OpenSG Symposium (2003)
D. Reiners (Editor)

Preserving Realism in real-time Rendering of Bidirectional Texture Functions

Jan Meseth and Gero Müller and Reinhard Klein

Computer Graphics Group, Bonn University, 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 is currently not feasible, approximations of the six-dimensional BTF are used such that the amount of data is reduced and current graphics hardware can be exploited. While existing methods work well for materials with low depth variation, realism is lost if the depth variation grows. In this paper we analyze this problem and devise a new real-time rendering method, which provides significant improvements with respect to realism for such highly structured materials without sacrificing the general applicability and speed of previous algorithms. We combine our approach with texture synthesis methods to drastically reduce the texture memory requirements and demonstrate the capabilities of our new rendering method with several examples.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

Realistic rendering of real-world objects incorporates complex geometric models and sophisticated modelling of the object's surface reflectance behavior. In the field of real-time rendering, for many years the latter task has been covered by the well-known Phong-Model [19] because of its simplicity and computational efficiency. The (lambertian) diffuse term of the model was allowed to vary spatially via texture mapping. Due to the dramatic development of rendering hardware in the last few years, it became possible to render surfaces using more enhanced and physically plausible approximations like the Lafortune [11] or the Ashikhmin model [1] and even arbitrary bi-directional reflectance distribution functions (BRDFs) [8] in real-time. Since the BRDF captures only the physical reflectance behavior of a uniform surface element, Dana et al. introduced the bidirectional texture function (BTF) which can roughly be interpreted as a tabulated BRDF-per-texel representation. Due to the pure size of a BTF (hundreds of megabytes) real-time rendering of the full data is currently not feasible. Hence approximations of the 6D-BTF are used.

Existing approaches to real-time BTF rendering assume common BRDF models on a per-texel basis. Implementing low-parameter but expressive models, this results in an extraordinary data compression as recently shown by McAllister et al. [17]. Although this kind of approach seems to work well for materials with low-depth variation, it leads to unsatisfying results if the depth variation grows. In fact, one experiences a significant loss of 3D-structure of the surface and therefore a loss of realism in the visualization of highly structured surfaces.

As main contributions of this paper, we first provide an *in-depth analysis* of fitting existing reflectance functions to BTF datasets. Based on the results and further observations, we devise a *new fitting function* that approximates a BTF by a set of reflectance fields. We show that this approach results in drastically reduced fitting errors and thus significantly improves the visual quality of rendered images, at the expense of slightly higher texture memory requirements. In addition, we propose a *real-time rendering algorithm* for the fitted data and present a method to *reduce the amount of texture memory* required by our data representation.

The rest of the paper is organized as follows: After reviewing related work in section 2 we analyze the BTFs of highly depth-varying surfaces in greater detail in section 3. In section 4 we present our new BTF approximation. In sec-

tion 5 our hardware-accelerated rendering algorithm and its integration within OpenSG are discussed. In section 6, we describe our approach to texture memory reduction. Several application examples of our method are shown in section 7. Finally, we conclude and describe directions for future research in section 8.

2. Related Work

Truly realistic renderings of real world materials have to simulate the four-dimensional BRDF(**l**,**v**) for every surface point of a material. Such an approach is infeasible given today's computing power and will likely remain in the near future.

Although simple texture and bump map representations lead to impressive results for very simple materials, more complex models are required to simulate the real appearance of natural materials. Early results approximated a single BRDF by a Ward [24] or Lafortune [11] model. In [8] Kautz and McCool approximate the four-dimensional BRDF by a product of two two-dimensional functions $g(\mathbf{l})$ and $h(\mathbf{v})$ which are stored as textures and combined during the rendering step. McCool et al. [18] improved the above method by employing homomorphic factorization, leading to approximations with user-controllable quality features. The above approaches were further improved by [20], [21] and [13], which all enable the BRDF to be lit by image-based lighting while relying on different approximation functions.

In the context of rendering spatially varying materials, the polynomial texture maps (PTM) method by Malzbender et al. [15] proved suitable for rendering scenes under varying lighting conditions, as long as the viewpoint remains fixed. Chen et al. [2] presented a different method for fixed illumination and variable viewpoint based on factorization methods.

