Pacific Graphics 2013
B. Levy, X. Tong, and K. Yin
(Guest Editors)

Volume 32 (2013), Number 7

Level-of-Detail Streaming and Rendering using Bidirectional Sparse Virtual Texture Functions

Christopher Schwartz, Roland Ruiters and Reinhard Klein

University of Bonn, Germany

Abstract

Bidirectional Texture Functions (BTFs) are among the highest quality material representations available today and thus well suited whenever an exact reproduction of the appearance of a material or complete object is required. In recent years, BTFs have started to find application in various industrial settings and there is also a growing interest in the cultural heritage domain. BTFs are usually measured from real-world samples and easily consist of tens or hundreds of gigabytes. By using data-driven compression schemes, such as matrix or tensor factorization, a more compact but still faithful representation can be derived. This way, BTFs can be employed for real-time rendering of photo-realistic materials on the GPU. However, scenes containing multiple BTFs or even single objects with high-resolution BTFs easily exceed available GPU memory on today's consumer graphics cards unless quality is drastically reduced by the compression. In this paper, we propose the Bidirectional Sparse Virtual Texture Function, a hierarchical level-of-detail approach for the real-time rendering of large BTFs that requires only a small amount of GPU memory. More importantly, for larger numbers or higher resolutions, the GPU and CPU memory demand grows only marginally and the GPU workload remains constant. For this, we extend the concept of sparse virtual textures by choosing an appropriate prioritization, finding a trade off between factorization components and spatial resolution. Besides GPU memory, the high demand on bandwidth poses a serious limitation for the deployment of conventional BTFs. We show that our proposed representation can be combined with an additional transmission compression and then be employed for streaming the BTF data to the GPU from from local storage media or over the Internet. In combination with the introduced prioritization this allows for the fast visualization of relevant content in the users field of view and a consecutive progressive refinement.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

1. Introduction

The realistic rendering of surface materials plays an important role in the generation of photo-realistic as well as predictive synthetic images. The appearance of real-world materials is often the result of complex light scattering interaction within small geometric structures on and under the surface. For a restricted set of materials, such as perfect mirrors, some metals or plastics, a visually pleasing rendering can be achieved by employing physically motivated analytical reflection models. However, the majority of the rich variety of material classes encountered in everyday life cannot as easily be represented by simple analytical models.

In 1997, Dana et al. [DvGNK97] proposed an image-based approach to acquire the appearance of real-world materials, the Bidirectional Texture Function (BTF). The six

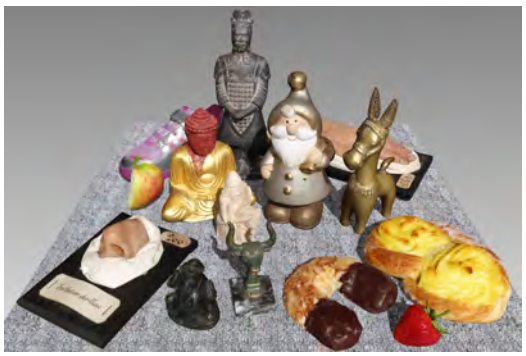


Figure 1: *A scene with* 14 *objects, all textured with BTF materials with up to* 2048×2048 *texels, rendered on the GPU at interactive frame-rates using our BSVTF approach.*

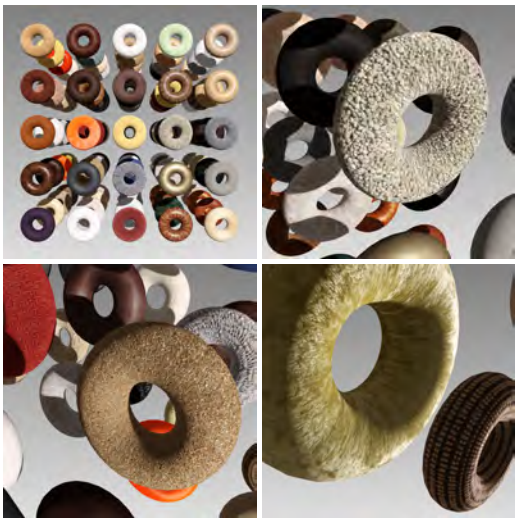


Figure 2: *Screenshots of a fly-through animation in a virtual scene with* 100 *different measured BTF material having a size of* 512×512 *texels. The scene renders with* 35 FPS *at* 1280×720 *pixel on a NVIDIA GeForce GTX 680 GPU. While the total amount of the factorized BTF data is* 6.2 GB*, the memory footprint on the GPU using BSVTFs is* 1.7 GB.

dimensional Bidirectional Texture Function $\rho(\mathbf{x}, \omega_l, \omega_v)$ describes the ratio of differential radiance that is scattered at a point $\mathbf{x}$ on a surface into direction ω_v to differential irradiance from direction ω_l. In contrast to the related Spatially Varying Bidirectional Reflectance Distribution Functions (SVBRDF), which are often represented as either spatially varying parameters of an analytical BRDF or spatially varying mixtures of a set of analytical or measured BRDFs, the BTF is characterized by a unique Apparent BRDF (ABRDF) at each point of the surface. Unlike the BRDF, the ABRDF may violate assumptions such as reciprocity or energy conservation and show a more general distribution of the reflected light. This way, the BTF is able to encode position dependent non-local effects, like interreflections, shadows, masking or subsurface scattering, cast from neighboring geometry onto the material surface.

Because of this property, BTFs are an excellent choice for the representation of many real-world materials. In recent years, BTFs have started to find application in industrial settings and in the domain of cultural heritage, as in these cases the exact reproduction of the appearance of materials or complete objects is desired. Although raw BTFs exhibit rather unhandy file-sizes of several tens to hundreds of Gigabytes – recent publications show high resolution BTFs with up to 500 GB – several compression techniques are available to cope with this problem. Among those, matrix factorization based approaches are capable of reducing the file-size up to a factor of 500 while still preserving the unique appearance of the BTF and allowing rendering in real-time.

Therefore, these factorized BTFs are currently the representation of choice for high quality materials in interactive applications as well as real-time graphics. Unfortunately, so far the usefulness of BTFs in real-time graphics is greatly hampered by the still rather large data sizes of up to several hundreds of megabytes per material. An additional entropy coding or lossy compression can be employed to improve the compression ratio over the factorized BTF, e.g. for the fast transmission over the internet. However, the data needs to be unpacked into the factorized representation again to support efficient random access for real-time rendering. In [SRWK11], using a lossy wavelet compression for transmission, factors of 10 for no perceivable up to 60 for a noticeable but still acceptable error were achieved. The authors reported that a BTF with a compressed size of 46.39 MB for the transmission had then to be stored in 2.5 GB of GPU memory. Therefore, rendering even a small scene containing a few objects with high resolution BTF materials on the GPU is simply impossible due to the high memory requirements that can even be hardly met by the latest professional hardware. When considering the trend towards high quality 3D graphic on tablets and mobile phones, which nowadays have performant graphics chips but a drastic shortage of memory, the problem of GPU memory consumption becomes even more severe. However, the familiar problem of rendering very large textures that exceed the available memory has already been successfully handled by employing a technique that is known as Clipmapping [TMJ98] or *Sparse Virtual Texturing* (SVT) [Bar08]. SVT utilizes a level-of-detail hierarchy in the spatial domain to only keep the required parts of the texture in the necessary resolution in GPU memory.

