Real-time BTF Rendering

Martin Schneider*

Institute of Computer Science II.
University of Bonn
Römerstr. 164, D-53177 Bonn,
Germany

Abstract

The Bidirectional Texture Function (BTF) is a suitable representation for the appearance of highly detailed surface structures under varying illumination and viewing conditions. Since real-time rendering of the full BTF data via linear interpolation is not feasible on current graphics hardware due to its huge size, suitable approximations are required in order to reduce the amount of data and to exploit the features of current graphics hardware.

In this paper we exploit the recently proposed Local Principal Component Analysis (LPCA) compression which provides a reasonable trade-off between visual-quality, rendering speed and size of the representation and present a new rendering method which is approximately three times faster than the originally proposed method. This is achieved by reparameterizing and rearranging the BTF-data appropriately so that built-in hardware interpolation between the values of the discrete LPCA representation can be exploited.

Keywords: Real-time rendering, graphics hardware, shading, texture, image-based rendering

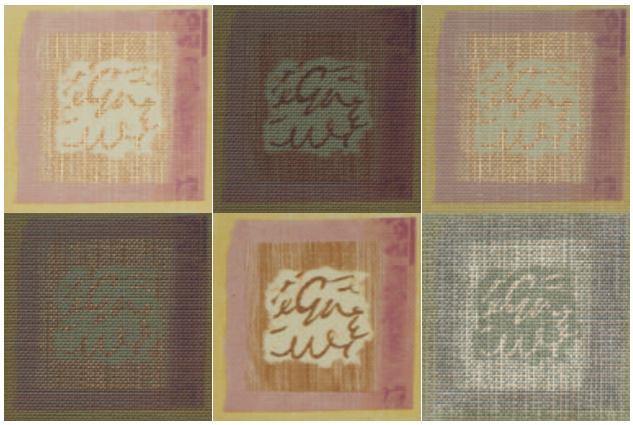

Figure 1: Six views of a wallpaper from various view and light directions. The appearance of the material changes drastically which cannot be reproduced by simple material representations. The BTF correctly represents and reproduces these effects.

1 Introduction

One of the main challenges in computer graphics is still the realistic reproduction of complex materials such as fabric or skin in a real-time rendering environment. Human observers are very sensitive to the subtle differences in the varying appearance of materials under varying light direction and viewpoint resulting from the complex interaction between light and material. These effects can be very strong even for relatively flat materials such as the wallpaper in figure 1. Since a full and physically correct simulation of the interaction between light and material is infeasible in practice, considerable effort has been spent on developing mathematical models which describe the reflection properties of a possibly wide range of materials with a small set of parameters.

For many years the reflectance behaviour of surfaces was modelled by the well known Phong Model because of its simplicity and computational efficiency. The diffuse term of the model was allowed to vary spatially via texture mapping. More convincing results can be achieved in combination with bump mapping or normal mapping techniques, which model the bumpiness of a surface by changing the surface normal. Unfortunately these results lack important effects such as self-shadowing, interreflections and subsurface-scattering. Because of the dramatic development of graphics hardware in recent years, it became possible to render surfaces using more enhanced and physically plausible approximations like the Lafortune[5] or the Ashikhmin[1] model and even arbitrary Bidirectional Reflectance Distribution Functions (BRDF)[4]. But since a BRDF describes reflectance on a microscale only, Dana et al. [2] introduced the Bidirectional Texture Function (BTF) which extends the concept of simple 2D-texture and also captures the mesostructure of a surface.

The BTF is a 6D function whose variables are the 2D surface position, the viewing and lighting directions. It can roughly be interpreted as a per-texel BRDF representation but including additional effects like self-shadowing, self-occlusion and subsurface scattering which are beyond single BRDFs. Since an easy and general method for modelling these effects mathematically seems out of reach,

*schneid3@cs.uni-bonn.de

current research concentrates on image-based methods. In this case the BTF is usually represented by a set of images taken under different light and camera positions. Unfortunately, many real world surfaces contain high frequencies in the spatial and angular domain, requiring a high density sampling consisting of thousands of images to represent the BTF with sufficient quality. Due to the pure size of a BTF (hundreds of megabytes) direct rendering of the BTF data via linear interpolation is currently not feasible in real-time even on todays graphics hardware. Therefore some sort of lossy compression is required that achieves good approximation quality together with high compression rates that allows real-time rendering. Recently several compression approaches for fast rendering have been proposed. In this paper we concentrate on Local Principal Component Analysis (LPCA) compression and present a new rendering method which is approximately three times faster than the originally proposed method.

