The 12th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST (2011)
M. Dellepiane, F. Niccolucci, S. Pena Serna, H. Rushmeier, and L. Van Gool (Editors)

WebGL-based Streaming and Presentation Framework for Bidirectional Texture Functions

C. Schwartz, R. Ruiters, M. Weinmann and R. Klein

University of Bonn, Germany

Abstract

Museums and Cultural Heritage institutions have a growing interest in presenting their collections to a broader community via the Internet. The photo-realistic presentation of interactively inspectable digital 3D replicas of artifacts is one of the most challenging problems in this field. For this purpose, we seek not only a 3D geometry but also a powerful material representation capable of reproducing the full visual appeal of an object. In this paper, we propose a WebGL-based presentation framework in which reflectance information is represented via Bidirectional Texture Functions. Our approach works out-of-the-box in modern web browsers and allows for the progressive transmission and interactive rendering of digitized artifacts consisting of 3D geometry and reflectance information. We handle the huge amount of data needed for this representation by employing a novel progressive streaming approach for BTFs which allows for the smooth interactive inspection of a steadily improving version during the download.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.2]: Graphics Systems—Distributed/network graphics; Computer Graphics [I.3.3]: Picture/Image Generation—Viewing algorithms; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture; Image Processing and Computer Vision [I.4.2]: Compression (Coding);—Information Storage and Retrieval [H.3.7]: Digital Libraries—Dissemination

1. Introduction

Through the steep technological advances of the last two decades, museums or other *Cultural Heritage* (CH) institutions find themselves equipped with the capabilities to reach out to much larger audiences than ever before. The public dissemination of CH content in the Internet is a quasi-standard nowadays. For many media types such as historic documents, books, pictures, audio and video recordings, the Internet already offers a compelling means of distribution and presentation. However, a very important goal in this field, namely the presentation of three-dimensional objects, ranging from the Plastic arts to archeological artifacts, was not very well supported until lately. So far, photographs have been in use as an unsatisfying substitution. Unfortunately, images will never be able to capture the full experience of three-dimensional exhibits. The ability to discover the objects from every angle and under different illumination does offer a considerably higher degree of insight, providing essential information about the shape, material and surface structure. Mainly due to recent progress in computer graphics and transmission speed, the virtual exhibition of objects or even whole collections has become a possibility. Textured 3D geometries and *Polynomial Texture Maps* (PTMs) are two prominent examples of modern applications of computer graphics in CH. Textured 3D geometries are very suitable for obtaining an impression of the object, allowing inspection from arbitrary viewpoints. On the other hand, PTMs facilitate changes in virtual lighting, providing clues about materials and fine details. Each of these techniques already offers a significant improvement over 2D pictures, however, the full visual impression can only be conveyed by combining both aspects. To obtain the full view- and light-dependent impression of the artifact's appearance, solutions based on 3D meshes combined with *Bidirectional Texture Functions* (BTFs) have been proposed. This approach also offers the possibility of rapid automatic digitization [SWRK11]. Figures 4 and 6 show the high quality that can be achieved even for challenging objects using this technique.

While web-based frameworks for textured 3D geometries or

DOI: 10.2312/VAST/VAST11/113-120

PTMs are already available, up to now the public presentation of geometry with BTFs via the Internet has not been realized. This is due to the large volume of the datasets, which remain unmanageable, even with the help of state of the art compression techniques.

In this paper, we present a novel progressive transmission method for the visualization of BTF textured objects over the Internet. By employing a suitable compression and streaming scheme, we are able to provide a high-quality preview of the object within a few seconds, allowing interactive exploration while the remaining details are transmitted in the background and the real-time visualization is successively updated. To demonstrate the ability to reach a broad target audience, we implement our technique in a browser-based viewer, using the emerging web-standard *WebGL* that will work cross-platform and without the need for installing any apps, plugins or extensions on all standard compliant browsers. Nonetheless, this technique could also be employed for browser-plugin-based 3D viewers or full-scale stand-alone applications, e.g. kiosk viewers, information panels, collection browsers, etc.. Although the main application envisaged in this paper is public dissemination, the presented technique is of course also applicable for expert use, e.g. for fast browsing of artifact databases or collaboration between institutions over the Internet.

In summary, our main contributions are

- a two-tiered BTF streaming scheme via successive transmission of SVD basis vectors, each of which is progressively downloaded
- considerable improvement of BTF compression ratio by employing an additional image compression on the singular vectors
- a wavelet-based codec for HDR image compression, optimized for the shader-based decoding in WebGL
- a sophisticated heuristic for determining an optimized streaming order, prioritizing the perceptually most important information by balancing the number of transmitted singular vectors against their respective accuracy
- a real-time WebGL object-exploration demo-application as proof-of-concept supporting concurrent rendering and transmission of BTFs out-of-the-box on standard compliant browsers

2. Previous Work

The most widespread modern technique for interactive inspection of an object from arbitrary viewpoints is the use of textured 3D meshes. There are already several web-based presentation applications in the context of CH, that make use of this technique, for example [DBPGS10, JBG11]. However, these approaches are not really suitable for a photo-realistic representation of objects with complex reflectance behavior.