In this paper, we propose the *Bidirectional Sparse Virtual Texture Function* (BSVTF), an adaption of the SVT technique to the context of real-time BTF rendering. In contrast to plain textures, which only have a level-of-detail hierarchy in their spatial resolution, a factorized BTF representation inherently includes a second level-of-detail domain of the ABRDF approximation quality. In this work, we demonstrate that both level-of-detail hierarchies can be combined in a consistent manner by reducing them to a single spatial level-of-detail problem. We show that this way the bottleneck of GPU memory can effectively be circumvented. In contrast to several tens to hundreds of Megabytes per high-resolution BTF, in our case the CPU and GPU memory demand is very moderate. More importantly, the memory demand grows only marginally with higher resolution and with increasing number of materials only additional storage space for the angular part of the factorized matrix is needed. Furthermore, the computational overhead on the GPU introduced by the approach remains constant regardless of the number of BSVTFs, allowing for rich virtual scenes that are textured with several high-resolution materials, such as the scene shown in Fig. 1. We demonstrate that BSVTFs can also be used for the efficient streaming over the Internet, allowing to display scenes with multiple high resolution materials without considerable delay. For this, we apply an additional streaming compression that utilizes the redundancy found in the level-of-detail hierarchy. To facilitate the fast

start of rendering, we interleave the angular factorization components with the transmission of the level-of-detail tiles of the spatial information.

In summary our contributions are

- A hierarchical level-of-detail approach for memory friendly real-time BTF rendering.
- An automatic weighting of the BTF compression approximation error and the spatial level-of-detail error of the SVT by formulating the approximation problem as a unified error minimization.
- A streaming approach utilizing a transmission compression based on the level-of-detail hierarchy, allowing rendering of scenes with BTF materials transmitted over a network without significant loading times.

2. Related Work

To the best of the authors knowledge, there exists no previous literature on a similar level-of-detail application on BTFs. However, there is a large body of related work in the fields of level-of-detail rendering as well as real-time rendering and streaming of BTFs.

Hierarchical level-of-detail: As early as 1976, Clark introduced the concept of hierarchical level-of-detail on geometric models [Cla76]. Here, the problem of considering only that parts of the geometry of a synthetic scene that are actually relevant for rendering the users viewport is solved by using an object hierarchy. The hierarchy holds the geometry of objects in the scene in different levels of detail. A *graphical working set* is built from the hierarchy by choosing exactly those objects that are visible on the screen in a level of detail that is sufficient for the required rendering resolution of the object. Since then, hierarchical level-of-detail has found a lot of application for scene geometries and terrain visualization and also for streaming these types of data over the Internet. More information on these research topics can be found in [LWC*02].

In real-time graphics, another level-of-detail hierarchy has also found very wide-spread application: in combination with trillinear interpolation, mip-maps of a texture, first introduced in [Wil83], are commonly applied to avoid aliasing artifacts arising from under-sampling textured area. In [TMJ98], Tanner et al. first make use of the mip-map hierarchy to allow for arbitrarily large virtual textures maintaining an active working set, similar to Clark. While Tanner et al. propose the use of a specialized graphics workstations, the concept of virtual texturing has in recent years regained popularity (e.g. [Bar08]) due to the increasing flexibility and general availability of GPUs.

BTF compression, streaming, and rendering: For the task of real-time rendering of BTF materials, a number of different solutions have been proposed. For a comprehensive overview we refer to [HF11]. At their core, almost all approaches have in common that they aim to reduce the huge amount of data in a BTF description to a more compact representation that will eventually fit on the GPU. One approach is to fit SVBRDFs to the BTF data. While this representation is well suited for evaluation on the GPU, the quality can suffer drastically by the reduction to an SVBRDF as the non-local effects of the light scattering in the material are lost. In a recent publication [WDR11], Wu et al. therefore combine a mixture of several fitted SVBRDF models with residual ABRDFs and propose to compress those via vector quantization. A second group of compression techniques is based on factorization. Here, the BTF is considered as a matrix or tensor of which a low-rank approximation is found. Recent comparisons [PSR13] indicate that on BTF data, *Full Matrix Factorization* (FMF) [KM03] often yields the best RMSE for a given compression ratio. The only mentioned exception is a BTF compression scheme based on *K-SVD* [RK09] that outperforms the FMF by a factor of 3 to 4 at comparable quality. However, an efficient real-time rendering technique for this compression has not yet been found. In [GMSK09], Guthe et al. employ a perceptually motivated BTF compression based on matrix factorization. Compression rates of about 500 : 1 are achieved with a high approximation quality. The authors observe that GPU memory can be saved by employing lower downsampled levels-of-details for some of the factorized data. In our paper, we will also save GPU memory by exploiting the fact that lower resolution versions of factorization components can be used. However, instead of reducing the level-of-detail once at compression-time, based on assumptions about viewing distance and angles, we store the factorized BTF data at multiple precomputed levels-of-detail. This allows us to dynamically decide at runtime which level-of-detail is necessary and can thus consider the actual view-point of the user.

Recently, data-driven compression methods for BTFs that are not based on factorization have been proposed as well. In [HF07], the authors follow a statistical modeling approach that achieves impressive compression ratios but in its nature is not capable of exactly reproducing the surface features of a given BTF. While this might be tolerable or even desired for the purpose of texture synthesis, it would for example not be applicable in the case of virtual surrogates for cultural heritage. In [HFM10], Havran et al. employ a compression based on multi-level vector quantization and in [TFLS11] Tsai et al. propose to use a decomposition in multivariate radial basis functions. Both methods provide high quality results for the reproduction of material appearance at real-time frame-rates. Unfortunately, no direct quality comparisons to FMF are given. However, the reported compression ratios are in the same region as achieved with FMF, so it is not to be expected that these techniques will reduce memory demand sufficiently to eliminate the memory issues of BTF rendering. In this paper, we use FMF compression, as it greatly facilitates the simplicity of the proposed progressive streaming and $L2$-norm based error-approximation for tile-prioritization. In future work one might consider the applicability of other compression methods for BSVTFs as well.

In [SRWK11], FMF compression is used for rendering BTFs in the Webbrowser via WebGL on commodity hard-

ware (NVIDIA GeForce 8800). The authors utilize the level-of-detail hierarchy implicated by the factorization to perform a progressive streaming of the BTF data over the internet. They employ an additional lossy image compression for the efficient transmission. However, the image compression does not allow fast random-access reconstruction of the compressed data any more, which is mandatory for the purpose of real-time rendering. Therefore, after transmission, the factorized data is unpacked into GPU memory again, occupying up to 2.5 GB for a single high quality BTF with 4 Megapixel. In contrast, with our proposed technique, scenes that contain several 4 Megapixel BTF materials with comparable compression ratios (660 : 1 in our case versus 428 : 1 in [SRWK11]), such as the one shown in Fig. 1 and 4, can be rendered in real-time with a much lower memory footprint (483 MB and 229 MB for the total scene).