Rendering spatially varying materials under varying light and view conditions was made possible with the six-dimensional BTF representation introduced by Dana et al. [3]. Due to the enormous amount of data in a BTF, only few real-time rendering algorithms have been published so far. Kautz and Seidel [9] proposed to factor the pixelwise BRDF - given as factors of simple reflectance models - into two-dimensional functions and storing the values in textures that are evaluated with hardware supported operations and dependent texture lookups. Unfortunately, their rendering algorithm yields unsatisfying results for more complex reflectance models which are not easily separable. In an approach similar to [21] Kautz et al. [10] rendered spatially varying BRDFs by simply employing higher-dimensional look-up tables. McAllister et al. [17] published a method that approximates the BTF by pixelwise Lafortune models, which can efficiently be evaluated in current graphics hardware. Daubert et al. [4] use a similar approach in rendering synthetic cloth BTFs, but additionally modulate the pixelwise Lafortune models with a view-dependent factor in order to cope with occlusion effects.

3. BTFs of Highly Depth-Varying Surfaces

The BTF can be defined as 2D RGB-texture that varies with 4D 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. Thus, some kind of lossy compression has to be used

If the per-texel data is assumed to exhibit a BRDF-like behavior, one can simply apply every known BRDF model/approximation, e.g. spherical harmonics [27], spherical wavelets [12] or analytical models like generalized cosine lobes [11, 1] to name a few, to each texel independently. The demand of real time rendering rules out linear-basis decompositions, since they tend to use too many coefficients in modelling specularities or may not yet be implementable in hardware. Unfortunately, analytical models are not always suitable either, especially in the case of rough and highly depth varying surfaces, for the following reasons:

- Forcing reciprocity - a basic assumption in analytic BRDF models - leads to significant errors, since the BTF of a rough surface captures also asymmetric effects like subsurface scattering. The measurement process and the required rectification of the reflectance images introduce additional asymmetry (see also section 4).
- The physical motivation of analytical models relies on the fact that the micro geometry of the surface can be modelled with a certain distribution of microfacets (see e.g. [1]), which is modulated by a shadowing and a Fresnel term. The (implicit) shadowing term of simple cosine-lobe models is insufficient to express the discontinuous shadowing and masking effects occurring on rough surfaces (see figure 1). In particular, they are not capable of modelling the view-dependent effect of changing perceived normals which is experienced for materials with high depth variation (see figure 2).

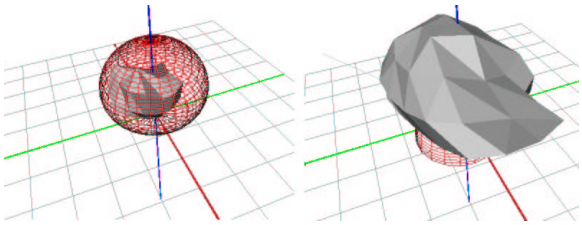

Figure 1: *Spherical plot of a knitted wool BTF texel for two different view directions (solid surface: measured data, red wireframe: two-lobe Lafortune-fit). The model is not expressive enough to model the discontinuous changes in the data and is bound to some average non-directional lobe.*

Since neither linear interpolation of measured values nor fitting of simple BRDF-style models can achieve both high

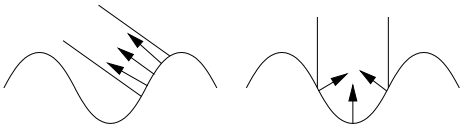

Figure 2: *Changing perceived normals. The perceived normal is averaged over all normals in the area of the beam passing through a pixel on the screen. For varying view direction, completely different normals are perceived.*

quality and real-time rendering, we propose in the following a combination of the memory inefficient but high quality linear interpolation strategy for rendering the full BTF data and the efficient yet low-quality fitting strategy.

3.1. 4D BTF-Slices

Fixing the incident direction $\mathbf{l}$ of a measured BTF dataset, we arrive at a 4D function called the *surface light field*:

$$LF_{\mathbf{l}}(\mathbf{x},\mathbf{v}) := BTF(\mathbf{x},\mathbf{l},\mathbf{v})$$

Surface light fields have been used in the task of 3D-photography, enabling the rendering of novel views of real-world objects under a complex but fixed illumination [28, 2].

