BTF Rendering for Virtual Environments

Jan Meseth, Gero Müller, Mirko Sattler, Reinhard Klein

Institute for Computer Science II, Römerstr. 164, 53117 Bonn, Germany
Phone/Fax: +49-228-73-4191/+49-228-73-4212
E-mail: {meseth,gero,sattler,rk}@cs.uni-bonn.de

Abstract :

Virtual Environments mostly try to convey as realistic as possible impressions. Among other senses, the eye provides the users with the most important inputs.
Achieving visual realism typically relies on highly accurate geometric models. Unfortunately unhandleable amounts of triangles need to be rendered to adequately represent mesostructure. Furthermore, sophisticated lighting calculations need to be computed to incorporate local self-shadowing and interreflection effects.
In current VR systems material representations like textures and bump-maps are utilized to compensate the abovementioned problems. While these approximations yield sufficient results for very simple materials, they are insufficient for more ambitious applications like cloth visualization or interior design, which require highly realistic material representations like the bi-directional texture function (BTF).
In this work, we compare several real-time BTF rendering methods concerning accuracy, runtime and memory requirements, and identify application areas for the different techniques. In addition, we show how BTF rendering can be integrated into scene-graph systems.

Key words : BTF rendering, realistic materials, real-time rendering, virtual reality, scene graphs.

1- Introduction

One of the possible goals of Virtual Environments (VEs) reproduced in Virtual Reality (VR) applications is to present the user an as realistic impression of the virtual world as possible. For humans, this especially concerns the visual realism of rendered scenes. Realistic, rendered images of real-world objects usually require complex geometric models and sophisticated modeling of the objects' surface reflectance behavior. Since the geometric complexity of an object's mesostructure is typically too complex to be modeled by geometric primitives, it is usually represented by suitable material representations that efficiently capture the surface's reflectance properties.
In existing VR applications, such materials are typically represented as coefficients of the well-known Phong-Model [1] because of its simplicity and computational efficiency. The (Lambertian) diffuse term of the model is allowed to vary spatially via texture mapping. Even more convincing results are achieved in combination with bump-mapping [2] or normal-mapping techniques [3], which model the bumpiness of surfaces by changing surface normals. Unfortunately, the results from these techniques still lack important local effects like view-dependent colors, self-shadowing, self-occlusion, inter-reflections and subsurface-scattering, which are commonly observed in real-world materials (compare figure 1).

Fig. 1: Six views of one wallpaper from various view and light directions. The appearance of the material changes drastically which cannot be reproduced by simple material representations like bump-mapped textures. The BTF [4] correctly represents and reproduces these effects.

The need for realistic materials has increased tremendously in the last years: Radiosity and Raytracing applications more and more rely on sophisticated, realistic material representations to enhance the realism in the resulting images. In the VR community, more and more ambitious application areas require steadily more accurate material representations. As an example, consider the interior design of a car in VR: objects have to be placed with great care since reflections in the windshield disturbing the driver have to be avoided at all costs. While simple reflectance models will fail to identify bad as well as favorable settings since their approximation of reality turns out too crude, highly accurate models can avoid these problems. Therefore, they potentially shorten the time-to-market by allowing early, well-founded judgment of the overall appearance of virtual prototypes.

Fig. 2: Gear box model (courtesy of DaimlerChrysler) with parts covered with lit textures and other parts lit by a Phong model. The structure of the model appears flat.

Fig. 3: Gear box model (courtesy of DaimlerChrysler) with parts covered with measured BTFs and other parts lit by a Phong model. While the reflection properties of the aluminium material appear much more realistic, the synthetic leather material nicely illustrates the enhanced depth impression generated by BTF rendering.

Stand-alone applications nowadays show the possibility to render surfaces using such high-quality and physically plausible approximations like spatially varying bi-directional reflectance distribution functions (BRDFs) (which describe the reflectance properties of single surface points) or bidirectional texture functions (BTFs) (which additionally include local self-shadowing, self-occlusion, inter-reflections and subsurface scattering) in real-time. As a result, the depth impression of the material increases drastically, leading to a realistic look-and-feel (for a comparison between the depth impression and the look-and-feel of BTF rendering with lit textures see figures 2 and 3). As a side-effect, correct depth impressions as perceived through local shadowing and inter-reflections significantly improve navigation in VEs. Unfortunately, due to the pure size of a BTF (hundreds of megabytes) real-time rendering of the full data is currently not feasible.

In this document, we review existing analytic and linear-basis-decomposition BTF rendering methods and provide a comparison in terms of accuracy and resources required for real-time rendering. Based on our results, we will identify application areas for the different techniques. Additionally, we will describe how BTF rendering can be integrated into the scene graph system OpenSG and provide example results.