Out-of-core rendering of reflectance data: For their editing system *BTFShop* [KBD07], Kautz et al. proposed an out-of-core rendering architecture for BTFs. For this, the uncompressed BTF data is split into tiles which are successively streamed to memory for editing and rendering. However, BTFShop is an editing application and not ment for real-time viewing purposes. The rendering relies on lazy updates and assumes that usually only a subset of pixels on the screen are changed and light and view directions remain constant. A slight rotation around the object would require to completely swap the cached tiles. This severely restricts the achievable frame-rates and prohibits streaming over a limited bandwidth network connection. In contrast, using the proposed BSVTFs allows changing the light and view directions even for scenes with many BTF materials in real-time with moderate bandwidth requirements.

On the related topic of surface light-field (SLF) rendering, Chen et al. presented the technique of *light field mapping* [CBCG02]. Here, the authors proposed to perform a spatial partition of the object's surface. In combination with factorization, vector quantization and image compression this allowed combining the SLFs for each such spatial part into textures that are suitable for rendering with the GPU. Images are then generated using multi-pass rendering, rasterizing the triangles of one spatial part at a time. While this algorithm allows for memory-friendly out-of-core rendering, it does not include level-of-detail and therefore requires the costly successive swapping of all textures for the visible parts of the object's surface in every frame.

3. Sparse Virtual Texturing

In this section, we briefly discuss the SVT algorithm [Bar08] and introduce our notation and implementation details.

We consider gray-scale images with $M \times N$ pixels depending on their context either as a matrix $\mathbf{I} = \mathbb{R}^{M \times N}$ or as a function $\mathcal{I} : (s,t) \in [0,M) \times [0,N) \subset \mathbb{R}^2 \mapsto i \in \mathbb{R}$, mapping from the continuous pixel-domain $[0,M) \times [0,N) \subset \mathbb{R}^2$ to intensity values in $\mathbb{R}$ by bi-linear interpolation of the matrix entries at the respective discrete rows and columns.

SVT considers the problem of representing a very large image $\mathbf{I}$ using a considerably smaller matrix $\mathbf{C} = \mathbb{R}^{O \times P}$, $O \ll M$ and $P \ll N$ as a cache. The technique exploits the fact that the display resolution itself is usually much smaller than the dimensions of $\mathbf{I}$. For rendering, it is therefore sufficient to hold only those parts of the image in memory, i.e. in the cache $\mathbf{C}$, that are visible on the screen at a given time. Additionally, they parts only have to be held in memory at the screen resolution, which has the additional benefit of avoiding aliasing artifacts due to under-sampling.

To this end, the original image $\mathbf{I}$ is decomposed into a set of disjoint tiles $\mathfrak{T} = \{\mathbf{T_i} \in \mathbb{R}^{T \times T} \subset \mathbf{I} | \forall_{\mathbf{i} \neq \mathbf{j}} \mathbf{T_i} \cap \mathbf{T_j} = \emptyset \wedge \mathbf{I} = \bigcup \mathbf{T_i}\}$ of size T. The tiles $\mathbf{T_i}$ are indexed by a two dimensional multi-index $\mathbf{i}$. More sets of tiles are generated for different resolutions $l = 0, \ldots, L$ by down-sampling the image $\mathbf{I}$, with L referring to the highest resolution, and are then denoted as $\mathfrak{T}_l$. In case portions of the image can sufficiently be represented in a lower resolution l, tiles from the set $\mathfrak{T}_l$ can be used. Note that tiles from this set will allow a larger coverage of the virtual image $\mathbf{I}$ at the same size T. We compute and decompose all down-sampled versions of the original image with resolutions of $\frac{M}{2^{L-l}} \times \frac{N}{2^{L-l}}$, $l = [0,1,...,L]$ until its content can eventually be expressed using the single tile in $\mathfrak{T}_0 = \{\mathbf{T}_{(0,0)}\}$, i.e. $\max(\frac{M}{2^L}, \frac{N}{2^L}) \leq T$. The content of the cache $\mathbf{C}$ is then compiled from that subset of tiles that form the visible part of the image $\mathbf{I}$ at a sufficient resolution. Hence, $\mathbf{C}$ is also referred to as the *tilecache*. If all space in the cache is already occupied on arrival of a new tile, free space will be made available by unloading existing tiles based on their priority (see Section 5.2). Tiles from multiple virtual textures are handled in a single tilecache. In our implementation, we take special care when manipulating the tilecache that at all times all parts of $\mathbf{I}$ are covered at least on a low-resolution level. This strategy prevents drawing errors due to cache misses in case of rapid user interaction.

To determine the information which tiles of which level have to be displayed, a *Feedback-image* $\mathcal{F} : (x,y) \in [0,X) \times [0,Y) \mapsto (\mathbf{i},l,\tau) \in \mathbb{R}^4$ is computed in regular intervals. Let $\Pi : \mathbb{R}^2 \to \mathbb{R}^2$ be a texture-mapping function that maps screen pixel coordinates (x,y) to texel coordinates $(s,t) \in [0,M) \times [0,N)$. For each pixel (x,y), the down-sampling level l and the index $\mathbf{i}$ of the tile $\mathbf{T_i}$ with the content for that pixel can be computed as

$$l = L - \log_2 \max\left(\left\|\frac{\partial \Pi(x,y)}{\partial x}\right\|, \left\|\frac{\partial \Pi(x,y)}{\partial y}\right\|\right) \quad (1)$$

$$\mathbf{i} = \left(\mathrm{mod}\left(\left\lfloor 2^l \frac{s}{M} \right\rfloor, 2^l\right), \mathrm{mod}\left(\left\lfloor 2^l \frac{t}{N} \right\rfloor, 2^l\right)\right)^T. \quad (2)$$

A pixel-shader is used to evaluate Equations 1 and 2. To support for multiple texture-images an additional texture-index τ is stored in the fourth channel.

In order to reassemble the original appearance of $\mathbf{I}$ from the possibly fragmented tiles that might also exhibit different resolutions, an indirection has to be performed for all texture-fetches during rendering. For each screen pixel (x,y), the texel coordinates $(s,t) = \Pi(x,y)$ are mapped to

coordinates in the tile-cache where the value for $\mathcal{I}(s,t)$ is stored. This requires to locate the appropriate tile in the tile-cache and find the correct offset within the tile itself. For this purpose, we maintain a *lookup-table* $\mathcal{L}:\mathbb{N}^2\rightarrow\mathbb{R}^3$ that holds the level l' in which a tile for (s,t) is available in the tile-cache (which might differ from the optimal level l) and the texel coordinates $\mathbf{i}'$ of its top-left corner in $\mathbf{C}$. Please note that $\mathcal{L}$ is considerably smaller than the original texture $\mathbf{I}$, as only one entry for every T^2-th texel is required. From this information, the coordinate $\mathbf{x}$ of the texel in the tilecache that needs to be fetched is computed as

$$\mathbf{x}=\mathbf{i}'+\left(2^{l'}s-M\left\lfloor 2^{l'}\frac{s}{M}\right\rfloor,2^{l'}t-N\left\lfloor 2^{l'}\frac{t}{N}\right\rfloor\right)^T. \quad (3)$$

We employ a separate lookup-table for each virtual texture.