The rest of the work is organized as follows: In the next section we will review previous work. In the following sections 3 and 4 we will discuss the LPCA compression in greater detail and present the originally proposed rendering method in section 5. Then in section 6 we introduce our revised and faster rendering algorithm. The results are given in section 7 and we end up with the conclusions in section 8.

2 Related Work

There is huge amount of related work in the field of real-time rendering, realistic material representations and image-based rendering and mentioning all of them is beyond the scope of this paper. Hence we consider only the papers concerned with rendering image-based material patches (i.e. BTFs) wrapped onto arbitrary geometry. This work can be roughly grouped into two categories.

The first group interprets the BTF as a set of spatially varying materials and fits simple analytic functions to the discrete BRDFs. Since only few parameters are required per BRDF impressive compression rates are achieved and efficient evaluation in graphics hardware is possible. The least complex model was suggested by McAllister et al. [8] and is directly based on the Lafortune model. The BTF is approximated by fitting Lafortune lobes to the BRDFs that they defined per texel. The model requires very few parameters to be stored per pixel resulting in a very compact material representation that can be efficiently rendered in hardware employing vertex and pixel shaders. The Lafortune Lobes model the variance of luminance of the surface point while the diffuse and specular albedo are stored as RGB color values. Already one year earlier Daubert et al. [3] proposed a similar but slightly more complicated approach which is also based on the Lafortune model. They additionally incorporated a view-dependent scaling factor per pixel stored in a lookup-table that modulates the pixel-wise Lafortune models in order to cope with self-occlusion effects. Since the lookup table is defined per pixel significantly more parameters have to be stored but real-time rendering is still possible by employing vertex and pixel shaders. Meseth et al. [9] proposed an improved material representation based on fitting a set of reflectance fields to the BTF. Each reflectance field describes the appearance of the surface for a view direction from the measurement process. Linear interpolation is employed to cover view directions not in the measured set. Since the reflectance fields are fitted per pixel and measured view direction, the amount of parameters is even higher than in the model by Daubert et al. but still allows real-time rendering. Nevertheless, these methods suffer from a lack of rendering quality due to the simplicity of the fitted analytic functions.

The second group of methods was developed in the context of pattern recognition and is intended to exploit the statistical properties of the BTFs. Sattler et al. [11] proposed a rendering method based on this kind of data driven approach by decomposing the BTF data into sets of principal components (called Eigen-Textures), one set for each measured view direction. As for the reflectance field based model linear interpolation is employed to yield light and view directions not in the measured set. Depending on the number of Eigen-Textures used, the number of parameters to be stored per pixel easily exceeds the number of parameters for the reflectance field based model. In contrast to these methods, the approach published by Müller et al. [10] is based on Eigen-BRDFs. The BTF is understood as a set of spatially varying BRDFs which is compressed using a combination of vertex quantization and PCA named Local-PCA yielding clusters of Eigen-BRDFs. Since the Eigen-BRDFs store values for measured view and light directions, this method requires linear interpolation as well to cover all possible view and light directions. The number of clusters can be adjusted accordingly to the structural complexity of the material. For structured materials this results in significantly reduced memory requirements compared to the previous method. Real-time rendering can be realized using the vertex and pixel shaders.

3 BTF-representation

In this work we adopt the BTF-compression method of [10]. They point out that a sampled BTF can be either interpreted as a collection of discrete textures (figure 2):

$$\mathbf{BTF}_{Tex} := \left\{ T_{(\mathbf{v},\mathbf{l})}(\mathbf{x}) \right\}_{(\mathbf{v},\mathbf{l})\in\mathcal{M}}$$

where $\mathcal{M}$ denotes the set of discrete measured view and light directions, or as a set of tabulated BRDFs:

$$\mathbf{BTF}_{Brdf} := \left\{ B_{(\mathbf{x})}(\mathbf{v},\mathbf{l}) \right\}_{\mathbf{x}\in I\subset\mathbb{N}^2} .$$

Please note that these BRDFs do not fulfill physically demanded properties like reciprocity. In special they already contain a factor $(\mathbf{n}\cdot\mathbf{l})$ between incident direction and surface normal. This is also nicely illustrated in figure 2. The

