Realistic Materials for Virtual Real-Time Environments

Marc Boßerhoff*
André Nicoll†

Computer Graphics Group
Bonn University
Bonn / Germany

Abstract

Bidirectional Texture Functions (BTF) can be used to render realistic surfaces without the need of modeling details in geometry. For efficient, real-time rendering on current graphics hardware it is necessary to find good approximations for the BTF data, because the full data is too large to be rendered. We will compare two approximations, the Polynomial Texture Map (PTM) and the Per-Pixel-Lafortune-BRDF. To use larger samples, we use texture synthesis methods based on BTF analysis with the BTF samples and compare the results obtained with different settings and materials. We combine these approaches with a rendering method that provides real-time rendering on large surfaces, but does only need little more memory than for the small samples.

Keywords: Image Based Rendering, BTF, real-time rendering, texture synthesis, appearance

1 Introduction

The objective in virtual environments is to create a most realistic representation of reality. To achieve this, we can model the complete geometry as good as possible and use raytracing-methods to simulate the radiation-exchange of all the surfaces. Due to restrictions of graphics hardware, this approach is not possible in real-time hardware in the moment and triangle based approaches use too much triangles to be rendered in real-time hardware.

To achieve the ability to render models in real-time, we must approximate geometry in less detail, but then we loose much of the highly detailed reflection information of realistic surfaces. To restore as much information as possible, we use a representation for the surface materials, which is not very memory-consuming, and can be computed in graphics hardware in real-time.

To use the capabilities of the latest generation graphics hardware, we decided to implement the Polynomial Texture Map [10] and the Per-Pixel-Lafortune-BRDF [6], to verify their suitability for realistic real-time environments. This means, that they must be efficient to compute in graphics-hardware and have a small visual error, compared to the original surface.

To gather the reflection information of real surfaces we use an image-based way. We take shots of real surfaces for a great set of varying light- and view-directions which are evenly spread over the hemisphere, at least 6000 images. We use our automated photo laboratory for this. The material is mounted on a movable robot and the light-source and the camera rotate around the robot. This procedure takes at least 12 hours to complete. These stacks of images are used as BTFs to fit the models for one view slot to represent a reflectance field.

Another problem with BTF samples is that they are taken from real world samples with limited sizes and that large samples need huge amounts of memory. Simple tiling of the measured samples usually leads to visible borders, so we decided to try texture synthesis algorithms to overcome this problem. For synthesizing BTFs instead of textures, we found the approach of Tong et al. [15] useful who use a BTF analysis before the synthesis.

We will show that we can use the combination of both ideas for a rendering algorithm that provides real-time rendering of large approximated BTFs. We also found that we can take advantage of the synthesized BTFs to render BTFs much larger than the samples, but only need additional memory for a small coordinate map.

2 Related Work

2.1 Representation

For rendering realistic materials we have to simulate the four-dimensional BRDF(ω_i, ω_e) for every point on the surface. But such a truly realistic simulation will exceed the capacities of modern graphics hardware.

With very simple geometries, texture- and bumpmapping yield good results for simple materials, but for more complex materials we will need the ability to change the appearance for varying light and view conditions. Early approaches simulated a single BRDF for the whole material (Ward [16] or Lafortune [6]). Kautz and McCool [4] approximated the BRDF by two functions, whose results are stored in textures and were combined by the graphics hardware. McCool [12] improved this method by homo-

*bosserho@cs.uni-bonn.de
†nicoll@cs.uni-bonn.de

morphic factorization, to give the user more control over the quality. These methods where improved by [13], [14] and [7] to BRDFs, lit by prefiltered environment maps, but their models are currently not capable for real-time rendering of BTFs.

For fixed viewpoint the polynomial texture map (PTM) by Malzbender et.al. [10] is very suitable for varying light conditions and can be easily computed in hardware, a reason why we implemented this method. Chen et al. [2] presented methods for surface light fields based on factorisation methods.

As the 6-dimensional BTF was introduced by Dana et al. [3], it was possible to render materials under varying light and view conditions. Due to the big amount of data in a BTF, it is needed to compress the data in specific models. But there are only few published. Kautz and Seidel [5] proposed a factorisation of the pixel wise BRDF in two dimensional functions whose coefficients are stored in textures to be computed with hardware supported operations and dependent texture lookups. McAllister et al. [11] published an approach with pixel wise Lafortune-BRDFs to approximate the BTF. This approach is perfect to reach the capacities of current graphics hardware, so we decided to implement this model as well.

