Eurographics Symposium on Rendering (2006)
Tomas Akenine-Möller and Wolfgang Heidrich (Editors)

Reconstruction of Volumetric Surface Textures for Real-Time Rendering

Sebastian Magda[1] and David Kriegman[2]

[1]Univ. of Illinois, Urbana-Champaign
[2]Univ. of California, San Diego

Abstract
Volumetric texturing is a popular technique for rendering rich 3-D detail when a polygonal surface representation would be ineffective. Although efficient algorithms for rendering volumetric textures have been known for years, capturing the richness of a real-life volumetric materials remains a challenging problem. In this paper we propose a technique for generating a volumetric representation of a complex 3-D texture with unknown reflectance and structure. From acquired reflectance data in the form of a 6-D Bidirectional Texture Function (BTF), the proposed algorithm creates an efficient volumetric representation in the form of a stack of semi-transparent layers each representing a slice through the texture's volume. In addition to negligible storage requirements, this representation is ideally suited for hardware-accelerated real-time rendering.

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

1. Introduction

Photorealism in computer graphics can only be achieved with a detailed representation of geometric and photometric properties of a scene. In general, the appearance of a surface material can be described with the *bidirectional texture function* (BTF) [DvNK99]. This 6-D function of light and viewing directions describes the apparent reflectance of a surface when reprojected onto a fixed plane. A real material can be sampled and then described with a BTF that implicitly encapsulates both the geometry and the reflectance. Although a BTF can be rendered directly, it requires efficient compression algorithms and rendering schemes for real-time applications. For many materials, this is the result of complex geometry, rather than the actual reflectance. Consider some examples in the lower right corner of Figure 2. Although these materials have relatively simple reflectance, their BTFs will be very complex due to geometry alone. By separating the geometry information from the reflectance data, we expect to obtain a much simpler representation. As a second benefit, extracting the geometry information also allows for a more realistic rendering of a material at grazing angles, where direct BTF rendering methods cannot reproduce the occluding contour. Yet, even though various methods have been proposed to render volumetric textures, there has been very little work on how to reconstruct them from image data. Existing rendering algorithms typically use computer-generated representations [MN98, Dis98, KK89, Ney98, DLHHP01, LPFH01, WTL*04, FHNK05], or, on rare occasions, volumes scanned using 3-D scanning techniques [CTW*04].

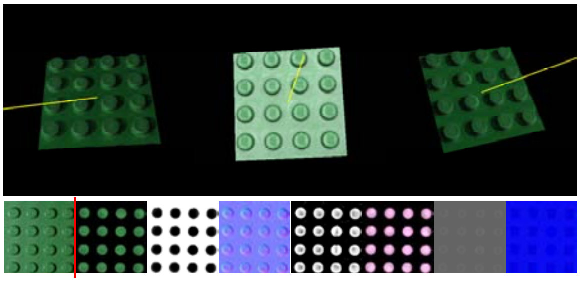

Figure 1: *Three views of a volumetric texture synthesized from a BTF set. The yellow line indicates the light direction. Bottom row shows parameter textures for a single layer with a sample input texture to the left of the line.*

The algorithm proposed in this paper allows volumetric representations to be generated from a set of images taken with an ordinary camera and parametrized as a BTF (see Figure 1). In addition to geometry, it also captures reflectance properties as a parametric BRDF model. This results in a compact representation, effectively compressing a BTF.

In interactive applications we are traditionally limited to a polygonal representation, which becomes very inefficient at surface material's macro scales. The standard solution to this problem is to approximate geometric detail with textures. Various texture representations typically offer a trade off between visual realism and rendering/storage costs. The complexity of the reproduced material is often the deciding factor. As such, relatively simple geometry of various rough materials, such as concrete, wood, painted surfaces, can be visually reproduced with a normal map. More complex geometry with a visible macro scale features require a more advanced representation for the same level of realism. Adding a displacement texture provides a way to represent materials for which geometry can be described by a depth map. More complicated materials typically require volumetric representations. For rendering, the volume can be re-sampled into a stack of semi-transparent textures simulating the original material. For greater realism, the re sampling is often done at run-time and perpendicular to the viewer's direction.

The algorithm proposed here focuses on this most generic form of materials, that can only be faithfully reproduced with a volumetric representation (refer again to Figure 2 for some examples). The model used by our algorithm consists of a stack of semi-transparent slices representing the original BTF data. At each layer point a vector of BRDF parameters approximates the local reflectance. The geometry is represented with volumetric attenuation sampled into each layer. At runtime, layers are rendered over a surface, giving an impression of a 3-D texture. We will refer to this model as Layered Volumetric Surface Texture (LVST). Similar volumetric representation have been shown t be useful for a number of volumetric textures [MN98, Ney98, LPFH01].

Key advantages of the proposed algorithm can be summarized as follows:

- Addresses an open problem of geometry extraction from BTFs.
- Works well for textures with complex geometry, reflectance, and shadowing.
- Separates the geometry from the reflectance into a very compact representation.
- Can be used to render the occluding contour in real-time applications.

2. Related Work

There are several other interesting methods that aim to infer the geometric structure of a texture. Liu et al. [LYS01] and Haindl et al. [HFA04] used shape-from-shading to construct a surface displacement map. Shape-from shading methods work for BTFs with simple reflectance for which the *Lambertian* reflectance assumption applies. Another approach taken by Yu and Chang [YC02] uses shadow graphs for height map reconstruction. Their method is a combination of shape-from-shading and shape-from-shadow with constraints enforcing consistency between the two. Wang and Dana [WD04] used a library of geometric textons to reconstruct an approximation of a BTF height map. Although the focus of their method is actually better reconstruction of shadowing effects rather than geometry, it generates a height map as well. It also works for non-*Lambertian* surfaces for which the reflectance model can be approximated by a known model. Unfortunately, displacement mapping methods are not very useful for textures with a more complex geometry.