In addition to the use of still images, there are also image-based techniques which take pictures of an object from several viewpoints and allow the user to make an interactive selection. Often, either *Quicktime VR* or *Flash* based solutions are employed for the presentation. While these approaches allow a very realistic depiction, the selection of viewpoints is limited to those views for which images have been captured. In-between views are either left out or have to be computed via interpolation. Other purely image-based techniques such as *Light Fields* [LH96] allow for arbitrary viewpoints, but suffer from large data sizes and are thus not well-suited for web-based applications. A more sophisticated approach that is capable of reproducing the view-dependent aspects of object appearance under fixed lighting are *Surface Light Fields* (SLF) [WAA*00]. They are not entirely image-based, but instead use an additional object geometry. As a result, classical computer graphics techniques can be employed, such as completely free camera movements and the composition of several objects within one scene.

A different avenue was followed in [DBPGS10], where a web-based viewer for *Polynomial Texture Maps* (PTMs) was presented. PTMs, which are also called *Reflectance Transformation Imaging* or *Reflectance Fields* in the literature, are the complementary technique for the photo-realistic depiction in the sense that they provide a fixed view under arbitrary illumination. By employing progressive downloads, the user is able to view large images and interactively change the light-direction. There are also works on multi-view PTMs [GWS*09], but, to the best of our knowledge, there is no solution for web-based distribution and viewing. In [MSE*10], an offline viewer was presented. However, due to the fact that it is difficult to take advantage of the coherence between different views and because flow-fields are used as an implicit geometry representation, a rather large amount of storage is required for every view. Additionally, a large number of views would be needed for quality results, especially if a completely arbitrary viewpoint selection is desired instead of limiting the viewer to a circular motion at fixed distance.

The Bidirectional Texture Function is a technique which combines the advantages of SLFs and PTMs. A BTF $\rho(x, \omega_i, \omega_o)$ is a function describing the appearance at each point x on a 3D surface, depending on both light-direction ω_i and view-direction ω_o. A survey on state of the art techniques for the acquisition and rendering of such functions can be found in [FH09]. So far, BTFs have mostly been applied as a representation of flat material samples. However, this technique has also been used to capture the appearance of CH artifacts, e.g. [FKIS02, MBK05, SWRK11], who propose capturing the BTF for the whole surface of non-planar objects. Since then, advances in capture setups have lead to a considerable increase in quality (see Figures 4 and 6). In this paper, we show that BTFs can be compressed sufficiently, so that streaming a dense sampling of view- and light-directions becomes practically attainable, allowing for virtually artifact-free interpolation. Furthermore, like SLF, the geometry with BTF representations is suited for working with scenes composed of multiple objects and free camera movements. An alternative approach makes use of *spatially varying BRDFs* (SVBRDFs). Here, techniques for the capture of objects with reflectance behavior (e.g. recently

[HLZ10]) are available, also. In addition, SVBRDFs are a reasonable choice for web-based applications due to their compactness and real-time rendering capabilities. However, since analytical BRDF models are employed, the complexity of reflectance behavior that can be represented is more restricted than for BTFs.

For the display of 3D content in web-applications one can choose from a wide variety of technical solutions and APIs. For a more comprehensive overview of 3D content in web-applications we refer the reader to [BEJZ09]. To reach the largest possible audience, it is desirable to avoid installation of third-party software or vendor specific solutions. A better choice would be a cross-platform and standardized solution. Furthermore, for our needs, a direct access to the graphics hardware is imperative. An API which has gained wide popularity over the last two years, accompanying the new web standard HTML5, is WebGL. It is has been standardized by the Khronos Group [Khr10] and several main-stream browser vendors (Mozilla, Google, Apple, Opera) have already announced their support.

3. Rendering

We use the *Decorrelated Full Matrix Factorization* (DFMF) [Mül09] to transmit the BTFs. Even though there are several different compression techniques available this technique offers several important advantages for our purposes. It provides a good compression ratio while allowing for decompression at a reasonable cost and allows for real-time rendering on the GPU. But more importantly, matrix factorization based techniques readily provide a level of detail representation sufficient for progressive transmission of the BTF. Additionally, this technique has the considerable advantage that the texture mapping units of the GPU can be utilized to perform interpolation both in the angular and spatial domain. This reduces the decompression costs considerably in comparison with techniques using clustering [MMK03], sparse representations [RK09] or vector quantization [HFM10], to name just a few. There are also tensor factorization based approaches [WXC*08], which are, like our wavelet compression, capable of further compressing the spatial dimensions. However, since these techniques are not well-suited for real-time rendering they are not directly applicable in our case.

When compressing a BTF via a matrix factorization, the original function is approximated as a sum of products of two functions, one of them depending only on the view- and light-directions ω_o, ω_i, the other on the spatial position x:

$$\rho(x,\omega_i,\omega_o) \approx \tilde{\rho}(x,\omega_i,\omega_o) = \sum_{c=1}^{C} u^{(c)}(\omega_i,\omega_o)\cdot v^{(c)}(x) \quad (1)$$

Here, the approximation quality versus the compression ratio is controlled by C, i.e. the number of functions used.

