Tileable BTF

Man-Kang Leung [†] Wai-Man Pang [‡] Chi-Wing Fu [†]
Tien-Tsin Wong [‡] Pheng-Ann Heng [‡]

[†] The Hong Kong University of Science and Technology

[‡] The Chinese University of Hong Kong

Abstract—**This paper presents a modular framework to efficiently apply the bidirectional texture functions (BTF) onto object surfaces. The basic building blocks are the BTF tiles. By constructing one set of BTF tiles, a wide variety of objects can be textured seamlessly** *without* **re-synthesizing the BTF. The proposed framework nicely decouples the surface appearance from the geometry. With this** *appearance-geometry decoupling***, one can build a library of BTF tile sets to instantaneously dress and render various objects under variable lighting and viewing conditions. The core of our framework is a novel method for synthesizing seamless high-dimensional BTF tiles, that are difficult for existing synthesis techniques. Its key is to shorten the cutting paths and broaden the choices of samples so as to increase the chance of synthesizing seamless BTF tiles. To tackle the enormous data, the tile synthesis process is performed in** *compressed* **domain. This not just allows the handling of large BTF data during the synthesis, but also facilitates compact storage of the BTF in GPU memory during the rendering.**

Keywords— **I.3.7.b Three-Dimensional Graphics and Realism – Color, shading, shadowing, and texture, I.3.3 Picture/Image Generation, I.3.6 Methodology and Techniques**

1 Introduction

Realistic modeling and rendering of surface-light interaction is one of the major goals in computer graphics. Several reflectance models of different levels of detail, such as the BRDF [5,43], BTF [12,20,47], and BSSRDF [24,50] have been proposed to address the problem. This paper introduces a modular framework to apply the bidirectional texture functions (BTF) in appearance modeling. Our goal is to *decouple the BTF synthesis from the surface geometry*, so that changing the surface geometry does not require re-synthesizing the BTF. To achieve this goal, we first construct the *BTF tiles* instead of directly synthesizing the BTF on the geometry surface.

Surface appearance modeling using the BTF can be roughly subdivided into the following phases: 1) *BTF acquisition* (real or synthetic data), 2) *BTF synthesis*, 3) *BTF compression*, and 4) *BTF rendering*. Once the raw BTF data is acquired, existing approaches normally synthesize the BTF *directly* onto the target geometry to avoid visible cutting seams and to minimize the geometric distortion. However, as the synthesis process is applied directly onto the target geometry, the synthesized BTF data is tied to the geometry surface and cannot be reused elsewhere. Furthermore, if we want to change the surface appearance with another BTF, we are forced to re-synthesize the BTF data even for the same target surface. Rather than having a surface-dependent BTF data, the proposed framework introduces a tile space to decouple the surface geometry from the synthesis process. The decoupling is done by replacing the target geometry surface with an intermediate tile space and by synthesizing the BTF in this tile space. Figure 1 outlines the proposed approach. Note that the proposed BTF synthesis framework is independent of the geometry. With this framework, we gain the following advantages:

Surface Independence and Reusability Since the synthesized BTF tiles are independent of the surface geometry, we can efficiently synthesize the BTF tiles without referring to any particular surface geometry. The tile set can be repeatedly used for dressing a wide variety of surface models; we do not need to modify any tile or synthesize more tiles. Hence, one can construct a library of BTF tile sets and use the tile sets over and over again.

Instant Re-dressing Furthermore, by defining a canonical organization of tiles so that all BTF tile sets share the same tile arrangement, we can instantaneously re-dress a tiled surface simply by looking up another tile set. No re-tiling nor extra computation is needed.

Aperiodic Even the total number of BTF tiles within a tile set is finite, the non-periodic property is achieved via Wang tiling [9, 46, 55]. As the conventional Wang tiling is only applicable for planar domain, we employ the techniques we devised in our previous work [18] to generalize Wang tiling on surfaces with more general topologies.

Compactness Since the BTF tiles are rectilinear in structure, they not just nicely fit into the memory, but also facilitate compression using standard block-based methods like S3TC. The entire BTF data is compact enough to be stored in current GPU memory for time-critical rendering.

The core of our work is the BTF tile synthesis. Due to the *high dimensionality* and *enormous storage* of the BTF, it is hard and computationally very expensive to find a synthesis solution without any obvious seam. Unlike the color (RGB) texture synthesis involving only 3 dimensions, BTF synthesis usually involves up to thousands of dimensions. While the BTF exemplar for synthesis is usually small in spatial resolution (probably due to acquisition difficulty and high storage requirement), finding a solution without obvious seams over thousands of dimensions is even more difficult. To solve the problem, we present a novel method for synthesizing high-dimensional BTF tiles. It consists of four major sub-steps: corner sampling, edge synthesis, frame construction, and interior area synthesis. To make the synthesis tractable, we also perform the synthesis in a *compressed* domain.

2 Related Work

2.1 Bidirectional Texture Function

In computer graphics, the use of BTF involves the following processes: 1) acquisition, 2) compression, 3) synthesis, and 4) rendering.

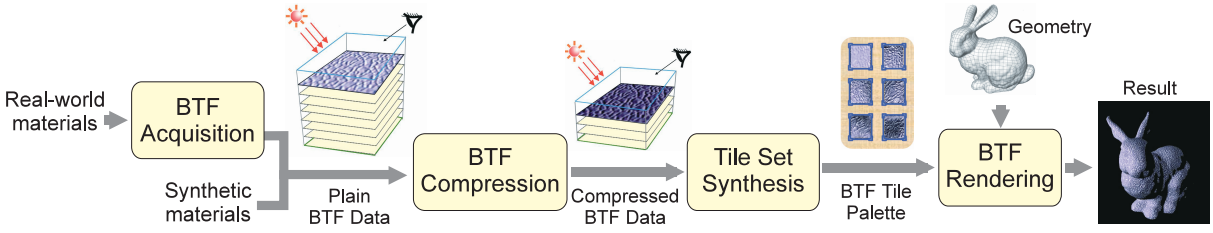

Fig. 1. Overview of the proposed framework.

BTF acquisition Dana et al. [12] were the first in capturing and modeling the BTF data from real-world materials. They built the first BTF database called *CUReT* [11]. Another database of higher-resolution and denser-sampled BTFs was recently collected by the University of Bonn [4, 36]. Furukawa et al. [19] devised an automatic method in capturing the BTF data from 3D models by using range cameras and reconfigurable camera array at the same time. Han and Perlin [23] developed a kaleidoscope-based capturing system for fast capturing of the BTF data without mechanical movement.

BTF compression Ginneken et al. [20] proposed to use texture histogram to correlate texture with viewing and irradiance changes. Leung and Malik [33] developed the 3D texton concept as clusters of filter output of textures under different viewing and lighting configurations. They further applied the method on the CUReT data for texture recognition. Suykens et al. [48] represented the BTF as spatially variant BRDF and applied the chained matrix factorization (CMF) to decompose it and render it using the GPU. On the other hand, the PCA method [10,26,56] and the multi-linear method [34, 53] were also used to compactly represent the high-dimensional and gigantic BTF data. Spherical harmonics [43] is another stream of research on compressing high-dimensional BTF. Inspired by the BRDF representation, Wong et al. [32, 60, 61] achieved compression in frequency domain using spherical harmonic transform.

BTF synthesis Before rendering the BTF data on a target geometry surface, the BTF has to be synthesized with reference to the target geometry. Liu et al. [35] were the first in applying texture synthesis methods [14, 15, 25, 29, 37, 52, 59, 62] to produce seamless BTF. Approximated geometry was first recovered using shape-from-shading and then served as a guidance for the BTF synthesis process. Tong et al. [51] improved this method and synthesized the BTF on arbitrary surfaces using the *k*-coherent search method. Zhou et al. [63] presented an interactive painting system to efficiently synthesize the BTF on arbitrary surfaces using graph-cut [29].

BTF rendering The BTF captures the surface appearance as well as its mesostructure; thus, it can greatly increase the surface realism in the rendering. Chen et al. [6] applied the BTF to render feather with a controllable parametric L-system. Sattler et al. [41] captured the mesostructure of fabric by using the BTF formulation and applied the compressed BTF data to render cloth. Sloan et al. [45] applied bi-scale decomposition on radiance transfer so as to add global transport effects to the BTF rendering. Wang et al. [57] proposed a real-time rendering framework for plant leaves using the spatially variant BRDFs along with subsurface scattering analysis.