2.2 BTF Synthesis

Several algorithms for 2D-texture and BTF synthesis were published in the past. A basic idea for texture synthesis using fast search for useful pixels in the sample was given by Wei and Levoy. [17]. They also give an approach to synthesize textures directly on surfaces. [18]

Their basic idea for synthesizing on surfaces was used by Tong et al. [15] for synthesizing BTFs on surfaces. For fast search for candidate pixels, they used the k-coherence synthesis as an extension of the Ashikhmin's synthesis algorithm for natural textures [1] instead of the methods proposed by Wei and Levoy. The initial BTF analysis using textons is described in [8].

A method for very fast texture synthesis is the Jump Map [19] that uses a random selection between precalculated candidate pixels.

Wei and Levoy also introduce a hierarchical approach for texture synthesis that uses all previously synthesized layers for calculating the distance function [17], but Tong et al. use a different approach that uses the pixels of the next lower resolution layer as preset to the output of the current layer. [15]

3 Representation

3.1 Polynomial Texture Map

The Polynomial Texture Map assumes a fixed front-view of the material-covered surface and thus requires less images, approximately 60 to 80. To approximate the reflection function of the surface, it uses a biquadratic polynomial whose coefficients are fitted per texel :

$$L(x,y,l_x,l_y) = a_0(x,y)l_x^2 + a_1(x,y)l_y^2 \\ +a_2(x,y)l_xl_y + a_3(x,y)l_x + a_4(x,y)l_y + a_5$$

L is the resultant luminance or color-value and l_x, l_y are the projection of the light-vector into the local coordinate system of the texture. It is possible to fit two different types of PTMs. We can either fit each color channel separately to keep the effect of changing colors for varying light directions or we can assume, that the color will be nearly constant under varying light directions and fit only the luminance value and modulate it with a base-color.

$$R(x,y) = L(x,y) * R_{base}(x,y)$$
$$G(x,y) = L(x,y) * G_{base}(x,y)$$
$$B(x,y) = L(x,y) * B_{base}(x,y)$$

The RGB-Fit will need 18 coefficients, 6 per color-channel. The luminance-fit will come along with 9 coefficients, 6 for the luminance and 3 for the base-color. The fitting algorithm uses a singular value decomposition (SVD) to solve the following system of equations, which leads to the minimal least squares error. The SVD can be computed once and be applied per pixel.

$$\begin{bmatrix} l_{x_0}^2 & l_{y_0}^2 & l_{x_0}l_{y_0} & l_{x_0} & l_{y_0} & 1 \\ l_{x_1}^2 & l_{y_1}^2 & l_{x_1}l_{y_1} & l_{x_1} & l_{y_1} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{x_N}^2 & l_{y_N}^2 & l_{x_N}l_{y_N} & l_{x_N} & l_{y_N} & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_5 \end{bmatrix} = \begin{bmatrix} L_0 \\ L_1 \\ \vdots \\ L_N \end{bmatrix}$$

$L_0, \ldots, L_N$ is the color- or luminance-data which is measured per pixel of varying light directions. l_{x0}, l_{y0} is the projection for the first light-direction to the local texture coordinate system, l_{x1}, l_{y1} the projection of the second and so on. The model is very simple and not capable of approximating highly specular materials and hard shadows, but yields good results for nearly diffuse materials. The reason is the polynomial model which can only contain one specularity due to mathematic restrictions and the low number of coefficients. As we are using images to fit the model, effects like self-shadowing (only soft shadows), sub-surface scattering and interreflections are preserved due to the characteristics of the material.