Such an approximation can be found by first storing the measured data samples in one matrix $\mathbf{M}$. Here, each column index represents a spatial position and each row index denotes a combination of light-direction and view-direction. Then the *Singular Value Decomposition* (SVD) $\mathbf{M} = \mathbf{USV}^T$ is computed. Each of the columns of $\mathbf{U}$ can now be regarded as a tabulated representation of one of the functions $u^{(c)}(\omega_i,\omega_o)$ and analogously the columns of $\mathbf{V}$ as representations of $v^{(c)}(x)$. Thus, in this approximation, C denotes the number of columns or *components* (SVD and PCA are closely related and the term component is in common use in the field of BTF compression [MMS*04]). $\mathbf{S}$ is a diagonal matrix, which can be stored by multiplication either with $\mathbf{U}$ or $\mathbf{V}$. Since many points on the surface exhibit similar reflectance behavior, there is a large redundancy between the columns of $\mathbf{M}$, allowing for a good approximation via a low rank matrix. For more details, we refer to [Mül09].

For rendering, it is necessary to reconstruct samples for arbitrary positions on the surface and arbitrary view- and light-directions. Therefore, one has to interpolate the available discretized representation. Here, the DFMF has the advantage that the interpolation can be performed independently for the spatial and angular dimensions by interpolating the 2D functions $u^{(c)}$ and the 4D functions v^c, respectively, instead of the actual 6D function $\tilde{\rho}(x,\omega_i,\omega_o)$. The GPU can be used to evaluate the sum (1) in real-time. Here, the tabulated functions $u^{(c)}$, $v^{(c)}$ are both stored in textures and can thus be evaluated per fragment by simply performing adequate texture fetches. For $v^{(c)}$ the 2D texture interpolation can be performed directly on the GPU, whereas for the angular components a 4D interpolation is necessary, which is not supported in hardware. Thus, for each available view-direction, we store a full light-hemisphere independently, represented via a parabolic parametrization. This way, we can utilize the GPU for light interpolation, but have to perform the view interpolation explicitly in the shader.

The DFMF does not compress the three color channels RGB together. Instead it first performs a decorrelation by transforming the colors into YUV color space, where each color is represented by a luminance Y and two chrominance components U and V. The rationale here is that for many materials there is a strong dependence of the intensity on the view- and light-direction, whereas the color remains largely constant. Thus, by performing this decorrelation, it is possible to store them independently and use fewer components to encode the color information than for the luminance.

Apart from the decorrelation, we additionally perform a reduction of the dynamic range before computing the SVD. For this, we apply a logarithmic transform to the Y channel and normalize the color channels as $\frac{U}{Y}$ and $\frac{V}{Y}$. This is necessary, since BTFs can exhibit a considerable dynamic range and the compression minimizes a squared error function. This means, that without this transformation, dark regions in the material would not be treated sufficiently, since they do not contribute much to the squared error, although they are still perceptually important.

4. Streaming

When transmitting BTFs over a network for interactive rendering, a progressive download is desirable due to the large size of BTFs. The full transmission of a BTF can require several minutes, whereas, using our technique, a high-quality

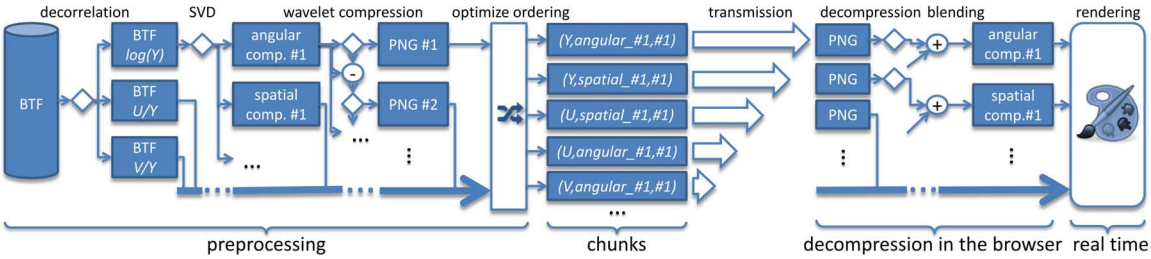


Figure 1: *Overview over our proposed compression, streaming and rendering pipeline.*

version is available after about seven seconds, a first preview even after less than a second (Timings depend on the connection speed. In the remainder of this paper, we will consider a commonplace 8 MBit/s connection). To achieve this, we both progressively increase the number of components C and the resolution of the textures by utilizing a wavelet codec.

By transmitting more components, the quality of the approximation is successively increased. In fact, it is possible to prove, that by using only the first C components, one obtains the best possible rank-C approximation of the original matrix under the L2-norm [EY36]. This can obviously be directly utilized for the progressive transmission of a BTF, by successively transferring the individual columns of the matrices $\mathbf{U}$ and $\mathbf{V}$, each time effectively increasing the rank of the approximation available for rendering.