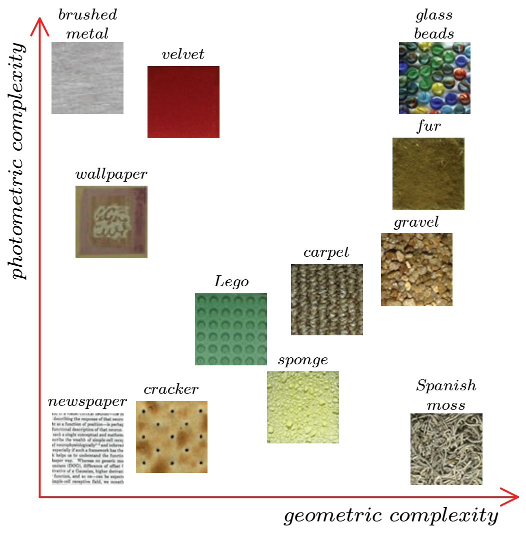

Figure 2: *Appearance complexity space for textures. Horizontal and vertical axes indicate increasing geometric and photometric complexities. Textures in the lower right corner have relatively simple reflectance, yet their BTFs are complex due to geometry.*

Although volumetric reconstruction into layers has not attracted much attention in the BTF setting, it has been explored in the area of stereo reconstruction. Stereo algorithms aim to estimate the depth of a scene using two or more images taken from different locations. This information can then be used to create a volumetric model of the scene. Over the years steady progress in the area resulted in improving accuracy and better handling of difficult settings [SS02]. Transparency has recently attracted more attention due to two reasons: some scenes actually contain transparent objects (glass windows, picture frames, etc.), but also in order to reconstruct layers with smoother semi-transparent boundaries. Of particular interest is the work by Szeliski and Golland [SG99]. Given a stereo sequence, their algorithm extracts layers with semi-transparent boundary regions. By performing a global optimization on the color and transparency value of boundaries, they create smooth, semi-transparent layer edges. However, as with most stereo reconstruction algorithms, reflectance is not modeled. Here we should also mention voxel-based reconstruction techniques, such as voxel coloring algorithms [SD99, SCM*04]. These rely heavily on photo consistency and therefore do not work well for non-*Lambertian* conditions.

Volumetric decomposition of BTFs can also be viewed as a compression scheme. Storage requirements typically make it impractical to use BTFs directly in real-time rendering. Over the past few years there has been an ongoing work on better compression and parametrization. Most compression solutions use some variation of eigen decomposition or principal component analysis [MMK03b, KMBK03, VT04, LHZ*04, MCT*05]. For additional compression, some also utilize *Laplacian* pyramids [HFA04, MCT*05]. A different approach is to view BTFs as a 2-D apparent BRDF (ABRDF) function [WHON97], that includes the effects of inter reflection and shadowing. Per-pixel view and lighting variations can then be approximated by ABRDF clustering [CD01, TZL*02, WD04]. ABRDFs can also be modeled using traditional parametric BRDF representations [MMK03a, FH04, KSS*04] or factorization techniques [SvBLD03].

3. The LVST Model

The volumetric model used by the algorithm consists of a stack of N layers, as pictured in Figure 3a. Each layer is parametrized with (u_n, v_n), where u and v are the 2-D texture dimensions, and n is the layer index. We number layers starting at one for the top layer. The bottommost layer is always fully opaque. Rendering involves superposition of each layer by re-projection into the view direction and computation of the reflectance values for each of them. This composing process can be expressed as a backward warping operation $\mathcal{W}_b$ applied to each layer projecting them into the viewing direction. This is accomplished using the shear-warp algorithm [LL94]. Under parallel projection this operation reduces to simple parallax shifts of layers relative to each other. The inverse of the backward warping operator $\mathcal{W}_f$, maps the camera view along $\vec{\omega}$ into coordinates of a layer n as:

$$\begin{bmatrix} u_n & v_n \end{bmatrix}^{\mathbf{T}} = \mathcal{W}_f(\vec{\omega}, n) \begin{bmatrix} \tilde{u} & \tilde{v} \end{bmatrix}^{\mathbf{T}} \tag{1}$$

where $(\tilde{u}, \tilde{v})$ are the coordinates in the camera view. In the composing process of N layers, the total camera-observed brightness I_o for the current viewing direction along $\vec{\omega}$ is the sum of brightness contributions from each individual layer I_l attenuated by the product of the projected attenuation factors a:

$$I_o(\tilde{u}, \tilde{v}, \vec{\omega}) = \sum_{n=1}^{N} I_{l,n}(u_n, v_n, \vec{\omega}) \prod_{n'=1}^{n-1} a_{n'}(u_{n'}, v_{n'}, \vec{\omega}) \tag{2}$$

This composing equation can be viewed as an approximation to a discretized form of the radiance equation commonly used in volumetric rendering. The outgoing radiance L_o is then equal to the sum of the incoming radiance $L_{i,n}$ in-scattered at each layer n according to the bidirectional scattering distribution function (BSDF) f_s:

$$L_o(\vec{\omega}) = \sum_{n=1}^{N} \int_{4\pi} f_{s,n}(\vec{\omega}' \to \vec{\omega}) L_{i,n}(\vec{\omega}') \vec{N}_n \cdot \vec{\omega}' \, d\vec{\omega}' \prod_{n'=1}^{n-1} a_{n'} \tag{3}$$

$\vec{N}_n$ is the normal vector. All parameters in Equation 3 are also functions of u and v. From now on, we will omit them from equations for better readability.

Equation 3 is only a discrete approximation of radiance transport. Light interactions along the direction $\vec{\omega}$ are limited to layer intersection points. Increasing the number of layers has the net effect of increasing the sampling rate along the ray. The integral accounts for light contributions from all directions and could be used to approximate arbitrary material. Yet a more application-specific approximation can lead to a more efficient implementation. The algorithm presented here is designed specifically for non-diffusive volumetric materials with strong light source direction correlation. Under these conditions, light interactions are dominated by absorption (as opposed to scattering) and light arriving at the camera is mainly due to the first scattering reflection event. The integral of the incoming radiance $L_{i,n}$ at each layer can be approximated with just the primary incoming light direction $\vec{\omega}_i$. We also add a constant diffuse radiance term $L_{d,n}$ to approximate any higher-order scattering events:

$$L_o(\vec{\omega}) = \sum_{n=1}^{N} f_{r,n}(\vec{\omega}_i \to \vec{\omega}) \Big[L_{i,n}(\vec{\omega}_i) + L_{d,n} \Big] \vec{N}_n \cdot \vec{\omega}_i \prod_{n'=0}^{n-1} a_{n'} \tag{4}$$

We replace the BSDF with a more appropriate reflectance term (BRDF) f_r. For BRDF, we are free to choose any parametric reflectance model that best suits a particular texture. BRDF models with fewer parameters have the advantage of faster conversion. In particular, we have been using the *Lafortune* model [LFTG97] which gives a good trade-off between generality and the number of required parameters.