2.2 *Wang Tiling*

The core of our BTF synthesis is Wang tiling [55]. The theory of Wang tiles can be traced back to the early 1960s when Wang [55] proposed non-periodic tiling of a plane. The tile set consists of a set of square tiles, known as Wang tiles, where edges of tiles are color-coded. In order to create a valid tiling of a plane by Wang tiles, all shared edges should have matched color. Grunbaum and Shepherd [21] examined this subject in depth, and presented the non-periodic tiling of a plane using a finite Wang tile set.

Stam [46] was the first in applying non-periodic Wang tiling to texture creation. Wang tiles were used as texture container for patterns such as water surface and caustic. Cohen et al. [9, 42] further investigated the use of Wang tiles in texture synthesis, and invented an automatic method to synthesize textures on Wang tiles. Stephen [7] later applied tiling to create animated flow pattern. Wang tiles were extended to contain flow information. Wei [58] devised a Wang tile arrangement scheme in texture memory so as to correct the texture filtering problem across tile images. Fu and Leung [18] later generalized the conventional Wang tiling mechanism, and made Wang tiling applicable to arbitrary topological surfaces. While [18] discusses only conventional texture tiling, this paper focuses on the more general BTF texture tiling and introduces a novel synthesis mechanism for making high-dimensional BTF tiles. Recently, Lagae and Dutré [31] invented an alternative Wang tile set by using colored corners instead of colored edges, while Kopf et al. [28] developed a recursive mechanism in Wang tiling.

3 BTF Tile Synthesis

The BTF is a six-dimensional function capturing the surface reflectance of a 2D texture under variable illumination and viewing configurations:

$$\text{The BTF data space} = V \times L \times T\ ,$$

where V is the viewing domain $\{\theta_v, \phi_v\}$, L is the lighting domain $\{\theta_l, \phi_l\}$, and T is the texture (spatial) domain (u, v). Azimuth ϕ_v (and ϕ_l) spans $[0, 2\pi)$ while altitude θ_v (and θ_l) spans $[0, \pi/2]$ over a hemisphere. Thus, an acquired BTF sample can be stored in the form of a six-dimensional table of pixels, or equivalently a four-dimensional array of texture images. Starting with a different point of view, Wong et al. [60] independently proposed the same 6D formulation known as the apparent BRDF of pixels (ABRDF) in the context of image-based rendering.

3.1 *BTF Compression*

Due to the enormous data volume of the BTF, processing of a plain BTF is computationally very expensive. To reduce the computational cost, we propose to synthesize the BTF in the

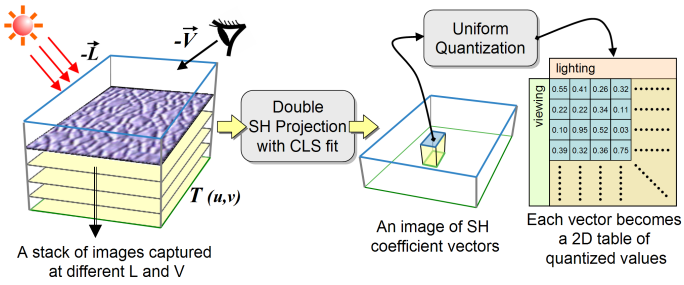

Fig. 2. Compressing the BTF data using the double spherical harmonic projection with the constrained least square method.

compressed domain. First, we apply the spherical harmonic (SH) transform on both the lighting (θ_l, ϕ_l) and viewing (θ_v, ϕ_v) dimensions, i.e., double SH projection [43]. The spatial dimension (u, v) is left untouched as the subsequent BTF synthesis will perform spatial segmentation. Hence, the outcome of this encoding is a 2D array of SH matrices. Each element (u, v) maintains a $k_l \times k_v$ matrix of the SH coefficients, where k_l and k_v are the number of SH coefficients kept for lighting and viewing dimensions, respectively (see Figure 2).

The double SH projection we employed is not obtained by spherical integration because the BTFs are normally acquired (sampled) over the upper hemisphere only, instead of the full sphere. Rather than using least square fitting as in [44], we applied the *constrained least square (CLS) method* to estimate the *noise-proof* SH coefficients [32]. It can be shown that SH coefficients with large magnitude are very sensitive to noise introduced by modern quantization and compression techniques. The CLS method noise-proves the SH coefficients by suppressing their magnitudes. Although we applied spherical harmonic transform to compress the BTF data in our current implementation, it may also be replaced by other sophisticated representations such as PCA or TensorTextures [53] because our framework is independent of the compression scheme being adopted. However, since the constrained SH projection we currently employed is noise-resistant, we can minimize the visual artifact introduced by the quantization process that follows.

3.2 *Synthesizing the BTF Tiles*

This subsection introduces a novel method for synthesizing high-dimensional BTF on rectangular tiles that can facilitate the following Wang tiling. Cohen et al. [9] synthesized texture patterns on tiles using the diamond-shaped samples as depicted in Figure 3. This method first extracts a set of diamond-shaped samples from the input texture. Then, it creates seamless tiles by merging four diamond-shaped samples side-by-side using dynamic programming [14] or standard graph-cut technique [29]. Cutting paths are traced in the overlapping regions between diamond-shaped samples to avoid seam. One advantage of this method is its fast computation as only small number of diamond-shaped samples is required.

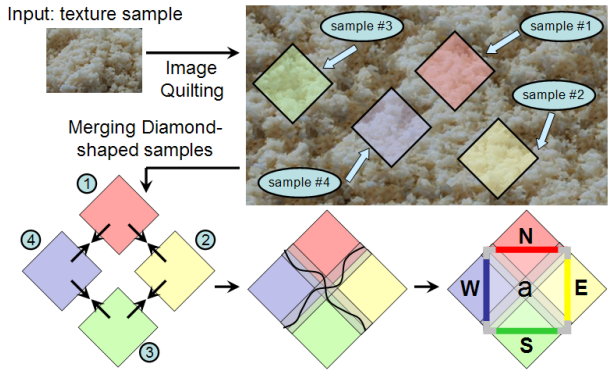

Fig. 3. Synthesizing seamless tile by merging four diamond-shaped samples [9].

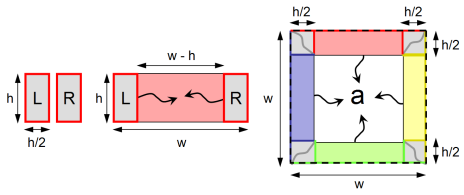

Fig. 5. Sizes (in pixel unit) of corner samples, edge samples, and a BTF tile (from left to right).

However, when applying it to high-dimensional BTF, we found that visible seams always pop up along the cutting paths on the synthesized tile images. The major reason is due to the high-dimensionality of BTF. Unlike the RGB texture which contains only 3 layers, the SH-projected BTF contains $k_l \times k_v \times 3$ coefficient layers, where k_v and k_l could be as small as 25 in order to achieve acceptable rendering quality. Hence, it is very difficult to always guarantee a seamless cutting path across all coefficient layers, especially when the cutting path is long.

Therefore, the key is to avoid long cutting paths. To address this issue, we divide the tile synthesis process into the following four major steps: corner sampling, edge synthesis, frame construction, and synthesis within frame. This idea was inspired by the formulation of the Poisson disk tiles [30] invented by Lagae and Dutré. Figure 4 illustrates these four steps. Our goal is to increase the chance of obtaining seamless merging. Note that the tile synthesis process is actually working on a volume with the third dimension spans the SH coefficients. For simplicity, we ignore this third dimension throughout the discussion below.