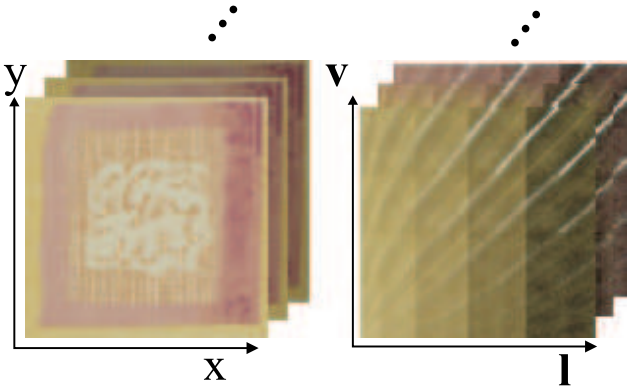

Figure 2: Two arrangements of the BTF data: As set of images (left) and as set of BRDFs (right).

BTF-data employed in this work is a high-quality RGB-sampling with $|\mathcal{M}| = 81 \times 81$ and $|I| = 256 \times 256$ leading to more than 1.2GB of data (consider [11] for details on the measurement process). In order to reduce the memory requirements we apply the Local-PCA algorithm to $\mathbf{BTF}_{Brdf}$ as in [10]. The resulting compressed representation has the form

$$\mathbf{BTF}_{Brdf} \approx \left\{ \tilde{B}^c_{(\mathbf{x})} \right\}_{\mathbf{x} \in I \subset \mathbb{N}^2},$$

where

$$\tilde{B}^c_{(\mathbf{x})} = \bar{B}_{k(x)} + \sum_{i=1}^{c} \langle B_{(\mathbf{x})} - \bar{B}_{k(x)}, E_{i,k(x)} \rangle * E_{i,k(x)}$$

denotes the reconstruction of the BRDF at texel x from c components of the PCA applied to the BRDFs in cluster $k(x)$ looked up from texel $\mathbf{x}$. The $E_{i,k}$ are the so called Eigen-BRDFs (i.e. the PCA-components in cluster k) and the mean is denoted by $\bar{B}_{k(x)}$. Adjusting the number of components c and the number of clusters $|k|$ allows for finding a flexible trade-off between memory-requirements, approximation quality and rendering speed. For real-time rendering c should be chosen as small as possible ($c < 8$), since rendering time depends on the number of components that have to be summed up. Depending on the material complexity compression ratios between 1:50 and 1:250 introducing a relative error of only about 2% can be achieved.

4 Rendering with LPCA-encoded BTFs

In general accurate rendering algorithms have to compute results approximating the rendering equation for every surface point $\mathbf{x}$ by computing outgoing radiance L_r as follows (neglecting emissive effects):

$$L_r(\mathbf{x}, \mathbf{v}) = \int_{\Omega_i} BRDF_{\mathbf{x}}(\mathbf{v}, \mathbf{l}) L_i(\mathbf{x}, \mathbf{l}) (\mathbf{n}_x \cdot \mathbf{l}) \, d\mathbf{l}$$

Here, $BRDF_{\mathbf{x}}$ denotes the BRDF for point $\mathbf{x}$, L_i denotes incoming radiance, n is the surface normal and Ω_i is the hemisphere over $\mathbf{x}$. Employing measured BTFs, the following approximation results:

$$L_r(\mathbf{x}, \mathbf{v}) \approx \int_{\Omega_i} \mathbf{BTF}(\mathbf{x}, \mathbf{v}, \mathbf{l}) L_i(\mathbf{x}, \mathbf{l}) \, d\mathbf{l}$$

The area foreshortening term is removed since this effect is represented in the measured BTFs already. In the presence of a finite number of point light sources only, the integral reduces to a sum. Approximating the BTF by the clustered Eigen-BRDFs, the above equation reduces to

$$L_r(\mathbf{x}, \mathbf{v}) \approx \sum_{j=1}^{n} \sum_{l=1}^{c} \alpha_{l,k(\mathbf{x})} E_{l,k(\mathbf{x})}(\mathbf{v}, \mathbf{l}_j) L_i(\mathbf{x}, \mathbf{l}_j)$$