Otherwise, fixing exitant direction $\mathbf{v}$, the resulting function is the patch's *surface reflectance field*:

$$RF_{\mathbf{v}}(\mathbf{x},\mathbf{l}) := BTF(\mathbf{x},\mathbf{l},\mathbf{v})$$

As expected, reflectance fields are used in rendering of real world objects under a fixed view with arbitrary illumination [7] and allow very compact representations [15] in the case of mainly diffuse reflection. Matusik et al. captured a set of sparse reflectance fields of an object enabling rendering of arbitrary views under arbitrary (but low-frequency) illumination [16].

Now we propose to implement BTF rendering as mapping and rendering a discrete set of discrete light fields or reflectance fields of the measured surface onto arbitrary geometry. Employing either light- or reflectance fields, the color of a BTF-textured surface element with texture coordinate $\mathbf{x}$ given local light and view direction $(\mathbf{l},\mathbf{v})$ can be computed as follows:

- Approximate the BTF by a set of independently fit light/reflectance fields.
- Compute the color of $\mathbf{x}$ according to every fitted light/reflectance field and interpolate the final color from the partial results.

4. Reflectance Field BTF Approximation

During our research, we tested both reflectance field and light field approximation and we found that the surface reflectance field is better suited for approximation in our approach. This can be explained by inherent BTF asymmetry, which will be discussed in the following subsection.

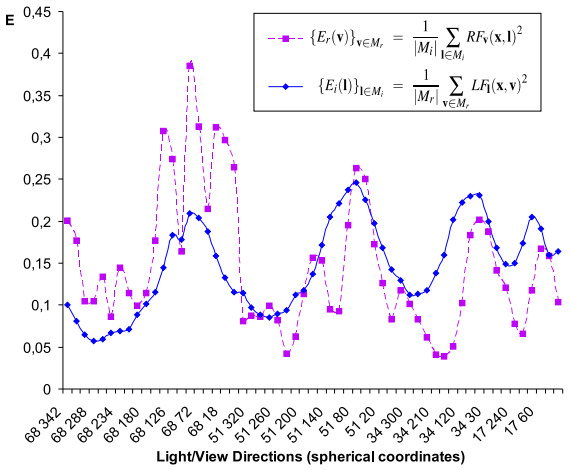

Figure 3: *Plot of the E_r, E_i series for one texel of the corduroy dataset.*

4.1. BTF Asymmetry

Figure 3 depicts a plot of the energy contained in a single texel in each light- or reflectance field of the corduroy BTF, where M_i, M_r denote the sets of measured light/view directions respectively. Note, that both series would be identical in the case of a reciprocal BRDF at the texel (and $M_i = M_r$). In our case, E_r exhibits much more discontinuities (arising from depth variation) than E_i especially for grazing angles. There are two main reasons for that:

- Interreflections and subsurface-scattering from neighboring texels and the fact that it is not possible to build a purely directional light source for the measurement process let the light act like a low-pass filter on the surface structure. In contrast, the small aperture of the camera provides a fairly directional view direction.
- Due to the rectification process the images of a light field are convolved with different, view-dependent smoothing kernels of increasing size. This increases the light-integration domain even further.

We conclude, that changes in light direction are smoother than changes in view direction. This was also noted by Malzbender et al.[15] in the case of mainly diffuse surfaces. Therefore the reflectance field is better suited for fitting by compact functional representations (e.g. polynomials) than the light field. The discontinuous view-dependence will be preserved by our approach, since the piecewise linear function as induced by the linear interpolation captures this high-frequency content of the data.

4.2. A Non-Linear Reflectance Field Approximation

An implementation of our approach can now be obtained by applying view interpolation and a suitable reflectance field approximation. This fit should be efficiently renderable on today's consumer graphics hardware and minimize the approximation error. At a first glance, PTMs appear to be a suitable candidate, since their hardware implementation is straight-forward. But the use of PTMs has a few shortcomings concerning realism:

- Specular peaks and hard shadows are blurred significantly (as mentioned in [15]).
- The polynomial doesn't fade away for grazing incident angles (compare figures 4 and 5).

