowing are still present, now it just comes pre-integrated, i.e. it can be defined through BSSRDF [22]:

$$BTF(x, \theta_i, \phi_i, \theta_o, \phi_o) = \int_S BSSRDF(x_i, x, \theta_i, \phi_i, \theta_o, \phi_o)dx_i$$

The assumption that $x_i = x_o$ simplifies measuring, modeling and rendering of the scattering function. As we can see the BTF integrates subsurface scattering from neighbouring surface locations.

2.5 Bidirectional Subsurface Scattering Distribution Function

Another possible reduction of the 8-D BSSRDF is to assume that we deal with a homogeneous surface [8], i.e.

- subsurface scattering depends only on relative surface positions of incoming and outgoing light $(x_i - x_o)$

Simply saying it means that scattering do not vary over a surface. With such assumption we get a 6D function that known as a bidirectional subsurface scattering distribution function (BSSDF).

$$BSSDF(x_i - x_o, \theta_i, \phi_i, \theta_o, \phi_o)$$

BSSDF represents homogeneous materials for which subsurface scattering is a significant feature of their overall appearance. For instance, BSSDF accounts for objects such as water, milk, human skin, and marble.

2.6 Bidirectional Reflectance Distribution Function

If we assume the following for a BSSDF that

- there is no spatial variation
- no self-shadowing
- no self-occlusion
- no inter-reflections
- no subsurface scattering
- energy conservation
- reciprocity $BRDF(\theta_i, \phi_i, \theta_o, \phi_o)$=$BRDF(\theta_o, \phi_o, \theta_i, \phi_i)$.

we get a 4-D bidirectional reflectance distribution function (BRDF)

$$BRDF(\theta_i, \phi_i, \theta_o, \phi_o).$$

Nicodemus et al. [25] was the one who proposed the BRDF. Two principal properties of the BRDF were introduced, i.e. *energy conservation* and *reciprocity*. *Energy conservation* law states that the total amount of outgoing light from a surface cannot exceed the original amount of light that arrives at the surface [35]. *Reciprocity* says that if we swap incoming and outgoing directions BRDF stays the same. If either of these conditions are not satisfied then such BRDF is called *apparent* BRDF (ABRDF) [31].

It is quite difficult to create a mathematical model for a BRDF that satisfies reciprocity, energy conservation and the same time produces realistic images. However, most BRDF's models do not satisfy these conditions and still get plausible rendering results. For instance, Phong model is the most well-known shading model in computer graphics. The traditional Phong model satisfy neither energy conservation nor reciprocity, but can still render many materials realistically plausible. Usually, such materials are of opaque and flat nature, for example plastic materials. The Phong model is an empirical model and is designed to fit the original function, often based on simple formulas which were derived from observations.

2.7 Spatially Varying Bidirectional Reflectance Distribution Function

If spatial dependence for BRDF takes place, we get a 6-D spatially varying BRDF (SVBRDF)

$$SVBRDF(x, \theta_i, \phi_i, \theta_o, \phi_o).$$

Assumptions are the same as for the BRDF, except now spatial dependence is present.

A SVBRDF is closely related to a BTF. The SVBRDF and the BTF almost the same scattering function, the difference is in scattering process. Changes in scattering at local position x for the BTF are influenced from neighbouring 3D surface geometry, as a result the self-shadowing, masking and inter-reflections are captured by the BTF. On the other side, the spatial dependence of a SVBRDF describes variations in the optical properties of a surface [10].

A SVBRDF represents structures at micro-scale level, which corresponds to near flat opaque materials. On the other side, a BTF capture structure both at macro and micro scales. That means that the BTF takes into account influences from local neighbourhood structures. Even though measurement, compression, rendering are more efficient for the SVBRDF, the BTF can produce better visual results. [10]

2.8 Surface Light Field

Consider BTF and assume

- fixed light direction $\theta_i = const$, $\phi_i = const$

we will get a 4-D Surface Light Field model (SLF)

$$SLF(x, \theta_o, \phi_o).$$

SLF model is a simple a textural model and is a subset of BTF. SLF is used when illumination direction is not varying in the rendering scene, but the view direction varies. Thus, such model is favoured for its greater computational efficiency for such cases.

2.9 Surface Reflectance Field

Analogously as for SLF, consider BTF and assume

- fixed camera direction $\theta_o = const$, $\phi_o = const$

we will get a 4-D Surface Reflectance Field model (SRF)

$$SRF(x, \theta_i, \phi_i).$$

SRF is another very popular variant of image-based rendering and is also a subset of BTF. In this case many images are taken under varying light directions with fixed view point[22]. For instance, Debevec et al. [7] recorded the appearance of human face while a light source was rotating around the face. Rendering the human face realistically is always a struggle in computer graphics. Due to complex reflectance characteristics of the human face, common texture mapping usually fails under varying illumination. Debevec et al. acquired approximately two thousand images under different light positions and could render the face under arbitrary lighting for original camera directions.

2.10 Summary of Scattering Functions

In practice, an advantage of simpler scattering function is a computation efficiency, while a disadvantage is a reduction of visual quality. But, development of the graphics hardware is always improving and as a result this encourages to use sophisticated scattering functions, which provide improvement in realistic material rendering.

However, a complex material representation requires sophisticated data measurement and modeling. For instance, till now a GRF has not been measured and still stays as a state-of-the-art problem [10]. In practice, the appropriate scattering function depends on the specific application. For instance, a scene with various textures can be rendered with different scattering functions. Simpler materials that do not have complex scattering features can be rendered with a 2-D textures in combination with BRDFs model, such as Phong model. If the material cannot be represented realistic without subsurface scattering, masking, self-reflections then typically such materials require high quality representations, e.g. BTF. However, these advanced material representations are very complex. In practice a tradeoff between visual quality and rendering cost is inevitable.

Chapter 3

State of Art a BTF

3.1 BTF Acquisition

BTF acquisition is not an easy task as it requires time for acquiring and post-processing the data, and also resources are needed to create acquisition setup. There are only a few measurement systems [22, 29, 6, 14, 18, 23] exist, but as the interest in the realistic material rendering using BTF is growing, measurement systems are developing. In this chapter we will review how in general BTF data acquisition is made, which post-processing steps are made, and pros and cons of existing measurement systems. Also, we will take a look at publicly available BTF datasets.

3.1.1 General Acquisition Methods

All the mentioned BTF acquisition systems share the same idea in the data acquisition, i.e. capturing the appearance of a flat square slice of the material surface under varying light and camera directions. The material surface is usually sampled over a hemisphere above the material slice as shown in Figure 3.1. Depending on the material reflectance properties sampling distribution may vary, e.g. sampling distribution can be dense in regions where specular peaks in light reflection occur. Then, if needed uniform distribution can be made in a post-processing step with a help of interpolation [10].

Digital cameras are used as capturing devices of the material appearance. Depending on the setup the number of cameras can vary. If there is only one camera [22, 23, 6], it usually moves around over the hemisphere above the sample with a help of robotic arm or the rail-trail system [22]. The advantage of such approach is that it is less expensive and can suit for low-budget applications. But, the disadvantage is the positioning errors that

can arise, which influence the overall measurement error. Depending on the application light sources can be fixed or moveable.

There are approaches which do not involve camera and light source movement at all. Schwartz et al. [29] developed a novel measurement system which uses a dense array of 151 camera, which are uniformly placed on hemispherical structure. Flashes of the cameras are used as light sources. Such setup provide high angular and spatial resolutions.