Here, n denotes the number of light sources, α denotes the projections on the respective basis vectors as in section 3, c is the number of components from the clustered Eigen-BRDFs $E_{k(\mathbf{x}),l}$ and $k(\mathbf{x})$ is the cluster or material index. Since the BTF was sampled for fixed light and view directions only, linear interpolation is employed to support arbitrary view and light directions, resulting in the final equation which has to be evaluated on the graphics hardware:

$$L_r(\mathbf{x}, \mathbf{v}) \approx \sum_{j=1}^{n} \sum_{\substack{\tilde{\mathbf{v}} \in N(\mathbf{v}) \\ \tilde{\mathbf{l}} \in N(\mathbf{l}_j)}} w_{\tilde{\mathbf{v}}, \tilde{\mathbf{l}}} \sum_{l=1}^{c} \alpha_{l,k(x)} E_{l,k(x)}(\tilde{\mathbf{v}}, \tilde{\mathbf{l}}) L_i(\mathbf{x}, \mathbf{l}_j)$$

By $N(\mathbf{v})$ we denote the set of neighboring view directions of $\mathbf{v}$, for which data was measured ($N(\mathbf{l})$ respectively), while w denotes an interpolation weight.

5 Original LPCA rendering method

In this section we will shortly review the rendering algorithm for LPCA encoded BTFs as presented in the original work by Müller et al. [10].

The BTF data needed for rendering is stored in three textures as depicted in figure 3. A rectangular texture T_1 stores material cluster indices together with sets of floating-point PCA weights α which determine, how the components of the indexed Eigen-BRDFs are to be combined. Another rectangular texture T_2 holds the floating-point valued components of the Eigen-BRDFs for the various clusters.

Two cube maps are employed to determine the three closest measured view and light directions for the current

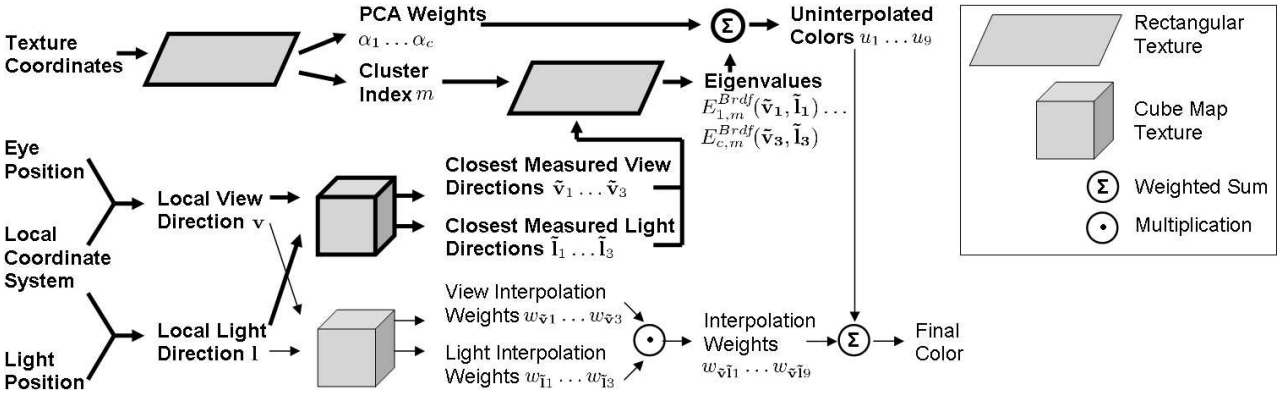

Figure 3: Layout of the original LPCA-rendering. The elements of the basic rendering algorithm are highlighted by bold font, thick arrows and thick borders. The other elements contribute to the view and light direction interpolation. The required data is stored in several textures. All processing takes place in the pixel shader.

view and light direction and the respective interpolation weights.

The rendering process is depicted in figure 3. The inputs are standard texture coordinates, the eye and light positions, and a per-pixel coordinate system, which is interpolated from the local coordinate systems at the vertices which are specified with the geometry and stored in display lists.

At first view and light directions are computed and transformed into the pixel's coordinate system. Using cube maps, the three closest view and light directions from the measurement process are looked up, together with their interpolation weights. The interpolation weights for each pair of closest view and light direction are multiplied to yield the final nine interpolation weights.

At the same time, the basic rendering algorithm is performed: using the texture coordinates, cluster indices and PCA weights for every pixel on the screen are looked up (just as we would lookup colors for texture mapping). The cluster index is used to select the appropriate 81x81x3 vectors representing the c Eigen-BRDFs of the current cluster from T_2. The view and light direction are used to select the appropriate RGB components from the large vectors. If interpolation is used, this lookup is repeated for every pair of closest view and light direction. Combining the PCA weights of the current pixel with the RGB components, the uninterpolated colors are computed (one for each pair of closest view and light direction). In a final step, these colors are multiplied with their respective interpolation weights and the results are summed to form the final color of the pixel.