4. BTF Real-time Rendering

We consider the discretized Bidirectional Texture Function $\rho(\mathbf{x},\omega_l,\omega_v)$ to be written as a matrix $\mathbf{B}_{i,j}$ with row indices i enumerating all combinations of view-/light-directions $\omega_\mathbf{l},\omega_\mathbf{v}$ and column indices j specifying the position $\mathbf{x}$ on the surface. For example, our highest resolution test datasets have 2048×2048 texels of spatial resolution, 151 view- and 151 light directions and three color channels. When arranging the colors as part of the ABRDFs, this results in a BTF matrix $\mathbf{B}\in\mathbb{R}^{68,403\times4,194,304}$ with about 287 billion entries. Compactly storing the matrix in half-precision floating point values, results in a datasize of 534.4 GB, which is certainly not applicable for real-time rendering.

In this paper, we build on the more compact FMF representation that can be obtained from $\mathbf{B}$ via singular value decomposition (SVD) [KM03]. Given the full SVD $\mathbf{B}=\mathbf{U}\Sigma\mathbf{V}^T$, a low-rank approximation is obtained by truncating the matrices $\mathbf{U}$ and $\mathbf{V}$ after C columns. Being a diagonal matrix, Σ can be multiplied with $\mathbf{V}$ prior to truncation, i.e. $\mathbf{V}:=\mathbf{V}\Sigma$. According to the Eckart-Young-Theorem [EY36] the SVD computes the best possible rank-C approximation of the original matrix under the L2-norm:

$$\arg\min_{\{\mathbf{U}_c,\mathbf{V}_c\}}\left\|\mathbf{B}-\sum_{c=1}^{C}\mathbf{U}_c\Sigma_{c,c}\mathbf{V}_c^T\right\|_F^2 \quad (4)$$

Here $\mathbf{U}_c$ and $\mathbf{V}_c$ denote the c-th column of the matrix $\mathbf{U}$ and $\mathbf{V}$ respectively, which in the context of BTFs are also referred to as Eigen-ABRDFs and Eigen-Textures, and $\Sigma_{c,c}$ denotes the c-th singular value. This way, only the more compact truncated matrices $\mathbf{U}'$ and $\mathbf{V}'$ have to be stored and an approximation of the BTF value can be obtained as $\mathbf{B}\approx\mathbf{B}'=\mathbf{U}'\mathbf{V}'^T$.

In the context of real-time rendering, the factorized representation has the important benefit of allowing random access to arbitrary values of the BTF without the necessity to reconstruct the full matrix $\mathbf{B}'$. Consider the c-th column of the matrices $\mathbf{U}'$ and $\mathbf{V}'$ as images $\mathcal{U}_c$ and $\mathcal{V}_c$. Then the BTF ρ can be approximated as

$$\rho'(\mathbf{x},\omega_l,\omega_v)=\sum_{c=1}^{C}\mathcal{U}_c(\omega_l,\omega_v)\cdot\mathcal{V}_c(\mathbf{x}). \quad (5)$$

If $\mathcal{U}$ and $\mathcal{V}$ are stored as textures on the GPU, Equation 5 can be efficiently evaluated in a shader-program. For directions and positions other than the discrete samples stored in $\mathbf{B}$, the values have to be interpolated. Instead of having to perform a costly $6D$ interpolation, in the factorized case a $2D$ interpolation for $\mathbf{x}$ in the spatial domain can be performed independently from a $4D$ interpolation in the angular domain.

We rely on the texture-units of the GPU to perform the spatial $2D$ interpolation for us when accessing the textures $\mathcal{V}$, by choosing a suitable layout of the Eigen-Textures. The four dimensional bidirectional interpolation in $\mathcal{U}$, however, has to be performed explicitly in the shader. For this, we follow an idea presented in [ND06] and pre-compute two separate two-dimensional Delaunay triangulations D_l and D_v for the sets of light and view direction samples of the BTF given in parabolic coordinates. We then raster each triangulation D into two RGB textures $\mathcal{D}$ and $\mathcal{B}$, containing the three direction indices of the enclosing Delaunay triangle and the three barycentric weights respectively. This way, during rendering the interpolated value for arbitrary view and light directions given in parabolic coordinates, can be evaluated in a GPU shader: For all 9 combinations of direction indices from $\mathcal{D}_l(\omega_\mathbf{l})$ and $\mathcal{D}_v(\omega_v)$, we perform a lookup into $\mathcal{U}$ and blend the values according to the barycentric weights. The small GPU memory overhead re-introduced by the index and weight textures can further be reduced in the case of rendering multiple BTF with the same angular sampling by sharing the textures between them.

5. Extension of SVT to BSVTFs

While for curved surfaces and perspective cameras almost all entries of the bi-directional reflectance properties in $\mathbf{U}'$ have to be accessed, the utilization of parts of the Eigen-Textures stored in $\mathbf{V}'$ follow the same consideration as conventional textures. Therefore, the idea of sparse virtual texturing could be directly applied in this case. The Eigen-Textures could be treated as an image with C channels and a spatial level-of-detail hierarchy can be constructed and decomposed into tiles.

However, this would not provide the best approximation as it does not take advantage of the property that the SVD compacts most of the information in the first few columns of $\mathbf{U}'$ and $\mathbf{V}'$, so that the contribution of later columns to the quality of the approximation decreases quickly. This observation has already found application in [SRWK11], where the columns of $\mathbf{U}'$ and $\mathbf{V}'$ were transmitted sequentially. In our case, the situation is far more general. Using SVT introduces a new degree of freedom, since every column of $\mathbf{V}'$ could be stored in a different spatial resolution.

We aim to combine both the spatial resolution and the approximation rank level-of-detail in a consistent manner. Instead of considering the matrix $\mathbf{V}'$ as one texture with multiple channels, we regard every column as an individual virtual $2D$ texture $\mathcal{V}'_c$. This way, the tiles of different columns are weighted against each other for utilization of the tilecache.

In principle, the goal of hierarchical level-of-detail rendering can be defined as the minimization of the rendering

error that can result from the restriction to a fixed tilecache size. In the case of BTFs, possible sources of error are an insufficient spatial resolution or insufficient number of factorization components for the low-rank approximation. Let image $\mathcal{S}$ denote the content of the screen when directly using the factorized BTF $\mathbf{B}'$ for rendering and $\tilde{\mathcal{S}}$ the content using SVT to access $\mathbf{V}'$. The rendering error under the L2 norm can be expressed as

$$\sum_{y=0}^{Y-1}\sum_{x=0}^{X-1}\left(\mathcal{S}(x,y)-\tilde{\mathcal{S}}(x,y)\right)^2, \tag{6}$$

i.e. the sum of squared differences over all screen pixels.

In our implementation, we do not directly minimize this term but instead propose a simplification. Let Π be the texture-mapping function used during rendering that maps from screen pixels (x,y) to the spatial position $\mathbf{x}$ in the BTF. Let furthermore $A_{\mathbf{x}}(\omega_i,\omega_o)=\rho'(\mathbf{x},\omega_i,\omega_o)$ denote the ABRDF encoded at that position in the factorized BTF. Then

$$\begin{aligned}&\sum_{y=0}^{Y-1}\sum_{x=0}^{X-1}\left\|A_{\Pi(x,y)}-\tilde{A}_{\Pi(x,y)}\right\|^2\\ =&\sum_{y=0}^{Y-1}\sum_{x=0}^{X-1}\left\|\sum_{c=0}^{C}\mathbf{U}'_c\mathcal{V}'_c(\Pi(x,y))-\sum_{c=0}^{C}\mathbf{U}'_c\tilde{\mathcal{V}}'_c(\Pi(x,y))\right\|^2\end{aligned}$$

denotes the L2 error of the ABRDF for every pixel reconstructed directly from the factorization (designated A) and reconstructed using SVT (designated $\tilde{A}$). Utilizing the SVD property of matrix $\mathbf{U}'$ being unitary, it is sufficient to consider the error for every individual virtual $2D$ texture $\mathcal{V}'_c$.

$$E=\left\|\left(\mathcal{V}'_c(\Pi(x,y))-\tilde{\mathcal{V}}'_c(\Pi(x,y))\right)\right\|^2. \tag{7}$$

Our proposed algorithm minimizes this error under the constraint of limited memory.