Instead we propose the following non-linear approximation for the reflectance field:

$$RF_{\mathbf{v}}(\mathbf{x},\mathbf{l}) \approx \sum_{i=1}^{k} s_{\mathbf{v},i}(\mathbf{x},\mathbf{l}) = \sum_{i=1}^{k} \left\langle \begin{pmatrix} a_{v,i}(\mathbf{x}) \\ b_{v,i}(\mathbf{x}) \\ c_{v,i}(\mathbf{x}) \end{pmatrix}, \mathbf{l} \right\rangle^{n_{v,i}(\mathbf{x})} \quad (1)$$

with $s_{\mathbf{v},i}(\mathbf{x},\mathbf{l})$ similar to a Lafortune lobe discarding the exitant direction. $\langle\ ,\ \rangle$ denotes the scalar product. This model is well suited for fitting specularities and directional diffuse lobes. The parameter k controls the number of lobes. We have found $k = 2$ as being sufficient for satisfying results (see table 1 and figure 5 for comparisons of fitting errors, and figure 4 for results of the 2-lobe fit). Since the fit is performed on the luminance data, the final color is computed as in [15]. A Levenberg-Marquardt algorithm is used for the fitting[11, 17, 14]. Convergence is improved via detecting principal directions with a high-pass filter and using the recovered directions as an initialization for the optimization.

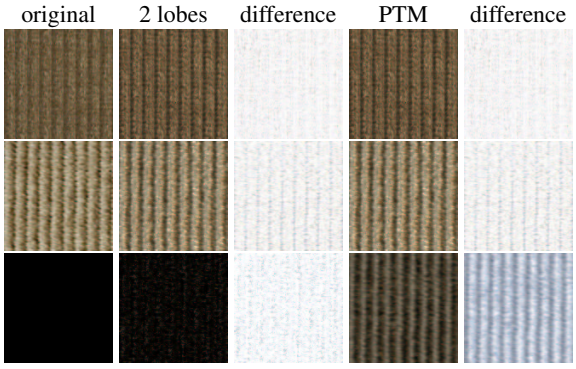

Figure 4: *Comparison of the fitting methods: results for different view directions (bottom: $\theta_l = 90^\circ$). While our 2-lobe model generates some specular highlights, the PTM shows the grazing angle problem.*

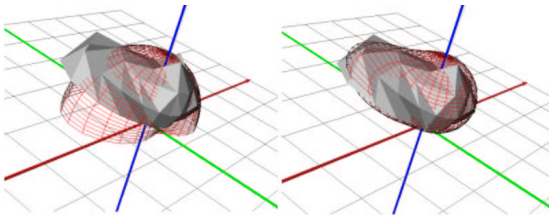

Figure 5: *Spherical plot of a reflectance field and the fit (In red wireframe. Left: PTM, Right: Non-linear, 2 lobes) for one texel. Note the undesirable behavior of the PTM at grazing angles and how well the non-linear method fits the directional lobe.*

Data set	Lafortune(2 lobes)	PTM	Non-linear(k=2)
curdoroy	0.11658	0.06176	0.04176
wool	0.08201	0.05183	0.03334
Proposte	0.12033	0.07459	0.06549

Table 1: *Average luminance difference between data and model. Applying our reflectance field approximation, the error is significantly reduced compared to fitting a single Lafortune model to the whole data set.*

5. Real-Time Rendering

Rendering BTF approximations in real-time still imposes a challenge both on the rendering method and the graphics hardware used for visualization. The advent of graphics processing units (GPUs) implementing the *Pixel Shader 2.0* specification opened up new possibilities for pixelwise real-time rendering by introducing new operations and increasing the amount of operations that can be processed per pixel. An additional, huge improvement for the quality of real-time rendered images was the introduction of a full floating point pixel pipeline, which omits frequent rounding and clamping errors in the processing stage.

In the first following subsection, we will describe the formula that our hardware-accelerated real-time rendering algorithm has to evaluate. Then we describe our data representation and present the rendering step for a single reflectance field. Afterwards, we introduce our view-interpolation scheme and point out its implications on the full rendering step. Finally, we describe how we combined our rendering method with the open source scene graph OpenSG.

5.1. Rendering Equation

The task of the rendering algorithm is to evaluate, for each surface point $\mathbf{x}$, the following formula:

$$L_r(\mathbf{x},\mathbf{v}) = \int_{\Omega_i} f_{r,x}(\mathbf{v},\mathbf{l}) L_i(\mathbf{l}) (\mathbf{n}\cdot\mathbf{l})\, d\mathbf{l} \quad (2)$$

where $f_{r,x}$ is the BRDF in point $\mathbf{x}$, L_i is the incoming radiance, Ω_i is the incident hemisphere over the surface point $\mathbf{x}$, $\mathbf{n}$ is the surface normal and L_r is the exitant radiance.