1. Corner Sampling To generate a tile set, we first extract a set of small *corner samples* from the raw BTF data. For each color-coded edge in the tile set, we extract a pair of rectangular corner samples from the BTF data volume. Each rectangular corner sample has size $h/2 \times h$, provided $(w - h) \times h$ is the size of an edge to be synthesized and $w \times w$ is the size of a resultant tile, see Figure 5. Note that the choice of the edge height, h, corresponds to the feature size in the BTF pattern. Since red and green (blue and yellow) color-coded edges are horizontal (vertical) in nature, we pick left and right (top and bottom) rectangular corners for them. Furthermore, when extracting corner samples from the input BTF volume, we try to maintain the similarity between corresponding corner samples so that by the time they are merged in step 3, we can ensure a seamless cutting path between corner samples in the combined tile frame.

To be precise, the word "similarity" refers to the seamlessness of the cutting paths (from graph-cut) between the corresponding corner samples that will be merged in step 3. In other words, we should find a set of corner samples in step 1 so that the cutting paths to be applied in step 3 are of very low matching error. Note that it is relatively easy to find such a set because the cutting paths here are much shorter as compared to the case of

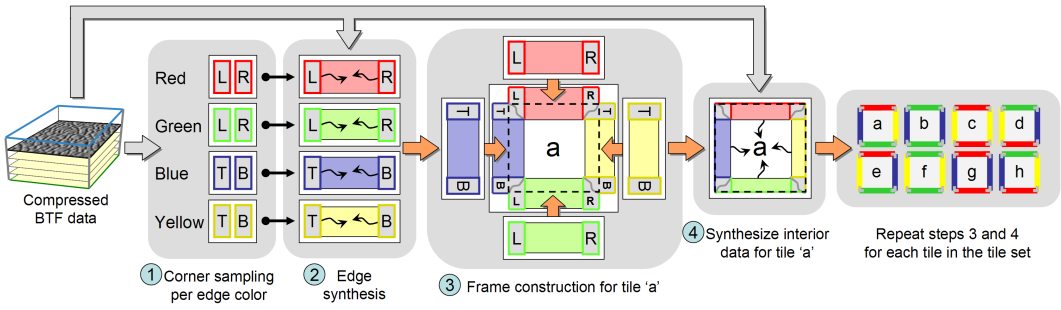

Fig. 4. Four steps for synthesizing a BTF tile: Corner sampling → Edge synthesis → Frame construction → Interior area synthesis.

diamond-based method.

2. Edge Synthesis The second step is to connect pairs of corner samples by synthesizing the $(w-h)\times h$ pixels in between them. The mechanism is achieved by applying the graph-cut algorithm iteratively to extract patches from the BTF volume and fill up the blank area in between the corner samples. Figure 6(a) shows the iterative synthesis of an edge. Also, we should avoid rotating the T and B corner samples as the SH coefficients are not rotation-invariant without proper rotation computation.

Note that in applying the graph-cut algorithm, we keep track of the similarity error (seamless-ness) for all pixels inside the blank area to be filled. All similarity errors are set to infinity at the beginning. When a candidate patch from the raw BTF data is considered, we first overlap it with the existing filled area, and apply graph-cut to find a cutting path between the patch and the filled area. Note that the error metric we used is a weighted sum of square difference between the SH coefficient vectors from the patch and the filled area. After that, the total error along the cutting path are summed and compared against the total error currently inside the fillable area to check if there is any improvement. Hence, if the candidate patch is good enough to be applied to fill the edge sample, the per-pixel similarity errors inside the filled area will be updated accordingly based on the errors previously found along the cutting path. This process is repeated until the whole area has been filled and the total (also individual) per-pixel similarity error falls below a certain user-defined threshold (as well as an individual threshold for each pixel).

3. Frame Construction After synthesizing all edges, we can construct the tile frame as illustrated in step 3 of Figure 4. At each tile corner, there is an overlapping area of size $h/2\times h/2$. A cutting path in this region is determined by the path previously found in step 1; note that to measure the similarity (seamless-ness) among corner samples in step 1, we have already applied the graph-cut algorithm between corresponding corner samples. Note again that since the cutting paths we used here are much shorter than that in the diamond-based method, we can easily find seamless cutting paths in all our experiments, even for the high-dimensional BTF. Once all edges are joined, the extra area outside the dotted line are clipped away.

4. Interior Area Synthesis Finally, the graph-cut filling algorithm (as illustrated in the edge synthesis part) is applied again to iteratively synthesize the interior area within the constructed frame. Figure 6(b) shows the iterative synthesis of the interior area.

The reasons why this synthesis method can handle high-dimensional BTF are that it *relaxes the constraints* (shortens the cutting paths) and *increases the choices* of samples (not just restricted to four selected diamond-shaped samples). The pair of corner samples corresponding to the same color-coded edge need not be paired up horizontally or vertically in the input BTF data volume, so we have more choices for our cornerstones. As our synthesis method does not limit us to choose four diamond-shaped samples, we can have numerous choices of samples, and hence, it substantially increases the chance to obtain seamless cutting paths. In all our experiments, the proposed method can successfully determine seamless cutting paths.

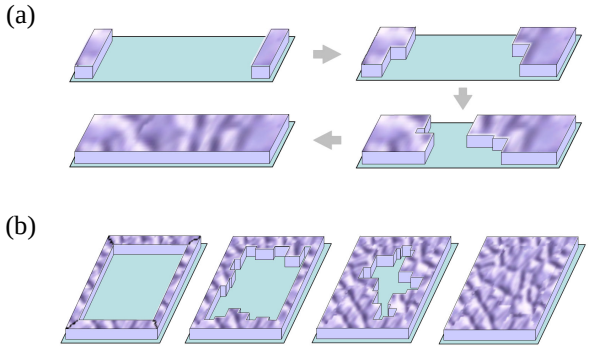

Fig. 6. (a) Edge synthesis and (b) interior area synthesis.

3.3 User-Controllable Tile Synthesis

In addition to the fully automatic tile synthesis method presented above, we also allow the users to interactively edit the features in order to fine tune the BTF tiles. To achieve this goal, we developed the user-controllable tile synthesis GUI as shown in Figure 7. Via this GUI, users can drag-and-drop features from the data sample onto the BTF tiles and constrain the BTF tile synthesis process [63] to preserve this user-edited content.

To illustrate how it works, we demonstrate how to synthesize tiles with this GUI. Note that the user can change the lighting and viewing on the synthesized BTF tile during the editing, but they are fixed in Figure 7 for simplicity.

1. Feature Extraction As shown in Figure 7(a), we first apply a user-guided image segmentation technique to extract features in the raw BTF, the "holes" in this particular example. After the segmentation, the GUI will highlight the extracted features in red color.

2. Corner Sampling Figure 7(b) presents the corner sampling process. These corner samples can be selected automatically or

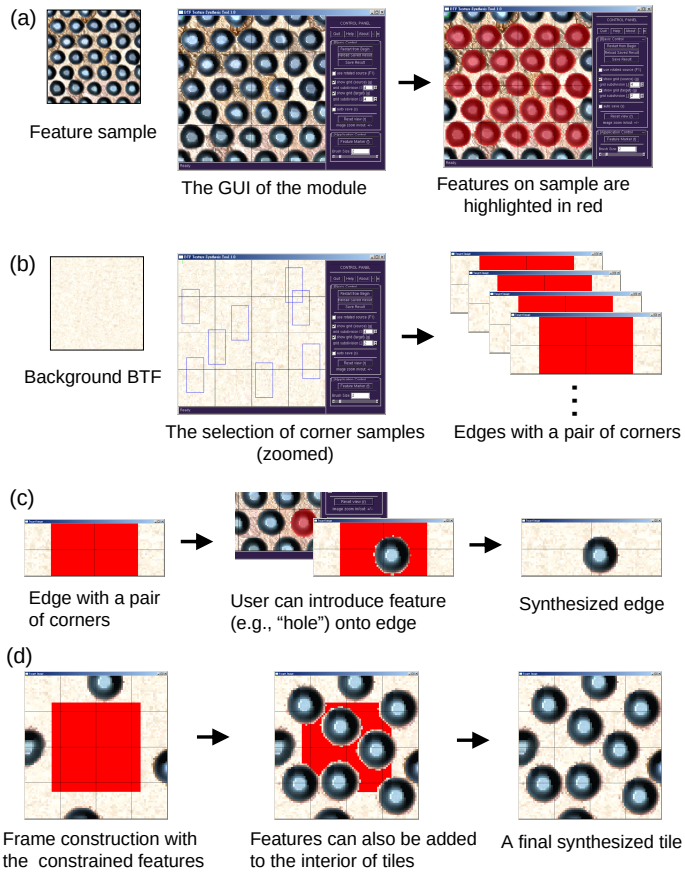

Fig. 7. Demonstration of our user-controllable tile synthesis GUI: (a) feature extraction, (b) corner sampling, (c) interactive edge synthesis, and (d) interactive interior area synthesis.

manually. To match the color tone of the background BTF with the color of the features, the user can optionally tune the hue and saturation of the background BTF. As the resultant color is basically a linear combination of SH coefficients, we can tune the color tone by tuning its SH coefficients directly. Note that background BTF refers to the BTF (usually low frequency) already synthesized on the BTF tiles.