The rest of the document is organized as follows: After reviewing related work in section 2, we review existing BTF rendering methods in section 3. In section 4, we compare the methods concerning accuracy, run-time, memory, and preprocessing requirements and identify application areas. In section 5, we describe the integration of BTF rendering into the scene graph system OpenSG. We conclude in section 6 and describe future directions of research.

2- Related Work

Material representations have been studied intensely in Computer Graphics for a long time already, although the focus of research varied substantially over the years. Here, we will concentrate on high-quality material representations for real-time rendering.

In the area of rendering single BRDFs a lot of research has been done in the last years and impressive results were achieved already. Early results fitted analytic functions to the BRDF resulting in the well-known Ward [5] and Lafortune [6] models. Kautz and McCool [7] approximated the four-dimensional *BRDF(**l**, **v**)* by a product of two two-dimensional functions *g(**l**)* and *h(**v**)* of the light direction ***l*** and view direction **v** which are stored as textures and combined during the rendering step. McCool et al. [8] improved the above method by employing homomorphic factorization, leading to approximations with user-controllable quality features. The above approaches were further improved by [9], [10], [11], [12] and [13] which all enable the BRDF to be lit by image-based lighting while relying on different approximation functions. Unfortunately, their representations can not easily be applied to real-time rendering of spatially varying materials.

In the context of rendering spatially varying materials, Debevec et al. [14] presented a method for reflectance field rendering (fixed view, varying light). They acquired and relighted human faces exploiting a human skin reflectance model. Malzbender et al. [15] compressed the reflectance fields of each texel of a measured material by fitting polynomials of low degree. Ashikhmin and Shirley [16] used basis textures lit by a steerable light basis for relighting. Levoy and Hanrahan [17] and Gortler et al. [18] simultaneously developed methods for light field rendering (fixed lighting, varying view). Miller et al. [19] parameterized light fields over surfaces introducing surface light fields and employed JPEG-like compression for the images. Following publications [20] [21] concentrated on the application of different data compression schemes.

BTF-rendering can be understood as rendering of spatially varying materials under varying light and view conditions. Due to the enormous amount of data in a BTF, only few real-time rendering algorithms have been published so far. A foundation was set by Kautz and Seidel [22] since they introduced techniques for evaluating spatially varying BRDFs on graphics hardware using a technique similar to Kautz and McCool [7]. This technique was recently improved by Suykens et al. [23] but even their improved method can achieve good results only for very simple synthetic BTFs. McAllister et al. [24] published a different method exploiting graphics hardware that approximates the BTF by pixelwise Lafortune models. One year earlier

already, Daubert et al. [25] published a similar approach in the context of rendering synthetic cloth BTFs: they additionally modulated the pixelwise Lafortune models with a view-dependent factor in order to cope with self-occlusion effects. Meseth et al. [26] proposed an improved material representation based on reflectance fields which increases the real-time rendering quality of materials, especially if they feature high depth variation. A different approach was introduced by Sattler et al. [27], who utilize principal component analysis (PCA) to compute Eigen-Textures that are composed during runtime based on view- and light-dependent weights. The method was improved in terms of memory requirements and approximation quality by an approach of Müller et al. [28]. They perform clustering of the spatially varying BDRFs by employing local PCA in order to compute Eigen-BRDFs.

3- BTF Rendering

Rendering BTF textured scenes implies evaluating the exitant radiance L_r for every surface point $\boldsymbol{x}$ following the formula

$$L_r(\boldsymbol{x},\boldsymbol{v}) = \int_{\Omega_i} BTF(\boldsymbol{x},\boldsymbol{v},\boldsymbol{l}) \cdot L_i(\boldsymbol{x},\boldsymbol{l}) \cdot (\boldsymbol{n}\cdot\boldsymbol{l})\, d\boldsymbol{l} \qquad (1)$$

where $\boldsymbol{v}$ is the view direction, L_i the incident radiance, $\boldsymbol{n}$ the surface normal, and Ω_i the hemisphere over $\boldsymbol{x}$. Please note that the term $(\boldsymbol{n}\cdot\boldsymbol{l})$ is included in measured BTFs already.
The six-dimensional BTF itself can be interpreted as a RGB-texture that varies with light and view direction. A high-quality sampling of this function consisting of 256x256 texels in size and 81x81 poses for light and viewing direction contains more than 1.2GB of data. Even with today's most powerful graphics hardware real-time BTF rendering via linear interpolation of the measured data is rather intractable, especially since VEs typically consist of several materials. Thus, some kind of lossy compression has to be employed that achieves high accuracy in real-time. In the following subsections, we will briefly review the existing techniques and emphasize their differences.