Please note, that even though the different columns in $\mathbf{U}'$ and $\mathbf{V}'$ have different importance for the quality of the BTF approximation, using the proposed minimization formulation we elegantly avoid the introduction of additional weighting-terms to balance the individual textures against each other. Furthermore, the proposed simplification of the rendering error minimization to an ABRDF error minimization has the additional advantage that the lighting in the virtual scene can be changed without the necessity to change anything in the tilecache utilization. Changes in view direction benefit from the availability of ABRDFs as well, as not all tiles in the tilecache have to be exchanged but only those which are affected by changes in visiblity or mip-level.

Without loss of generality, in the remainder of this paper we will assume that the individual Eigen-textures $\mathcal{V}'_c$ are laid out side-by-side in a large enough virtual texture $\mathcal{I}$ that will be used for SVT.

5.1. Level-of-detail strategy

While in the case of level-of-detail on geometry a variety of strategies for the artifact free refinement without inconsistencies exists, for SVT not too many details can be found in the literature. In this work, we essentially distinguish between two operations: `add` and `swap`.

The operation `add` will insert a tile $\mathbf{T}$ at free space in the tilecache. As a post-condition, we check whether any ancestor tile of $\mathbf{T}$ in the tile-hierarchy is now completely covered by its children. If so, the ancestor is removed from the tilecache, as it will not contribute to the pixels drawn on screen any more. `add` operations are only performed on tiles that have an ancestor in the tilecache. After the operation one or none (if an ancestor has been removed) of the free entries in the tilecache will be occupied.

The operation `swap` will remove two tiles $\mathbf{T}_{\mathbf{i}_1,l_1},\mathbf{T}_{\mathbf{i}_2,l_2}$ from the tilecache and instead insert a tile $T_{\mathbf{i}',l'}$ from a lower level $l'<\min(l_1,l_2)$ in the tile-hierarchy that covers those parts of $\mathbf{I}$ that were shown in $\mathbf{T}_{\mathbf{i}_1,l_1}$ and $\mathbf{T}_{\mathbf{i}_2,l_2}$, that is $\mathbf{i}'=\left\lfloor 2^{l'-l_1}\mathbf{i}_1\right\rfloor=\left\lfloor 2^{l'-l_2}\mathbf{i}_2\right\rfloor$. This operations will result in one free entry in the tilecache.

After all operations have been performed, the entries in the lookup-table are updated accordingly.

5.2. Tile prioritization

In order to minimize the ABRDF error from Equation 7, we weight the possible tiles that can be loaded into the tilecache against each other. For this, we roughly follow two measures:

1. The number of the pixels on the screen covered by the tile
2. The average reduction of the approximation error E for a pixel covered by the tile

As long as there is still free space in the tilecache we perform the `add` operations on tiles prioritized by these two criteria. For this we set the priority P of a tile $\mathbf{T}_{\mathbf{i}}$ at level l as $P=w(\mathbf{i},l,l-1)\cdot v(\mathbf{i},l)$, where

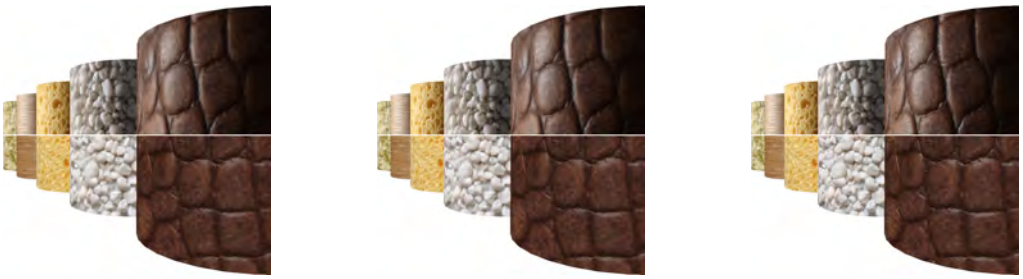
$$w(\mathbf{i},l,l')=\frac{1}{T^2}\sum_{x=0}^{T-1}\sum_{y=0}^{T-1}\left(\mathcal{T}_{\mathbf{i},l}(x,y)-\mathcal{T}_{2^{l-l'}\mathbf{i},l'}\left(x',y'\right)\right)^2 \tag{8}$$

denotes the maximum L2 difference of the tile to its lower resolution ancestor at level l' in the tile-hierarchy and

$$v(\mathbf{i},l)=\left|\left\{(\mathbf{i}',l')\in\mathcal{F}|l'\geq l\wedge\mathbf{i}=\left\lfloor 2^{l-l'}\mathbf{i}'\right\rfloor\right\}\right| \tag{9}$$

denotes the number of votes, i.e. pixels in the feedback image (see Section 3) that show the index values $\mathbf{i}$ and l of the tile or its descendants in the tile-hierarchy. (x',y') in Equation 8 denote the point in the lower resolution tile $\mathcal{T}_{2^{l-l'}\mathbf{i},l'}$ that maps to the same point in the virtual texture as (x,y) does in $\mathcal{T}_{\mathbf{i},l}$. The value P approximates the reduction in the error E in Equation 7 if $\mathbf{T}_{\mathbf{i},l}$ would be in the tilecache.

This definition for P is only valid for `add` operations on tiles of level l for which the parent at level $l-1$ is currently visible. Otherwise computing the votes v would be more complex, as several in-between steps would have to be considered. Since, `add` operations for a tile with a directly available parent are favorable for the application of streaming in Section 6, we restrict ourselves to the simple case.



BSVTF (244 MB) $C = 100$, 4032^2 pixel tilecache size | FMF BTF (316 MB) $C = 100$ | uncompressed (167 GB)

Figure 3: *BTF renderings using BSVTFs (left), FMF compression (center) and no compression (right). Upper and lower half of the images are lit from different light directions. All of the materials (front to back: leather, gravel, sponge, wood, velvet) exhibit complex view and light dependent material appearance. Whereas the uncompressed materials appear to be visibly sharper, there is hardly any noticeable difference between our technique and the FMF.*

While w is pre-computed for every tile of every Eigen-Texture, the pixel votes v, obtained at runtime from the feedback buffer, are simply repeated in the spatial layout of the components. This particular choice of v will also make sure that no space is wasted on tiles with unnecessarily high resolutions, i.e. levels that are higher than the ones in the feedback buffer, since those will have a priority of $P = 0$.