The proposed LVST model can be summed up as consisting of a stack of layers with spatially-varying parameters. Each layer location (u_n, v_n) is described by several parameters:

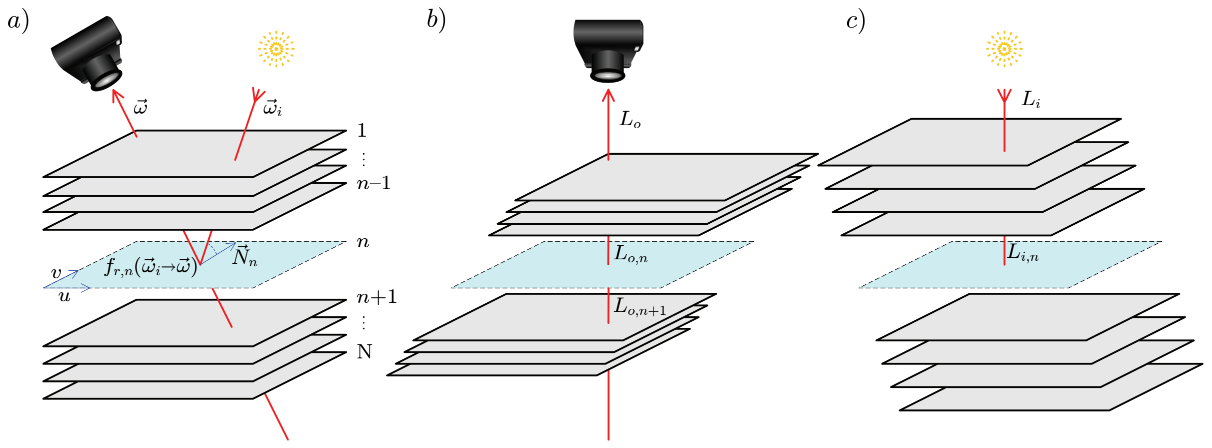

Figure 3: *The LVST model. (a) Layer parameters relate the total outgoing radiance in terms of the reflected illumination and the total outgoing radiance from layers below. (b) Layers warped into the camera direction. (c) Layers warped into the light source direction.*

- The surface normal $\vec{N}_n$.
- Parameters of the chosen BRDF $f_{r,n}$.
- The attenuation factor a_n.
- The diffuse radiance term $L_{d,n}$.

In the next section we will show how to iteratively solve for these unknown parameters.

4. Estimating the LVST Parameters from a BTF Dataset

Equation 4 expresses the total outgoing radiance along a single ray. A captured BTF samples the outgoing radiance values L_o for discrete light and camera directions $(\vec{\omega}_i, \vec{\omega})$. The total incoming radiance L_i can also be measured. For each layer n, we would like to find the best estimate for the BRDF $f_{r,n}$, attenuation a_n, surface normal $\vec{N}_n$, and the diffuse light term $L_{d,n}$, so that the observed error between rendered and originally measured radiance values is minimized. Globally we are trying to solve the inverse image synthesis problem with unknown reflectance and geometry. This problem appears in vision applications and, in general, is difficult to solve. For a particular BRDF model with f parameters, we have to solve for $n \cdot u \cdot v \cdot (f+4)$ unknowns. For the trichromatic RGB model, the number of unknows is even greater. Solving for them directly, even for models with just a few layers, can be prohibitive. Instead, we solve this high dimensionality problem by expressing it as a series of local optimization steps. In particular, we iteratively perform local optimizations on individual layers.

4.1. Layer Optimization

Consider the radiance $L_{o,n}$ leaving a particular layer n toward the camera along $\vec{\omega}$ (see Figure 3). By looking back at Equation 4 we can see, that it is equal to the sum of the incoming radiance $L_{i,n}(\vec{\omega}_i) + L_{d,n}$ reflected by the layer and the radiance transmitted from the layers below $L_{o,n+1}$. We will denote the reflected radiance with $L_{l,n}$:

$$\begin{aligned} L_{o,n}(\vec{\omega}) &= L_{l,n}(\vec{\omega}) + a_n L_{o,n+1}(\vec{\omega}) \\ L_{l,n}(\vec{\omega}) &= f_{r,n}(\vec{\omega}_i \rightarrow \vec{\omega}) \left[L_{i,n}(\vec{\omega}_i) + L_{d,n} \right] \vec{N}_n \cdot \vec{\omega}_i \end{aligned} \tag{5}$$

Lets assume, that all incoming and outgoing radiance values are known and we are given a set of $(L_{i,n}, L_{o,n+1}, L_{o,n})$ triplets for K different light and viewing directions. For each discrete position on a layer n we want to solve for the unknown parametric BRDF function $f_{r,n}$, the diffuse radiance constant $L_{d,n}$, the surface normal $\vec{N}$, and the attenuation coefficient a_n. The least-squares solution can be obtained by minimizig objective function $\mathcal{O}$ of the form:

$$\mathcal{O} = \sum_{k=1}^{K} \left[L_{l,n,k} + a_n L_{o,n+1,k} - L_{o,n,k} \right]^2 w_{n,k} \tag{6}$$

We have dropped some notation from Equation 5 for readability.