3.2 Lafortune-BTF

Another approach is the Lafortune-Model which was introduced by Lafortune et al. [6]. It is a generalization of the cosine lobe model [9] and is usable for varying light- and view-directions. The reflection-function for the model is written as follows :

$$f_r(u,v) = \sum_i^M [C_{x,i}u_xv_x + C_{y,i}u_yv_y + C_{z,i}u_zv_z]^{n_i}$$

(u_x, u_y, u_z) and (v_x, v_y, v_z) are projections of the light- and view-direction into the local texture coordinate system. $C_{x_i}, C_{y_i}, C_{z_i}, n_i$ are the parameter which must be fitted where n_i is the specular exponent and M is the number of lobes. This model is much more flexible than the PTM, as it allows to increase the number of lobes to the needs of a material, to get a good fit with a small error. If a material is more specular the number of lobes will increase and exceed the limits of actual hardware, so we must limit the maximal number of lobes.

Like the PTM, we can also fit two models, a RGB- and a luminance-model, doing the same modulation with a base-color like the PTM. As we have no linear equation system to solve, a non-linear approximation is needed to fit the measured BTFs to the Lafortune-model. We use the Levenberg-Marquardt-method as it is easy to use for fitting arbitrary functions and it is perfect for our needs. Using the RGB-model we have 24 coefficients, 8 per color channel with 2 lobes per channel. The luminance-model does not support color changes of the surface for varying light- and view-directions but allows much more lobes (up to 7), to approximate more specular materials with less error. In contrast to the PTM the Lafortune-BTF supports effects like specularity at grazing angles, off-specular reflection, retro-reflection and anisotropy.

4 BTF Synthesis

The BTF synthesis consists of two steps. First, the BTF sample is analyzed using k-means clustering to increase the synthesis speed and reduce memory usage. The synthesis then uses the data from the previous step to generate a larger BTF. Tong et al. used their k-coherence synthesis directly on surfaces [15]. We used their k-coherence synthesis, however we only needed to synthesize a 2D-BTF and do not synthesize directly on a surface. This simplified our implementation.

We also tried the Jump-Map synthesis proposed in [19] to produce fast synthesis results, but with our materials the results show too many visible artifacts. Thus, we will only mention the common map with precalculated nearest neighbors that can be used for both methods.

4.1 Analysis

A BTF can be regarded as 2D-texture where each pixel (x, y) is not a color value, but approximately a four dimensional BRDF $(\theta_v, \phi_v, \theta_l, \phi_l)$. In a sampled BTF, one has a number of images for each light and view direction. These stacked images form the BTF sample and give a high dimensional vector for each pixel. We used approximately 6000 images for our BTF samples. For the synthesis, a distance function between the BRDFs of two pixels must be found.

Tong et al. suggest the application of 3D-texton analysis to a BTF sample [15]. Each pixel of each image is filtered with n Gaussian filters. So the dimension of each vector is multiplied by n. However, we omitted the filters because the results for our materials are much better without these filters and we save memory and computation time. As figure 2 shows, the application of filters destroys the fine structures of the BTF.

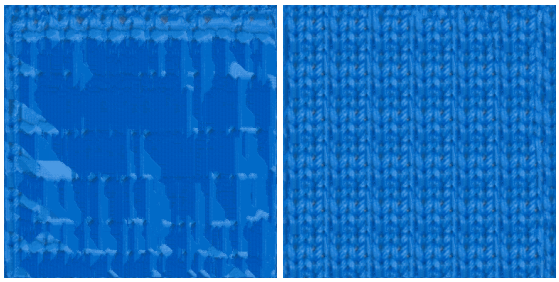


Figure 2: Synthesis results with filters (left) and without filters (right).

To decrease the number of vectors, we apply k-means clustering to the vectors. This algorithm assigns the index (called texton) t_i of a cluster center to each pixel and stores the mean vector (called appearance vector) v_i for each center. The dot products then are calculated only between the appearance vectors. We usually used between 1000 and 2000 cluster centers.

Tong et al. define the dot product between two textons as the dot product of their appearance vectors: $t_i \cdot t_j := v_i \cdot v_j$. The squared euclidean distance between two pixels p_i, p_j with assigned texton indices t_i, t_j now can be calculated as

$$dist(p_i, p_j) = \|t_i - t_j\|^2 = t_i \cdot t_i - 2 \cdot t_i \cdot t_j + t_j \cdot t_j$$

To increase synthesis speed, we precalculate the dot product between each pair of textons and store them into a matrix M.