Also, Ngan et al. [23] made a setup which does not involve camera movement by placing the planar slices of the material in a form of the pyramid. Thus, such setup captures the material appearance for 13 different camera views at once. Light directions are sampled by hand-moved electronic flash. The disadvantage of such setup is that it provides sparse angular resolutions, but depending on the application such approach can be plausible.

The material sample is commonly flat and squared slice, which is placed the holder plate. To conduct automatic post-processing borderline markers are placed on the holder. Those markers provide the important information for further post-processing steps such as image rectification and registration.

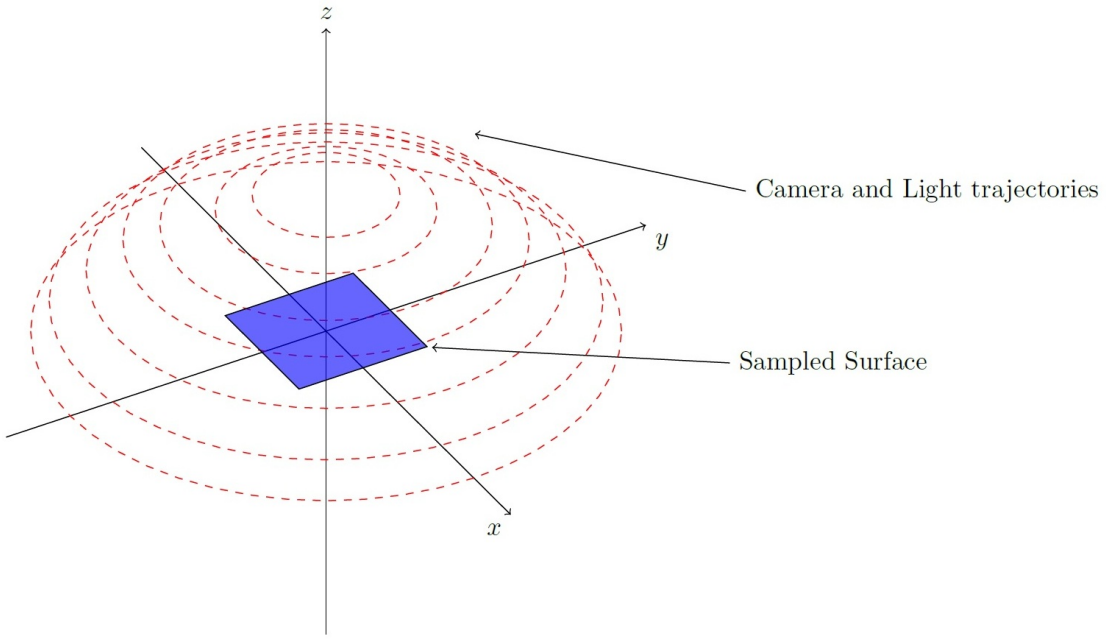

FIGURE 3.1: **Example of BTF measurement.**
Camera and light positions share the same trajectories. Red dashed circles are the sample positions on the hemisphere.

3.1.2 Post-processing

After the measurement is done, the raw data has to be further post-processed, because typically such such data is not ready for further modeling/compression or rendering. Raw data is a set of images that are not aligned with each other and images are not mutually registered.

When the raw images are obtained under different camera angles (θ, ϕ) they are perspectively distorted [26]. Thus, sample image have to be aligned with each other and spatially registered to be further exploited. Firstly, borderline markers that were placed around the material sample on the holder plate aid the automatic detection of the material slice. Then, after the material slices are detected and cropped, they are ready to for mutual alignment. This process is called *rectification*. *Rectification* is a process which involves projecting all sample images onto the plane which is defined by the frontal view, e.g. $(\theta_o = 0, \phi_o = 0)$. In other words, all normals of sample images have to be aligned with their corresponding camera directions, i.e. as if all sample images were taken from frontal view $(\theta = 0, \phi = 0)$. The last step is image *registration*, a process of getting pixel-to-pixel correspondence between the images. As, all transformation were done, it is only enough to rescale all images to some equal resolution.

Even after the proper rectification and registering of the measured data, registration errors can be still present between individual camera directions [10]. This happens due to structural occlusions of the material surface. Because, of such self-occlusions some geometry structures are not captured by certain camera directions, but can be captured with other camera directions. That is why even after recti?cation images captured from completely different directions are not correctly mutually align. Also, registration errors can be caused both by inaccurate camera and material sample positions happened during the measurement processes. One way to avoid artifacts in rendering caused by registration errors is to employ a compression step separately for each fixed camera positions, i.e. subsets of BTF data. For instance, Sattler *et. al.* [26] has done this approach.

If needed, further processing steps can be done, for instance *linear edger blending* to reduce tiling artifacts [26]. Also, typical image processing steps may be employed, e.g. noise reduction filters.

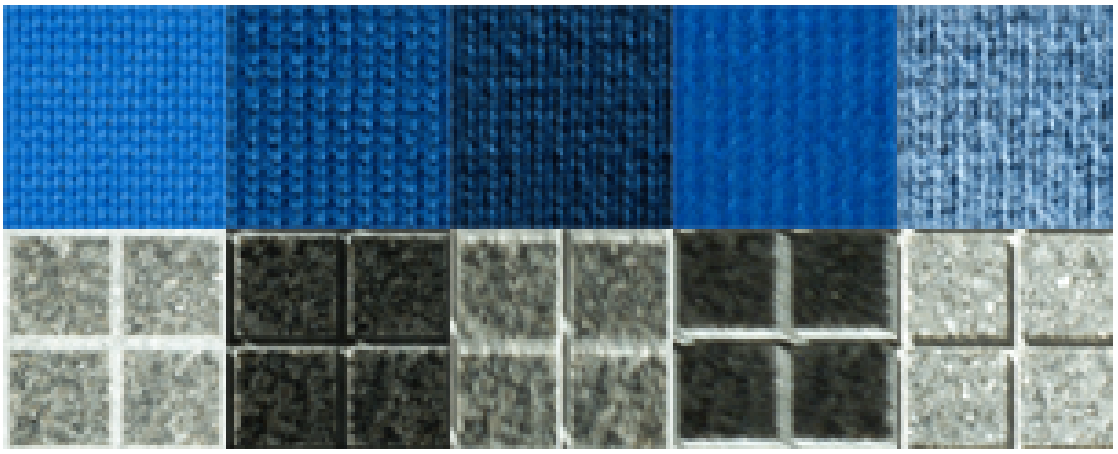

FIGURE 3.2: **BTF example of Bonn Database [1].**
Example how BTF catches rich appearance of the material due to dependencies on light and camera directions. Upper row is a knitted wool, lower is a graved granite stone.

3.1.3 Publicly Available BTF Datasets

The accurate rendering of the material surface is highly depended on the quality of acquired data, especially for BTF. There are several properties that are vital for reproducing quality rendering results. The BTF datasets can be distinguished by how well image post-processing were done and how good spatial and angular resolutions are. So, depending on the application trade-off between high and low spatial or angular resolutions is done. For example, some materials with a low range of reflectance may benefit from sparse resolutions, e.g. wool, plastic, etc.

A pioneer in the BTF acquisition was Dana *et. al.* [2], who measured 61 materials with fixed light and moving camera aided by a robotic arm. Such procedure resulted in a set of images, which can be regarded as a subset of BTF, which is called surface light field (SLF), see Chapter 2.8. Data *et. al.* *CUReT* database is publicly available [2]. For each measured surface Dana *et. al.* used 205 different combinations of camera and light directions, which resulted in relatively sparse angular resolution. Dana's *et. al.* BTF database are not rectified, but the authors provide image coordinates to allow their further rectification. Because, of this limitations such BTF dataset usually used for computer vision purposes, i.e. texture classification [10].