The major drawback of this method is that all Eigen-BRDFs are interpolated manually, which is very expensive. In the following section we will present a new rendering method that employs the built-in interpolation capabilities of modern graphics boards resulting in a significant rendering speed-up.

6 Our new rendering method

In the hardware rendering approach presented in the previous section BTFs are approximated by weighted sums of clustered Eigen-BRDFs. Since the BTF was sampled for fixed light and view directions, linear interpolation has to be employed to support arbitrary viewing and lighting directions. Since 4D texture mapping is not supported on current graphics hardware interpolation cannot be directly performed in graphics hardware and was therefore performed manually in the mentioned approach. To accomplish this the three closest measured view and light directions are looked up together with their corresponding interpolation weights. Since each pair of closest view and light direction has to be considered, this results in a total of nine uninterpolated color values and nine interpolation weights that have to be multiplied and summed up.

In order to speed up this time consuming process we will incorporate the approach by Liu et al.[6] into the Local PCA rendering algorithm. Therefore we reparameterize the Eigen-BRDFs in a way that allows us to make use of the volume texture capabilities of current graphics hardware. Since the need for interpolation affects only the Eigen-BRDFs, this approach is also applicable for common BRDF rendering.

6.1 Reparameterization

For rendering efficiency it is important to find a good parameterization for the viewing and lighting directions so that we can achieve a good resampling of the Eigen-BRDFs on a regular grid. With a good parameterization, the viewing and lighting directions corresponding to a uniform sampling grid of the parameter space are evenly distributed on the hemisphere. Experience has shown that a good map should have the following properties:

Preserve fractional area: This property ensures that a "fair" set of points on the square will map to a fair set on the hemisphere.

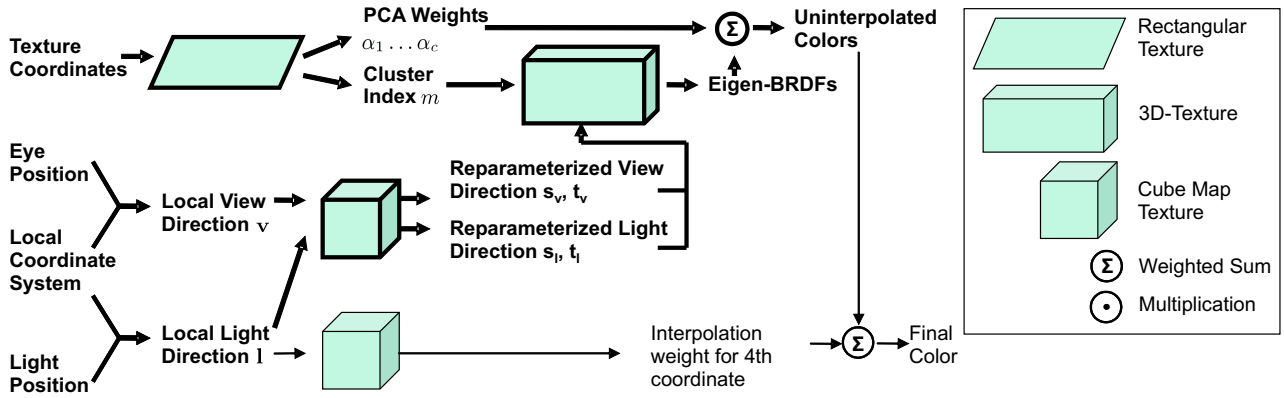

Figure 4: Layout of the revised LPCA-rendering: Now two 3D texture coordinates are looked up together with the corresponding linear interpolation weight. Please note that only one direction is interpolated manually.

Bicontinuous: A map is bicontinuous if the map and its inverse are both continuous. Such a map will preserve adjacency. It is necessary since we wish to use the linear distance on the square as an estimate of angular distance on the hemisphere.

Low distortion: By low distortion, we mean that shapes are reasonably well preserved.

The elevated concentric map has the properties listed above and maps uniformly distributed points on the square to uniformly distributed points on the hemisphere. Therefore it is a suitable parameterization for our needs.