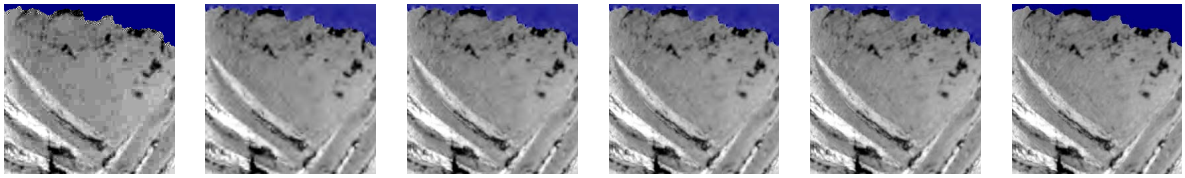
Instead of employing a specialized streaming protocol, our viewer application simply requests small chunks in a specified order, which is comparable to current HTTP streaming approaches for other multimedia content [VDVLVdW10]. This has the advantage that it can work over HTTP, which allows delivery with an ordinary web server, omitting firewall problems or the need for a particular streaming server.

Wavelet Compression. The rendering can start as soon as the first component for each channel Y, U and V is available. Each additional component that has been transmitted, can then be utilized directly for rendering to further increase the quality of the approximation. The individual components $u^{(c)}$ and $v^{(c)}$, however, are still very large. Especially the spatial components can require considerable space, as each one is an high dynamic range (HDR) gray-scale image with the full resolution of the texture. Thus, for a 2048×2048 pixel BTF, uncompressed transmission of only one component for one of the channels still requires 8MB. Since the angular components show rather low frequencies and the spatial components exhibit frequency characteristics similar to natural images (cf. Figure 3), usual image compression and transmission techniques can be applied here. We thus utilize a wavelet codec to send each of the individual component textures progressively. We start with a highly compressed version and then gradually send several difference images, each also encoded with the wavelet codec, until the original has been reconstructed to a sufficient level of accuracy.

There is a wide range of techniques, both for image compression and for progressive transmission of images. Even giving a short overview would by far exceed the scope of this paper, and hence, as possible starting points, we refer the reader to [DN98, RY00]. However, for our purposes, many of these techniques are not directly applicable, as we have two important constraints. Firstly, we have to transmit HDR textures. Even though the dynamic range of the resulting textures has been reduced considerably by the logarithmic transform, for full quality display of the BTF the higher precision of floating point values is still desirable. And secondly, our codec must be suitable for fast decompression in the browser. There are several elaborate encoding schemes, such as the SPHIT [SP96] codec, which achieve very good compression ratios and allow for elegant progressive transmission. However, JavaScript, being an interpreted language (though JIT is available in newer browsers), is still not fast enough for decompressing data encoded with these techniques. Instead we need a codec that can either be decoded by native browser-functions or via shaders in WebGL.

Unfortunately, the browser does not support a decompression codec which is directly suited to our purposes. Usually, one can only rely on support for JPEG and PNG images, both providing neither HDR encoding nor progressive transmission (though possible in both PNG and JPEG, it cannot be used for WebGL textures). We therefore decided to implement a simple wavelet codec ourselves. The restoration of the HDR signal from LDR images and the decompression of the wavelet transform is performed in a fragment shader using the render-to-texture capabilities of WebGL. Consequently, this step is no longer limited by the execution speed of the JavaScript interpreter. For the actual stream-decoding, on the other hand, WebGL is not suitable at all, since a sequential decompression of the bit-stream is necessary, which cannot easily be performed on the GPU. Hence, we store the quantized wavelet components in a PNG image, utilizing the LZ77 compression and Huffman entropy encoding that is part of PNG. This approach is not the best available image-codec w.r.t. compression ratio. However, while still compressing reasonably well it allows for the progressive transmission of HDR data and can be efficiently decoded with the limited resources available to a JavaScript application. In our experiments, it performed better than JPEG compression regarding RMS error (see Figure 2).

For image compression we apply a straightforward wavelet transform coder. As first compression step, we perform a dyadic wavelet decomposition of our texture using the popular CDF 9/7 wavelet [CDF92], which has found widespread



JPEG(0.4bpp) RMSE 22.1427 | **Wavelet** (0.4bpp) RMSE 10.8259 | **Wavelet** (0.8bpp) RMSE 6.7880 | **Wavelet** (1.2bpp) RMSE 4.9895 | **Wavelet** (1.6bpp) RMSE 3.9339 | **Original** (16bpp)

Figure 2: *Enlarged views of a detail in a spatial texture of the Buddha dataset. The left-most image shows as comparison a uniformly quantized and JPEG compressed LDR version. The next four images show how our wavelet codec continuously refines the texture compared to the original on the right. The last row gives the RMS error computed for only those regions of the texture which are occupied (unoccupied texels are marked in blue).*

application in image compression, for example in the JPEG2000 image format. This decomposition is performed directly on the HDR floating point data. To compress these floating point values, we then use a deadzone uniform scalar-quantizer to obtain 16 Bit integer values. For each of the wavelet bands, we choose an individual threshold, using the algorithm from [SG88] to perform the bit allocation for obtaining chunks of a fixed size. During this allocation, we compute the rates under the assumption that each of the sub-bands is compressed individually via LZ77 and Huffman encoding. This is obviously only an approximation, since the coefficients for all sub-bands are stored together in the final PNG, but we found that it is an acceptable approximation, resulting in files of almost correct size. Finally, the quantized coefficients are stored in one PNG image, storing the high and low bytes separately from each other, as this provided the best compression results in our experiments.