To further decrease computation time in the analysis step, one can select a few hundred representative images with the Gaussian filters from above to calculate a vector for each image and cluster these images [15]. The final synthesis results are only slightly worse if they are different at all, and the time for analysis is reduced. For example, the precalculation for a 64x64 sample takes about an hour if all BTF images are used, but we need only about 10 minutes for selection of 150 images *and* precalculation on this selection.

Our BTF analysis can be summarized as follows:

- Optional: Select representative images.
- Apply k-means clustering to the BRDF vectors of the pixels.
- Calculate the dot products between the appearance vectors from clustering.

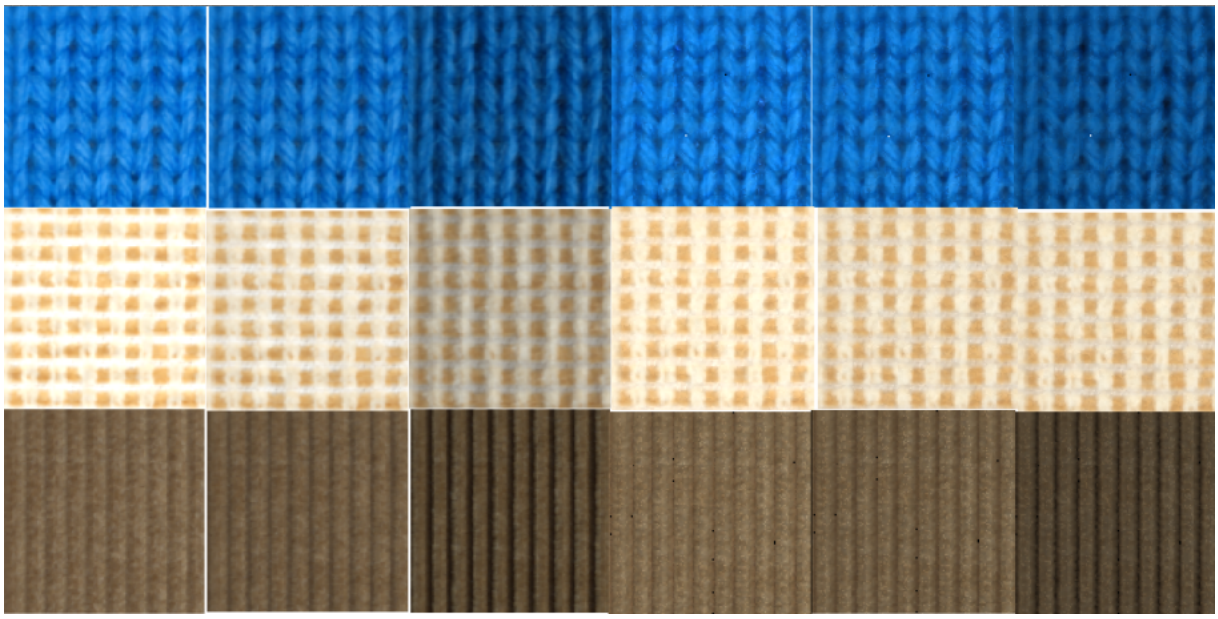


Figure 1: Fitting results for spherical angles (0,0), (25,0), (50,0). Left 3 images are the PTM models and the right 3 images are the Lafortune-BTF.

4.2 k-coherence synthesis

Suppose, one has already synthesized a part of the output. Now, to synthesize the next pixel p_0 in scanline order, we want to find pixel s_0 in the input sample that is the best match for a new pixel. To find this pixel, we use a set of pixels $N(p_0) = \{p_1, ..., p_n\}$ in the neighborhood of p_0 to find a neighborhood $N(s_0) = \{s_1, ..., s_n\}$ of s_0 in the input sample with minimal distance between $N(p_0)$ and $N(s_0)$. The border of the output is initialized with random pixels from the sample.