The weighting factor $w_{n,k}$ in the Equation 6 is crucial, as it provides the means to distribute radiance into appropriate layers. It is used to penalize data when a layer point is not in the direct line of sight of the camera and represents visibility. It can be written as a product of attenuations:

$$w_{n,k} = \prod_{n'=1}^{n-1} a_{n'} \tag{7}$$

Unlike the global problem, the local objective function is only non-linear in the BRDF model parameters. We will directly take advantage of that fact later on. The optimization process can broken down into a series of layer updates. For each layer, the algorithm uses the current estimate of the model to compute necessary layer radiance values. From each input image in the sampled BTF, we can obtain the outgoing radiance from the top layer $L_{o,1} = L_o$. Similarly, the total incoming radiance from the light source is the same as the incoming radiance to the top layer $L_{i,1} = L_i$. If the current state of the LVST model is known, we can also compute radiance values for other layers. In general, the outgoing radiance $L_{o,n}$ from layer n is equal to the difference between the measured radiance L_o and radiance contributions from layers above n:

$$L_{o,n} = \frac{1}{a_{n-1}} \left[L_o - \sum_{n'=1}^{n-1} L_{l,n'} \prod_{n''=1}^{n'-1} a_{n''} \right] \tag{8}$$

In a similar way, we can express the radiance incoming from below the layer n as the sum:

$$L_{o,n+1} = L_{l,n+1} + \sum_{n'=n+2}^{N} L_{l,n'} \prod_{n''=n+1}^{n'-1} a_{n''} \tag{9}$$

Finally, to compute the radiance due to direct illumination, we first perform a warping operation similar to $\mathcal{W}_f$, but with respect to the light source direction (see Figure 3c). We can express the direct illumination reaching the layer n as:

$$L_{i,n} = L_i \prod_{n_i=1}^{n-1} s_{n_i} \tag{10}$$

The shadowing coefficient s_{n_i} is eventually equal to the attenuation a_{n_i}. Initially model optimization is done without shadowing, which is then estimated in the final pass. The reasons for that will be explained in Section 4.4.

In summary, the algorithm begins from the bottommost layer and iterates through all layers until convergence is achieved. All layers are initialized to be fully transparent. For each BTF sample image, the layer stack undergoes viewer and camera direction warping operation $\mathcal{W}_f$ and the three layer radiance values are computed. These can then used to solve the least-squares problem from Equation 6.

4.2. BRDF Model Optimization

The majority of BRDF models are non-linear and therefore minimization of the objective function from Equation 6 generally requires a non-linear solution. Unfortunately non-linear minimization algorithms are also much slower than their linear counterparts, as they typically require multiple iterations over the input data. This is problematic, as typical input BTF data sets are very large, containing on the order of thousands of images. Iterating over the whole input data can therefore be very time consuming.

When dealing with large data sets, it is better to use incremental algorithms, such as the Extended Kalman Filter [Ber96], conceptually equivalent to an incremental version of the Gauss-Newton method. The idea behind incremental solvers is to deal with large sets by incrementally iterating over smaller chunks of data. In particular, an incremental version of the Gauss-Newton method works by progressively updating the current gradient covariance matrix $\mathcal{C}_m$, as new data vectors $\mathbf{x}$ becomes available. A single iteration over the parameter vector $\mathbf{P}$ is performed every M such updates. It can be conceptually written it as:

$$\mathcal{C}_m = \lambda \mathcal{C}_{m-1} + \nabla \mathbf{x}_m \nabla \mathbf{x}_m^T \tag{11}$$

$$\mathbf{P}_i = \mathbf{P}_{i-1} - \mathcal{C}_{Mi}^{-1} \sum_{m'=M(i-1)}^{Mi} \nabla \mathbf{x}_{m'} \mathbf{x}_{m'} \tag{12}$$

The parameter vector $\mathbf{P}_i$ is updated every M data vectors $\mathbf{x}$. For faster computation we can also take advantage of the Sherman-Morrison update formula [GL96] instead of directly computing the inverse of $\mathcal{C}_m$. The smoothing parameter λ controls "forgetfulness" of the algorithm with respect to the past data. It is desirable to smoothly decay its value in the course of the algorithm from some initial constant. This results in faster initial convergence. For more details on the algorithm refer to the work of Bertsekas [Ber96].

Given radiance values computed using Equations 8, 9, and 10, the algorithm evaluates the gradient of Equation 6 and subsequently updates the incremental solver. As an example, in Appendix A we will show to derive the gradient function for the *Lafortune* model.

4.3. Recovering the Attenuation Parameter

In general, the spatially varying attenuation parameter a_n is difficult to recover using the incremental solver directly. The problem is ill-posed, as multiple combinations of BRDF parameters and the attenuation can result in many local minima. Since Equation 6 in linear in the attenuation parameter, we will take advantage of it. We use the non-linear incremental solver to initially estimate the layer radiance $L_{l,n}$, while a_n is set to maximum opacity. We use this result to subsequently solve a linear problem of the form:

$$a_n = \operatorname*{argmin}_{\alpha_2} \sum_{k=1}^{K} \left[\alpha_1 L_{l,n,k} + \alpha_2 L_{o,n+1,k} - L_{o,n,k} \right]^2 w_{n,k} \tag{13}$$