Based on Dana *et. al.* BTF measurement system, Sattler from Bonn University made his own measuring system [26]. The main difference in that system is that a camera moves on a semi-circle rail around material sample. Such setup provides spatially rectificated and mutually registered data, with reasonable angular and spatial resolutions. Datasets of Bonn University [1] are publicly available and were used in this thesis.

Consider Figure 3.2, which illustrates one of the sampled materials of Bonn database. The measured surface is being fixed all the time on the sampler holder as shown in Figure 3.1. For each light position, a camera takes a shot of the material while moving from point to point of the hemisphere. Bonn database has the same trajectory for camera and light positions, i.e. 81 positions on the hemisphere, which resulted in $81 \times 81 = 6561$ total number of acquired images. After that the sample images were rectificated and registered, resulting in a set of images with spatial resolution 256×256. Typically, the size of one uncompressed BTF is around 1.2 Gb.

3.2 BTF Data Representations

Before doing any further operation on acquired BTF it is important to chose a suitable data representation for BTF data. Suitable presentation can enormously influence the final quality of BTF rendering and compression ratio.

3.2.1 Texture Representation

The first one representation is a straightforward representation, i.e as a set of original rectified and registered textures. Mathematically this representation expressed as

$$BTF_{Texture} = \left\{ I_{(\theta_i,\phi_i,\theta_o,\phi_o)} \mid (\theta_i, \phi_i, \theta_o, \phi_o) \in M \right\}$$

where M denotes a set of images $I_{(\theta_i,\phi_i,\theta_o,\phi_o)}$ measured for different light and camera directions (i, o) accordingly.

Basically, $BTF_{Texture}$ is used for compression methods that do analysis on the whole sample plane.

3.2.2 ABRDF Representation

ABRDF representation is a set of ABRDFs, one ABRDF for each sample position of the material plane. ABRDF was defined in Chapter 2.6. In this case it is called *apparent*, because BTF includes effects such self-occlusions, sub-surface scattering and other complex effects which violate two basic properties of BRDF.

$$BTF_{ABRDF} = \left\{ P_{(x)} \quad \mid (x) \in I_{(\theta_i,\phi_i,\theta_o,\phi_o)} \subset \mathbb{N}^2 \right\}$$

BTF_{ABRDF} denotes a set of $P_{(x)}$ images, i.e. ABRDF for spatial position x.

BTF_{ABRDF} representation allows better pixel-to-pixel comparisons, which can give a big advantage for methods that employ pixel-wise compression, e.g. BRDF based models.

Also, such arrangement provides images with lower variance [12], which can allow better compression results in certain scenarios, e.g. when the material surface is not smooth. Anyway, both representations posses the same information, and any compression method can use either of them.

3.3 BTF Compression Methods

BTF data consists of thousands of images, which means single BTF requires lots of storage space. Bonn Database [1] consists of 8-bit PNG images with a resolution of 256×256 sampled for 81×81 different camera and light directions. The uncompressed data of one BTF occupies approximately 1,2 GB of space. To render the scene with several BTFs and to achieve acceptable frame-rates becomes practically impossible task, especially for low-end hardware. Also, as we intent to render BTF in a web-browser, BTF data would have to be transferred from a server to a client, which means the compact representation of BTF is inevitable in such case. For our scenario it is important to chose the right compression method, i.e. the one that can allows real-time decompression, high compression rate while preserving good quality, and separability of compressed BTF data. Separability is needed for real-time streaming in a web-browser. As the data is streamed in small chunks and at some point it is important to have an option to render the preview of BTF based on the partial data.

There are different types of methods applied for compressing the BTF. Those methods can be categorized the following way:

- *Analytic methods* group, where BTF is represented by analytic BRDF models. Analytic BRDF models are the functions which fit separately each texel of BTF. Such functions store few parameters, thus high real-time performance is easily achieved. However, these group of methods can suffer from decreased quality [12]. Also, it is hard to change parameters in order to control the visual quality. Chapter 3.3.1.

- *Statistical methods*, which belong to methods that reduce dimensionality based on statistics, for instance based linear basis decompression method such as PCA (Principal component analysis). PCA based methods are frequently used, because its parameters directly correspond to the trade-off between compression ratio and reconstruction quality. Also, PCA is frequently a basis point for some more sophisticated methods [28]. Chapter 3.3.2.

- *Probabilistic models*, which can spatially enlarge BTF to any arbitrary size without visible discontinuities and extremely compress the original BTF. However, the resulted quality usually suits for flat surfaces and there are problems of implementing in GPU for real-time rendering. Chapter 3.3.3.

3.3.1 Analytic methods

This group of methods take advantage of ABRDF representation of BTF. There is a large number of techniques which allow compactly represent BRDF, which in essence can be also applied for ABRDF. Each texel of BTF, i.e. spatial position of surface are represented as ABRDF. Each of these ABRDFs can be modeled and compressed by any BRDF model.

One of the possible ways to model ABRDF is to use *Polynomial Texture Mapping* (PTM) approach. Malzbender *et. al.* [20] used this approach which allows for high compression rates and generally good quality. However, PTM requires to compute specular and diffuse effects separately. PTM model assumes that the input surfaces has to be either diffuse or their specular contribution is separated beforehand. For BTF it can be quite difficult to separate specular highlights [12].

Haindl *et. al.* [12] applied PTM for a fixed camera positions of BTF, i.e. for reflectance fields (RF). General formula looks the following way (PTM RF):

$$R_o(r,i) \approx a_0(r)u_x^2 + a_1(r)u_y^2 + a_2(r)u_x u_y + a_3(r)u_x + a_4(r)u_y + a_5(r)$$

where R_o is approximated RF for fixed camera direction o and u_x, u_y are projections of the normalized light vector into the local coordinate system $r = (x, y)$. Set of all possible R_o is the number of all camera positions, i.e. n_o. Coefficients a_p are fitted by the use of *singular value decomposition* SVD for each R_o and parameters stored as a spatial map referred to as a PTM.

However, Malzbender *et. al.* [20] claims that this method produce considerable errors for high grazing angles. But, nevertheless this method enables fast rendering and generally suited for smooth material surfaces.

Another model which produces slightly better visual quality is the polynomial extension of one-lobe Lafortune model (PLM) [12]. One-lobe Lafortune model (LM) looks the following way [9]:

$$Y_o(r,i) = \rho_{o,r}(C_{o,r,x}u_x + C_{o,r,y}u_y + C_{o,r,z}u_z)^{n_{o,r}}$$

where $w_i(\theta_i, \phi_i) = [u_1, u_2, u_3]^T$ is a unit vector pointing to light position. Parameters ρ, C_x, C_y, C_z, n can be computed with a Levenberg-Marquardt non-linear optimisation

algorithm [9]. During testing of this model Filip and Haindl [9] claim that LM produce unsatisfactory results for complex ABRDFs. Thus, the polynomial extension of one-lobe Lafortune was introduced (PLM RF):

$$R_o(r,i) \approx \sum_{j=0}^{n} a_{r,o,i,j} Y_o(r,i)^j$$

PLM RF solves the problem of bad quality for grazing angles and improves the rendering quality compared to PTM RF. However, statistical based methods produce even better quality compared to above methods but with lower compression rates [12].

3.3.2 Statistical methods

Statistical methods group belong to the context of pattern recognition and aimed to make benefit from the statistical properties of a BTF [27]. This group of methods mostly based on a dimensionality reduction of a data, while preserving the most important information. The goal of the dimensionality reduction is to find a new basis which would represent the data with less dimensions and at the same time preserving the important features of the original data. The size of the data with the new basis will be decreased.