3. Edge Synthesis After generating the corner samples, we synthesize the edges. As depicted in Figure 7(c), we can interactively drag-and-drop previously segmented features onto the edge using the GUI and apply a constrained texture synthesis to fill the blank area (highlighted in red). Since each synthesized edge is ultimately divided into two halves during the frame formation (see Figure 4), we can match individual features even they cross the edges of two tiles.

4. Interior Area Synthesis Figure 7(d) shows a resultant tile frame constructed using the edge generated in Figure 7(c). The lower half of the edge corresponds to the top edge of the frame. Similarly, we can also introduce features into the interior of a tile, before applying the constrained texture synthesis to fill its interior blank area (highlighted in red).

With this GUI, we allow the user to take *full control* on the generation of the desired BTF in practical applications. Features can lie on the edges of tiles and are still matchable after the tiling. Sometimes, it could be hard to synthesize features with regular structures that are relatively large compared to the size of tiles, but with this GUI, we can constrain the features on edges and tiles, and thus can preserve features originally in the raw BTF data.

4 Tiling and Rendering

4.1 *Tiling*

Before dressing a geometry surface with the BTF tiles, we have to parameterize the geometry surface. Techniques concerning surface parameterization [3, 13, 17] were studied intensively in recent years. Examples include the shell map structure [16, 39] on various 3D models as well as the PolyCube-Map method [49] for efficient texture mapping. In our current implementation, we adopt the PolyCube-Map method to install a quad-based structure on the input meshes. However, we have to emphasize that any low-distortion surface parameterization can be adopted in our tileable BTF framework.

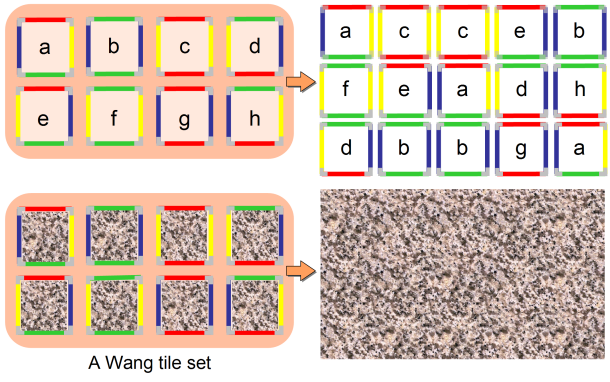

Fig. 8. A set of Wang tiles (left) and a valid Wang tiling (right).

After the surface parameterization, Wang tiling [9, 18, 46, 58] is employed to dress up the geometry surfaces with the matchable BTF tiles, and this useful tiling tool also helps to formulate the BTF tile arrangement onto the parameterized surfaces. As illustrated by Figure 8, Wang tiles have color-coded edges and the matching of edge color between tiles leads to a match in the texture pattern contained in the tiles. Thus, we can create a seamless texture pattern non-periodically on the tiled region.

4.2 *Rendering*

Once we complete the above offline pre-computation, the rendering of BTF tiles on geometry surfaces is straightforward. Figure 9 illustrates the basic rendering mechanism: 1) the TBN transform [2, 38] and 2) the double SH reconstruction. The abbreviation TBN refers to the local tangent space on object surface, formed by tangent ("T"), binormal ("B"), and normal ("N").

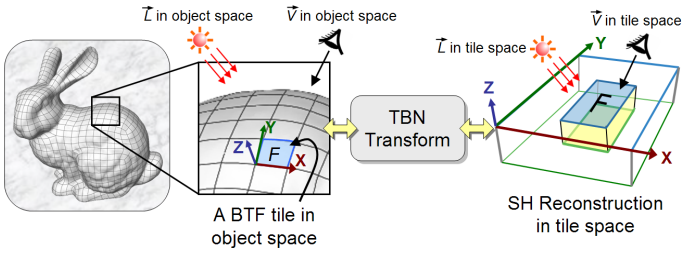

Fig. 9. Transformation between tile space and object space.

Local Illumination To ensure the SH coefficients to be reusable for tiling different surfaces, they have to be encoded in the local tile space. Hence, the transformation of both light and viewing vectors from the object space to the local tile space is required for computing the local illumination. It is important to note that the tangents and binormals are not defined by the local curvature as in general TBN transform [2, 38]. Rather, we align them with the local surface parameterization grid so that the transformed light and viewing vectors confirm to the coordinate system of the tile space as defined in the raw BTF samples. After the transformation, we look up the corresponding quantized SH coefficients, unquantize them, perform double SH reconstruction (inverse transform) to obtain the reflectance, and compute the final pixel color.

Distant Environment To render the BTF-tiled objects illuminated in a distant environment, our system currently supports two approaches: 1) importance sampling [1] and 2) frequency-domain approach [43]. The importance sampling approach approximates the illumination of distant environment using a limited number of directional lights, say 200 lights. Efficient sampling algorithms have been proposed recently [1,8,54]. We used the Spherical Q^2-Tree sampling technique [54] in generating the samples. For each sample (directional light), we render an image by local illumination. The final result is produced by summing the rendering results from multiple passes of such local illumination.

The frequency-domain approach first encodes the distant environment as a SH coefficient vector. The pixel color is computed by performing the inner product between the SH matrix (BRDF), $\vec{c}_\rho$, and the SH vector (distant environment), $\vec{c}_e$, in frequency domain. However, a rotation, $\mathbf{R}$, on $\vec{c}_e$ has to be carried out at each pixel, as the local tile space BTF ($\vec{c}_\rho$) and the environment ($\vec{c}_e$) are SH-encoded in two different coordinate systems. Further note that the CLS we used actually encodes the BRDF in the hemispherical SH domain; the lower hemispheres of the basis functions are all zeros. An autocorrelation matrix, $\mathbf{A}$, is required to convert the full-sphere SH coefficients of distant environment before performing the inner product. Readers are referred to [32] for the mathematical details. In matrix form,

$$\text{the final pixel radiance } p \;=\; \vec{c}_\rho(\vec{v})^T[\mathbf{A}][\mathbf{R}]\vec{c}_e\,, \qquad (1)$$

where $\vec{v}$ is the viewing direction corresponding to the pixel in tile space; $\vec{c}_\rho(\vec{v})$ is a k_l-dimensional vector reconstructed given the current viewing direction $\vec{v}$; $\mathbf{A}$ is a $k_l \times k_l$ matrix with elements,

$$a_{ij} = \int_{\Omega_H} y_i(\vec{s})y_j(\vec{s})d\vec{s}\,, \qquad (2)$$

for the integral of two spherical harmonic basis functions over the hemisphere Ω_H. This matrix of integrals can be precomputed numerically (see Appendix for the definition of y_i). $\mathbf{R}$ is a $k_l \times k_l$ matrix that rotates the environment coefficient vector $\vec{c}_e$ to align with $\vec{c}_\rho$ in local tile space. The elements of matrix $\mathbf{R}$ can be determined as described in [22]. However, a more efficient way to rotate $\vec{c}_e$ is to compute it analytically [27, 40]. In our current implementation, we compute the reflected radiance using this frequency-domain approach and the rotation of $\vec{c}_e$ is computed analytically. In addition, our current implementation does not render the macro-scale shadow due to the object. To account for this kind of shadow, the bi-scale radiance transfer [45] can be employed.

TABLE I

PROPERTIES OF RAW BTF DATA SETS IN OUR EXPERIMENTS.