The elevated concentric map Ω can be represented as a concatenation of an elevation map Ψ and concentric map Φ.

$$\Omega(s,t) = (\Psi \circ \Phi)(s,t)$$

The concentric map $\Phi(s,t) : [0,1] \times [0,1] \rightarrow [0,1] \times [0,2\pi]$ is a function from the unit square to the unit disk, mapping concentric squares to concentric circles. At first the square is mapped to $(s,t) \in [-1,1]^2$ and is divided into four regions by the lines $s = t$ and $s = -t$. For the first region the map $\Phi(s,t) = (\rho(s,t), \phi(s,t))$ is

$$\begin{aligned} \rho(s,t) &= s \\ \phi(s,t) &= \tfrac{\pi}{4}\tfrac{t}{s} \end{aligned}$$

This produces an angle of $\phi \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ The other three regions have analogous transforms (see [12] for details).

The concentric map can be extended to the hemisphere by projecting the points up from the unit disk to the unit hemisphere. In order to create a uniform distribution on the hemisphere we generate a uniform distribution of the z-coordinate. This can be accomplished by noting that if we have a uniform distribution on a disk, then the distribution of ρ^2 is also uniform. So given a uniformly distributed point on a disk, we can generate a uniformly distributed point on the hemisphere by first generating the z-coordinate from ρ^2, and then assigning x and y such that the point is on the hemisphere. As a result the elevation map $\Psi(\rho,\phi) : [0,1] \times [0,2\pi] \rightarrow \mathbb{R}^3$ is defined as

$$\begin{aligned} x &= \rho\sqrt{2-\rho^2}\cos\phi \\ y &= \rho\sqrt{2-\rho^2}\sin\phi \\ z &= 1-\rho^2 \end{aligned}$$

Since the elevated concentric map $\Omega(s,t)$ is invertible with

$$(s,t) = \Omega^{-1}(x,y,z) = \Omega^{-1}(x,y,\sqrt{1-x^2-y^2})$$

we are able to reparameterize the Eigen-BRDFs as follows

$$\mathbf{E}_{i,k(x)}(\theta_v,\phi_v,\theta_l,\phi_l) = \tilde{\mathbf{E}}_{i,k(x)}(s_v,t_v,s_l,t_l)$$

Now we can use this reparameterized BTF for hardware-accelerated rendering.

6.2 Data organization

In this section we describe how the 4D Eigen-BRDFs are organized in a 3D texture, so that the volume texture capabilities of current graphics hardware can be exploited.

For a given 4D reparameterized Eigen-BRDF $\tilde{E}_{j,k(x)}(s_v,t_v,s_l,t_l)$, we discretize the lighting parameter t_l as $\{t_{l_0},\dots,t_{n(t_l)-1}\}$ so we create a set of functions with fixed t_l for all $n(t_l)$ provided values of t_l. In other words, we create a set of volume textures $\{S_0,\dots,S_{n(t_l)-1}\}$, where

$$S_i(s_v,t_v,s_l) = \tilde{E}_{j,k(x)}(s_v,t_v,s_l,t_{l_i})$$

Now the interpolation in the s_v,t_v and s_l dimension are performed by the hardware and so we only have to handle the interpolation in the fourth dimension t_l by ourselves. This is done by performing two texture accesses and linearly interpolating the results afterwards.

$$\tilde{E}_{j,k(x)}(s_v,t_v,s_l,t_l) = \begin{cases} S_0(s_v,t_v,s_l) & ,t_l < t_{l_0} \\ S_{n(t_l)-1}(s_v,t_v,s_l) & ,t_l \geq t_{l_{n(t_l)-1}} \\ (1-w)S_i(s_v,t_v,s_l) & \\ +wS_{i+1}(s_v,t_v,s_l) & ,t_{l_i} \leq t_l < t_{l_{i+1}} \end{cases}$$

with $w = (t_l - t_{l_i})/(t_{l_{i+1}} - t_{l_i})$.

In practice the sequence $S_i(s_v,t_v,s_l)$ of volume textures is combined into a single volume texture and loaded into graphics hardware. The texture is indexed by the texture coordinates

$$(s_v, t_v, z_1 = t_{l_i} + \tfrac{s_l}{n(t_l)})$$
$$(s_v, t_v, z_2 = t_{l_{i+1}} + \tfrac{s_l}{n(t_l)})$$

The cost of an Eigen-BRDF interpolation is therefore two texture accesses and one linear interpolation.