The parabolic parameterization of the hemispherical angular components only occupies circular regions in the texture representing the u-vectors, and often the spatial components contain an only partially filled texture atlas representing the v-vectors. Therefore, in both textures we have entries which are of no importance to the final appearance of the object, denoted as "Don't Care" values (see Figure 3 with "Don't Care" areas marked in blue). However, when these entries are simply set to 0, the resulting texture contains sharp edges, which are not well-suited for the wavelet compression, as many coefficients are necessary to encode the resulting high frequencies. To avoid these problems, we use the approach described in [BHH*98] to fill these areas in such a way that the number of coefficients needed for compression is minimized as far as possible.

Transmission and Decompression. The individual components are loaded from the JavaScript by successively requesting the compressed chunks from the server. The quantization thresholds can be transmitted separately in a JSON file. As soon as it has been received, each chunk is decompressed from PNG into a WebGL texture natively by the browser. Further decompression is then performed in two steps by a shader. First, the low and high bytes are combined and the quantization is reversed to obtain a floating point texture of the wavelet transformed image. Secondly, the original texture is reconstructed by applying the inverse wavelet transform. This is done successively for the horizontal and vertical directions on each scale. We perform the transformation via direct convolution, instead of using a lifting scheme, to avoid the need for even more render-passes. Each of these steps is performed in a fragment shader on the GPU, by using a WebGL framebuffer to render a suitable quad into a texture. Care has to be taken to correctly interleave these individual decoding tasks with the actual rendering of the object to avoid noticeable drops in the frame rate. To achieve progressive transmission, we successively transmit encoded difference images, which are then joined by the shader using additive blending.

Transmission Order. One important remaining question is the order in which the components are to be transmitted to the client. At any time during streaming, there are several possibilities for choosing the next chunk. For each of the channels Y, U and V, it is either possible to transmit a new component or to increase the quality of an already transmitted component by loading the next difference image for either the angular or spatial domain. We determine this order in a preprocessing step. Here, we sort the textures by employing a greedy scheme in such a way that the total RMS error for the whole BTF is decreased as much as possible with each transmitted chunk. This order could be found by computing the sum of squared errors (SSE) between the original BTF and the reconstruction with the transmitted components explicitly. However, this would be extremely costly, as it would require decompressing and computing the SSE for the whole BTF file for every possible decision. We instead use an approximation, which takes advantage of the fact that the BTF is represented via a SVD. For this approximation, we consider the errors in $\mathbf{U}$ and $\mathbf{V}$ independently. Assuming that the compression had only been performed for the columns in matrix $\mathbf{U}$, this results in a distorted matrix $\tilde{\mathbf{U}}$, the error of which is given by

$$\|\mathbf{U}\mathbf{S}\mathbf{V}^T - \tilde{\mathbf{U}}\mathbf{S}\mathbf{V}^T\|_F^2 = \|(\mathbf{U}-\tilde{\mathbf{U}})\mathbf{S}\mathbf{V}^T\|_F^2 = \|(\mathbf{U}-\tilde{\mathbf{U}})\mathbf{S}\|_F^2.$$

By construction, $\mathbf{V}$ is orthogonal, and the second equality holds as a result of the fact that the Frobenius norm is invariant under orthonormal transformations. We can thus compute the SSE for the whole BTF dataset by computing the errors for each of the component textures individually and weighting them with the weights in $\mathbf{S}$. The same

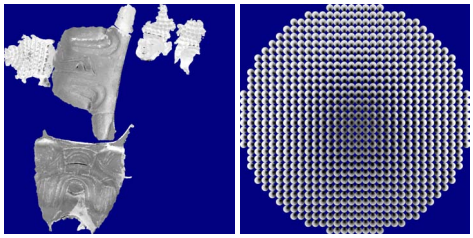


Figure 3: *Uncompressed first spatial (left) and angular (right) luminance components for the Buddha dataset. "Don't Cares" are marked in blue. The angular component texture in fact exhibits rather low frequencies, as the blue border around the light-hemispheres are "Don't Care" values and thus do not need to be taken into account.*

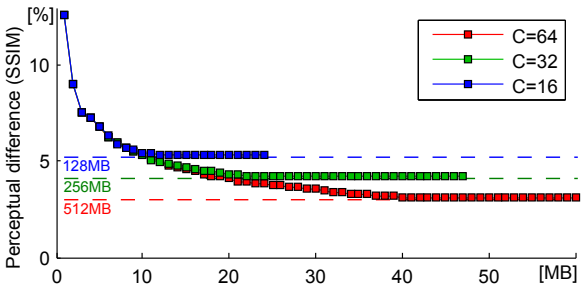


Figure 5: *The perceptual error [WBSS04] in dependence of the amount of transmitted data for different versions of the 1.05MP Buddha BTF. The dashed lines correspond to the DFMF compressed BTFs without employing further wavelet-compression. The error is computed w.r.t. the uncompressed (133GB) dataset and averaged over 5 representative view- and light-combinations.*