BTF	Original Spatial Image Resolution	Light Sampled	View Sampled	Total Size
FLOORTILE	800×800	81	81	12.6GB
IMPALLA	256×256	81	81	1.29GB
WRINKLES	128×128	50	60	147MB
HOLES	128×128	51	51	128MB
REACTDIFFUSE	128×128	81	81	321MB

TABLE II

PERFORMANCE OF DATA COMPRESSION (INCLUDING DOUBLE SH PROJECTION AND UNIFORM QUANTIZATION).

Resolution × light × view	No. of Coefficients		Timing (h:m:s)	Compress Rate
	View(k_v)	Light(k_l)		
$128 \times 128 \times 81 \times 81$ (321MB)	25	25	10 : 13	9.53%
	36	25	12 : 41	13.73%
$256 \times 256 \times 81 \times 81$ (1.29GB)	25	25	17 : 47	9.53%
	36	25	20 : 51	13.73%
$800 \times 800 \times 81 \times 81$ (12.60GB)	25	25	2 : 51 : 29	9.53%
	36	25	3 : 22 : 00	13.73%

5 IMPLEMENTATION AND RESULTS

5.1 Data Sources

We have tested the proposed framework on both real and synthetic BTF data. The real data employed in our experiments is mainly from the Bonn BTF database [4, 41]: FLOORTILE and IMPALLA. These BTF data samples are captured with 81 sample lighting and 81 sample viewing directions, resulted in a set of total 6,561 sample images in RGB format. In addition to the Bonn BTF database, we also used two BTF data obtained from MSRA [35, 51]: WRINKLES and HOLES, and another synthetic BTF data produced ourselves: REACTDIFFUSE. The total size of the BTF data sets ranges from 128 MB to 12.6 GB. Table I summarizes the properties of all raw BTF data used in our experiments.

5.2 Compact Data Representation

Since the plain BTF data samples could be too large to be fit into the conventional PC memory, our data compression engine performs the double SH projection and the uniform quantization in a scanline-wise fashion. This approach can greatly reduce the amount of disk I/O and optimize the data processing speed. Then, each coefficient is quantized to an 8-bit integer so that we can hold all quantized SH coefficients in memory afterwards. Table II shows the timing and performance statistics of the compression including the double SH projection and the uniform quantization tested on a PC with Pentium IV 3.2 GHz CPU and 1 GB memory. Different number of SH coefficients are tested. Column "Timing" shows the time to perform double SH projection and uniform quantization. It increases as the raw BTF data size or the number of SH coefficients employed increases. "Compression ratio" is measured with respect to the raw BTF data size. It is mainly affected by the number of SH coefficients employed and the original sampling rates along lighting and viewing.

5.3 Tile Synthesis

In our experiments, the time needed to arrange the BTF tiles on geometry surfaces is negligible as compared to the time needed to synthesize the BTF tiles. Fortunately, we only need to perform the synthesis once in offline. Table III lists the total time for synthesizing each BTF tile set tested in our experiment (column "Total Synthesis Time"). The statistics are recorded on a PC with Pentium IV 3.2 GHz CPU and 1GB memory.

After the BTF tile synthesis, not all BTF elements (BTFels) could be finally used in a tile set while many other BTFels could be repeatedly used in different BTF tiles. Keeping all SH coefficients for each BTF tile is wasteful. To efficiently store the data, we construct the BTFel table which stores only those referenced BTFels without duplication. Then, for each BTF tile, we only store a 2D array of BTFel index pointing to elements in the BTFel table, instead of directly storing the full SH coefficients. By this means, storing those synthesized BTF tiles in GPU memory can be highly efficient. Column "Percentage of Referenced BTFels" in Table III lists the percentage of BTFels referenced, hence stored, for each synthesized BTF tile set in our experiments, and also the amount of GPU memory needed to store the BTFels together with the uniform quantization coefficients. The ratio (the 4th column) is measured with respect to the total number of texels in the raw BTF data. In addition, note that all these BTF tile sets have 96 BTF tiles and use 25×25 lighting and viewing SH coefficients, except for the FLOORTILE data set, which has only 16×16 coefficients so that it can be fitted into the texture memory.

5.4 Results

To demonstrate the renderings of BTF-dressed surface under different lighting and viewing conditions, two kinds of lighting configurations are used in our experiments: local illumination and distant environment lighting. Figures 10-12 present the rendering results of BTF-dressed objects illuminated by a point light source and as viewed from different orientations. From Figures 10 to 12, the BTF tile sets shown are IMPALLA, HOLES, and WRINKLES. In each figure, the same BTF tile set is repeatedly used to seamlessly dress three objects, BUNNY (the top row), 3-HOLES (the middle row) and LAURANA (the bottom row). For each generated image, two boxed regions are blown up for inspection.

Figure 13 shows the distant environment lit BUNNY. In the upper row, BUNNY is dressed with the BTF tile set WRINKLES and lit by GRACE. The lower row shows the BUNNY dressed with REACTDIFFUSE lit in GALILEO. Importance sampling approach is used in order to render the macro-scale shadow due to the object. In particular, we applied the Spherical Q^2-Tree [54] to generate the samples (directions). For all results in Figure 13, 50 samples are used and the corresponding 50 locally illuminated images are summed to generate the final result. The total time taken to render a 640×480 image are 3.2 min. and 4.5 min. for WRINKLES-dressed and REACTDIFFUSE-dressed BUNNY, respectively.

With the appearance-geometry decoupling, we can see that the same BTF tile set can be repeatedly used to dress up surfaces seamlessly even the geometries are substantially different.

5.5 Limitations

One major limitation of our method is the BTF being synthesized must be more or less "isotropic" spatially. If the BTF exhibits an obvious anisotropic pattern, extra effort is needed to maintain the anisotropic behavior over the object surface. This is because our tiling approach only ensures the matching locally among tile edges.

Another restriction is due to the nature of the BTF. Unlike simple color textures (diffuse reflection) that are independent of viewing and lighting directions, there is always an intrinsic orientation associated with the BTF as the BTF is acquired under a specific orientation (coordinate framework). Note that the BTF captures all frequency components including specular component as well as the diffuse component. During the synthesis, we have to match the pixel values together with the orientation (the spatial ordering of the pixels). Therefore, all synthesized BTF tiles must be laid on the tile surface with a consistent orientation, unlike the color texture tiles that can be randomly oriented provided the edges are matched.

The proposed framework relies on a proper quad-based parameterization over the object surface. If the parameterized quads are not maintained with similar sizes or the quads are over-distorted, the final dressing may be malformed. This is mainly due to the fact that BTF tiles are synthesized in a single scale and in a square shape.

6 CONCLUSION

In conclusion, this paper presents a novel and modular framework for efficiently applying the BTF on geometry surfaces. Given M different surfaces and N different BTF data samples, if we apply conventional BTF approach, we need to perform the BTF synthesis $M \times N$ times so as to produce all different BTF dressings on the M different surfaces. In contrast, with the appearance-geometry decoupling, the proposed framework only needs to perform the synthesis process N times so as to generate N sets of BTF tiles corresponding to the N BTF inputs. Once these tile sets are constructed, we do not need to change them any more. They can be repeatedly used to dress up M or more 3D models without retiling a model or resynthesizing a tile set. In this way, the BTF becomes highly reusable. Game developers can stock a library of BTF tiles and conveniently apply them to dress up different models in a highly cost-effective manner.

To make this modular framework practical, we also introduce an original tile synthesis algorithm for synthesizing the BTF tiles. It divides the BTF tile synthesis into four sub-steps: corner sampling, edge synthesis, frame construction, and interior area synthesis. Such approach relaxes the constraints (by avoiding long cutting paths) and increases the choices of samples, in order to maximize the chance of synthesizing seamless tiles for the high-dimensional BTF.

Acknowledgments

In this research work, we would like to thank MSRA and the University of Bonn for their high-quality BTF data, Paul Debevec for the HDR environment maps, Liang Wan and Guangyu Wang for helping us to prepare the companion video, Marco Tarini of the Visual Computing Lab, ISTI/CNR, Pisa, for the

TABLE III
STATISTICS OF SYNTHESIZED BTF TILE SETS.