One of the popular methods used for a BTF compression which belong to this group of methods is *Principal component analysis* (PCA) [5, 27, 12, 28, 26, 21, 24]. PCA is a linear transformation of the data to a new basis, where each new coordinate (variable) is called *principal component* [5]. Each new coordinate are mutually orthogonal, i.e. uncorrelated. Components are sorted by decreasing variance, i.e. first component posses the greatest variance compared to other components, second one holds the second greatest variance and so on. The last components which hold small variations are discarded, because they do not hold important information. The ones that are left for a new basis are called *principal components*. In more detailed form PCA method is explained in Chapter 4, which was used in our work.

The reason why PCA is a popular approach for compressing BTF are because of the following reasons. Firstly, PCA allows for a strong correspondence between the number of components and the decompression error. The correspondence lies in the amount of components used. The bigger quantity of components are used to gain better decompression error, and vise-verse the smaller number of components gives better compression ratio, but worse decompression error. This makes easier to regulate the estimated parameters, i.e. to do a trade-off between a rendering quality and a compression ratio. Secondly, reasonable compression ratio along with low decompression error can be achieved, e.g. compression ratio 1 : 100 with an average decompression error 2%- 4% [27, 12]. Thirdly, PCA is suitable for real-time decompression for average GPU

[27, 12]. It is suitable, because decompression of one single pixel is not depended on the other pixels, which suits the nature of GPU. And computation of a single pixel for PCA decompression on the GPU is done by means of linear combination of estimated PCA parameters, so the speed of decompression depends on number of components used.

There exist different methods based on PCA, i.e. PCA *Reflactance Field* (RF), PCA BTF and local PCA (LPCA) [26, 27, 21, 12]. Sattler *et. al.* [26] developed PCA RF. This method ensures a pixel position coherence, by employing PCA per camera direction. The advantage is that on average 8 components are enough to get good quality results with decompression error 2%- 4%. On the other hand, PCA BTF method, which does PCA on all camera directions at once, requires on average 41 components [12]. Müller *et. al.* [27] improved PCA BTF by exploiting the vector quantization technique and applying PCA for resulted clusters. The name of this method is local PCA (LPCA). On average this method requires 19 components [12]. Even though compression ratio of PCA RF is lower than PCA BTF and LPCA, it is less computational demanding.

Last but not least, any PCA method is suitable for streaming purpose, as it is possible to stream components separately and progressively enhance the rendering quality.

3.3.3 Probabilistic models

Compression methods based on probabilistic models [11, 16, 12] provide much higher compression rate than most others methods and allow for a seamless enlargement of textures of any size [12]. Usual compression compression ratio achieved by such models are $1:6-8\times10^4$. But, the visual quality of rendered results suits best for smooth material surfaces, while materials with high frequencies can appear flat using probabilistic methods [12]. Also, these type of models can be problematic for implementation on low-end GPUs. The problem arise in shaders, when probabilistic model requires information of spatial neighbours to synthesize the texture [16, 12]. Even though solutions exists how to handle such problem, e.g. take advantage of Framebuffers Objects, which allow access for previously rendered results [12]. Such solution may be time-consuming for low-end GPUs.

A common algorithm that employs probabilistic model combines a material range map and synthetic smooth texture produced by a probabilistic model. Range map is a monochrome image that stores the depth of each spatial position of the material surface [16]. If the synthetic texture is larger than range map, it can be enlarged using the Roller technique[13]. Currently there are available several probabilistic models, i.e. Gaussian Markov Random Field (GMRF) and Causal Autoregressive (CAR) models [12, 5]. Commonly these models are 3D models which further factorized and modelled by

less dimensional models, i.e. by a set of 2D models. 2D models requires less parameters to store. Such models are very complex, because they can store up to thousand 2D models [10]. The learning process of such models is not easy as well, because it requires large training data set. Thus, probabilistic models can be applied to some simpler materials, e.g. smooth materials. Materials with high frequencies can result in a flat look, as shown by Haindl *et. al.* [12]. Also, the set of 2D models has to be trained simultaneously, thus it demands high computational effort.

Chapter 4

PCA

In this chapter we explain the derivation of PCA and the algorithm for compressing and decompressing the BTF. We have chosen PCA for BTF compression, which belong to statistical methods, see Chapter 3.3.2. While analytical methods and probabilistic methods have better compression ratio, they can produce worse rendering quality than PCA [12].

4.1 Derivation of PCA

Derivation of PCA can be done by means of a maximum variance formulation, as defined by Bishop [5]. Consider a set of variables $\{x_n\}$, where $n = 1..N$. x_n are D-dimensional vectors. The aim is to project this data to an orthogonal basis while maximizing variation of new variables. For the sake of simplicity, consider projection to one-dimensional space, i.e. to a new basis which consist of one vector u_1. As the magnitude of vector is not important in this case, let it be a unit vector, i.e. $u_1^T u_1 = 1$. Then, each variable x_n is projected onto a new basis, i.e. a scalar $u_1^T x_n$.

To compute the variance of the projected data, first we need to define the mean of projected data:

$$\frac{1}{N}\sum_{n=1}^{N} u_1^T x_n = u_1^T(\frac{1}{N}\sum_{n=1}^{N} x_n) = u_1^T \overline{x}.$$

Then, the variance of the projected data looks the following way:

$$\sum_{n=1}^{N}(u_1^T x_n - u_1^T \overline{x})^2 = u_1^T S u_1$$

where S is the covariance matrix defined as:

$$S = \sum_{n=1}^{N}(x_n - \overline{x})(x_n - \overline{x})^T.$$

The next step is to maximize the variance $u_1^T S u_1$ with respect to u_1. In order to avoid $\|u_1\|$ growing to infinity, additional constrain has to be used, i.e. normalization constrain $u_1^T u_1 = 1$.

Then, the problem looks the following way:

$$\begin{aligned} &maximize \quad u_1^T S u_1 \\ &subject\ to \quad u_1^T u_1 = 1 \end{aligned}$$

This can be solved with Lagrangian multiplier:

$$u_1^T S u_1 + \lambda_1 (1 - u_1^T u_1).$$

By taking the derivative with respect to u_1 and setting it to zero, the local extremum can be found:

$$\frac{\partial}{\partial u_1} u_1^T S u_1 - \lambda_1 \frac{\partial}{\partial u_1} u_1^T u_1 = 0.$$

The result of taking derivative will be the following equation [5]:

$$S u_1 = \lambda_1 u_1$$

which implies that this is an eigenvector and an eigenvalue problem, where u_1 is the eigenvector and λ_1 is the eigenvalue. So, the maximum will be when eigenvector u_1 will have the largest eigenvalue λ_1. Then, such vector u_1 will be a *principal component*.

In the same way it is possible to define the rest of principal components, which maximize the variance of projected data and with a condition that new components are orthogonal to other components. After all, PCA involves evaluating the mean $\overline{x}$ and covariance matrix S and finding eigenvectors with corresponding eigenvalues.