This equation can be interpreted as a blending optimization

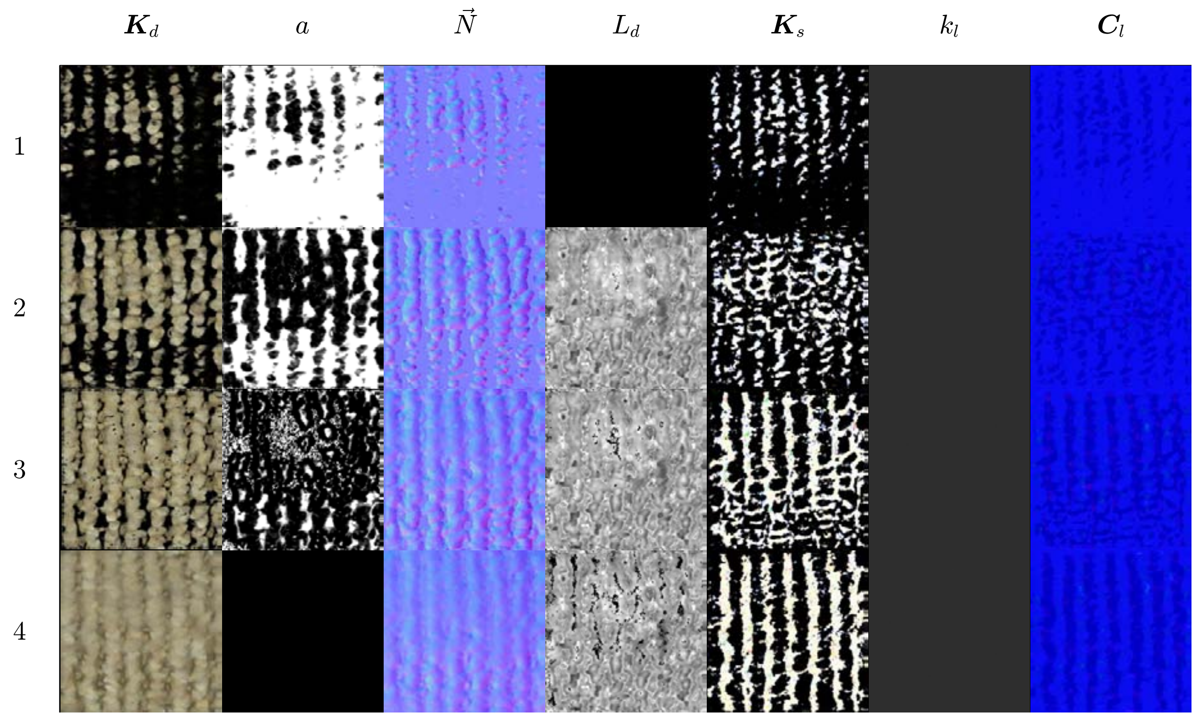

Figure 4: *LVST parameter textures recovered from the carpet BTF set using the* Lafortune *reflectance model. Each row corresponds to one layer. Parameters are in columns. From left to right, they are: color diffuse term, attenuation coefficient, normal vector (color represented as $(r,g,b) = (N_x,N_y,N_z)$), diffuse radiance factor, specular term, specular exponent, and* Lafortune *lobe coefficients. Attenuation, diffuse radiance, and specular exponent are scalar values. Refer to Appendix A for more details on the* Lafortune *model.*

between two one-layer BTF representations at slightly different offsets. α_1 is a dummy blending factor for $L_{l,n}$ that we do not have to solve for. Once we solve Equation 13 for α_2, $L_{l,n}$ must be updated making α_1 redundant.

Recovering geometry from images can at most be guaranteed visual consistency with the originals. Recovering the attenuation parameter can be viewed as solving a blending equation for two BTF representations at different depth offsets. Yet visual consistency can often be achieved in a non-unique way, allowing multiple solutions. For example, a constant radiance offset present in the input data could be arbitrary distributed between layers. This is the case for flat, featureless texture regions in the BTF. Although visually consistent with the input data, unexpected artifacts may appear, when rendered for viewing directions that were not part of the input. Moreover, as the distance between layers is decreased, so will the differences in their appearance, until they become dominated by noise and re-projection error. Increasing the number of layers can therefore be beneficial only up to a certain point. We can conclude, that the resolution limit, defined as the minimum useful separation distance between layers, is directly related to the strength of the noise present in the BTF and inversely related to the local texture gradient measured against the relative warping displacement for that separation distance. Below this threshold, the attenuation parameter cannot be meaningfully estimated. Alternatively, we could modify the algorithm to achieve better consistency. For example, adding a smoothness constraint based upon the local neighborhood information, such as the surface gradient continuity, could lead to a more consistent solution and is an interesting area to explore.

4.4. Shadows and the Diffuse Radiance Term

Unlike some other phenomena, shadows cannot be easily parametrized. Shadows are created when light is attenuated on its way to a surface and depend on geometry. Although the LVST model stores spatially-varying attenuation parameters at each layer, shadows generated that way will rarely be consistent with the original shadows due to geometry being flattened into layers. Yet sharp shadow boundaries that do not match exactly may still look more “natural” to an observer. Following this argument, we model shadows using the actual attenuation values stored at each layer. As this is not consistent with least-squares optimization used to re-

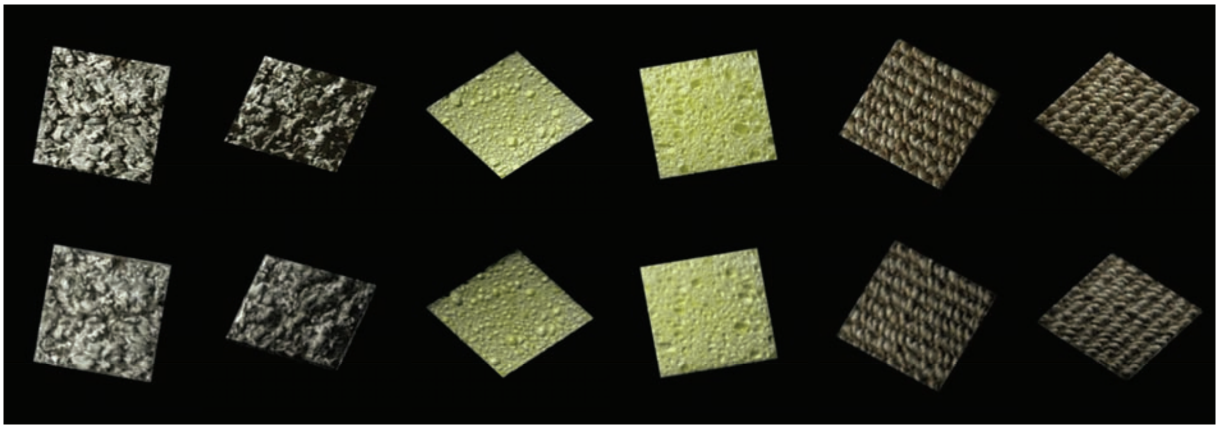

Figure 5: *Images synthesized using the LVST model. The original BTF samples are at the top with corresponding synthesized equivalents at the bottom.*

cover other parameters, shadowing is initially ignored and is only taken into account in the final pass of the algorithm.