	Tile Resolution	Total Synthesis Time (min.)	Percentage of Referenced BTFels	GPU memory needed (in MB)
FLOORTILE	128×128	33	37.86%	186.10
IMPALLA	256×256	51	49.92%	61.36
WRINKLES	64×64	15	76.92%	23.64
HOLES	100×100	25	34.86%	10.72
REACTDIFFUSE	64×64	16	55.01%	16.91

PolyCube-Mapped models, and Yu-Wing Tai for his advice in implementing the graph-cut algorithm. This project is supported by the Research Grants Council of the Hong Kong Special Administrative Region, under RGC Earmarked Grants (Project No. HKUST612706 and CUHK416806), and affiliated with the CUHK Virtual Reality, Visualization and Imaging Research Centre as well as the Microsoft-CUHK Joint Laboratory for Human-Centric Computing and Interface Technologies.

REFERENCES

[1] Sameer Agarwal, Ravi Ramamoorthi, Serge Belongie, and Henrik Wann Jensen. Structured importance sampling of environment maps. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):605–612, 2003.

[2] James F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH 78 Proceedings)*, volume 12, pages 286–292, August 1978.

[3] Ioana Boier-Martin, Holly Rushmeier, and Jingyi Jin. Parameterization of triangle meshes over quadrilateral domains. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 193–203, 2004.

[4] Bonn BTF database. University of Bonn. Webpage: `http://btf.cs.uni-bonn.de`.

[5] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflection functions from surface bump maps. In *Computer Graphics (SIGGRAPH 87 Proceedings)*, volume 21, pages 273–281, July 1987.

[6] Yanyun Chen, Yingqing Xu, Baining Guo, and Heung-Yeung Shum. Modeling and rendering of realistic feathers. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):630–636, 2002.

[7] Stephen Chenney. Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242, 2004.

[8] Jonathan Cohen and Paul Debevec. LightGen HDRShop plugin, 2001. `http://gl.ict.usc.edu/HDRShop/lightgen/lightgen.html`.

[9] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):287–294, 2003.

[10] Oana G. Cula and Kristin J. Dana. Compact representation of bidirectional texture functions. In *CVPR: Computer Vision and Pattern Recognition, 2001*, pages 1041–1047, 2001.

[11] CUReT. Columbia-Utrech reflectance and texture. Webpage: `http://www.cs.columbia.edu/CAVE/curet`.

[12] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and texture of real world surfaces. *ACM Transactions on Graphics*, 18(1):1–34, January 1999.

[13] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, ACM, pages 173–182, 1995.

[14] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, ACM, pages 341–346, 2001.

[15] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, 1999.

[16] Gershon Elber. Geometric texture modeling. *IEEE Computer Graphics and Applications*, 25(4):66–76, July/August 2005.

[17] Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 157–186. Springer-Verlag, 2005.

[18] Chi-Wing Fu and Man-Kang Leung. Texture tiling on arbitrary topological surfaces using Wang tiles. In *EGSR 2005: Eurographics Symposium on Rendering 2005*, pages 99–104, June 2005.

[19] Ryo Furukawa, Hiroshi Kawasaki, Katsushi Ikeuchi, and Masao Sakauchi. Appearance based object modeling using texture database: acquisition, compression and rendering. In *EGRW 2002: Proceedings of the 13th Eurographics Workshop on Rendering*, pages 257–266, 2002.

[20] Bram Van Ginneken, Jan J. Koenderink, and Kristin J. Dana. Texture histograms as a function of irradiation and viewing direction. *International Journal of Computer Vision*, 31(2-3):169–184, 1999.

[21] Branko Grünbaum and Geoffrey C. Shephard. *Tilings and patterns*. W. H. Freeman & Co., 1986.

[22] Robin Green. Spherical harmonic lighting: The gritty details. In *Proc. of Game Developer Conference 2003 (GDC2003)*, March 2003.

[23] Jefferson Y. Han and Ken Perlin. Measuring bidirectional texture reflectance with a kaleidoscope. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):741–748, 2003.

[24] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In *Proceedings of ACM SIGGRAPH 93*, Annual Conference Series, ACM, pages 165–174, 1993.

[25] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of SIGGRAPH 2001*, Annual Conference Series, ACM, pages 327–340, August 2001.

[26] Pun-Mo Ho, Tien-Tsin Wong, and Chi-Sing Leung. Compressing the illumination-adjustable images with Principal Component Analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(3):355–364, March 2005.

[27] Joseph Ivanic and Klaus Ruedenberg. Rotation matrices for real spherical harmonics. direct determination by recursion. *Journal of Physical Chemistry*, 100(15):6342–6347, 1999.

[28] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive Wang tiles for real-time blue noise. *ACM Transactions on Graphics (SIGGRAPH 2006)*, 25(3):509–518, 2006.

[29] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):277–286, 2003.

[30] Ares Lagae and Philip Dutré. A procedural object distribution function. *ACM Transactions on Graphics*, 24(4):1442–1461, October 2005.

[31] Ares Lagae and Philip Dutré. An alternative for Wang tiles: colored edges versus colored corners. *ACM Transactions on Graphics*, 25(4):1442–1459, Oct 2006.

[32] Ping-Man Lam, Chi-Sing Leung, and Tien-Tsin Wong. Noise-resistant fitting for spherical harmonics. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):254–265, March/April 2006.

[33] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[34] Xinguo Liu, Yaohua Hu, Jingdan Zhang, Xin Tong, Baining Guo, and Heung-Yeung Shum. Synthesis and rendering of bidirectional texture functions on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):278–289, 2004.

[35] Xinguo Liu, Yizhou Yu, and Heung-Yeung Shum. Synthesizing bidirectional texture functions for real-world surfaces. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, ACM, pages 97–106, 2001.

[36] Gero Müller, Jan Meseth, Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Acquisition, synthesis and rendering of bidirectional texture functions. *Computer Graphics Forum*, 24(1):83–109, March 2005.

[37] Andrew Nealen and Marc Alexa. Hybrid texture synthesis. In *EGSR 2003: Eurographics Symposium on Rendering 2003*, pages 97–105, 2003.

[38] Mark Peercy, John Airey, and Brian Cabral. Efficient bump mapping hardware. In *Proceedings of ACM SIGGRAPH 97*, Annual Conference Series, ACM, pages 303–306, 1997.

[39] Serban D. Porumbescu, Brian Budge, Louis Feng, and Kenneth I. Joy. Shell maps. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3):626–633, 2005.

[40] David W. Ritchie and Graham J. L. Kemp. Fast computation, rotation and comparison of low resolution spherical harmonic molecular surfaces. *Journal of Computational Chemistry*, 20(4):383–395, 1999.

[41] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Efficient and realistic visualization of cloth. In *EGSR 2003: Eurographics Symposium on Rendering 2003*, pages 167–177, 2003. `http://btf.cs.uni-bonn.de/download.html`.

[42] Jonathan Shade, Michael F. Cohen, and Don P. Mitchell. Tiling layered depth images, 2002. Technical report, University of Washington, Seattle.

[43] François X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A global illumination solution for general reflectance distributions. In *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 187–196, July 1991.

[44] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):382–391, 2003.

[45] Peter-Pike Sloan, Xinguo Liu, Heung-Yeung Shum, and John Snyder. Bi-scale radiance transfer. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):370–375, 2003.

[46] Jos Stam. Aperiodic texture mapping. Technical Report R046, 1997. European Research Consortium for Informatics and Mathematics.

[47] Pei-hsiu Suen and Glenn Healey. Analyzing the bidirectional texture function. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 753, 1998.

[48] Frank Suykens, Karl vom Berge, Ares Lagae, and Philip Dutré. Interactive rendering with bidirectional texture functions. *Computer Graphics Forum*, 22(3), September 2003.

[49] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. PolyCube-Maps. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3):853–860, 2004.

[50] Xin Tong, Jiaping Wang, Stephen Lin, Baining Guo, and Heung-Yeung Shum. Modeling and rendering of quasi-homogeneous materials. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3):1054–1061, 2005.

[51] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):665–672, 2002.

[52] Greg Turk. Texture synthesis on surfaces. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, ACM, pages 347–354, 2001.

[53] M. Alex O. Vasilescu and Demetri Terzopoulos. Tensortextures: multilinear image-based rendering. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3):336–342, 2004.