3.1 – Lafortune Lobes (LAF)

Fitting analytic functions to spatially varying materials has been done by several publications. The least complex model was suggested by McAllister et al. [24] and is directly based on the Lafortune [6] model. Following their approach, the BTF is evaluated as follows:

$$BTF(\boldsymbol{x},\boldsymbol{v},\boldsymbol{l}) \approx \rho_d(\boldsymbol{x}) + \sum_{i=1}^{k} \rho_{s,i}(\boldsymbol{x}) \cdot s_i(\boldsymbol{x},\tilde{\boldsymbol{v}},\tilde{\boldsymbol{l}}) \qquad (2)$$

where ρ_d and ρ_s denote diffuse and specular albedo, $\tilde{\boldsymbol{l}}$ and $\tilde{\boldsymbol{v}}$ denote the light and view direction transformed to the surface point's local coordinate system, and k is the number of Lafortune lobes

$$s(\boldsymbol{x},\tilde{\boldsymbol{v}},\tilde{\boldsymbol{l}}) = \left(\tilde{\boldsymbol{v}}^t \cdot \begin{bmatrix} C_x(\boldsymbol{x}) & 0 & 0 \\ 0 & C_y(\boldsymbol{x}) & 0 \\ 0 & 0 & C_z(\boldsymbol{x}) \end{bmatrix} \cdot \tilde{\boldsymbol{l}} \right)^{n(\boldsymbol{x})} \qquad (3)$$

fitted to the surface point $\boldsymbol{x}$.
The model requires very few parameters to be stored per pixel resulting in a very compact material representation and can be rendered efficiently in real-time employing vertex- and pixel-shaders that are nowadays available on standard PC graphics boards. The Lafortune lobes model the variance of luminance of the surface point while the diffuse and specular albedos are stored as RGB color values.

3.2 – Scaled Lafortune Lobes (SLAF)

Already one year earlier, Daubert et al. [25] proposed a slightly more complicated material representation, which is also based on the Lafortune model. Following their proposal, the BTF is evaluated as follows:

$$BTF(\boldsymbol{x},\boldsymbol{v},\boldsymbol{l}) \approx T(\boldsymbol{x},\tilde{\boldsymbol{v}}) \cdot \left(\rho_d(\boldsymbol{x}) + \sum_{i=1}^{k} s_i(\boldsymbol{x},\tilde{\boldsymbol{v}},\tilde{\boldsymbol{l}}) \right) \qquad (4)$$

where $T(\boldsymbol{x},\boldsymbol{v})$ represents a lookup table storing view-dependent scaling factors.[1]
Since the lookup-table is defined per pixel, significantly more parameters have to be stored for this model but real-time rendering employing graphics hardware is still possible. Originally, the model was intended to independently represent the three channels of the RGB model by fitting individual Lafortune lobes and lookup-tables for each channel. In our comparison, we will limit this model to luminance calculations, which requires the introduction of an additional specular albedo per lobe (like in the LAF approach).

3.3 – Reflectance Fields (RF)

Recently, Meseth et al. [26] published an approach to BTF rendering 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. Following their proposal, the BTF is evaluated as follows:

$$BTF(\boldsymbol{x},\boldsymbol{v},\boldsymbol{l}) \approx \sum_{v \in N(\tilde{\boldsymbol{v}})} w_v(\boldsymbol{x}) \cdot RF_v(\boldsymbol{x},\tilde{\boldsymbol{l}}) \qquad (4)$$

Here, $N(\tilde{\boldsymbol{v}})$ denotes the set of closest view directions (which is a subset of the measured view directions), w_v denotes the interpolation weight, and RF is the reflectance field which is approximated by

$$RF_v(\boldsymbol{x},\tilde{\boldsymbol{l}}) \approx \rho_d(\boldsymbol{x}) + \rho_v(\boldsymbol{x},\tilde{\boldsymbol{l}}) \cdot \sum_{i=1}^{k} \left(\begin{bmatrix} a_{v,i}(\boldsymbol{x}) \\ b_{v,i}(\boldsymbol{x}) \\ c_{v,i}(\boldsymbol{x}) \end{bmatrix} \cdot \tilde{\boldsymbol{l}} \right)^{n_{v,i}(\boldsymbol{x})} \qquad (5)$$

with the summed term being similar to the Lafortune lobes but excluding exitant direction and k being the number of lobes. Please note that we use a slightly extended version of

[1] Please note that the term $\tilde{l}_z$ from the original definition is included in formula 1 as $(\boldsymbol{n}\cdot\boldsymbol{l})$ already.

the reflectance fields which contains diffuse albedo as well.
Since the reflectance fields are fitted per pixel and measured view direction, the amount of parameters necessary for evaluation of the model is even higher than for the SLAF model but still allows real-time evaluation using the vertex- and pixel-shaders. Like the LAF model, the lobes are intended to compute luminance values that scale the RGB color albedo.