6.3 Shader design

In order to avoid the pixel shader's online effort of calculating the viewing parameters (s_v,t_v) from (θ_v,ϕ_v) we precompute the mapping and store it in a texture. Therefore we densely sample the hemisphere of directions and store for every direction the corresponding (s_v,t_v) in a cube map. In addition to (s_v,t_v) we also precompute z_1, z_2 and w values and put them into a second cube map. Another issue is dealing with the floating point representation of the Eigen-BRDFs. Since Nvidia's GeForce FX only supports 3D textures with 8-bit precision we quantize every Eigen-BRDF separately to 8-bit yielding scaling factors $s_{j,k}$. To compensate for this we divide every PCA weight $\alpha_{j,k}$ by the corresponding scaling factor $s_{j,k}$ in order to avoid performing the rescaling at runtime.

The new rendering process is depicted in figure 4. It differs in representation of the Eigen-BRDFs. Now we fetch the (s_v,t_v) values by accessing the first cube map via the view vector. The light vector is used as a lookup index into the second cube map and fetches the z_1, z_2 and w values. Now these values are combined in the pixel shader to yield the two texture coordinates $(s_v, t_v, z_1 = t_{l_i} + \frac{s_l}{n(t_l)})$ and $(s_v, t_v, z_2 = t_{l_{i+1}} + \frac{s_l}{n(t_l)})$. With them the 3D texture is accessed and the results are linearly interpolated by the interpolation weight w.

7 Results

We have implemented the presented rendering method on a NVidia Geforce FX 5950 graphics board and tested it with several models and BTF-data from BTF Database Bonn[1]. Table 7 enlists frame rates for the two models depicted in figure 5.

[1]http:\\btf.cs.uni-bonn.de

	Local-PCA		revised Local-PCA	
c	cloth	statue	cloth	statue
2	21	17	67	50
4	15	11	42	34
8	7	4	31	24

Table 1: Rendering performance in frames per second for the models depicted in figure 5 and different numbers of LPCA components.

8 Conclusions

In this work we presented significant improvements in rendering LPCA compressed BTFs on graphics hardware. We have shown how the necessary manual interpolations between measured light and view directions can be handed over to the build-in hardware interpolation by using a reparameterization and the volume texture capabilities of graphics hardware. Without loss of quality the new shader performed approximately three times faster in our tests compared to the straightforward implementation.

In the future we will extend our method to support mip-mapping. Unfortunately, the standard mip-mapping capabilities of graphics hardware cannot be employed due to the clustering step. We therefore have to compute new material cluster indices and cluster weights for each mip-map level. To smoothly interpolate between different mip-map levels, we will determine the currently desired exact texture resolution level and need to manually interpolate between the discrete mip-map levels by essentially executing the described rendering algorithm twice.

References

[1] M. Ashikhmin, S. Premoze and P. Shirley. A Microfacet-based BRDF Generator. In *Proceedings of Siggraph 2000*, pp. 65–74

[2] K. J. Dana, B. van Ginneken, S. K. Nayra, and J. J. Koenderink. Reflectance and Texture of Real World Surfaces. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 151–157, 1997

[3] K. Daubert, H. Lensch, W. Heidrich and H. P. Seidel. Efficient Cloth Modeling and Rendering. In *12th Eurographics Workshop on Rendering*, pp. 63–70, 2001

[4] J. Kautz, P.-P. Sloan, J. Snyder Fast, Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonics In *12th Eurographics Workshop on Rendering*, pp. 301–308, 2002

[5] E. Lafortune, S. Foo, K. Torrance and D. Greenberg Non-linear approximation of reflectance functions. In *IEEE Int. Conf. on Computer Vision*, pp. 1010–1017, 1999

[6] X. Liu, Y. Hu, J. Zhang, X. Tong, B. Guo and H. Shum Synthesis and Rendering of Bidirectional Texture Functions on Arbitrary Surfaces. Submitted to IEEE TVCG, 2002

[7] D. K. McAllister. A Generalized Representation of Surface Appearance. *PhD thesis, University of North Carolina*, 2002

[8] D. K. McAllister, A. Lastra, and W. Heidrich. Efficient Rendering of Spatial Bi-directional Reflectance Distribution Functions. In *Graphics Hardware 2002*, pp. 78–88, 2002

[9] J. Meseth, G. Müller, R. Klein. Preserving Realism in real-time Rendering of Bidirectional Texture Functions. In *OpenSG Symposium 2003*, pp. 89-96, April 2003.