4.2 SVD as PCA

Consider a set of variables x_n be columns of matrix X. To compute covariance matrix S, matrix X has to be centred, i.e.

$$X_c = X - 1\overline{x}$$

where $\overline{x}$ is the vector of column-averages of matrix X and matrix 1 is the matrix of ones. Then, the covariance matrix can be calculated the following way:

$$S = X_c X_c^T$$

In practice to find eigenvectors and eigenvalues of covariance matrix S can be done by means of *singular value decomposition*(SVD). SVD does not require S to be computed, instead it is enough to perform SVD on matrix X_c. For any real matrix X_c there is exist a decomposition [34]:

$$X_c = U\Sigma V^T$$

where U and V orthogonal matrices and S diagonal matrix consisting of singular values. The diagonal values of Σ are the square roots of the eigenvalues of $X_c X_c^T$ [19]. Consider SVD of $X_c X_c^T$:

$$X_c X_c^T = (U\Sigma V^T)(U\Sigma V^T)^T$$
$$X_c X_c^T = (U\Sigma V^T)(V\Sigma U)$$
$$V^T V = I$$
$$X_c X_c^T = U\Sigma^2 U^T$$

From eigen decomposition theorem [33] it is implied that matrix U hold orthogonal eigenvectors of $X_c X_c^T$ and Σ contains square roots of the eigenvalues. Thus, decomposition of $X_c = U\Sigma V^T$ gives eigenvectors and eigenvalues needed for PCA.

4.3 Algorithm

The algorithm of PCA for BTF compression and decompression is mainly borrowed from Borshukov *et. al.* [24, Ch. 15]. We apply PCA per camera direction, which is why it can be called PCA RF. We also experimented and applied PCA per several directions at once.

4.3.1 Compression

In the *first* step of compression algorithm we built ABRDF representation of the BTF data. Let matrix A denote the stored BTF data. We consider each image I as three column vectors a_i (red, green, and blue channels), where $i = 0..(3 * N)$. N is the total number of images in BTF dataset and 3 is the number of channels per image. The size of a_i is $W \times H$, where W and H are dimensions of the image. So, the matrix A has the following dimensions $(W * H) \times (3 * N)$, which consist of such columns a_i. Rows of matrix A are ABRDF representation of BTF, which will be dimensionally reduced.

The *second* step is called "centring" of the data. We compute the average value of each row of matrix A

$$m_i = \frac{1}{3N} \sum_{i=1}^{3N} A_{i,j}$$

Then, we subtract mean vector from each column of matrix A

$$B_{i,j} = A_{i,j} - m_i.$$

At the *last* step, we compute singular value decomposition (SVD) of matrix B. The result of which will be the following decomposition

$$B = U\Sigma V^T$$

where matrix U holds *principal components* of size $W \times H$. Σ is a diagonal matrix and holds the "importance" value of each principal components, and matrix V stores weights that are needed for reconstructing matrix B.

4.3.2 Decompression

The decompression step is simply matrix operations, which require to combine 3 matrices U, Σ, V and mean vector m. To make it a bit easier to decompress on GPU we construct new matrices as Borshukov *et. al.*

$$L = \left[m \mid U\Sigma \right]$$
$$R = \begin{bmatrix} 1...1 \\ V^T \end{bmatrix}.$$

Matrix A takes such form of decompression $A = LR$. In detail, to decompress texture with index i for first C components, we do it separately for each color channel

$$red(x,y) = \sum_{k=1}^{C} L_{xy,k} R_{k,3i+0}$$
$$green(x,y) = \sum_{k=1}^{C} L_{xy,k} R_{k,3i+1}$$
$$blue(x,y) = \sum_{k=1}^{C} L_{xy,k} R_{k,3i+2}$$

4.4 Angular Interpolation

BTF data is measured for a discrete set of light and camera directions, thus it is necessary to perform an interpolation to find the color value for unmeasured directions. We propose a 2-D linear interpolation using barycentric weights [10].

Before computing interpolation weights for input direction P, it is required to find closest measured directions to it. The determination of three closest directions is described in Chapter 4.4.1. After three closest directions $P_1P_2P_3$ are known for input direction P, interpolation weights have to be computed.

It is assumed that sampled measured directions form a convex hull, and found closest directions, i.e a triangle lies on the sampled hemisphere, as in Figure 3.1. Thus, it is possible to use barycentric coordinates to compute interpolation weights w_1, w_2, w_3. Chapter 4.4.2 explains how to compute barycentric coordinates.

To compute the final interpolated color we combine interpolation weights of light and camera directions. Assume that I and O are input light and camera directions. Then, bounding triangles for both of them are $I_1I_2I_3$ and $O_1O_2O_3$ accordingly. Corresponding

barycentric weights then are $b_i = [b_{i1}, b_{i2}, b_{i3}]^T$ and $b_o = [b_{o1}, b_{o2}, b_{o3}]^T$. The final color is the linear combination of b_i, b_o weights and known measured color values

$$C_f = \sum_{u=1}^{3} b_{iu} \sum_{v=1}^{3} b_{ov} C_{uv},$$

where C_{uv} is a color value that corresponds for a light direction I_u and a camera direction O_v.

4.4.1 Finding Closest Directions

Depending on the material sampling, either uniform or non-uniform, different strategies can be applied for finding the closest directions for an input angle. One of the strategies is to compute it in real-time each time or precomputed and stored in a cubemap [12].

BTF Bonn database [1] provides the data with uniform sampling per longitude and per latitude. Depending on the longitude position, different quantization step is applied. For areas closer to the bottom of the hemisphere quantisation step for latitude gets increased. This is done to have relatively equal distance between the directions for any longitude position. Also, more dense sampling needed at important areas (e.g. specular peak areas) to avoid interpolation artifacts. [12].

Assume, a set of quantisation steps $S = \{(\Delta\theta * n, \Delta\phi_n) \mid n \in M \subseteq \mathbb{N}\}$, where M specifies the number of steps. For $\theta = 0^\circ$ only one image is taken, i.e for $(0^\circ, 0^\circ)$ direction.

First, we find four closest directions, and then decrease it three directions. Let the direction for which we need to find the closest direction will be denoted as P=(θ^p, ϕ^p). To find closest points for θ^p, we use algorithm described in Figure 4.1.

Algorithm 1 Find bounding points

```
1: procedure FindBound(angle, step)
2:     lowerBound = step * floor(angle/step)
3:     upperBound = lowerBound + step
4:     Return (lowerBound, upperBound)
```

FIGURE 4.1: **Finding Bounds**

We get that $(\theta_L, \theta_U) = FindBound(\theta_p, \Delta\theta)$. Because, the sampling steps are different for specific θ, first we find closest points that lie at longitude θ_L. In the same manner as for θ, i.e. $(\phi_L^1, \phi_U^1) = FindBound(\phi_p, \Delta\phi_n)$, where $n = \frac{\theta_L}{\Delta\theta}$. And, finally for θ_U we get $(\phi_L^2, \phi_U^2) = FindBound(\phi_p, \Delta\phi_{n+1})$.

So, resulting four close directions to direction P are: $A = (\theta_L, \phi_L^1)$, $B = (\theta_L, \phi_U^1)$, $C = (\theta_U, \phi_L^2)$ and $D = (\theta_U, \phi_U^2)$, as shown in Figure 4.2.

We, then reduce to three closest directions, so then less weights would be computed.

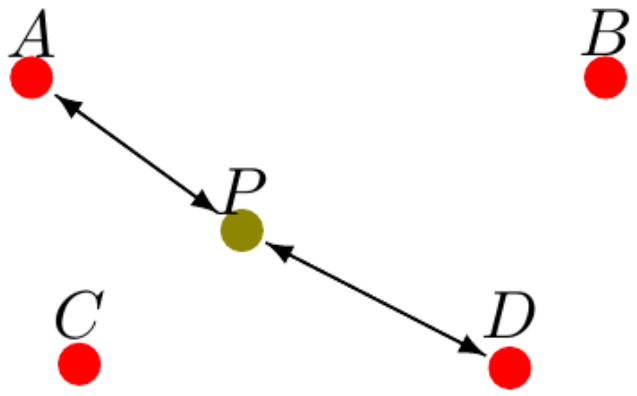

FIGURE 4.2: **Closest Directions**

Consider, Figure 4.2, which shows an example of four close directions. Our aim is to find to which three directions P is closer.