computation is also possible for $\mathbf{V}$, under the assumption that $\mathbf{U}$ is orthonormal. When there are distortions in both matrices, this no longer holds exactly. However, we found that deviations are very small. The difference between the correctly computed error $\|\mathbf{USV}^T - \tilde{\mathbf{U}}\tilde{\mathbf{S}}\tilde{\mathbf{V}}^T\|_F^2$ and the approximation $\|(\mathbf{US} - \tilde{\mathbf{U}}\tilde{\mathbf{S}})\|_F^2 + \|\mathbf{VS} - \tilde{\mathbf{V}}\tilde{\mathbf{S}}\|_F^2$ was below 0.3% when comparing the DFMF compressed dataset with one with additional wavelet compression applied. Currently, we simply compare the errors in the channels Y, U and V directly. However, since luminance and chrominance are actually represented using different units, one being the logarithm of the intensity, the other being normalized color components, proper weighting factors, obtained by perceptual experiments, should be employed instead. Even better results might be possible by using a more sophisticated BTF error metric, e.g. [GMSK09]. Unfortunately, for such an approach our heuristic would no longer be applicable, increasing computation time drastically, as the error would have to be recomputed for many different combinations of transmitted components with varying approximation qualities.

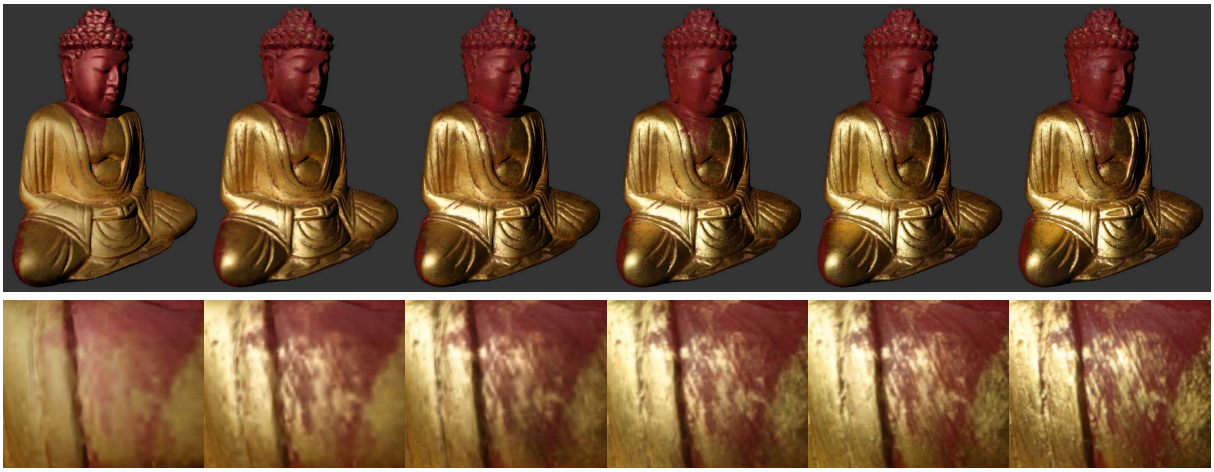
5. Evaluation

We evaluate our approach on several challenging real-world datasets captured using a highly parallelized photographic setup [SWRK11], which offers an angular sampling of 151×151, high geometry and texture resolution and high dynamic range. We used flat material samples as well as BTF texture atlases for whole 3D objects. The view- and light-directions are resampled into parabolic maps with a radius of 16 pixels, resulting in angular textures of 1024×1024 pixels. For the texture atlases of the objects, 1024×1024 pixels were employed, except for the Buddha dataset, for which a higher spatial resolution of 2048×2048 was available. The material samples (two fabrics and one leather) have spatial resolutions of 346×335, 573×463 and 512×512. Uncompressed, the size of the data reaches from 534.4GB for the 4.2 Megapixel (MP) BTF over 133.6GB for 1.05MP down to 15.6GB for the smallest material sample.

For testing the transmission performance and real-time capabilities, we created an HTML5 based BTF viewer employing WebGL for rendering. Datasets presented in this paper and the interactive viewer are available at `http://btf.cs.uni-bonn.de`. Currently, we do not employ asynchronous transfer for loading additional information, e.g. the 3D geometries or the quantization thresholds, as this would be beyond the scope of this paper, but instead include them directly in the HTML file. Even though, in real-life applications these files should be transmitted asynchronously on request, (or in the case of geometry even progressively downloaded) we found this setting suitable for evaluating the performance of our rendering and BTF streaming technique. Please note that we do not include the quantization thresholds ($\approx$ 35-64KB) in our transmission-size considerations.

Unfortunately, the missing support for 3D textures forces us to store the angular and spatial components in several tiles of one 2D texture, restricting the maximum renderable spatial- and angular-resolution and increasing the number of necessary texture-fetches. Although these constraints could be partially handled by using large textures and multiple texture-units, the most crucial remaining limitation is the available GPU memory, especially since half-precision floating point textures are not yet supported in any browser. Just the 4.2MP BTF data, disregarding any additional buffer-textures, would require 2.5GB of GPU memory for 64 components for the Y and 32 for the U and V channels, by far exceeding a mainstream configuration.