3.4 – Principal Component Analysis (PCA)

Another recent publication in the area of BTF rendering by Sattler et al. [27] proposes a very different approach: instead of fitting analytic functions to the BTF, they decompose 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 cover light- and view-directions not in the measured set. Following their proposal, the BTF is evaluated as follows:

$$BTF(\boldsymbol{x},\boldsymbol{v},\boldsymbol{l}) \approx \sum_{\substack{v\in N(\tilde{\boldsymbol{v}})\\ l\in N(\tilde{\boldsymbol{l}})}} w_{v,l}(\boldsymbol{x}) \cdot \sum_{i=1}^{c} \alpha_i(v,l) \cdot E_i^{Tex}(\boldsymbol{x},v) \qquad (6)$$

where - as before - $N(\tilde{\boldsymbol{v}})$ denotes the set of closest view directions, $N(\tilde{\boldsymbol{l}})$ denotes the set of closest light directions, $w_{v,l}$ denotes the interpolation weight, and α_i is the weight for the i-th Eigen-Texture E_i^{Tex}.
Depending on the number of Eigen-Textures to be used, the number of parameters to be stored per pixel easily exceeds the number of parameters for the reflectance field based model. Real-time rendering is made possible by a combination of CPU and GPU computations.

		LAF	SLAF	RF	PCA	LPCA
Proposte	avg	0.0999	0.0825	0.0745	0.0150	0.0293
	min	0.0655	0.0545	0.0494	0.0090	0.0160
	max	0.1180	0.1044	0.0934	0.0237	0.0648
knitted Wool	avg	0.0833	0.0736	0.0590	0.0164	0.0235
	min	0.0607	0.0568	0.0464	0.0097	0.0127
	max	0.1158	0.0967	0.0745	0.0237	0.0359
Wallpaper	avg	0.0630	0.0593	0.0459	0.0158	0.0222
	min	0.0386	0.0378	0.0325	0.0089	0.0110
	max	0.1020	0.0928	0.0682	0.0315	0.0430
Stone	avg	0.0954	0.0853	0.0642	0.0166	0.0197
	min	0.0392	0.0354	0.0287	0.0086	0.0099
	max	0.1892	0.1814	0.1264	0.0434	0.0410
Alu-minium	avg	0.0572	0.0558	0.0479	0.0142	0.0109
	min	0.0391	0.0389	0.0324	0.0065	0.0062
	max	0.0935	0.0889	0.0782	0.0236	0.0176
synth. Leather	avg	0.0499	0.0489	0.0392	0.0163	0.0172
	min	0.0391	0.0388	0.0310	0.0100	0.0092
	max	0.0658	0.0641	0.0529	0.0247	0.0270

Tab. 1: ε for a selection of measured materials as average, minimum and maximum over all texels of the samples. While the linear-decomposition methods achieve good results for all materials, the analytic function fitting methods yield rather bad results for highly diffuse materials like Proposte and knitted Wool or with strong self-shadowing effects like Stone.

3.5 – Local Principal Component Analysis (LPCA)

In contrast to the PCA based method, the approach published by Müller et al. [28] is based on Eigen-BRDFs. The BTF is interpreted as a set of spatially varying BRDFs which is compressed using a combination of vertex quantization and PCA named local PCA [29] yielding clusters of Eigen-BRDFs. Since the Eigen-BRDFs store discrete values for the measured view- and light-directions only, this method requires view- and light-interpolation as well. The BTF is approximated as:

$$BTF(\boldsymbol{x},\boldsymbol{v},\boldsymbol{l}) \approx \sum_{\substack{v\in N(\tilde{\boldsymbol{v}})\\ l\in N(\tilde{\boldsymbol{l}})}} w_{v,l}(\boldsymbol{x}) \cdot \sum_{i=1}^{c} \alpha_{i,m}(\boldsymbol{x}) \cdot E_{i,m}^{Brdf}(v,l) \qquad (7)$$

where both the weights $\alpha_{i,m}$ and Eigen-BRDFs $E_{i,m}^{Brdf}$ depend on the cluster or material index m.
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 achieved employing the vertex- and pixel-shaders.