In the presence of a finite number of point light sources only, the integral reduces to a sum. Substituting $f_{r,x}$ by our reflectance fields approximation from equation 1 and interpolating the view direction, we obtain the final equation:

$$\begin{aligned} L_r(\mathbf{x},\mathbf{v}) &= \sum_{\mathbf{l}\in L}\left(\sum_{v\in N(\mathbf{v})} w_v(\mathbf{v})RF_v(\mathbf{x},\mathbf{l})\right)L_i(\mathbf{l})(\mathbf{n}\cdot\mathbf{l}) \\ &= \sum_{v\in N(\mathbf{v})} w_v(\mathbf{v})\sum_{\mathbf{l}\in L}\left(RF_v(\mathbf{x},\mathbf{l})L_i(\mathbf{l})(\mathbf{n}\cdot\mathbf{l})\right) \end{aligned} \quad (3)$$

where L represents the set of light sources, $N(\mathbf{v})$ denotes the index set of view directions from the measured BTF data that are neighboring $\mathbf{v}$, and w_v denotes the weight for the reflectance field RF_v. Please note that the current view direction $\mathbf{v}$ denotes a different entity than the index v to the view directions from the measured data.

5.2. Data Representation

Since the evaluation of our rendering equation requires the lookup of several, reflectance field dependent values, we store them in textures that can be accessed by the pixel shaders. In order to lookup the weights w_v for the specific reflectance fields given a view direction $\mathbf{v}$, we utilize cube maps storing pairs of weights and identifiers for the reflectance fields.

The parameters $a_{v,k}$, $b_{v,k}$, $c_{v,k}$, and $n_{v,k}$ that determine the shape of the k-th lobe for a given reflectance field RF_v are stored in 2D textures. Gathering all those 2D textures for a predefined lobe k over all reflectance fields and stacking them, we arrive at a 3D texture. Using this 3D texture representation, we can access all reflectance fields in the pixel shader program by binding a single texture only. In an analogous way, we store the average per pixel color per reflectance field in a 3D texture with the fourth texture component remaining for an additional alpha component.

Since the evaluation of our lobes requires the view vector to be transformed into a local coordinate system, we precompute these transformation matrices for the triangles, interpolate them at vertices and pass them as multitexturing coordinates. The mesh is compiled into a display list.

5.3. Rendering a Single Reflectance Field

Our rendering equation is evaluated completely in the pixel shaders. First the view-vector and the light-directions (either point or directional light sources) are transformed into the local coordinate system interpolated from the local coordinate systems specified per vertex. Next, we determine the identifiers of the closest reflectance fields for the current pixel by using the view direction as index into the above mentioned cube maps. By assigning the identifier value to the z coordinate of our current 2D texture coordinates, we get 3D texture coordinates which can be used to lookup the color and parameters for the surface point from the 3D textures. We evaluate the pixel's luminance following equation 1 and additionally check for every light source, whether the angle between the triangle's normal and the light direction is greater than 90°, in which case we neglect the contribution of this light source. After summing up the contributions, we multiply with the surface point's average color to get the final output of the fragment shader.

5.4. Reflectance Field Interpolation

One main task that remains for rendering is the computation of the weights $w_v(\mathbf{v})$ and the selection of the number of reflectance fields to interpolate from. We found that a simple scheme interpolating the four closest measured view directions yields excellent results.

As mentioned above, we utilize cube maps to lookup closest measured view directions and their interpolation weights. For every pixel (i.e. view direction) in the cube map, we first transform the view vector from cartesian to spherical coordinates (θ_v, ϕ_v). We then choose the closest θ values from the existing measurement data. Next, for each such θ, we determine the closest ϕ values. These four samples determine the four closest measured view directions and their reflectance fields. The according weights w_v are computed via bilinear interpolation (please note that the measured samples are non-uniformly spread). Two pairs (w_v, v) are stored in each cube map, resulting in a total number of two cube maps for storing the four closest entities.