Therefore, we tested our streaming approach with 32 Y and half as many U and V components, resulting in about 1.4GB of GPU memory consumption. In principle higher numbers of components as well as higher texture resolutions could also be streamed efficiently. This is especially since the wavelet codec performs a multi-scale analysis and transmits the most important wavelet coefficients more accurately (see Figure 2). Thus, a perceptually acceptable version, only lacking high frequency details, is usually available very fast. To allow for a meaningful visualization as early as possible, we constrain the wavelet compression to produce chunk sizes of 100KB. Rendering can start as soon as six chunks are available. As shown in Figure 4, after transmitting just 1MB of chunks, i.e. 1s of transmission, the overall



0.87MB 0.99MB 1.91MB 6.97MB 46.39MB 534.4GB

Figure 4: *A sequence of screenshots showing the refinement of the appearance of the* 4.2*MP Buddha dataset over the streaming progress. With* 0.87*MB enough chunks were transmitted to start rendering. From* 6.97 *to* 46.39*MB only minor changes in appearance are noticeable and merely remaining fine details are loaded (reducing the SSIM error [WBSS04] from 3.84% to 2.54% on the full image and from 12.54% to 8.28% on the detail view). For comparison, a raytraced image of the uncompressed dataset is shown on the right.*

impression of the object's appearance is already successfully recovered. The progressive download of additional fine details is perceivable until about 7MB are transmitted. The full dataset with applied wavelet compression occupies as little as 46.4MB, while approximating the captured appearance with a perceptual error [WBSS04] of 2.5%. A more detailed analysis is given in Figure 5.

While rendering the BTF is mainly limited by the number of components and screen resolution, decompression depends on the resolution of the angular and spatial textures. On a current GPU (GeForce GTX560 TI), rendering 640×640 pixels was possible with at least 65 FPS for 1.05MP with up to 32 components. Higher spatial resolutions (4.2MP,$C = 32$) are still renderable with 60 FPS, while a higher number of components ($C = 64$) leads to a slightly worse result of 25 FPS. During decompression of transmitted chunks the frame rate remains constant for $C = 16$ and $C = 32$ and decreases to 25 FPS for $C = 64$ at 1.05MP. For the 4.2MP BTF with $C = 32$, 14 FPS were achieved. The decompression times directly influence the maximum possible transmission rate, which was at about 1.9MB/s for the smallest dataset and 0.4MB/s for the largest. Even on 5 year old graphics hardware (GeForce 8800GTX) the $C = 16$, 1.05MP datasets were transmitted with 7 and rendered with 25 FPS. Therefore, we recommend to offer datasets at multiple quality levels to accommodate older hardware.

Compatibility. So far, four of the five major mainstream browsers have committed themselves to supporting WebGL, however only Chrome and Firefox have WebGL readily available in their current release versions. Additionally, so far only Google Chrome and the nightly build of Firefox do implement the `oes_texture_float` extension, which

Figure 6: *A screenshot of our application and enlarged views of the presented object from two arbitrarily selected viewpoints with two freely chosen light directions, respectively.*

our current implementation needs, as our BTF representation makes heavy use of HDR data. It should be possible to circumvent this limitation by using several LDR textures and combining these in the shader to obtain HDR data, although we have not yet implemented such an approach. We hope that the support for WebGL will improve in future browsers, but would like to stress that the basic streaming technology we presented is in no way limited to WebGL. Our decompression and viewing shaders are based on OpenGL ES2.0, a standard specifically designed to be supported by as many devices as possible, including mobile platforms.

In our proposed implementation, a major challenge for mainstream deployment would be the large amount of texture memory that is needed to render even one BTF-textured object, making scenes composed of multiple objects not yet feasible. Thus, a more memory efficient compression scheme would be desirable.

6. Conclusions and Future Work

We presented a WebGL framework for the public dissemination of cultural heritage artifacts as digitized 3D objects textured with Bidirectional Texture Functions. This repre-

sentation allows for the generation of highly accurate digital replicas with complex reflectance behavior, including visual hints about manufacturing techniques and signs of wear. By streaming the individual components obtained by the SVD based compression of the BTF together using a wavelet-based image compression, we are able to present high-quality previews of the BTF after just a few seconds. The remaining data is progressively loaded until a full quality presentation is obtained. Even though we used a WebGL based implementation, the presented technique is not limited to web browsers, but could be used by a wide range of clients for the streaming and photo-realistic depiction of objects.

Apart from CH, this technique also has the potential to find application in fields such as virtual prototyping, entertainment and advertisement.

In our implementation, we include the geometry of the objects in the HTML file. It would be an obvious first extension of our technique to also employ progressive geometry transmission. Furthermore, it is conceivable, that in the future a full-fledged hierarchical level of detail renderer could be implemented, employing view-dependent refinement to allow for the presentation of large objects in extremely high resolution or even complete scenes, such as virtual excavation sites or historical settings, at very high levels of detail.

Acknowledgements. We would like to thank Raoul Wessel for proposing the idea and Nils Jenniche for helping with the implementation. The research leading to these results was partially funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 231809; 2008-2012 and by the German Science Foundation (DFG) under research grant KL 1142/4-1.