Note, that in a shadowed area, the diffuse radiance $L_{d,n}$ becomes the primary source of illumination. In the final step of the algorithm we solve for the diffuse radiance term, which essentially serves as a correction factor for shadows. We again linearize Equation 6 solving for $L_{d,n}$:

$$L_{d,n} = \underset{\alpha}{\operatorname{argmin}} \sum_{k=1}^{K} w'_{n,k} \left[f_{r,n} \left[L_{i,n,k} + \alpha \right] \vec{n} \cdot \vec{\omega}_i + L_{o,n+1,k} - L_{o,n,k} \right]^2 \tag{14}$$

The main difference this time is that we only consider data points for which $L_{i,n,k} < 1$, i.e. at least some shadow is cast by layers above. Many other ways of rendering shadows are definitely possible and it can be an interesting area to investigate.

Summing it up, the algorithm consists of a series of local optimization steps that are repeated until convergance is achieved:

1. *Initialize all layers and make them fully transparent.*
2. *Starting from the deepest layer N, iterate over the layer stack (with no shadowing):*
    a. *Solve for the attenuation coefficient (Section 4.3):*
        i. *Generate an opaque layer for the current layer location.*
        ii. *Solve Equation 13 for the attenuation.*
    b. *Estimate the BRDF and surface normal parameters (Section 4.2).*
3. *Solve for the diffuse radiance component and update the model for shadowing (Section 4.4).*

5. Input BTF Data Sets

In our experiments we have been using textures from the UIUC BTF database [KMBK03]. It contains some complex volumetric surface textures and a few are shown in Figure 2. Each BTF was sampled over a range of discrete viewing and lighting directions resulting in over 10,000 images.

Images required some additional post-processing, most importantly, inversion of the gamma correction that was automatically applied by the camera. Some image artifacts, which could not be corrected for, included video compression artifacts and camera noise. Errors in geometric alignment, including camera and light orientations, lens distortions, approximation of the perspective with parallel projections, and errors in the BTF rectification process are also problematic.

6. Results

Although the algorithm could be applied to BTFs with simple geometry, it does not provide any advantage over other compression schemes mentioned in Section 2. Since it was designed to specifically handle textures with complex volumetric structures, we have tested the algorithm on several captured BTFs with such characteristcs. In addition to BTF data, the algorithm requires a choice of number of layers and the height bounds. These can be manually selected by examining the data. Since all layer updates are independent of each other in the (u, v) space, we take an advantage of this parallelism and solve for all layer parameters simultaneously. The number of iterations required before convergence conditions are detected depends on the complexity of the texture and the chosen BRDF, but is on the order of a

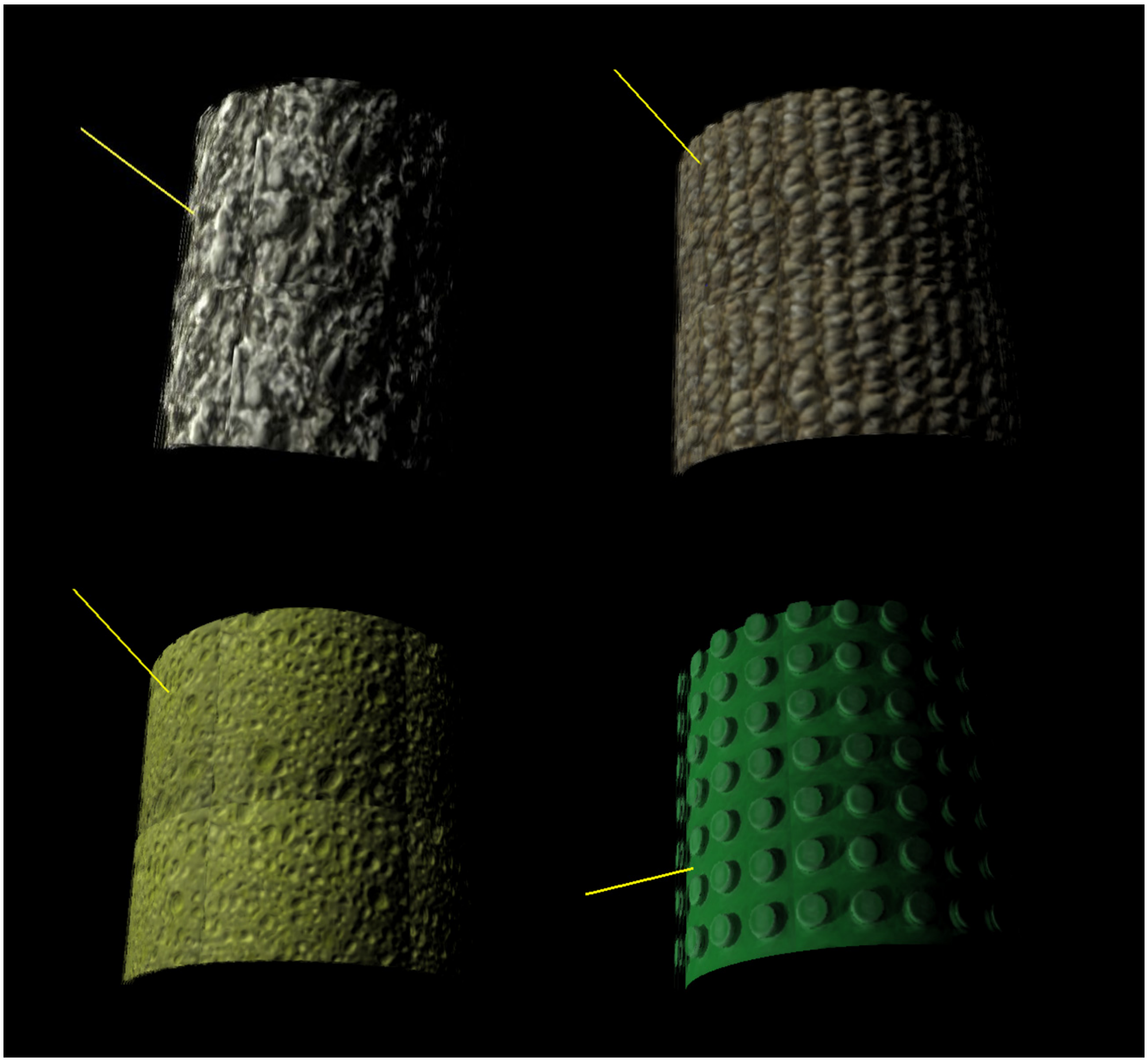

Figure 6: *LVSTs tiled on a cylinder and rendered in real-time. The two examples in the left column were fitted using the diffuse model and the two on the right use the* Lafortune *model. The yellow line indicates the light direction. Note the 3-D texture at the occluding contour.*