4- Comparisons and Application Areas

In this section a comparison of the strengths and weaknesses of the available rendering methods will be provided. Depending on our evaluation, we will determine application areas for the different approaches, i.e. we will give recommendations, which specific models should be applied to which kinds of materials assuming characteristics of the VEs and special quality and performance criteria.

4.1 – Approximation Quality

To compare the approximation quality of the different models, we used a simple expression, which measures the average reconstruction error per texel:

$$\varepsilon_{texel}^{M}(\boldsymbol{x}) = \sum_{(v,l)\in\Delta} \frac{\left|BTF(\boldsymbol{x},\boldsymbol{v},\boldsymbol{l}) - M(\boldsymbol{x},\boldsymbol{v},\boldsymbol{l})\right|}{\left|\Delta\right|}$$

Here, Δ denotes the set of discrete measured view and light directions (in our case |Δ|=81x81 as mentioned in section 3), M denotes the corresponding BTF-approximation.
Table 1 enlists ε for some of the measured materials. The numbers show that the lobe based models (we used 2 lobes for the evaluation) yield worse quality than the linear-decomposition based models. The best approximation quality is achieved by the PCA method (16 Eigen-Textures were used per measured view direction), while the LPCA method (32 clusters with 8 Eigen-BRDFs per cluster were used) still achieves good quality. Although the approximation quality of the three different lobe-based models appears very similar, the visual impression is quite different. For most materials, the RF method achieves pleasing results while the SLAF and the LAF method usually fail to reproduce acceptable quality for our sample materials (please consider figures 5 and 6).

Model	Values/Texel	Bytes/Texel
LAF (2 lobes)	14	22
SLAF (2 lobes)	257	508
RF (2 lobes)	894	1785
PCA (16 comp.)	3888	3888
LPCA (32 clust., 8 comp.)	85.9	170.8

Tab. 2: Memory requirements of the different models. For the LPCA model, a 256x256 base texture was assumed.

4.2 – Memory Requirements

A very important factor for the use of BTF rendering methods in VEs that usually contain several materials is the memory consumption of the BTF data. Table 2 provides the per-texel memory requirements listing both the number of required values and the number of bytes required in our implementation (the numbers are different since values either require 8 bit fixed-point or 16 bit floating-point precision).
As expected, the better the approximation quality, the more BTF data is required. Only the LPCA makes a notable difference since the main amount of data required by this method does not directly depend on the number of pixels in the base texture (i.e. the size of the pictures taken during the measurement process of the material) but on the number of clusters for the local PCA. This results in a memory reduction of about 1:10 and even 1:20 compared the RF and PCA methods while achieving nearly similar (PCA) or even better (RF) approximation quality.

4.3 – Run-Time Requirements

All of the above methods can be implemented to work completely on the GPU by employing the vertex- and pixel-shaders: while the vertex-shader programs compute local light- and view-directions, the pixel-shader programs evaluate the respective equation for BTF rendering. In order to provide a future-proof comparison we enlisted the number of required operations in table 3 in addition to an average frame-rate that we measured for our implementation on a NVidia GeForce FX 5900 graphics board (rendering the shirt model from figure 6). The poor frame rates strongly result from the OpenGL drivers currently available for Windows systems, which appear not to be optimized to handle large amounts of texture memory – especially if floating point textures are employed. We expect better frame-rates reflecting the numbers from table 3 for upcoming driver versions. Please note that the frame-rates are mainly independent of the polygon count of the rendered models at least for the computationally expensive methods which are completely fill-rate-limited.
In contrast to all other methods that are inherently pixel-based, the PCA method can as well be evaluated in a per-triangle manner rather than in a per-pixel one (see [27]). Therefore it can easily outperform the LPCA method if the rendered models have a small triangle count.

4.4 – Precomputation

Another distinctive feature between the methods is the required amount of preprocessing.
The lobe-based methods demand non-linear optimization in order to fit the analytical model parameters to the measured data.

		LAF	SLAF	RF	PCA	LPCA
fixed	ALU	5	34	43	5	$\lceil k/4 \rceil+6$
	TEX	$l+2$	$l+7$	$3l+6$	2	$\lceil k/4 \rceil+3$
per light	ALU	$5l+6$	$5l+6$	$12l+12$	$3k+3\lceil k/2 \rceil+18\lceil k/4 \rceil+11$	$7\lceil k/2 \rceil+41$
	TEX	-	-	-	$3k+9\lceil k/4 \rceil+5$	$9k+11$
typical	ALU	21	50	79	160	56
	TEX	4	9	12	91	88
	fps	100	50	25	6(12)	11

Tab. 3: Number of pixel-shader operations for the different BTF rendering algorithms. For every algorithm, the fixed number of operations and the additional number per light-source are given. The operations are divided into texture lookups (TEX) and other operations (ALU). l is the number of lobes, k is the number of PCA components. The final line gives a typical frame rate for a single light source. For the PCA method, the number in parenthesis denotes the frame rate for per-triangle evaluation (instead of per-pixel).