One of the possible ways to find closest three directions is to compute a distance between P and all other four directions and discard the furtherest one. However, this is computational heavy for real-time. The less computational demanding way could be to find approximately to which triangle direction P belongs, i.e. ABC or CBD. The following method produces unnoticeable difference of visual results compared to the method which tests if the point P belong to ABC or CBD.

We compute to which direction P is closer,i.e. whether to A or D. A distance is computed as for Cartesian coordinates, but variables are changed for spherical coordinates: $x = rsin(\theta)cos(\phi)$, $y = rsin(\theta)sin(\phi)$, $z = rcos(\theta)$. So, the distance then will be:

$$d = \sqrt{(x - x^{'})^2 + (y - y^{'})^2 + (z - z^{'})^2} =$$
$$\sqrt{r^2 + r^{'2} - 2rr^{'}(sin(\theta)sin(\theta^{'})cos(\phi)cos(\phi^{'}) + sin(\theta)sin(\theta^{'})sin(\phi)sin(\phi^{'}) + cos(\theta)cos(\theta^{'}))} =$$
$$\sqrt{r^2 + r^{'2} - 2rr^{'}(sin(\theta)sin(\theta^{'})cos(\phi - \phi^{'}) + cos(\theta)cos(\theta^{'}))}$$

Note that, in practice r and $r^{'}$ are equal 1 for simplicity. As, we are interested in comparing distances, it is only enough to compare this term:

$$d^{'} = sin(\theta)sin(\theta^{'})cos(\phi - \phi^{'}) + cos(\theta)cos(\theta^{'})$$

The bigger term $d^{'}$, the smaller the overall distance, because there is are negative sign in the formula d. So, if P is closer to A, the resulting three directions are A, B, C. If P closer to D then B, C, D.

If the input direction P is beyond the measuring directions, i.e. $\theta_p > \Delta\theta * n$ for any n. In this case, we take two closer measured directions, i.e. $A = (\theta_L, \phi_L^1)$ and $B = (\theta_L, \phi_U^1)$ and perform linear interpolation between these two directions.

4.4.2 Barycentric Coordinates

Common interpolation technique for a BTF is barycentric coordinates interpolation. However, it is computational heavy, so the following approximation algorithm proposed by Hatka and Haindl [15] will be used.

Assume that a triangle $P_1P_2P_3$ bounds input point P, for which we want to compute interpolation weights. Figure 3.1 demonstrates the hemisphere on which triangle $P_1P_2P_3$ lies. C_P denotes desired pixel color. So, generally speaking linearly interpolation of that pixel will be $C_P = w_1C_{P1} + w_2C_{P2} + w_1C_{P2}$, where C_{P1}, C_{P2}, C_{P3} correspond to color values of the found triangle $P_1P_2P_3$. Weights w_1, w_2, w_3 are normalized and sum up to 1.

Weights w_1, w_2, w_3 defined as volumes V_1, V_2, V_3 which correspond to PP_2P_3O, PP_3P_1O, PP_1P_2O tetrahedrons, where $O = (0, 0, 0)$. Volumes calculated as determinates of 4×4 vectors

$$w_1 := V_1 = \frac{1}{6} |det(PP_2P_3O)|$$
$$w_2 := V_2 = \frac{1}{6} |det(PP_3P_1O)|$$
$$w_3 := V_3 = \frac{1}{6} |det(PP_1P_2O)|$$

Last step is the normalization of found volumes, i.e. $V_i = \frac{V_i}{\sum_{i=1}^{3} V_i}$.

Chapter 5

Streaming

When rendering a BTF in a web-browser the data has to be transferred before the rendering can start on a client side. Even compressed BTF data of size 10-20 Mb will take time for full transmission. Assume an average user, with a 8Mbit/s connection speed. Depending on the internet connection speed of a user, the full transmission of hight quality BTF could take several minutes. The solution to skip this waiting time, we propose to use progressive streaming of BTF data. Implementing a streaming technique a user will be able to see a low quality preview of original BTF just in a few seconds. In our case, principal components that cover full angular domain will be streamed one by one using WebSockets [32]. The rendering will be refreshed whenever a new component arrives. With each new component the quality of rendering will be progressively improved. Also, it is possible to show an overall progress for an user, which makes the process of streaming even more interactive.

5.0.3 Web Sockets

We use WebSockets for streaming the data, as it is most efficient and elegant way of communicating between a server and a client. The following advantages of WebSocket technology [32, Ch. 1]:

- **Delivers high *Performace*** for real-time server-client connections. Usually, web developers used well known methods such as polling, long polling, and HTTP streaming. However, WebSocket saves bandwidth, CPU power, and latency compared to those methods. For example, polling method makes requests to the server and has to wait for the response. With WebSockets the client does not need to wait for the response, because WebSockets reuse the same connection from client to the server and wise-verse. This single connection reduces the latency.

- ***Saves time*** to develop web-applications. *Simplicity* of is one of the main advantage over older methods for server-client communication.

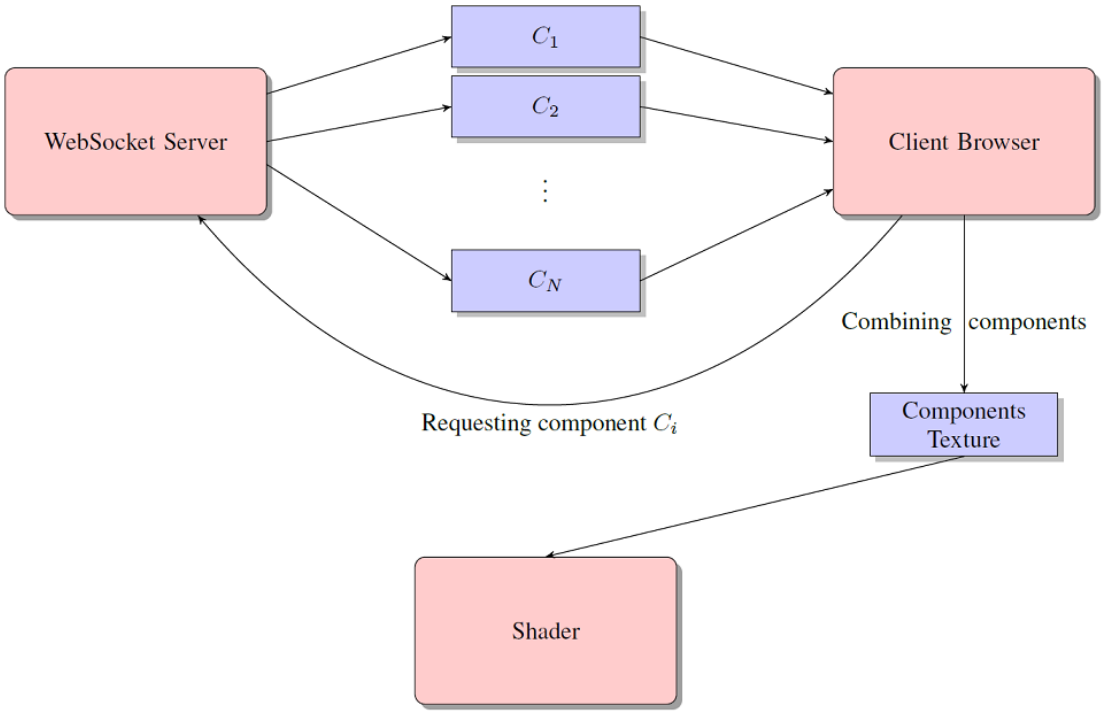

FIGURE 5.1: **Streaming process illustration**

As it was described in Section 4.3 matrices U, Σ and V store all needed data to reconstruct BTF. Matrices Σ and V are quite small in size, they are sent at first place. On the contrary, matrix U stores principal components of size $W \times H$ which makes matrix U the biggest in size of all. On a client side before streaming, matrix U is initialized with blank values, for example zeros. After a Socket connection was established, the client side requests one component at a time. Each new arrived component saved in matrix U and the client side refreshes the shader to render BTF. Figure 5.1 illustrates this process. Also, each component is saved as PNG image on the WebSocket server, which makes a bit further compression of the BTF.