References

[BEJZ09] BEHR J., ESCHLER P., JUNG Y., ZÖLLNER M.: X3dom: a dom-based html5/x3d integration model. In *Web3D* (2009), pp. 127–135. 3

[BHH*98] BOTTOU L., HAFFNER P., HOWARD P. G., SIMARD P., BENGIO Y., LECUN Y.: High quality document image compression with djvu. *J. Elec. Imag. 7* (1998), 410–425. 5

[CDF92] COHEN A., DAUBECHIES I., FEAUVEAU J. C.: Biorthogonal bases of compactly supported wavelets. *Information Technology* (1992). 4

[DBPGS10] DI BENEDETTO M., PONCHIO F., GANOVELLI F., SCOPIGNO R.: Spidergl: a javascript 3d graphics library for next-generation www. In *Web3D* (2010), pp. 165–174. 2

[DN98] DAVIS G. M., NOSRATINIA A.: Wavelet-based image coding: An overview. *Applied and Computational Control, Signals, and Circuits 1* (1998), 205–269. 4

[EY36] ECKART C., YOUNG G.: The approximation of one matrix by another of lower rank. *Psychometrika 1* (1936), 211–218. 4

[FH09] FILIP J., HAINDL M.: Bidirectional texture function modeling: A state of the art survey. *PAMI 31* (2009), 1921–1940. 2

[FKIS02] FURUKAWA R., KAWASAKI H., IKEUCHI K., SAKAUCHI M.: Appearance based object modeling using texture database: acquisition, compression and rendering. In *EGRW* (2002), pp. 257–266. 2

[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. 6

[GWS*09] GUNAWARDANE P., WANG O., SCHER S., RICKARDS I., DAVIS J., MALZBENDER T.: Optimized image sampling for view and light interpolation. In *VAST* (2009), pp. 93–100. 2

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

[HLZ10] HOLROYD M., LAWRENCE J., ZICKLER T.: A coaxial optical scanner for synchronous acquisition of 3d geometry and surface reflectance. In *SIGGRAPH* (2010), pp. 99:1–99:12. 3

[JBG11] JUNG Y., BEHR J., GRAF H.: X3DOM as carrier of the virtual heritage. In *3D-ARCH* (2011). 2

[Khr10] KHRONOS GROUP: *WebGL Specification 1.0*, 2010. http://www.khronos.org/webgl/. 3

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *SIGGRAPH* (1996), pp. 31–42. 2

[MBK05] MÜLLER G., BENDELS G. H., KLEIN R.: Rapid synchronous acquisition of geometry and BTF for cultural heritage artefacts. In *VAST* (2005), pp. 13–20. 2

[MMK03] MÜLLER G., MESETH J., KLEIN R.: Compression and real-time rendering of measured btfs using local pca. In *VMV* (2003), pp. 271–280. 3

[MMS*04] MÜLLER G., MESETH J., SATTLER M., SARLETTE R., KLEIN R.: Acquisition, synthesis and rendering of bidirectional texture functions. In *EG STAR* (2004), pp. 69–94. 3

[MSE*10] MUDGE M., SCHROER C., EARL G., MARTINEZ K., PAGI H., TOLER-FRANKLIN C., RUSINKIEWICZ S., PALMA G., WACHOWIAK M., ASHLEY M., MATTHEWS N., NOBLE T., DELLEPIANE M.: Principles and practices of robust, photography-based digital imaging techniques for museums. In *VAST* (2010), pp. 111–137. 2

[Mül09] MÜLLER G.: *Data-Driven Methods for Compression and Editing of Spatially Varying Appearance*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2009. 3

[RK09] RUITERS R., KLEIN R.: Btf compression via sparse tensor decomposition. *CGF 28*, 4 (2009), 1181–1188. 3

[RY00] RAO K. R., YIP P.: *The Transform and Data Compression Handbook*. CRC Press, 2000. 4

[SG88] SHOHAM Y., GERSHO A.: Efficient bit allocation for an arbitrary set of quantizers. In *International Conference on Acoustics, Speech, and Signal Processing* (1988). 5

[SP96] SAID A., PEARLMAN W.: A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology 6*, 3 (1996), 243 –250. 4

[SWRK11] SCHWARTZ C., WEINMANN M., RUITERS R., KLEIN R.: Integrated high-quality acquisition of geometry and appearance for cultural heritage. In *VAST* (2011). 1, 2, 6

[VDVLVdW10] VAN DEURSEN D., VAN LANCKER W., VAN DE WALLE R.: On media delivery protocols in the web. In *ICME* (2010), pp. 1028 –1033. 4

[WAA*00] WOOD D. N., AZUMA D. I., ALDINGER K., CURLESS B., DUCHAMP T., SALESIN D. H., STUETZLE W.: Surface light fields for 3d photography. In *SIGGRAPH* (2000), pp. 287–296. 2

[WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13*, 4 (2004), 600 –612. 6, 7

[WXC*08] WU Q., XIA T., CHEN C., LIN H.-Y. S., WANG H., YU Y.: Hierarchical tensor approximation of multi-dimensional visual data. *Vis. and CG 14*, 1 (2008), 186 –199. 3