In case there is no free space left in the tilecache but further tiles could be added, we have to decide whether a `swap` operation should be performed to free space or not. Naturally, the `swap` operation will increase the error E, as it replaces higher resolution tiles $\mathbf{T}_{\mathbf{i},l}$ with a lower resolution substitute $\mathbf{T}_{\mathbf{i}',l'}$. We can approximate the rise in error by

$$\begin{aligned} c(l',\mathbf{i},l) &= \sum_{k=l'}^{l} w\left(\left\lfloor 2^{k-l}\mathbf{i}\right\rfloor, k, k-1\right) v(\mathbf{i},l) \\ &\approx w(\mathbf{i},l,l')v(\mathbf{i},l) \end{aligned}, \quad (10)$$

which is the accumulated approximated error of the portion of all in-between tiles with levels $k = l', \ldots, l$ that are currently covered by pixels from the high resolution tile and would therefore be revealed in case of a `swap`. Even though this particular approximation is not very accurate, it has the benefit that only one weight value $w(\mathbf{i},l,l-1)$, which is the same as the weight that we employ for computing P, has to be computed and stored per tile. Since we will replace exactly two tiles $\mathbf{T}_{\mathbf{i}_1,l_1}$ and $\mathbf{T}_{\mathbf{i}_2,l_2}$, the total increase in error or *cost* of this operation can be expressed by $c = c(l',\mathbf{i}_1,l_1) + c(l',\mathbf{i}_2,l_2)$.

In order to decide whether to perform a `swap` operation, we first find the three swap-candidates with the lowest cost $c^\star$ and compare this value with the highest priority $P^\star$ of the tiles that could be added. If $c^\star < P^\star$, this means that the approximated error of not having the tile with priority $P^\star$ in the tilecache is higher than the error induced by performing the `swap` operation with cost $c^\star$. Hence, we will reduce the total error E by first performing the least costly `swap` operation to obtain free space and then performing the highest priority `add` operation. Otherwise, we already found the best solution and will not perform any operation.

6. Streaming

Similar to other hierarchical level-of-detail techniques, the proposed BSVTFs are very well suited for streaming over a network. Tiles that have to be inserted into the tilecache by the `swap` or `add` operation are in this case requested from a streaming server.

To facilitate the transmission of tiles over a low-bandwidth network, we employ an additional compression to the tiles prior to submission that is inverted before the tile is inserted into the tilecache. As observed in [SRWK11], the Eigen-Textures obtained by the SVD show similar image statistics as natural images. Therefore, in principle every image-compression technique could be employed for this purpose. For example, in [KM03] Koudelka et al. utilize JPEG compression while in [SRWK11] Schwartz et al. employed a wavelet codec similar to JPEG2000.

For the compression we perform a discrete cosine transformation (DCT) on 8×8 pixel blocks of the tile-images and then apply a quantization with respect to a quality threshold similar to JPEG. The quantized data is then stored using deflate. The only mentionable difference to other off-the-shelve implementations is the fact that our compression operates on floating point values (half-precision).

To further improve the compression ratio and exploit the large redundancy present in the sets of tiles for different resolutions, instead of directly compressing the tiles, we compress the differences $\mathcal{T}'$ of a tile to its up-sampled parent in the tile-hierarchy $\mathcal{T}'_{\mathbf{i},l}(x,y) = \mathcal{T}_{\mathbf{i},l}(x,y) - \mathcal{T}_{\lfloor\frac{1}{2}\mathbf{i}\rfloor,l-1}(\frac{x}{2},\frac{y}{2})$. This procedure exploits the fact that due to our construction of the tile-hierarchy, most of the low-frequency components of the DCT are already covered by the parent tile. Thus, the amount of information that needs to be compressed is drastically reduced by using the difference image. The compressed size of a compressed tile depends on the choice of T and the user-determined quality threshold for the quantization. In our experiments we were able to obtain a compression ratio up to $6:1$ with no perceivable artifacts.

In order to unpack the DCT compressed difference images after transmission, the respective parent tile is required. During an `add` operation this does not pose a problem, since we

decided in Section 5.2 that this operation should only be performed if a parent of the tile is still in the tilecache and hence available at the client side. In case of the `swap` operation, in the worst case all ancestor tiles will have to be requested as well in order to sequentially unpack all of them until the parent is available. However, in order for a `swap` operation to be possible, higher resolution tiles had to be added to the tilecache first. In turn this means that the full branch of the level-of-detail hierarchy up to this resolution and thus also all ancestors of the tile that has to be swapped in, had to be previously transmitted to the client. We therefore employ a *least recently used* caching strategy to keep as many tiles that have been received as possible in the client-side RAM.

Even before applying the transmission compression the size of the tiles is only in the order of a few Kilobytes. The Eigen-ABRDFs in $\mathbf{U}'$ on the other hand have a size of a few Megabytes per color channel (4.4 MB for the UBO2011 objects). Fortunately, in contrast to the tiles that have to be swapped in and out on demand, $\mathbf{U}'$ only has to be transmitted once and does not change during the rendering process. Still, loading this amount of data for multiple objects in advance over a low-bandwidth connection is not a good solution.

We therefore transmit the columns $\mathbf{U}'_c$ sequentially and interleave them in the tile-datastream. This way, only a few hundred Kilobytes have to be transmitted at once, allowing to start rendering considerably faster. In this case, the vectors $\mathbf{U}'_c$ have to be prioritized in a similar fashion as the tiles to decide whether to stream the next tiles or another column of $\mathbf{U}'$. From Equation 5, it becomes apparent that $\mathbf{U}'_c$ can only contribute to the BTF approximation if $\mathbf{V}'_c$ is available as well. We can therefore approximate the priority of the Eigen-ABRDFs by the sum of votes for all tiles $\mathbf{T}_{\mathbf{i},l}$ in the tilecache that are currently used to represent $\tilde{\mathcal{V}}'_c$, weighted by the average intensity of the tile, i.e. $\sum v(\mathbf{i},l)\|\mathbf{T}_{\mathbf{i},l}\|_F^2$. This weighting can be understood as the contribution the tile makes to not having a value for $\mathbf{V}'_c$ available at all, which would in turn render the request for $\mathbf{U}'_c$ pointless.

7. Evaluation

To assess the feasibility of our approach, we tested the level-of-detail rendering and streaming on several high-resolution BTFs from the OBJECTS2011 and OBJECTS2012 databases of the University of Bonn [SWRK11] as well as a collection of 100 measured BTFs obtained from material samples. Please see the additional multimedia material to get an impression of the complex material appearance effects captured in the BTF datasets.

All of our BTF materials have a high dynamic range and are represented in RGB color. In all cases, the angular sampling contained the same set of 151×151 directions ω_v, ω_v. Before uploading $\mathbf{U}$ to the GPU we furthermore compute an additional 152-th basis illumination in which we stored a pre-integrated value of all other lights for the efficient evaluation of a view-dependent ambient term in the fragment shader. The spatial resolution of the datasets varies from 512×512 to 2048×2048 texels (see Table 1 for details).

resolution	uncomp.	FMF	BSVTF	pre-proc.
Buddha, Donkey, Minotaur, Terracotta Soldier				
2048^2 px	534 GB	813 MB	460 MB	20 min
Almond Horn, Apple, Moulage 2, Pudding Pastry Strawberry				
1600^2 px	326 GB	501 MB	262 MB	12.5 min
Chess Piece, Ganesh, Moulage 1, Shoe				
1024^2 px	133 GB	213 MB	168 MB	5 min
Santa, 100 Materials				
512^2 px	33 GB	63 MB	53 MB	1 min