This task is usually performed by a Levenberg-Marquardt algorithm implying long fitting times (several hours) while the results heavily depend on an appropriate initialization. It also prevents raising the quality of the models by using more than 2-3 lobes, since fitting times become impractical. Furthermore, we sometimes experienced pixel-artifacts resulting from a divergent fit in under-sampled regions. Super-sampling alleviates the problem but increases fitting time even further.
The LPCA method also involves non-linear minimization but the employed generalized Lloyd-algorithm turned out to be very robust and even does not require the full dataset for training. Depending on the number of clusters the fitting times for the LPCA method range from one to two hours.
Since the fitting procedure of the pure PCA approach is totally linear, it takes only a few minutes and does not depend on any kind of initialization.

4.5 – Application Areas

As demonstrated, the presented methods differ significantly in approximation quality and consumption of computing power. In the following we give a short summary for each method in order to identify application areas:

LAF/SLAF

Both models are well suited for speed-critical applications that use many different materials, since their memory and run-time requirements are very low. In cases of materials with simple reflectance behavior (e.g. highly specular like the Aluminium in figure 3) and approximately flat surface structure, the quality of rendered images will be very convincing. For more complex materials which are commonly used in interior design, the approximation quality is too bad.

RF

The reflectance field based method mainly suffers from its high memory requirements, the run-time requirements are still acceptable for real-time applications. The approximation

quality is sufficient for nearly all materials (see figures 5 and 6), but many will still show an artificial touch, especially since resulting images may appear too crisp due to the limited modeling probabilities of Lafortune lobes.
The approximation quality and run-time behavior render this technique appropriate for high-quality VR styling reviews such as interior design. Users will have to cope with the pixel errors described in [26] and will need to implement some memory reduction technique if materials with low-frequency material structure are used.

PCA

This method mostly suffers from its tremendous memory requirements, making the method suitable only for applications with very few materials (like virtual try-on of cloths). The technique achieves photorealistic results if a high number of PCA components is employed, yet the number of components can be adjusted based on the complexity of the material, the desired approximation quality and rendering speed. Combining the technique with image-based lighting leads to realistic results as experienced in the real world (compare figure 4).
The triangle based implementation described in [27] increases the frame rates but is limited to models with low polygon count (up to some thousand triangles).

LPCA

The LPCA method is the most generally applicable one since its rendering speed, memory requirements and approximation quality are adjustable by choosing appropriate numbers of clusters and components. Photorealistic results can be achieved. Nevertheless, the run-time requirements are currently too high for real-time virtual reality applications. Yet, it can be used to either enhance the appearance of a restricted number of virtual objects or to easily generate high-quality images for scenarios that require no real-time performance. Additionally, we expect this method to achieve real-time performance in combination with future graphics hardware.

5- Integration into OpenSG

In this section we will describe how the BTF rendering methods can be integrated into the open source scene graph system OpenSG [30]. The platform independent scene graph system is based on OpenGL and was especially designed to support multithreading, multi-pipeline and multi-machine rendering. Its clear design makes it very easy to include the newest features of graphics boards into it.
The basic control element for rendering in OpenSG is the Chunk, which is used to control the OpenGL state. An implementation of BTF rendering would especially use the subclasses TextureChunk, CubeTextureChunk, VertexProgramChunk and FragmentProgramChunk. The first two ones encapsulate the various texture formats offered by OpenGL (2D, 3D, cube) whereas the second two encapsulate vertex- and fragment-programs that get executed by modern graphics boards.
Since most of the above BTF rendering algorithms at least partially rely on highly accurate data, a few new classes need to be implemented that are currently not supported by the above standard features of OpenSG. OpenGL offers high precision by providing 16 bit and 32 bit floating point values to be stored in textures. OpenSG textures are rather tightly bound to the Image class, which unfortunately does not support floating point formats explicitly so far. Therefore, either the Image class needs to be extended or a new class FloatImage needs to be implemented. Additionally, loaders have to be implemented for the floating point file formats in which the BTF data is stored. Finally, a new texture class has to be implemented that encapsulates rectangular textures. These textures allow more efficient storage by allowing arbitrary heights and widths and – for NVidia based graphics boards – are currently the only way to access floating point valued textures.
The higher level primitive which would be used to finally encapsulate the BTF rendering is the Material. For every BTF rendering algorithm presented above, the ChunkMaterial class should be employed. At initialization time, the class gets handed the correct textures, fragment- and vertex program as Chunks. For implementation details concerning the texture formats and the fragment and vertex programs, we refer to the respective papers, where the methods were introduced ([24],[25],[26],[27],[28]).