This implies that we have to repeat the above rendering for a single reflectance field four times. During the first step, we compute the pixel's color according to the first and second of the four closest reflectance fields, multiply the resulting luminances by the average colors for the reflectance field and the weights w_v, and add them. In the second step, the pixel's color according to the third and fourth closest reflectance field are computed and added to the previous result, either by a direct sum or by blending if multi-pass rendering has to be employed. Since our interpolation scheme guarantees the weights to accumulate to one and since the weights change smoothly with varying directions, the result looks very convincing.

5.5. Rendering within OpenSG

Integrating our rendering algorithm into OpenSG unfortunately poses various problems on the developer, since neither multipass-rendering nor floating-point image formats are currently supported. Despite these challenges, we decided to use it as the rendering environment since this way we can easily utilize the already implemented features of the

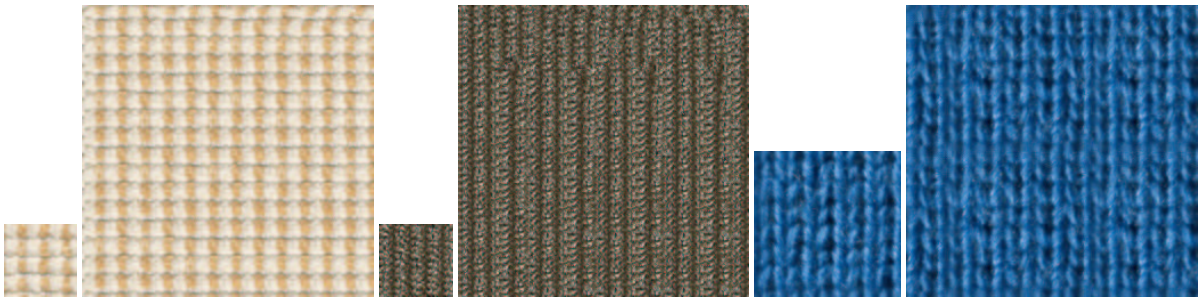

Figure 6: *Results from texture synthesis (left: Proposte, middle: corduroy, right: knitted wool). The small images represent exemplary views of the samples, the large images the according views from the synthesized images.*

scene graph system. In order to implement BTF rendering, we derived several classes from the OSGStateChunk, one for each fitting method that we tested. Since the OSGImage class does not support floating-point valued images so far, we extended our chunks by the ability, to read their own file format (the number of parameters is different every time) and to manage the textures by themselves. In addition, the chunks are used to load and manage the cube map data.

To get around the multi-pass rendering problem, we derived a class from OSGDrawableMaterial, which switches the state between the two different rendering passes. Unfortunately, this becomes only possible by handling the display lists of the BTF textured models ourselves. The second task of this class is to compute the local texture coordinate systems for every triangle of the model and to specify these as multi-texture coordinates in the display lists.

6. Memory Reduction

Due to the frequent use of 3D texture maps and floating-point entries in the texture slices, our rendering method turns out to be rather memory consuming (about 400 MB for a 256×256 BTF with 81 reflectance fields). In addition, the BTF renderer has to cope with the problem of applying materials of large extent in the presence of small samples only without introducing neither noticeable cracks or seams nor obvious repetitions.

The solution for both problems are texture synthesis algorithms, which generate textures of arbitrary size similar in appearance to example textures, which are provided as input. Several such algorithms were published in the past, most for synthesis of flat 2D textures (e.g. [25, 6, 30]), others directly synthesize on geometric models (e.g. [23, 26, 29]). Recently, Tong et al. [22] published a specialized texture synthesis algorithm for BTF data, which we used in our implementation.

In their analysis step, the BTF is considered a 2D texture with entries of dimension n, where n equals the number of different view- and lighting conditions during the measurement process. To reduce this huge amount of data, the m most significant view- and lighting conditions are determined and only their values are used for future computations. With a k-means clustering algorithm, r so called textons are computed that represent the cluster centers. Similarities among these textons are precomputed. Every entry in the 2D texture representing the BTF is finally assigned its closest texton.

The synthesis step presented originally directly works on arbitrary objects, but for our purpose, a planar synthesis is sufficient. Like in other texture synthesis algorithms, a new texture is generated by starting with a random texton and subsequent addition of textons that fit their already synthesized neighborhood. Other than texture synthesis algorithms, Tong's algorithm does not copy texton values from the example to the new texture but instead stores texture coordinates referencing the example texture. This indirection necessitates an additional dependent texture lookup during rendering but saves huge amounts of texture memory especially for large BTF textures.