Table 1: *The employed datasets (Fig. 1 and 2).* Uncompressed, FMF *and* BSVTF *gives the filesizes for the different levels of compression. Here,* BSVTF *refers to the streaming ready file including headers, pre-computed weights, level-of-detail hierarchy and DCT compression. The* pre-processing *time refers to computing the BSVTF from the FMF BTF.*

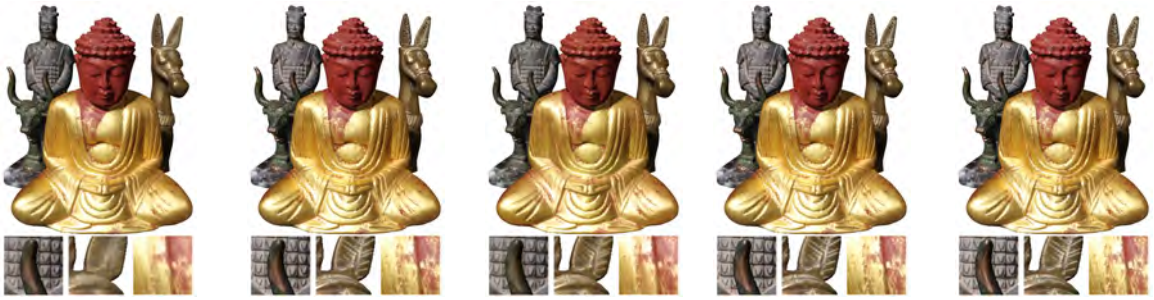
As factorization compression we compute the SVD on the full BTF matrix $\mathbf{B}$ to obtain a rank $C = 100$ approximation according to Equation 4, which we will denote *FMF BTF*.

From the FMF BTFs, we generate the BSVTF by first creating a layout of the Eigen-Textures. To save texture-lookups in the shader, we store four values $\mathcal{V}'(\mathbf{x})$ per pixel as RGBA channels. Then we compute the sets of tiles for different resolutions $\mathfrak{T}_l$. In our experiments, we use a tile-size $T = 64$. We additionally extend the tile with 4 pixels of padding at each border to allow for trilinear filtering using the tilecache texture. This results in 40.5 KB per tile when employing 16 bit floating point numbers. Using the DCT compression this size is reduced to about 7 KB to 10 KB. We also pre-compute the weights between direct descendents $w(\mathbf{i},l,l-1)$ from Equation 8 and store them as 16 bit float as well. Finally, the Eigen-ABRDFs with 151×151 angular directions are stored for the three color channels. Here we employ the strategy of packing four components into the RGBA channels as well, resulting in packets of 534 KB for interleaving with the tile transmission. Details on the processing times and the resulting total file-sizes can be found in Table 1. The costs for generating the level-of-detail representation from factorized BTFs is negligible compared to the time requirements of the factorization. While computing the FMF compression for a 512×512 texel BTF took 20 minutes using a highly optimized GPU implementation, generating the DCT compressed tiles with a single-threaded CPU implementation took only one additional minute on a 2.4 GHz Intel Xeon.

We compiled six test-scenes from the available datasets:

1. all 100 materials, arranged on a grid of tori (Fig. 2),
2. all available objects from OBJECTS2011 and OBJECTS2012 on a BTF textured plane (Fig. 1),
3. the four 4 Megapixel objects (Fig. 4 and 5),
4. five selected materials, presented on cylinders (Fig. 3),
5. only the Buddha object,
6. only the Terracotta Soldier object.

The performance of the BSVTFs was measured using fly-through animations. Please see the videos in the additional multimedia material for an impression of the animations.



BSVTF (77 MB) $C = 100$, 1008^2 tilecache | BSVTF (229 MB) $C = 100$, 4032^2 tilecache | FMF (227 MB) $C = 7$ | FMF (3.17 GB) $C = 100$ | uncompressed (2.1 TB)

Figure 4: *Quality comparison on a scene with four* 4 Megapixel *BTFs that would exceed the memory of most GPUs. With a too small tilecache size (first image), our technique is not able to resolve the fine mesoscopic details. Using an appropriate tilecache size (second image), the BSVTFs still have a small GPU memory footprint and at the same time achieve a comparable quality to directly rendering the FMF BTF data (fourth image). The third image demonstrates the loss in quality when using FMF with a higher compression ratio to achieve the same small memory footprint. Due to the insufficient number of $C = 7$ columns, this rendering shows blurred meso-structure details, washed-out highlights and false-colors.*

Rendering with a screen resolution of 1280×720 and using a tilecache the size of 4032×4032 pixels, we achieve a comparable quality to FMF BTFs at real-time frame-rates. All tests were conducted on an Intel Core i7 950 with an NVIDIA GeForce GTX 680 GPU with 4 GB of GPU memory. Details on rendering performance and GPU memory consumption can be found in Table 2.

In all of our experiments with BSVTFs, the average total CPU utilization of the system was at 23% with 51% of the time spent on rendering and GUI, 46% on evaluating the feedback-image and deriving the list of operations, 2.4% in image decompression and the remaining 0.6% in network- or disk-IO. As expected, rendering with FMF BTFs resulted 12% CPU load, since here no other tasks than rendering and GUI had to be performed. The recorded frame-rates suggest that rendering with BSVTFs is about 28% less efficient than using FMF BTFs while requiring 23%-93% less GPU memory. In both cases the performance is mainly correlated with the triangle count of the scenes, not the number of BTFs.

Fig. 3 and 4, which depict scene 4 and 3 respectively, offer a qualitative comparison of BSVTFs with FMF BTFs and uncompressed BTFs. Whereas the uncompressed BTFs appear to be visibly sharper, there is hardly any noticeable difference between BSVTFs and FMF BTFs. Note that rendering the uncompressed BTFs has been performed using deferred shading from out-of-core data and is prohibitively costly. The hard-disk is a severe bottle-neck, resulting in several hours for one image with solely local illumination.

While we employed a tilecache with 4032×4032 pixels for our evaluation of the performance, the GPU memory footprint could be reduced even further by choosing a smaller cache size. Fig. 4 demonstrates the influence of a reduced tilecache size. Although the most obvious difference can be observed in the spatial resolution of surface details, the quality of the reflectance also suffers for too small tilecache sizes. For example, the copper parts of the Minotaur object show a shift in color and appear more dull. For more tilecache size comparison images, please refer to the additional multimedia material.

	#		FPS		GPU Memory	
	Δ	BTFs	BSVTF	FMF	BSVTF	FMF
1	180K	100	34 ±10	-	1.7 GB	6.2 GB
2	3.7M	15	10 ±2	-	483 MB	6.6 GB
3	2.3M	4	18 ±5	-	229 MB	3.2 GB
4	4.6K	5	72 ±16	106 ±23	244 MB	316 MB
5	50K	1	46 ±9	69 ±14	181 MB	0.8 GB
6	1.1M	1	24 ±3	29 ±6	181 MB	0.8 GB

Table 2: *Results of the performance evaluation on the test-scenes described in Section 7. The columns* #Δ *and* #BTF *denote the number of triangles and BTF materials in the scene.* FPS *are the average and standard deviation on the tested animation sequence. FPS for FMF BTFs are only available for scenes that fit into the GPU memory of our test system.*

In Fig. 5 we demonstrate the streaming over the network. After a transmission of 25 MB the scene already achieves a perceptual similarity, measured by the structural similarity index (SSIM) [WBS*04], of 95.4% to the converged BSVTF (i.e. no `add` or `swap` operation would further reduce the error). After transmitting about 100 MB the images become virtually indistinguishable.