Fig 4: If combined with image based lighting and correct shadows, BTF rendering via the PCA method leads to very realistic results.

6- Conclusions and Future Work

In this work we presented several BTF rendering methods for VEs. We compared these methods concerning approximation quality, run-time, memory, and preprocessing requirements and identified application areas for the different techniques. A possible integration of BTF rendering into a scene-graph system has also been proposed.
For BTF rendering, essentially two modeling paradigms have been identified: analytic function modeling and linear basis decompositions. Whilst due to their relatively moderate hardware requirements analytic modeling approaches have the capability of extending current systems by more sophisticated material representations, we strongly believe that statistical analysis of the large BTF dataset as done for

instance by the PCA will be an essential part of future BTF rendering methods, since only such an analysis can account for the large amount of redundancy in both the angular and spatial domain of a typical BTF.
A promising approach could be the combination of classical BRDF rendering such as [13] and LPCA regarding the special properties of the resulting Eigen-BRDFs.
An open problem of BTF-rendering in general is the availability of an encompassing material library. Therefore an interesting aspect of future research will be the generation of novel BTFs from a limited set of "base"-Materials.

7- Acknowledgements

This work was partially funded by the European Union under the project RealReflect (IST-2001-34744) and the German Ministry of Education and Science (BMBF) under the project Virtual Try-On (01IRA01A).

8- Bibliography

[1] Phong T. Illumination for computer generated pictures. In Communications of the ACM, 18(6), 311-317, 1975

[2] Blinn J. Simulation of wrinkled surfaces. In Computer Graphics (SIGGRAPH '77 Proceedings), 286-292, 1978

[3] Heidrich W. and Seidel H. Realistic, Hardware accelerated Shading and Lighting. In SIGGRAPH 1999, 171-178, 1999

[4] Dana K., van Ginneken B., Nayra S. and Koenderink J. Reflectance and Texture of Real World Surfaces. In IEEE Conference on Computer Vision and Pattern Recognition, 151-157, 1997

[5] Ward G. Measuring and modeling anisotropic reflection. In SIGGRAPH 1992, 265-272, 1992

[6] Lafortune E., Foo S., Torrance K., and Greenberg D. Non-linear approximation of reflectance functions. In SIGGRAPH 1997, 117-126, 1997

[7] Kautz J. and McCool M. Interactive Rendering with Arbitrary BRDFs using Separable Approximations. In Tenth Eurographics Workshop on Rendering, 281-292, 1999

[8] McCool M., Ang J. and Ahmad A. Homomorphic factorization of BRDFs for high-performance rendering. In SIGGRAPH 2001, 171-178, 2001

[9] Ramamoorthi R. and Hanrahan P. Frequency space environment map rendering. In SIGGRAPH 2002, 517-526, 2002

[10] Sloan P., Kautz J. and Snyder J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In SIGGRAPH 2002, 527-536, 2002

[11] Latta L. and Kolb A. Homomorphic factorization of BRDF-based lighting computation. In SIGGRAPH 2002, 509-516, 2002

[12] Lehtinen J. and Kautz J. Matrix Radiance Transfer. In Symposium on Interactive 3D Graphics 2003, 57-64, 2003

[13] Sloan P., Hall J., Hart J. and Snyder J. Clustered Principal Components for Precomputed Radiance Transfer. In SIGGRAPH 2003, 382-391, 2003

[14] Debevec P., Hawkins T., Tchou C., Duiker H., Sarokin W. and Sagar M. Acquiring the reflectance field of a human face. In SIGGRAPH 2000, 145-156, 2000

[15] Malzbender T., Gelb D. and Wolters H. Polynomial texture maps. In SIGGRAPH 2001, 519-528, 2001

[16] Ashikhmin M. and Shirley P. Steerable illumination textures. In ACM Transactions on Graphics, 21(1), 1-19, 2002

[17] Levoy M. and Hanrahan P. Light Field Rendering. In SIGGRAPH 1996, 31-42, 1996

[18] Gortler S., Grzeszczuk R., Szeliski R. and Cohen M. The Lumigraph. In SIGGRAPH 1996. 43-54, 1996

[19] Miller G., Rubin S. and Ponceleon D. Lazy Decompression of Surface Light Fields for Precomputed Global Illumination. In 9th Eurographics Workshop on Rendering, 281-292, June 1998