For the materials we measured, we found that even using small samples only, the overall structure and appearance was well preserved in the synthesized BTF textures (Proposte and corduroy require samples of size 64^2 while the higher structured knitted wool requires 96^2). Figure 6 shows synthesis examples for the three different materials. While the algorithm achieved good results for knitted wool and Proposte, the random component of the algorithm destroys the uniformity of the corduroy sample which contains a single, uniform orientation direction. Even the hierarchical approach from [22] is not powerful enough to solve this problem. Since the underlying algorithm used to generate the synthesis result does not directly influence our rendering method, we still expect our approach to texture memory reduction to be useful for all kinds of materials.

The additional indirection from the synthesized texture reduces the memory requirements per material to about 25 MB (24 MB for a 64^2 BTF base texture including 81 different reflection fields and floating-point precision values; less than 1 MB for a 256^2 index texture from the texture synthesis step).

7. Results

We implemented our fitting framework on an Intel Pentium IV 1.4 GHz machine with 512 MB RAM. Fitting times for the different methods varied from few minutes to several hours, depending on the either linear- or non-linear fitting-algorithm. These times can easily be reduced utilizing multiple computers in parallel. Unfortunately, the results of the fitting procedure sometimes lead to floating point overflows in the graphics hardware during rendering, which is partially due to the specific graphics hardware and partially to the rather high exponents for spatially restricted lobes. These overflows lead to single incorrect pixel colors (either black or white). Using 16 bit floating point values, these problems occurred more frequently.

Our rendering algorithm was implemented using a ATI Radeon 9700 graphics board for visualization, whose limited number of dependent texture lookups requires two separate rendering passes. Figures 7 and 8 show example images of models that were rendered in real-time.

8. Conclusions

In this work, we presented an in-depth analysis of the real-time rendering problem for highly depth varying BTF materials. We demonstrated why existing algorithms have problems with such materials and proposed a new method that achieves high-quality results at the expense of consuming more texture memory than existing real-time methods. To alleviate the memory problem, we showed how our algorithm can easily be combined with existing texture synthesis algorithms to significantly reduce the memory requirements.

Unfortunately, the memory reduction approach is not suitable for materials with high-level structure like newspapers or paintings, which are found frequently in the real world. Nevertheless, virtual environments are usually dominated by rather homogenous materials, for which the approach yields good results.

For future work, we will combine our BTF rendering method with image based lighting, test several different texture synthesis algorithms, try to further reduce the amount of memory required by our method, and fully integrate our method into OpenSG.

Acknowledgements

This work was partially funded by the European Union under the project RealReflect (IST-2001-34744). We want to thank André Nicoll and Marc Boßerhoff for helping with the implementation and Mirko Sattler for fruitful discussions. Special thanks belong to Ralf Sarlette who provided the BTF measurements.

References

1. M. Ashikhmin and P. Shirley. An Anisotropic Phong BRDF Model. *Journal of Graphics Tools: JGT*, **5**(2), pp. 25–32, 2000
2. W.-C. Chen, J.-Y. Bouguet, M. H. Chu, and R. Grzeszczuk. Light field mapping: Efficient Representation and Hardware Rendering of Surface Light Fields. In *SIGGRAPH 2002*, pp. 447–456, 2002
3. K. J. Dana, B. van Ginneken, S. K. Nayra, and J. J. Koenderink. Reflectance and Texture of Real World Surfaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 151–157, 1997
4. 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
5. P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. In *SIGGRAPH 2000*, pp. 145–156, 2000
6. A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH 2001*, pp. 341–346, 2001
7. T. Hawkins, J. Cohen, and P. Debevec. A photometric approach to digitizing cultural artifacts. In *2001 conference on Virtual reality, archeology, and cultural heritage*, pp. 333–342, 2001
8. J. Kautz and M. McCool. Interactive Rendering with Arbitrary BRDFs using Separable Approximations. In *Tenth Eurographics Workshop on Rendering*, pp. 281–292, 1999
9. J. Kautz and H.-P. Seidel. Towards Interactive Bump Mapping with Anisotropic Shift-Variant BRDFs. In *SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware*, pp. 51–58, 2000
10. J. Kautz, P.-P. Sloan and J. Snyder. Fast, Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonic. In *13th Eurographics Workshop on Rendering*, pp. 301–308, 2002
11. E. P. F. Lafortune, S.-C. Foo, K. E. Torrance, and D. P. Greenberg. Non-linear approximation of reflectance functions. In *SIGGRAPH 1997*, pp. 117–126, 1997
12. P. Lalonde and A. Fournier. A Wavelet Representation of Reflectance Functions. *IEEE Transactions on Visualization and Computer Graphics* **3**(4), pp. 329–336, 1997
13. L. Latta and A. Kolb. Homomorphic factorization of BRDF-based lighting computation. In *SIGGRAPH 2002*, pp. 509–516, 2002
14. H. Lensch, M. Goesele, J. Kautz, W. Heidrich, and H.-P. Seidel. Image-Based Reconstruction of Spatially Varying Materials. In *12th Eurographics Workshop on Rendering*, pp. 103–114, 2001
15. T. Malzbender, D. Gelb, and H. Wolters. Polynomial texture maps. In *SIGGRAPH 2001*, pp. 519–528, 2001
16. W. Matusik, H.P. Pfister, A. Ngan, P. Beardsley, R. Ziegler and L. McMillan. Image-Based 3D Photography using Opacity Hulls. In *ACM Transactions on Graphics*, 21(3), pp. 427–437, 2002