Limitations: Although our evaluation shows that the proposed BSVTF is applicable in a number of scenarios and performs very well, the method also has a few limitations that need to be considered as well. First, the additional buffer updates, the regular texture-data uploads, and the additional texture fetches due to the indirection in the fragment shader show an unavoidable and significant impact on the frame-rate. Second, our current approach only uses a level-of-detail hierarchy on the Eigen-Textures. While this is very feasible for few but high resolution BTFs (e.g. scenes 2 and 3), it is less efficient in scenes with many but comparably low spatial resolution materials (e.g. scene 1). It will not help at all if instead of spatial resolution a high angular resolution of the BTF data would become the bottleneck. Finally, unless the

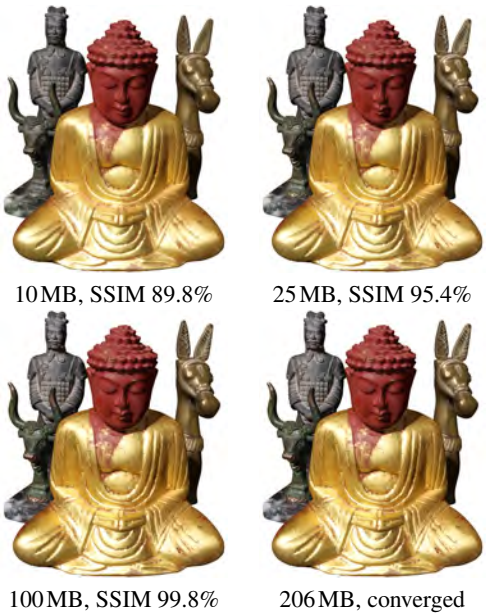


Figure 5: *Rendering quality after streaming different amounts of data over the network. The SSIM values predict the perceptual similarity between the images and are computed with respect to the converged version.*

movement of the user is somehow anticipated, a pre-fetching of data is hard to implement and resolution popping artifacts can not completely be eliminated, especially when streaming from a network connection with high latency.

8. Conclusion and Future Work

In this paper, we demonstrated that by adapting sparse virtual textures to factorized BTFs it becomes possible to render scenes with a large number of high resolution BTFs efficiently on the GPU. For this, we suggested a strategy to trade of spatial resolution and the accuracy of the reflectance representation. Furthermore, we demonstrated that this technique can be combined with an additional image compression codec and used for network transmission.

An important consideration of the proposed BSVTFs with regard to GPU memory is that only the spatial domain is covered by the level-of-detail hierarchy. We envision to overcome this limitation by extending the level-of-detail approach to the Eigen-ABRDFs as well, keeping only those that are most important to the current view-point in GPU memory. For this, a hierarchical factorization could be used, in which first the whole BTF is represented by a small number of columns C and then the residuum is subdivided into smaller subsets which are factorized individually.

Another direction of future research will be improving the network streaming by integrating a progressive refinement of the tiles, similar to [SRWK11]. This would allow smaller tile-sizes and thus faster responses to changes in view-point over low-bandwidth networks.

Acknowledgements: We would like to thank Nils Jenniche for his groundwork on the combination of SVT with BTFs. The research leading to these results was funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 323567 (Harvest4D); 2013-2016.

References

[Bar08] BARRETT S.: Sparse virtual texture memory. In *Game Developer Conference* (2008). 2, 3, 4

[CBCG02] CHEN W.-C., BOUGUET J.-Y., CHU M. H., GRZESZCZUK R.: Light field mapping: efficient representation and hardware rendering of surface light fields. In *SIGGRAPH* (2002), pp. 447–456. 4

[Cla76] CLARK J. H.: Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10 (1976), 547–554. 3

[DvGNK97] DANA K. J., VAN GINNEKEN B., NAYAR S. K., KOENDERINK J. J.: Reflectance and texture of real-world surfaces. In *IEEE Conference on Computer Vision and Pattern Recognition* (1997), pp. 151–157. 1

[EY36] ECKART C., YOUNG G.: The approximation of one matrix by another of lower rank. *Psychometrika 1* (1936), 211–218. 5

[GMSK09] GUTHE M., MÜLLER G., SCHNEIDER M., KLEIN R.: Btf-cielab: A perceptual difference measure for quality assessment and compression of btfs. *Computer Graphics Forum 28*, 1 (2009), 101–113. 3

[HF07] HAINDL M., FILIP J.: Extreme compression and modeling of bidirectional texture function. *PAMI 29*, 10 (2007), 1859–1865. 3

[HF11] HAINDL M., FILIP J.: Advanced textural representation of materials appearance. In *SIGGRAPH Asia Courses* (2011). 3

[HFM10] HAVRAN V., FILIP J., MYSZKOWSKI K.: Bidirectional texture function compression based on multi-level vector quantization. *CGF 29*, 1 (2010), 175–190. 3

[KBD07] KAUTZ J., BOULOS S., DURAND F.: Interactive editing and modeling of bidirectional texture functions. In *SIGGRAPH* (2007). 4

[KM03] KOUDELKA M. L., MAGDA S.: Acquisition, compression, and synthesis of bidirectional texture functions. In *Texture 2003 Workshop* (2003), pp. 59–64. 3, 5, 7

[LWC*02] LUEBKE D., WATSON B., COHEN J. D., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002. 3

[ND06] NGAN A., DURAND F.: Statistical acquisition of texture appearance. In *EGSR* (2006), pp. 31–40. 5

[PSR13] PAJAROLA R., SUTER S. K., RUITERS R.: Tensor approximation in visualization and computer graphics. In *EG Tutorials* (2013). 3

[RK09] RUITERS R., KLEIN R.: Btf compression via sparse tensor decomposition. *EGSR* (2009), 1181–1188. 3

[SRWK11] SCHWARTZ C., RUITERS R., WEINMANN M., KLEIN R.: Webgl-based streaming and presentation framework for bidirectional texture functions. In *VAST* (2011), pp. 113–120. 2, 3, 4, 5, 7, 10

[SWRK11] SCHWARTZ C., WEINMANN M., RUITERS R., KLEIN R.: Integrated high-quality acquisition of geometry and appearance for cultural heritage. In *VAST* (2011), pp. 25–32. 8

[TFLS11] TSAI Y.-T., FANG K.-L., LIN W.-C., SHIH Z.-C.: Modeling bidirectional texture functions with multivariate spherical radial basis functions. *PAMI 33*, 7 (2011), 1356–1369. 3

[TMJ98] TANNER C. C., MIGDAL C. J., JONES M. T.: The clipmap: a virtual mipmap. In *SIGGRAPH* (1998), pp. 151–158. 2, 3

[WBS*04] WANG Z., BOVIK A. C., SHEIKH H. R., MEMBER S., SIMONCELLI E. P., MEMBER S.: Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13* (2004), 600–612. 9

[WDR11] WU H., DORSEY J., RUSHMEIER H.: A sparse parametric mixture model for btf compression, editing and rendering. *Computer Graphics Forum 30*, 2 (2011), 465–473. 3

[Wil83] WILLIAMS L.: Pyramidal parametrics. In *SIGGRAPH* (1983), pp. 1–11. 3