[20] Wood D., Azuma D., Aldinger K., Curless B., Duchamp T., Salesin D. and Stuetzle W. Surface Light Fields for 3D Photography. In SIGGRAPH 2000, 287-296, 2000

[21] Chen W., Bouguet J., Chu M. and Grzeszczuk R. Light field mapping: Efficient Representation and Hardware Rendering of Surface Light Fields. In SIGGRAPH 2002, 447-456, 2002

[22] Kautz J. and Seidel H. Towards Interactive Bump Mapping with Anisotropic Shift-Variant BRDFs. In SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware, 51-58, 2000

[23] Suykens F., vom Berge K., Lagae A. and Dutré P. Interactive rendering of bidirectional texture functions. In Eurographics 2003, 463-472, 2003

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

[25] Daubert K., Lensch H., Heidrich W. and Seidel H. Efficient Cloth Modeling and Rendering. In 12th Eurographics Workshop on Rendering, 63-70, 2001

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

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

[28] Müller G., Meseth J. and Klein R. Compression and Real-Time Rendering of Measured BTFs using Local PCA. To appear in Vision, Modeling and Visualization 2003.

[29] Kambhatla N. and Leen T. Dimension Reduction by Local PCA. In Neural Computation, 9 (7), 1493-1516, 1997

[30] www.opensg.org

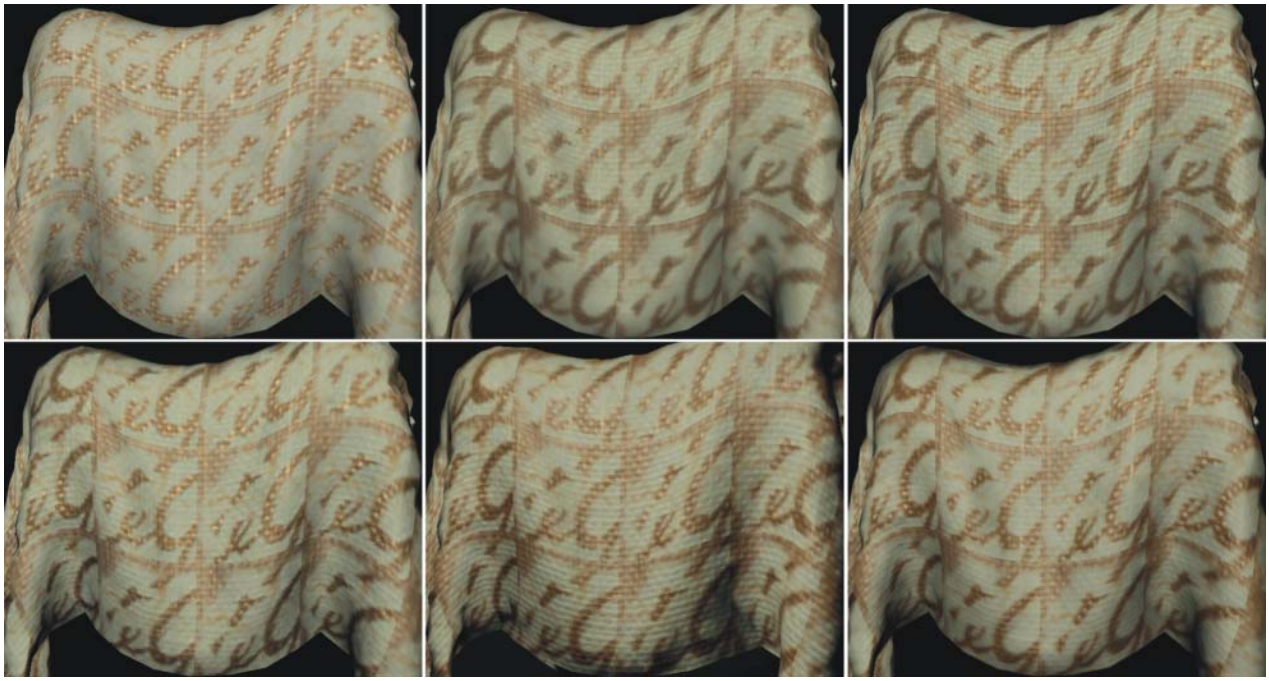

Fig.5: Results from the different rendering algorithms for a piece of cloth covered with wallpaper. From left to right, top first: lit texture, LAF model, SLAF model, RF model, PCA model, LPCA model.

Fig.6: Results from the different rendering algorithms for a shirt made of knitted wool. From left to right: lit texture, LAF model, SLAF model, RF model, PCA model, LPCA model.