Consider Figure 5.2 that shows how the streaming works on practice. We can see that even with first components the resulted texture looks quite decent. With further components the overall quality of texture improves, e.g. specularities are increasing, small micro-structures become more visible and emphasized. To make the streaming process a bit entertaining, the client also can see the progress-bar of the streaming progress. Also, some of the mid-results were skipped in the Figure 5.2 for the sake of simplicity.

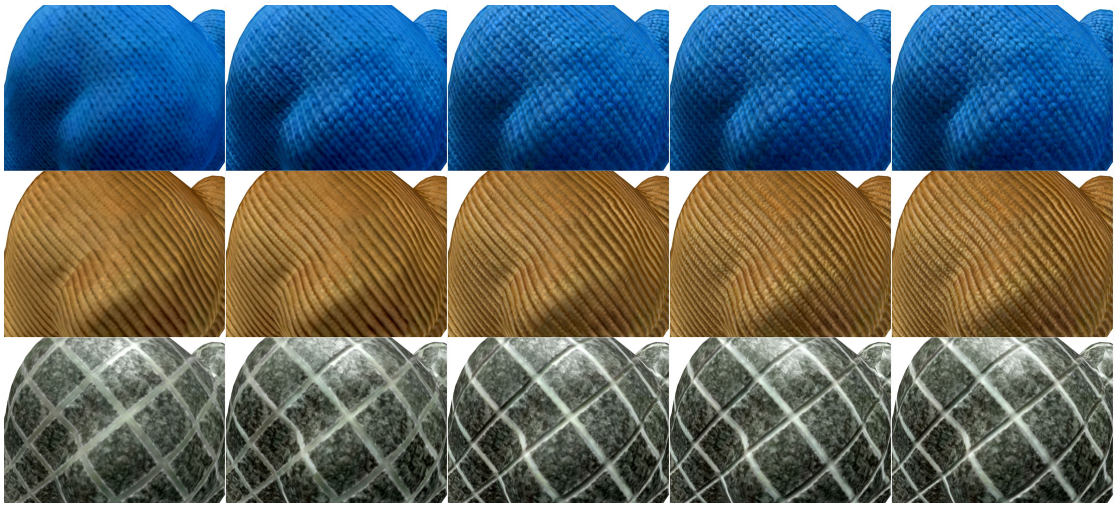

FIGURE 5.2: **Example of Progressive Streaming**
From left to right: *1*, *2*, *4*, *6*, *7* components rendered at the same time.
Note: *8th* component is average grey value, which sends at first place.

5.0.4 Transmission

Chapter 6

Implementation

6.1 Compression

To compress the BTF data we used Java programming language. In our case, the main problem of PCA lies in implementation of singular value decomposition (SVD). To solve this problem, we used a fast linear library *jblas* [3] developed by Mikio Braun. The *jblas* library is gaining popularity in scientific computing. This library is one of the most fastest library for the Java programming that can solve various linear algebra problems.

Compressed data has to be sent to a shader. The best way to send matrices to a shader, is to send them as textures. So, after we perform SVD, we save our resulted matrices U, Σ, V as textures. Matrices U and V are in range $[-1; 1]$, so we map the data into image domain $[0; 1]$. We store each component of matrix U separately as PNG images. We store each component separately as we would need to stream the data. Consider an example shown in Figure 6.1, which shows first components of some materials. Matrices Σ and V are stored together in one texture as they are small enough. Note that the values of matrix Σ are not in range $[-1; 1]$, but we are still able to map them and store in the texture. We will not go in details here, as it is a trivial task.

Also, one practical consideration when using *jblas* is to scale the data for better reconstructing quality in the shader. We found out that tenths of values of U and V matrices are zeros. So, basically we can scale the data by multiplying it with 10 and at the same time matrix Σ by 0.1. This way the reconstruced resulted will the same. But, with scaling we improves precision of the data when we map it.

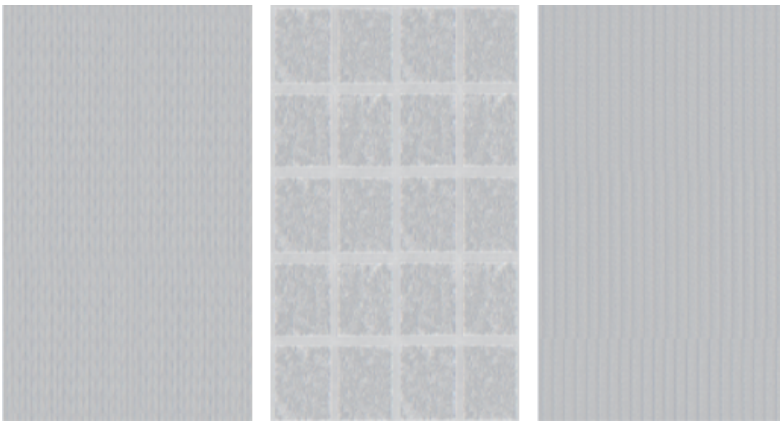

FIGURE 6.1: **Example of Principal Components**
From left to right: wool, impalla, corduroy.

6.2 Rendering

The aim of this thesis was to implement efficient BTF-shader for XML3D [30]. XML3D platform was implemented to deploy 3D graphics in web browsers. This technology is based on WebGL and JavaScript. We also use Xflow [17] to combine principal component textures in one texture, which further is needed for BTF-shader. The shader is written in OpenGL Shading Language (GLSL). The rendering process of the shader is depicted in figure 6.2.

The compressed BTF data is stored in two textures. One texture L stores principal components, the other R stores PCA weights, which determine how the components have to be summed up. The other inputs are texture coordinates, eye and light positions. Eye and light positions are transformed to spherical coordinates. Then, we lookup the three closest views from the measured BTF data, which are will be needed for interpolation purpose. This lookup process is fairly simple as we have a static array in the shader, which stores sample intervals of the measured BTF data.

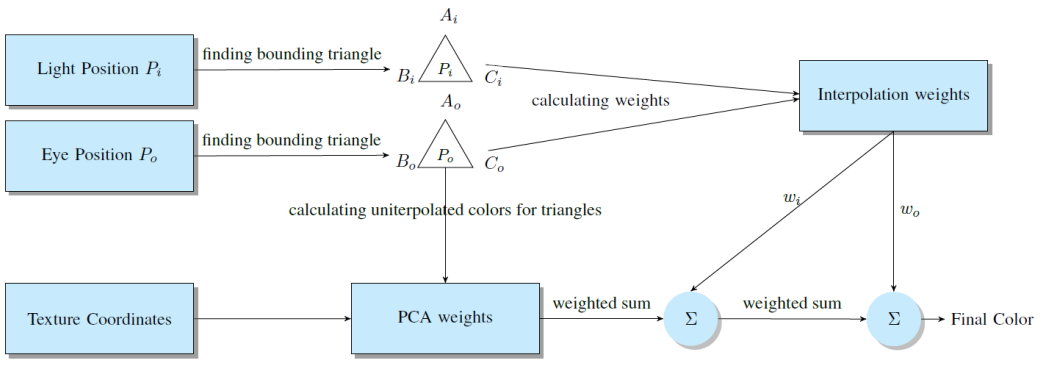

FIGURE 6.2: **Shader Design**

6.3 Streaming

Xflow is a part of XML3D implementation, which allows to process the data on flow, i.e. in runtime. At start, on a client side we create array *texData* of RGB colors, which further will be fulfilled with newly arrived principal components. As it was described in chapter **??**, we stream principal components one by one. Each component C_i arrives as a PNG image. Then, on a client side we read PNG image using PNG decoder written in JavaScript by Arian Stolwijk [4]. PNG images are decoded to pure array of RGB colors. This new data of component C_i is then placed to its place in *texData* array. Finally, using Xflow we create from *texData* array a texture with which we update our BTF-shader.