[10] G. Müller, J. Meseth, R. Klein Compression and real-time Rendering of measured BTFs using local PCA. In *Vision, Modeling and Visualization*, pp. 271–279, 2003

[11] M. Sattler, R. Sarlette, and R. Klein. Efficient and Realistic Visualization of Cloth. In *Eurographics Symposium on Rendering*, 2003

[12] P. Shirley, K. Chiu. A Low Distortion Map between Disk and Square. In *Journal of Graphics Tools*, vol. 2, no.3, pp. 45–52, 1997

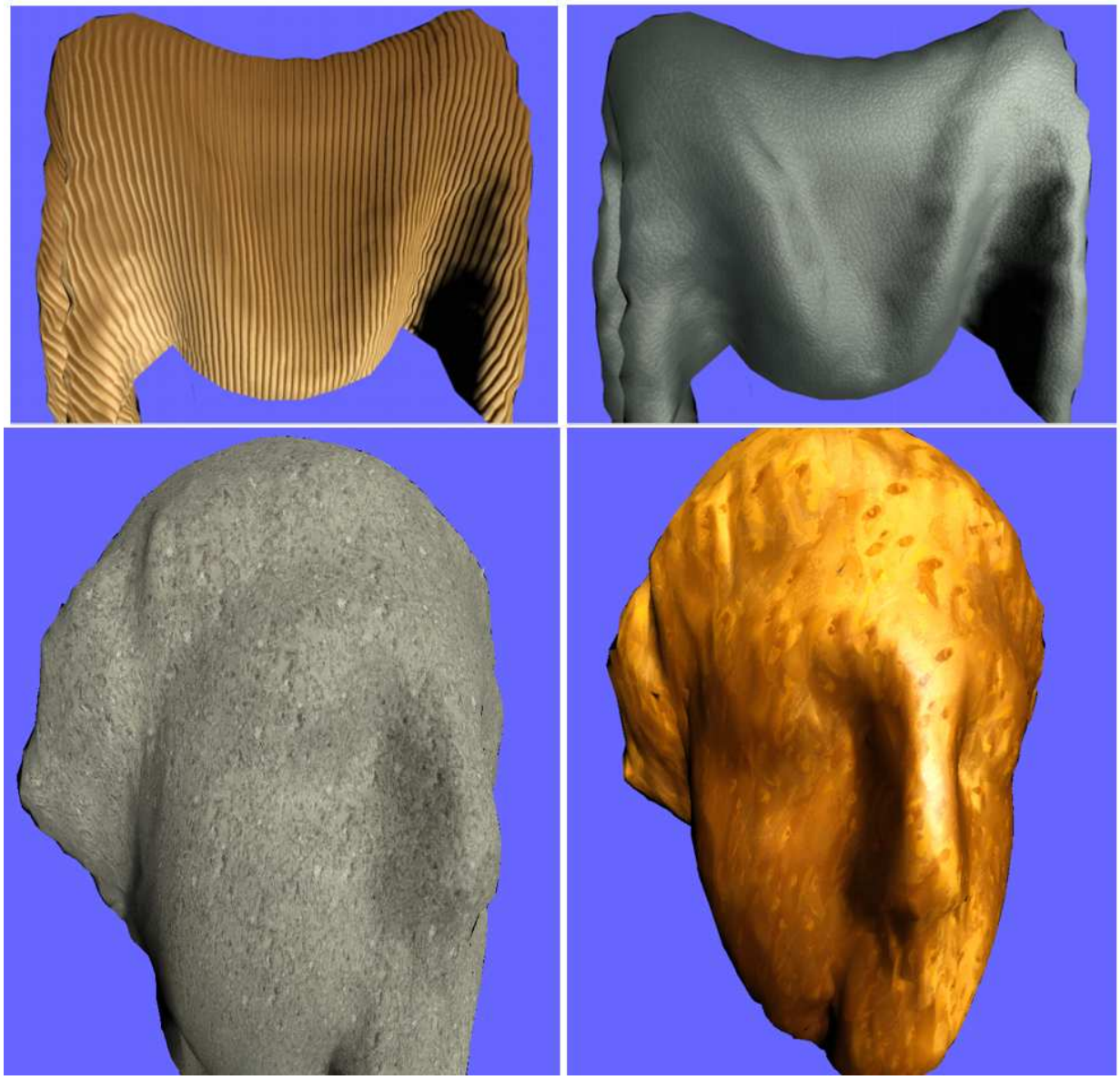

Figure 5: Some results of the rendering method. In the top row we have a cloth covered with a corduroy and synthetic grey. The bottom row depicts a weathered statue head covered with stone and varnished wood.