In order to avoid time consuming search in the complete sample for each output pixel, we used the k-coherence synthesis [15] and also tried the Jump Map [19]. The idea of both methods is to find, for each pixel in the sample, a number of (for example 10) pixels with the most similar neighborhoods. For small samples up to 128x128 this can be done by simple brute force search in the sample using the following algorithm:

For each pixel p with neighborhood $N(p)$ do

- For each pixel q with neighborhood $N(q)$ calculate $dist(N(p), N(q))$ and store the coordinates of the m best pixels $q_1, ...q_m$.
- To support the Jump Map: Calculate a probability for each $q_1, ...q_m$ depending on the distance. [19]

With these m nearest neighbors for each pixel, a small candidate set for the best pixel can be built during the synthesis. We search only these candidates for the best match. The number of candidates depends on the value of k and the neighborhood size:

- $k = 1$: Each pixel $p \in N(p_0)$ has a texture coordinate in the sample. The neighborhoods of these pixels $s_1, ..., s_n$ at these coordinates form the candidate set. [1]
- $2 \leq k \leq m$: For each pixel $s_1, ..., s_n$ found in the sample as in $k = 1$, we add $k - 1$ nearest neighbors to the candidate set.

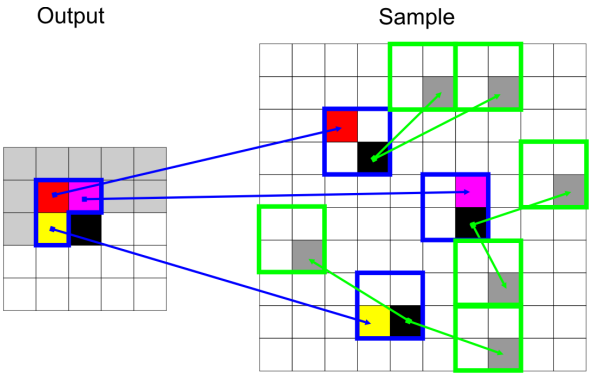


Figure 3: Candidate set for k-coherence synthesis. The blue areas are the candidates for $k = 1$. For $k = 3$ for each of these candidates the two most similar neighborhoods are added as additional candidates (green).

We haven chosen a data independent implementation for the synthesis algorithm, so we can use it for both BTF and texture synthesis. The sample data must provide a distance function between two pixels that are selected using their coordinates. This works, because the synthesis result contains only values from the original sample.

Hierarchical synthesis To improve the synthesis results, especially to better preserve the structure of a ma-

terial, we have implemented a hierarchical synthesis. For each image of the sample BTF an image pyramid is build, where each layer has half the resolution of the previous layer. We perform the analysis on each layer independent from the other layers. For the synthesis, the output is also an image pyramid with the same number of layers as the input. The first layer is synthesized without modification.

For the next layers, we use the information from the layer with the next lower resolution: To one quarter of the pixels in the current layer, we assign the sample texture coordinates of the corresponding pixel in the previous output layer, multiplied by 2 because of the lower resolution. Because the lower resolution sample was created by bilinear interpolation, we will find a similar pixel at these coordinates in the current layer. These pixels now will be omitted in following synthesis. After the synthesis of the missing pixels, the synthesis is again performed on the previously omitted pixels to reduce visible errors. Our method is a 2D adaptation of the hierarchical synthesis on surfaces by Tong et al.

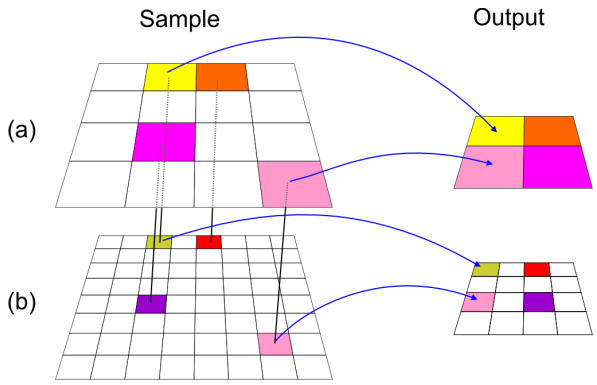


Figure 4: Hierarchical Synthesis with two layers. (a) Lower resolution after synthesis. The pixels of the output have texture coordinates in the sample. (b) Higher resolution before synthesis. The texture coordinates from the lower layer are transformed to the higher resolution and used as already synthesized pixels.

Using large BTFs for rendering One purpose of our BTF synthesis was to have large BTFs for rendering a scene. We take advantage of the fact, that the used synthesis method does not generate pixels with new values, but used only pixels from the small sample to generate the result. Thus, the key for saving memory is to store only the coordinates to the BTF sample in a coordinate map as synthesis result instead of copying the large vectors from the pixels of the BTF sample to the output. So we can use large BTFs without needing several gigabytes of memory. Of course, for rendering an additional level of indirection is necessary.