Chapter 7

Evaluation

Chapter 8

Conclusions and Future Work

8.1 Summary

8.2 Future work

Bibliography

[1] Btf database bonn 2003. http://cg.cs.uni-bonn.de/en/projects/btfdbb/download/ubo2003/. Accessed on January 2015.

[2] Curret btf database. http://www1.cs.columbia.edu/CAVE/software/curet/index.php. Accessed on January 2015.

[3] Linear algebra for java - jblas. http://mikiobraun.github.io/jblas/. Accessed on January 2015.

[4] Pure javascript png decoder. https://github.com/arian/pngjs. Accessed on January 2015.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] K.J. Dana, B. Van-Ginneken, S.K. Nayar, and J.J. Koenderink. Reflectance and Texture of Real World Surfaces. *ACM Transactions on Graphics (TOG)*, 18(1):1–34, Jan 1999.

[7] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, and Westley Sarokin. Acquiring the reectance field of a human face. In *SIGGRAPH*, New Orleans, LA, July 2000.

[8] Y. Dong, S. Lin, and B. Guo. *Material Appearance Modeling: A Data-Coherent Approach: A Data-coherent Approach*. SpringerLink : Bücher. Springer, 2013.

[9] J. Filip and Michal Haindl. Non-linear reflectance model for bidirectional texture function synthesis. In J. Kittler, M. Petrou, and M. Nixon, editors, *Proceedings of the 17th IAPR International Conference on Pattern Recognition*, pages 80–83, Los Alamitos, August 2004. IEEE, IEEE.

[10] M. Haindl and J. Filip. *Visual Texture*. Advances in Computer Vision and Pattern Recognition. Springer-Verlag, London, 2013.

[11] Michal Haindl and J. Filip. A fast probabilistic bidirectional texture function model. In A. J. C. Campilho and M. Kamel, editors, *Image Analysis and Recognition*, pages 298–305, Heidelberg, September 2004. Springer, Springer.

[12] Michal Haindl and Jiri Filip. Advanced textural representation of materials appearance. In *SIGGRAPH Asia 2011 Courses*, SA '11, pages 1:1–1:84, New York, NY, USA, 2011. ACM.

[13] Michal Haindl and M. Hatka. Btf roller. In M. Chantler and O. Drbohlav, editors, *Texture 2005: Proceedings of 4th Internatinal Workshop on Texture Analysis and Synthesis*, pages 89–94, Edinburgh, October 2005. Heriot-Watt University, Heriot-Watt University.

[14] Jefferson Y. Han and Ken Perlin. Measuring bidirectional texture reflectance with a kaleidoscope. *ACM Trans. Graph.*, 22(3):741–748, July 2003.

[15] Martin Hatka and Michal Haindl. Btf rendering in blender. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, VRCAI '11, pages 265–272, New York, NY, USA, 2011. ACM.

[16] Michal Havlíček. Bidirectional texture function three dimensional pseudo gaussian markov random field model. In *Doktorandské dny 2012*, pages 53–62, Praha, 11/2012 2012. České vysoké učení technické v Praze, České vysoké učení technické v Praze.

[17] Felix Klein, Kristian Sons, Dmitri Rubinstein, and Philipp Slusallek. Xml3d and xflow: Combining declarative 3d for the web with generic data flows. *Computer Graphics and Applications, IEEE*, 33(5):38–47, 2013.

[18] Melissa L. Koudelka, Sebastian Magda, Peter N. Belhumeur, and David J. Kriegman. Acquisition, compression, and synthesis of bidirectional texture functions. In *In ICCV 03 Workshop on Texture Analysis and Synthesis*, 2003.

[19] Feifei Li. Advanced topics in data management. University Lecture, 2008.

[20] Tom Malzbender, Tom Malzbender, Dan Gelb, Dan Gelb, Hans Wolters, and Hans Wolters. Polynomial texture maps. In *In Computer Graphics, SIGGRAPH 2001 Proceedings*, pages 519–528, 2001.

[21] Gero Müller, Jan Meseth, and Reinhard Klein. Compression and real-time rendering of measured btfs using local pca. In T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, and R. Westermann, editors, *Vision, Modeling and Visualisation 2003*, pages 271–280. Akademische Verlagsgesellschaft Aka GmbH, Berlin, November 2003.

[22] Gero Müller, Jan Meseth, Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Acquisition, synthesis and rendering of bidirectional texture functions. In Christophe Schlick and Werner Purgathofer, editors, *Eurographics 2004, State of the Art Reports*, pages 69–94. INRIA and Eurographics Association, September 2004.

[23] Addy Ngan and Frdo Durand. Statistical Acquisition of Texture Appearance. pages 31–40.

[24] Hubert Nguyen. *GPU Gems 3*. Addison-Wesley Professional, August 2007.

[25] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Radiometry. chapter Geometrical Considerations and Nomenclature for Reflectance, pages 94–145. Jones and Bartlett Publishers, Inc., USA, 1992.

[26] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Efficient and realistic visualization of cloth. In *Eurographics Symposium on Rendering 2003*, June 2003.

[27] Martin Schneider. Real-time btf rendering. In *The 8th Central European Seminar on Computer Graphics*, pages 79–86, April 2004.

[28] Christopher Schwartz, Roland Ruiters, Michael Weinmann, and Reinhard Klein. Webgl-based streaming and presentation of objects with bidirectional texture functions. *Journal on Computing and Cultural Heritage (JOCCH)*, 6(3):11:1–11:21, July 2013.

[29] Christopher Schwartz, Ralf Sarlette, Michael Weinmann, and Reinhard Klein. Dome ii: A parallelized btf acquisition system. In Holly Rushmeier and Reinhard Klein, editors, *Eurographics Workshop on Material Appearance Modeling: Issues and Acquisition*, pages 25–31. Eurographics Association, June 2013.

[30] Kristian Sons, Felix Klein, Dmitri Rubinstein, Sergiy Byelozyorov, and Philipp Slusallek. Xml3d: interactive 3d graphics for the web. In *Web3D '10: Proceedings of the 15th International Conference on Web 3D Technology*, pages 175–184, New York, NY, USA, 2010. ACM.

[31] Frank Suykens, Karl vom Berge, Ares Lagae, and Philip Dutr. Interactive rendering with bidirectional texture functions. *Comput. Graph. Forum*, 22(3):463–472, 2003.

[32] V. Wang, F. Salim, and P. Moskovits. *The Definitive Guide to HTML5 WebSocket*. Apressus Series. Apress, 2012.

[33] Eric W. Weisstein. Eigen decomposition theorem. From MathWorld—A Wolfram Web Resource.

[34] Eric W. Weisstein. Singular value decomposition.From MathWorld—A Wolfram Web Resource.

[35] Chris Wynn. An introduction to brdf-based lighting. *NVIDIA Corporation*.