[54] Liang Wan, Tien-Tsin Wong, and Chi-Sing Leung. Spherical Q2-tree for sampling dynamic environment sequences. In *Proceedings of Eurographics Symposium on Rendering 2005 (EGSR 2005)*, pages 21–30, June 2005.

[55] Hao Wang. Proving theorems by pattern recognition II. *Bell Systems Technical Journal*, 40:1–42, 1961.

[56] Hongcheng Wang, Qing Wu, Lin Shi, Yizhou Yu, and Narendra Ahuja. Out-of-core tensor approximation of multi-dimensional matrices of visual data. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3):527–535, 2005.

[57] Lifeng Wang, Wenle Wang, Julie Dorsey, Xu Yang, Baining Guo, and Heung-Yeung Shum. Real-time rendering of plant leaves. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3):712–719, 2005.

[58] Li-Yi Wei. Tile-based texture mapping on graphics hardware. In *SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 55–63, 2004.

[59] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, ACM, pages 355–360, 2001.

[60] Tien-Tsin Wong, Pheng-Ann Heng, Siu-Hang Or, and Wai-Yin Ng. Image-based rendering with controllable illumination. In *Proceedings of the 8-th Eurographics Workshop on Rendering (Rendering Techniques 97)*, pages 13–22, St. Etienne, France, June 1997. Springer-Verlag.

[61] Tien-Tsin Wong and Chi-Sing Leung. Compression of illumination-adjustable images. *IEEE Transactions on Circuits and Systems for Video Technology (Special issue on Image-based Modeling, Rendering and Animation)*, 13(11):1107–1118, November 2003.

[62] Steve Zelinka and Michael Garland. Jump map-based interactive texture synthesis. *ACM Transactions on Graphics*, 23(4):930–962, 2004.

[63] Kun Zhou, Peng Du, Lifeng Wang, Jiaoying Shi, Baining Guo, and Heung-Yeung Shum. Decorating surfaces with bidirectional texture functions. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):519–528, 2005.

APPENDIX

Constrained Least Square Estimation for SH

Given M samples of a spherical function f, $\vec{\beta} = \left[f(\vec{s}_1), \cdots, f(\vec{s}_M)\right]^T$, sampled at directions $\{\vec{s}_1, \cdots, \vec{s}_M\}$, we want to estimate an n-dimensional SH coefficient vector $\vec{c}$ that satisfies the following linear system,

$$\vec{\beta} = \mathbf{Y}\,\vec{c}, \tag{3}$$

where $y_i(\vec{s})$ is the i-th SH basis function evaluated at direction $\vec{s}$ and

$$\mathbf{Y} = \begin{bmatrix} y_1(\vec{s}_1) & \cdots & y_n(\vec{s}_1) \\ \vdots & \ddots & \vdots \\ y_1(\vec{s}_M) & \cdots & y_n(\vec{s}_M) \end{bmatrix}.$$

We can set up the following constrained least square (CLS) cost function,

$$J(\vec{c}) = \|\vec{\beta} - \mathbf{Y}\,\vec{c}\|^2 + \lambda(\vec{c}^T\vec{c} - E), \tag{4}$$

where E is the energy of function f and can be estimated from the sampled values of $f(\vec{s})$; $\lambda > 0$ is a parameter to search. Its constrained solution is given by

$$\vec{c} = \left(A + \lambda I_{n\times n}\right)^{-1}\vec{b}, \tag{5}$$

where $A = \mathbf{Y}^T\mathbf{Y}$; $I_{n\times n}$ is an $n \times n$ identity matrix; and $\vec{b} = \mathbf{Y}^T\vec{\beta}$. According to the constraint

$$\vec{c}^T\vec{c} = \vec{b}^T\left(A + \lambda I_{n\times n}\right)^{-2}\vec{b} \le E\ , \tag{6}$$

λ should satisfy the following inequality

$$\vec{b}^T\left(A + \lambda I_{n\times n}\right)^{-2}\vec{b} \le E\ . \tag{7}$$

If we set $\lambda = 0$, the least square solution is obtained. If we set $\lambda \to \infty$, $\vec{c}$ is a zero vector. As λ increases, the norm of $\vec{c}$ decreases monotonically. The goal is to find out the smallest value of λ such that (7) is satisfied. As J can be proved to be an increasing function of λ, we use an iterative approach to determine λ by first assigning an initial value of λ and then using a simple binary search to estimate a suitable value of λ based on (7). For detail proof, readers are referred to [32].

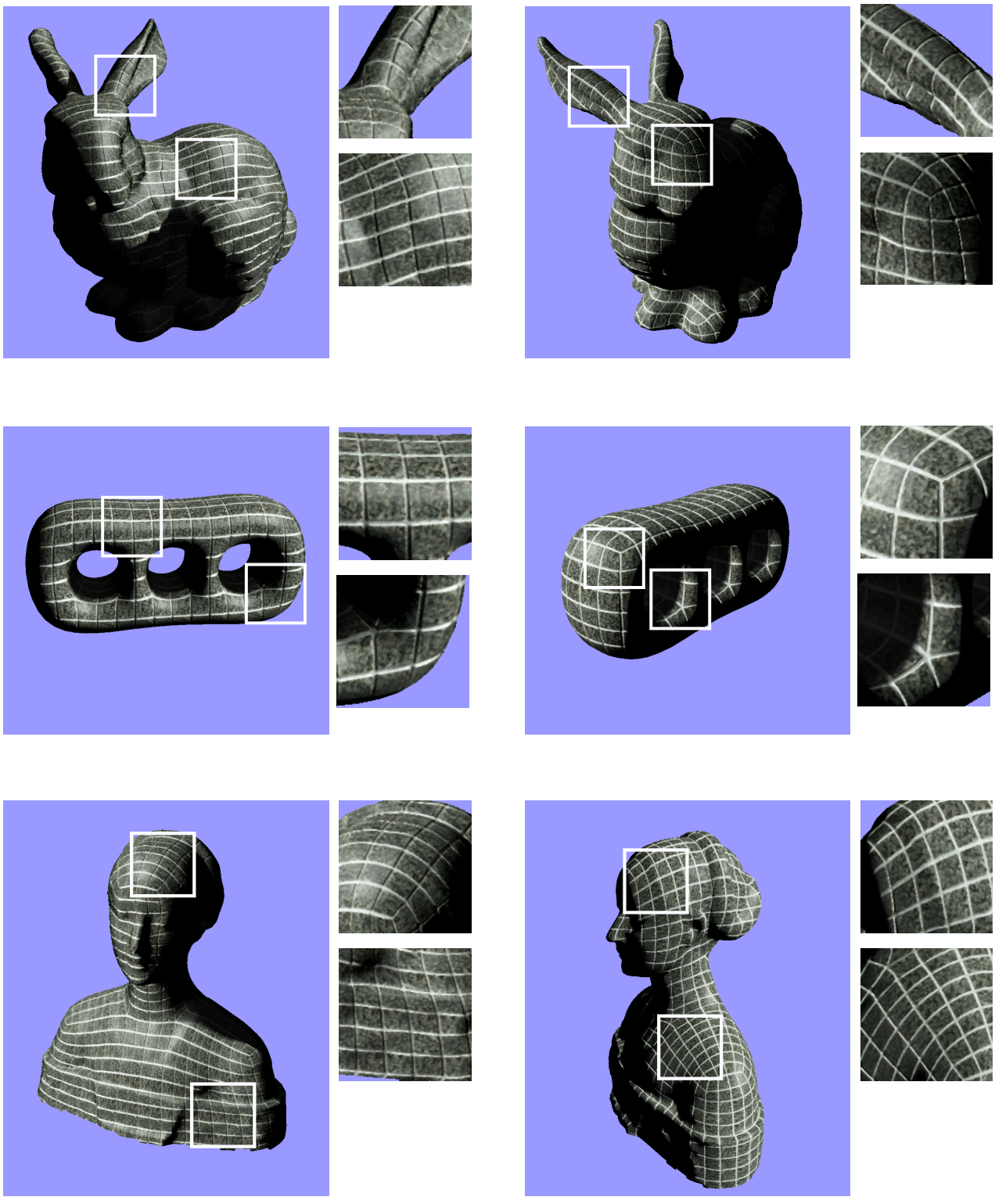

Fig. 10. The BTF tile set: IMPALLA.