Results The results of our synthesis depend on the structure of the input and on the values for k and the neighborhood size. Larger neighborhoods, higher neighborhoods, and hierarchical synthesis generally improve the results. However, for materials with irregular structures, for example many natural materials as in [1], lower values for k can be better. Figure 6 shows the results for proposte with different settings for these values.

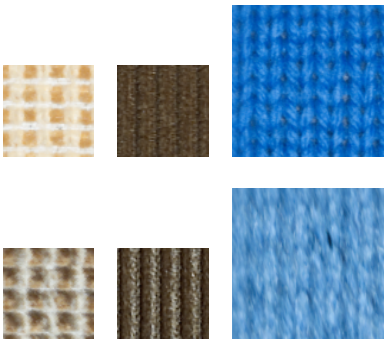


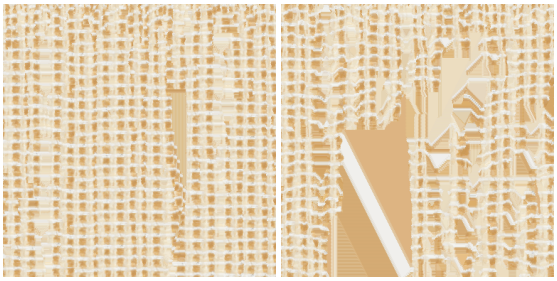
Figure 5: Samples we used for our results. From left to right: Proposte 64x64, Corduroy 64x64, Knitted wool 128x128. First row: view and light vector orthogonal to the plane. Second row: view and light from small angles above the plane.

For materials with very complex structures like knitted wool the synthesis results are currently not very good for certain light and view angles, but we obtained good results with more simple materials. One problem that could lead to the problems with small viewing angles is that the original sample images must be taken from small angles and have in one direction a very small resolution that is only interpolated to a higher resolution. This is hardly taken into account by the distance function and might lead to the visible sample borders. Figure 7 shows synthesis results with different materials and light and view angles.

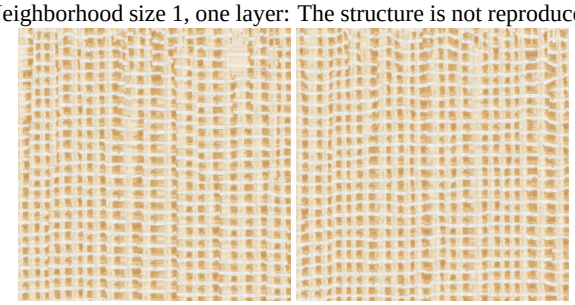
Size	k	Neighborhood size	Time
256x256	1	2	1.9 s
256x256	3	2	2.7 s
256x256	6	2	3.7 s
256x256	6	3	13.6 s
512x512	6	2	13.4 s

Table 1: Synthesis time with 3 layers.

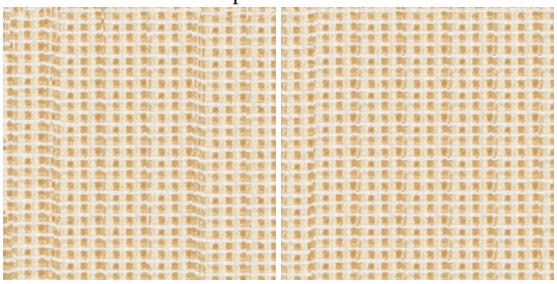
Table 1 shows the time for the BTF synthesis with three layers with different output sizes and values for k without the time for calculating the texton map. Due to the precalcualtion of the dot products, the sample resolution and the number of selected BTF images have no effect on the synthesis time. The timings where obtained on a Pentium 4, 2.4 GHz without special code optimizations for Pentium 4.



Neighborhood size 1, one layer: The structure is not reproduced.