few thousands input BTF samples. Quite often convergence of a single layer is achieved before the majority of the input data is even evaluated, suggesting significant data redundancy. Depending on the number of layers and their dimensions, our C++ implementation typically takes a few hours to fully converge when run on a desktop PC.

Several examples of synthesized views along with the original data are shown in Figure 5. All textures in the figure have the same spatial resolution as the original BTF data (192x192, with a possible margin of a few pixels). They are modeled with four to five layers, except the lichen texture, which uses six layers. Since the algorithm relies on parametric modeling, it cannot represent phenomena not represented by the model. The resulting reconstructions are typically blurred versions of the originals. However, a major advantage of this representation is a more natural interpolation and even extrapolation from the original data.

The LVST representation can be used to render BTFs on polygonal meshes. Figure 6 demonstrates smooth changes in texture appearance on a smoothly-varying surface with tiled LVSTs. Please note silhouettes visible near the upper edge of the cylinder. To further improve the visual quality at grazing angles, the algorithm can be trivially extended to additionally construct vertical layers.

6.1. Compression and Real-Time Rendering

The result of the algorithm is a compact texture representation highly suitable for real-time applications. From the original sets of over 10,000 images, corresponding LVST representations are on the order of 10-20 textures. Such high compression ratio and easily parallelizable rendering process is perfectly suitable for graphics hardware acceleration. The rendering process simply involves evaluation of Equation 4. For each layer, we have to evaluate the BRDF model stored in several textures and modulate it with the shadow cast by the layers above. In practice, it can be performed in a few simple steps:

1. *Starting from the bottommost layer N, evaluate the BRDF for the given light and camera directions.*
2. *Compute the shadow cast by all layers above the current layer and use it to compute the final layer radiance.*
3. *Render the layer into the screen buffer blending it according to the attenuation factor.*

These three steps can be evaluated usng graphics hardware in a pixel shader. We have implemented a real-time application to demonstrate this. In a vertex program it computes layer offsets and evaluates the BRDF model in a fragment program. Figure 7 shows several screen captures of textured objects using a patch-based texture synthesis algorithm [MK03]. The synthesis is done on vectors of LVST parameters in a fashion similar to regular texture synthesis, that operates on vectors of RGB color values. In LVST synthesis, we have used vectors of concatenated values of attenuation coefficients a_n and some number of BRDF parameters from every layer. Typically the diffuse term $K_{d,n}$ alone was sufficient. Note, that the texture synthesis algorithm from [MK03] trades-off synthesis quality for speed and therefore we expect even better visual results from other synthesis algorithms.

7. Conclusions

In this paper, we have described an algorithm that separates the geometric information from the reflectance data present in a BTF. It is especially destined for volumetric surface textures with complex, opaque microstructure, shadowing, and unknown reflectance properties. These types of textures are especially difficult to compress and reconstruct by existing BTF compression schemes. By contrast, our representation naturally offers very high compression of the original data and is fit for real-time rendering applications. On top of that, by providing a geometric representation, it allows for realistic rendering for textures at grazing angles. Even higher coherence of the rendered texture could be achieved by trivially extending the algorithm to re sample the BTF into vertical layers. At a negligible additional storage cost, textures could be fully rendered at any orientation to the viewer.

Although we have applied the underlying algorithm to decomposition of BTFs into layers, its applications could potentially be useful in other areas. The concept of recursively updating data slices could be used in other volume reconstruction applications, where transparency, shadowing, and complicated reflectance properties play an important role. Finally, it could likely be extended to voxelized representations as well.

8. Acknowledgements

All data sets used in this paper have come from the UIUC BTF database [KMBK03]. This work was supported by the National Science Foundation grants IIS-0116560, IIS-0308185, and EIA-0224431.

References

[Ber96] BERTSEKAS D. P.: Incremental least-squares methods and the extended kalman filter. *SIAM Journal on Optimization 6* (1996), 807–822.

[CD01] CULA O. G., DANA K. J.: Compact representation of bidirectional texture functions. In *Conference on Computer Vision and Pattern Recognition* (2001), vol. 1, pp. 1041–1047.

[CTW*04] CHEN Y., TONG X., WANG J., LIN S., SHUM B. G. H.-Y.: Shell texture functions. In *SIGGRAPH* (2004), vol. 23, pp. 343–353.

[Dis98] DISCHLER J.-M.: Efficiently rendering macro geometric surface structures with bi-directionaltexture functions. In *Eurographics Workshop on Rendering* (1998), Springer, pp. 169–180.

[DLHHP01] DAUBERT K., LENSCH H. P. A., HEIDRICH W., HANS-PETERSEIDEL: Efficient cloth modeling and rendering. In *Eurographics Workshop on Rendering* (London, UK, 2001), Springer-Verlag, pp. 63–70.

[DvNK99] DANA K. J., VAN GINNEKEN B., NAYAR S. K., KOENDERINK J. J.: Reflectance and texture of real-world surfaces. In *SIGGRAPH* (1999), vol. 18, pp. 1–34.

[FH04] FILIP J., HAINDL M.: Non-linear reflectance model for bidirectional texture function synthesis. In *International Conference on Pattern Recognition* (Washington, DC, 2004), vol. 1, IEEE Computer Society, pp. 80–83.