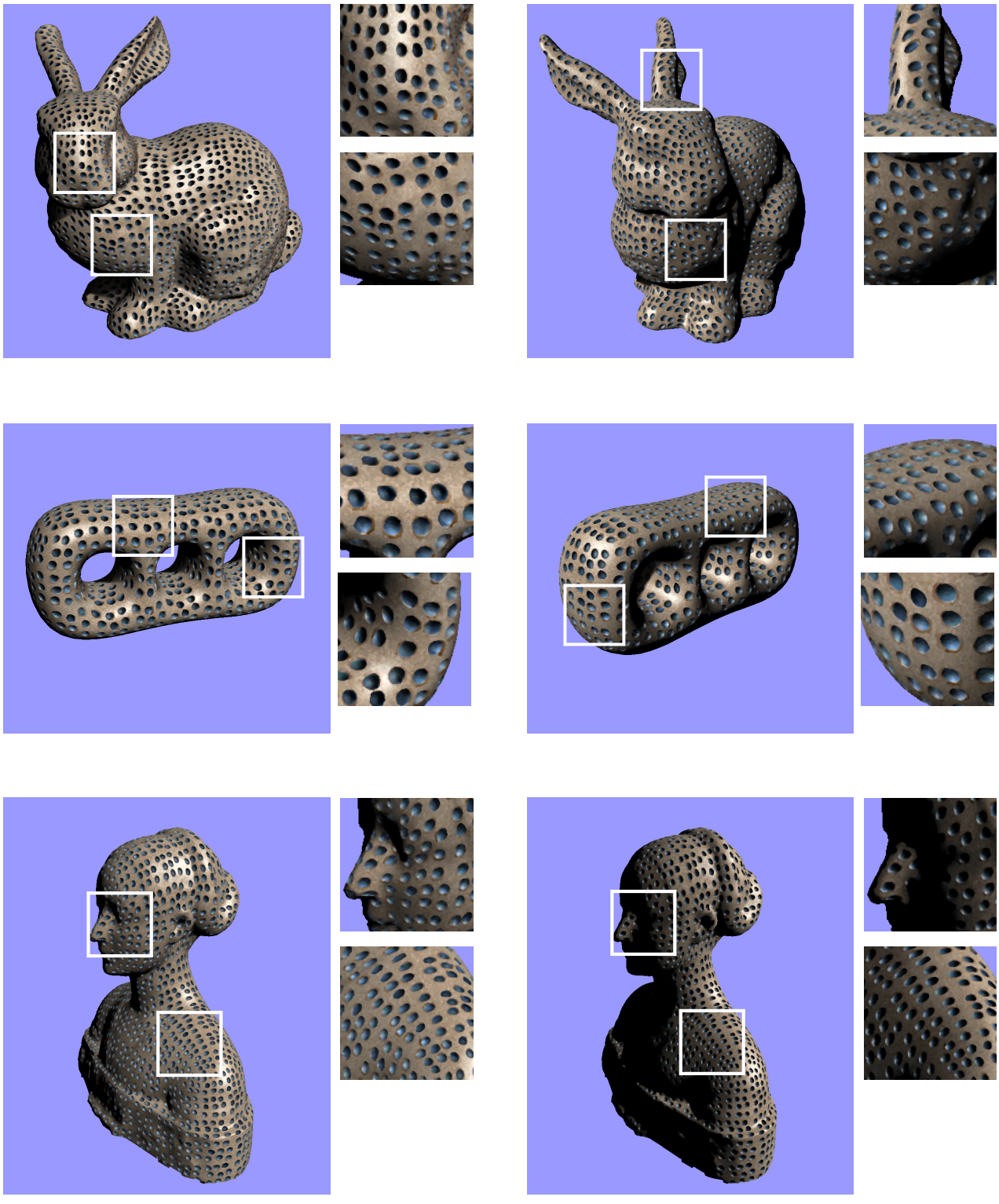

Fig. 11. The BTF tile set: HOLES.

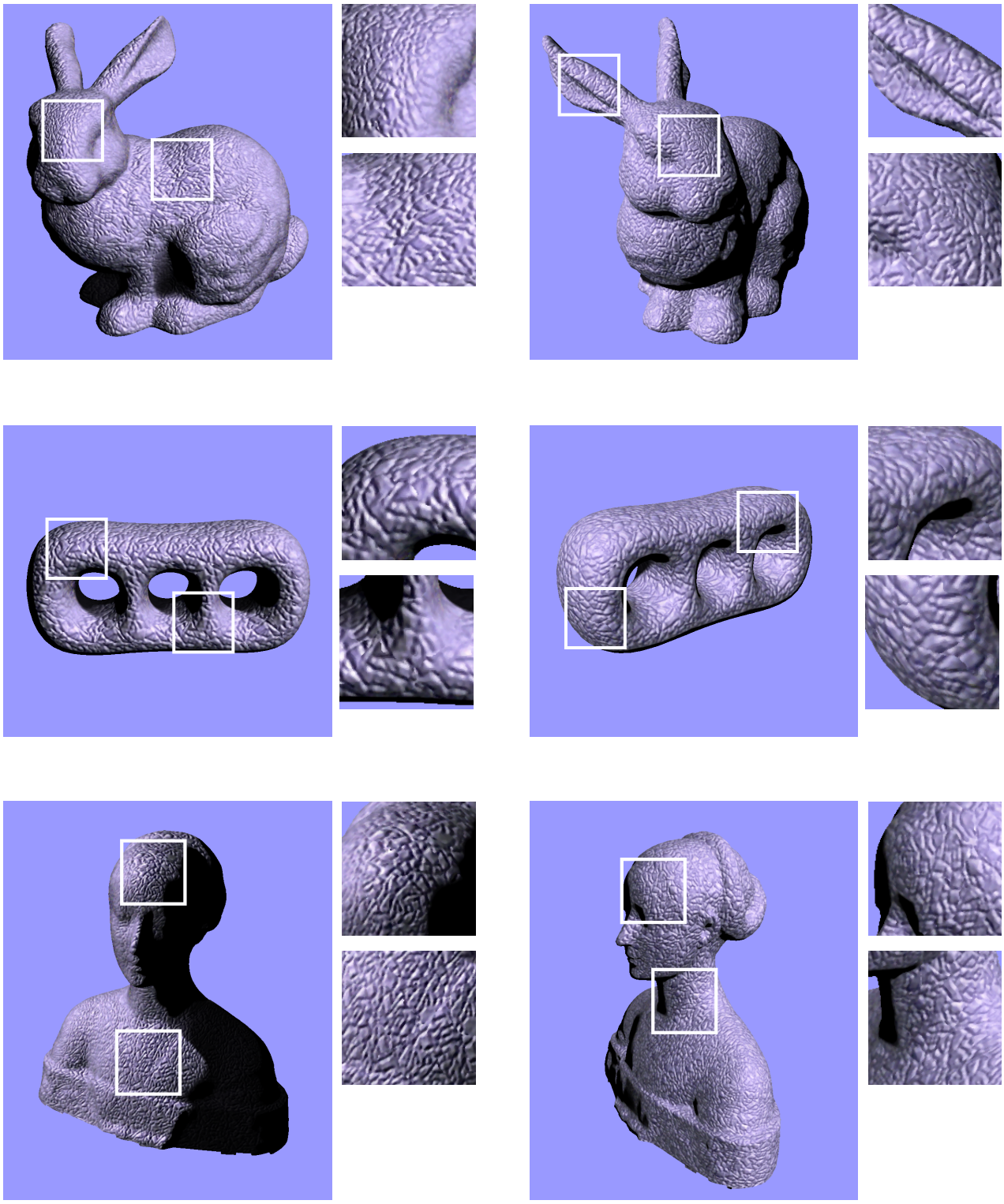

Fig. 12. The BTF tile set: WRINKLES.

Fig. 13. Distant environment lighting: a BTF-dressed BUNNY illuminated by the HDR environments. Upper row: WRINKLES lit by GRACE. Lower row: REACTDIFFUSE lit by GALILEO.