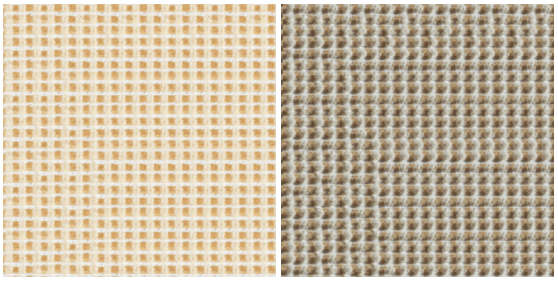


Neighborhood size 2, one layer: Visible errors in the reproduction.

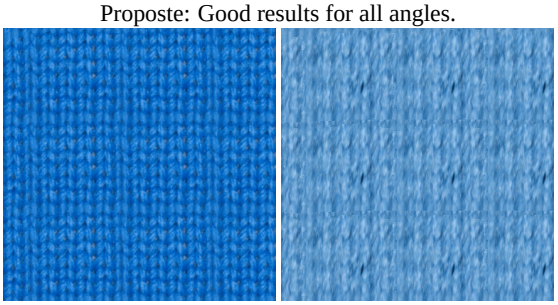


Neighborhood size 2, three layers: Very few errors, especially for $k = 5$.

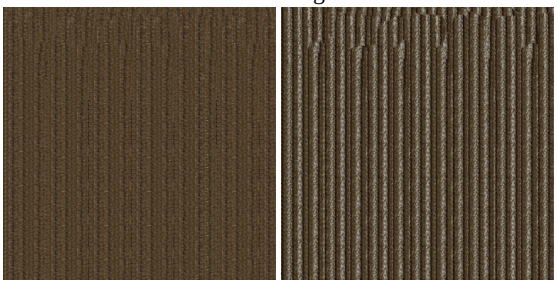
Figure 6: Synthesis results for proposte with different parameters. Left column: $k = 1$. Right column: $k = 5$



Proposte: Good results for all angles.



Knitted wool: Good results for views from top but visible tiling for small angles.



Corduroy: Reproducing the structure causes problems for all views.

Figure 7: Synthesis results for different materials. Left column: view and light vector orthogonal to the plane. Right column: view and light from small angles above the plane.

5 Combination

The combination of the BTF approximation and BTF synthesis yields a method for fast rendering of large BTFs. We use the synthesis to generate a coordinate map from a BTF sample and the PTM- or Lafortune-Model to render the result of the synthesis in real-time. For the synthesis process it is not necessary to know which model we will use for real-time rendering, as the synthesis uses the original BTF sample, but does not use the fitted sample. Also, the PTM- and Lafortune-fitting works directly on the original BTF sample. So the rendering can also be used without the synthesis. Because the synthesis result is a texture coordinate map to the BTF, we will need a texture lookup to render the result. This step can easily be integrated in the rendering step as an additional texture lookup. If we do not use the synthesis, we simply omit this step and do a lookup directly into the PTM- or the Lafortune-BTF. As we can use one texture unit to sample from the coordinate map and the other units to sample from our rendering models, it is possible to render large textures using the small BTF sample with only one additional texture lookup. Therefore the rendering works in real-time.

6 Results

Due to the rendering models we achieved a very good compression of the original BTFs as we are using only one view slot. We got the following results :

The fitting was done on a AMD 1.3 GHz. For rendering we will need the ATI Radeon 9500/9700 series, as they support Pixel Shader 2.0. They support operations to make real-time rendering of the Lafortune model possible. The PTM is possible on earlier graphics hardware like a GeForce2, but it is much easier to implement in the newer, as we have floating-point color registers, need no scaling and biasing and can use the full range of 32-bit floats. This leads to a better quality due to the absence of rounding errors.

As we have no quantitative error measure to compara-

Image size	Type	Memory	Fitting time
256x256	BTF	58 MB	-
256x256	PTM-RGB	6 MB	22 sec
256x256	PTM-Lum	3 MB	15 sec
256x256	LaF.-Lum-2 Lobes	2.8 MB	112 sec

Table 2: Memory consumption of the different models

ble the visual quality, we can describe only our subjective impression. As described the PTM is limited by their polynomial model and can gather only one specularity. But the fitting method with SVD is very stable if we clamp small eigenvalues to zero. The result offers no visual artifacts.