Figure 7: *Comparison of our rendering technique (left) with approximated bump-mapping (right). The same light configurations were used in both pictures. Using our technique, the 3D structure of the corduroy material on the car seat appears realistic, while bump-mapping clearly misses the highlights for grazing light angles.*

17. 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

18. M. D. McCool, J. Ang, and A. Ahmad. Homomorphic factorization of BRDFs for high-performance rendering. In *SIGGRAPH 2001*, pp. 171–178, 2001

19. B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, **18**(6), pp. 311–317, 1975

20. R. Ramamoorthi and P. Hanrahan. Frequency space environment map rendering. In *SIGGRAPH 2002*, pp. 517–526, 2002

21. P.-P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH 2002*, pp. 527–536, 2002

22. X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH 2002*, pp. 665–672, 2002

23. G. Turk. Texture synthesis on surfaces. In *SIGGRAPH 2001*, pp. 347–354, 2001

24. G. J. Ward. Measuring and modeling anisotropic reflection. In *SIGGRAPH 1992*, pp. 265–272, 1992

25. L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH 2000*, pp. 479–488, 2000

26. L.-Y. Wei and M. Levoy. Texture Synthesis Over Arbitrary Manifold Surfaces. In *SIGGRAPH 2001*, pp. 355–360, 2001

27. S.H. Westin, J.R. Arvo, and K.E. Torrance. Predicting reflectance functions from complex surfaces. In *SIGGRAPH 1992*, pp. 255–264, 1992

28. D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. H. Salesin, and W. Stuetzle. Surface Light Fields for 3D Photography. In *SIGGRAPH 2000*, pp. 287–296, 2000

29. L. Ying, A. Hertzmann, H. Biermann, and D. Zorin. Texture and Shape Synthesis on Surfaces. In *12th Eurographics Workshop on Rendering*, pp. 301–312, 2001

30. S. Zelinka and M. Garland. Towards Real-Time Texture Synthesis with the Jump Map. In *Eurographics Workshop on Rendering 2002*, 2002

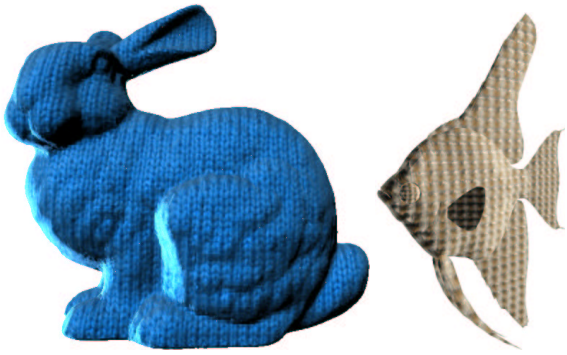

Figure 8: *Examples: left knitted wool, right Proposte. Note the desired flattening effects for grazing viewing angles on the bunny and the specular highlights on the fish, which are both experienced in reality. Flattening effects (see figure 2) occur because the small creases in the material are no longer visible for grazing viewing angles.*