The Lafortune-model in contrast is very scalable due to the fact, that we can add as much lobes as we want (7 if we want real-time single pass rendering) and the sharpness of the specularity can be determined with the exponent in the sum. But we experienced some difficulties with the fitting method. It is very slow and leads to visual artifacts if the sampling grid is not small enough. As we have a stack of images with discreet light and view vectors, the sampling over the hemisphere is often not evenly spread and the Levenberg-Marquardt-method produce visual artifacts. So we have to interpolate between the images to generate more light and view vectors and provider a better sampling grid.

7 Conclusions

BTF approximation functions can be used for realistic real-time rendering in current graphics hardware. While the PTM has problems with many structures (for example hard shadows), the Per-Pixel-Lafortune-BRDF produces better results. Modified approximation functions could lead to even better results.

As we have only one view slot, the limits of our models are not exhausted. The quality of the fitted data is good, as long as we use a single view slot. More view slots will lead to many visual artifacts and exceed the limits of the model.

But it is possible to fit every single view slot, and store the result as a texture stack in a 3d texture and select the appropriate texture for the viewing direction, and render it with the method for a static view direction.

The successful application of texture synthesis to BTFs allows the rendering of large BTFs still in real-time and without memory problems, but the quality of the results with the k-coherence synthesis depends on the used material. However, with better synthesis methods, for example using a distance function better than the euclidean distance, it should be possible to produce good results for more difficult materials, too. As long as a synthesis method is based on copying pixels from the original sample, our coordinate map can be used for memory efficient rendering of large BTFs.

The implementation of our rendering framework allows to implement new methods easily, as it supports many basic functions for model-fitting. It is possible to invent new models and combine them with the texture synthesis for real-time rendering.

References

[1] Michael Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[2] Wei-Chao Chen, Jean-Yves Bouguet, Michael H. Chu, and Radek Grzeszczuk. Light field mapping: efficient representation and hardware rendering of surface light fields. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 447–456. ACM Press, 2002.

[3] Kristin J. Dana, Bram van Ginneken, Shree K. Nayra, and Jan J. Koenderink. Reflectance and texture of real world surfaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 151–157, San Juan, Puerto Rico, June 1997.

[4] Jan Kautz and Michael McCool. Interactive rendering with arbitrary BRDFs using separable approximations. In *Tenth Eurographics Workshop on Rendering*, pages 281–292, 1999.

[5] Jan Kautz and Hans-Peter Seidel. Towards interactive bump mapping with anisotropic shift-variant BRDFs. In *2000 SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware*, pages 51–58, Interlaken, Switzerland, 2000. ACM Press.

[6] Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-linear approximation of reflectance functions. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 117–126. ACM Press/Addison-Wesley Publishing Co., 1997.

[7] Lutz Latta and Andreas Kolb. Homomorphic factorization of BRDF-based lighting computation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 509–516. ACM Press, 2002.

[8] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using 3d textons. pages 29–44, June 2001.

[9] Robert R. Lewis. Making Shaders More Physically Plausible. In *Fourth Eurographics Workshop on Rendering*, number Series EG 93 RW, pages 47–62, Paris, France, 1993.

[10] Tom Malzbender, Dan Gelb, and Hans Wolters. Polynomial texture maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 519–528. ACM Press, 2001.

[11] David K. McAllister, Anselmo Lastra, and Wolfgang Heidrich. Efficient rendering of spatial bi-directional reflectance distribution functions. In *Graphics hardware 2002*, pages 79–88, Saarbrucken, Germany, 2002. Eurographics Association. ISBN:1-58113-580-7.

[12] Michael D. McCool, Jason Ang, and Anis Ahmad. Homomorphic factorization of BRDFs for high-performance rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 171–178. ACM Press, 2001.

[13] Ravi Ramamoorthi and Pat Hanrahan. Frequency space environment map rendering. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 517–526. ACM Press, 2002.

[14] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536. ACM Press, 2002.

[15] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 665–672. ACM Press, 2002.

[16] Gregory J. Ward. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 265–272. ACM Press, 1992.

[17] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000.

[18] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 355–360. ACM Press / ACM SIGGRAPH, 2001.

[19] Steve Zelinka and Michael Garland. Towards real-time texture synthesis with the jump map. In *Eurographics Workshop on Rendering 2002*, Pisa, Italy, June 2002.