[FHNK05] FURUKAWA R., HARADA M., NAKAMURA Y., KAWASAKI H.: Synthesis of textures with intricate geometries using BTF and large number of textured micropolygons. In *Proc. of the 4th International Workshop on Texture Analysis and Synthesis* (2005), pp. 77–82.

[GL96] GOLUB G. H., LOAN C. F. V.: *Matrix Computations, Johns Hopkins Studies in Mathematical Sciences*. Johns Hopkins University Press, 1996.

[HFA04] HAINDL M., FILIP J., ARNOLD M.: BTF image space utmost compression and modelling method. In *International Conference on Pattern Recognition* (Washington, DC, 2004), vol. 3, IEEE Computer Society, pp. 194–197.

[KK89] KAJIYA J. T., KAY T. L.: Rendering fur with three dimensional textures. In *SIGGRAPH* (1989), pp. 271–280.

[KMBK03] KOUDELKA M. L., MAGDA S., BELHUMEUR P. N., KRIEGMAN D.: Acquisition, compression and synthesis of bidirectional texture functions. In *International Workshop on Texture Analysis and Synthesis* (2003), pp. 59–64.

[KSS*04] KAUTZ J., SATTLER M., SARLETTE R., KLEIN R., HANS-PETERSEIDEL: Decoupling BRDFs from surface mesostructures. In *Conference on Graphics Interface* (2004), Canadian Human-Computer Communications Society, pp. 177–182.

[LFTG97] LAFORTUNE E. P. F., FOO S.-C., TORRANCE K. E., GREENBERG D.: Non-linear approximation of reflectance functions. In *SIGGRAPH* (1997), pp. 117–126.

[LHZ*04] LIU X., HU Y., ZHANG J., TONG X., SHUM B. G. H.-Y.: Synthesis and rendering of bidirectional texture functions on arbitrarysurfaces. *Transactions on Visualization and Computer Graphics 10*, 3 (2004), 278–289.

[LL94] LACROUTE P., LEVOY M.: Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH* (1994), pp. 451–458.

[LPFH01] LENGYEL J., PRAUN E., FINKELSTEIN A., HOPPE H.: Real-time fur over arbitrary surfaces. In *Symposium on Interactive 3D Graphics* (2001), ACM Press, pp. 227–232.

[LYS01] LIU X., YU Y., SHUM H.-Y.: Synthesizing bidirectional texture functions for real-world surfaces. In *SIGGRAPH* (2001), pp. 97–106.

[MCT*05] MA W.-C., CHAO S.-H., TSENG Y.-T., ANDCHUN FA CHANG Y.-Y. C., CHEN B.-Y., OUHYOUNG M.: Level-of-detail representation of bidirectional texture functions for real-timerendering. In *Symposium on Interactive 3D Graphics* (New York, NY, 2005), ACM Press, pp. 187–194.

[MK03] MAGDA S., KRIEGMAN D.: Fast texture synthesis on arbitrary meshes. In *Eurographics Symposium on Rendering* (2003), pp. 82–89.

[MMK03a] MESETH J., MÜLLER G., KLEIN R.: Preserving realism in real-time rendering of bidirectional texture functions. In *OpenSG Symposium* (2003), Eurographics Association, pp. 89–96.

[MMK03b] MÜLLER G., MESETH J., KLEIN R.: Compression and real-time rendering of measured BTFs using local PCA. In *In Proceedings of The 8th International Fall Workshop on Vision, Modelingand Visualisation* (Berlin, Germany, 2003), Akademische Verlagsgesellschaft Aka GmbH, pp. 271–280.

[MN98] MEYER A., NEYRET F.: Interactive volumetric textures. In *Eurographics Workshop on Rendering* (1998), Eurographics Association, pp. 157–168.

[Ney98] NEYRET F.: Modeling, animating, and rendering complex scenes using volumetric textures. *Transactions on Visualization and Computer Graphics 4*, 2 (1998), 55–70.

[SCM*04] SLABAUGH G. G., CULBERTSON W. B., MALZBENDER T., STEVENS M., SCHAFER R. W.: Methods for volumetric reconstruction of visual scenes. *International Journal of Computer Vision 57*, 3 (2004), 179–199.

[SD99] SEITZ S. M., DYER C. R.: Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision 35*, 2 (1999), 151–173.

[SG99] SZELISKI R., GOLLAND P.: Stereo matching with transparency and matting. *International Journal of Computer Vision 32*, 1 (1999), 45–61.

[SS02] SCHARSTEIN D., SZELISKI R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision 47*, 1-3 (2002), 7–42.

[SvBLD03] SUYKENS F., VOM BERGE K., LAGAE A., DUTRÉ P.: Interactive rendering with bidirectional texture functions. *Computer Graphics Forum 22*, 3 (2003), 463–472.

[TZL*02] TONG X., ZHANG J., LIU L., WANG X., ANDHEUNG YEUNG SHUM B. G.: Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH* (2002), pp. 665–672.

[VT04] VASILESCU M. A. O., TERZOPOULOS D.: TensorTextures: multilinear image-based rendering. In *SIGGRAPH* (2004), vol. 23, pp. 336–342.

[WD04] WANG J., DANA K. J.: Hybrid textons: modeling surfaces with reflectance and geometry. In *Conference on Computer Vision and Pattern Recognition* (2004), vol. 1, pp. 372–378.

[WHON97] WONG T.-T., HENG P.-A., OR S.-H., NG W.-Y.: Image-based rendering with controllable illumination. In *Eurographics Workshop on Rendering* (London, UK, 1997), Springer-Verlag, pp. 13–22.

[WTL*04] WANG X., TONG X., LIN S., HU S., GUO B., HEUNG-YEUNGSHUM: Generalized displacement maps. In *Eurographics Symposium on Rendering* (2004), Eurographics Association, pp. 227–234.

[YC02] YU Y., CHANG J. T.: Shadow graphs and surface reconstruction. In *European Conference on Computer Vision* (London, UK, 2002), Springer-Verlag, pp. 31–45.

Appendix A: Solving for the *Lafortune* Model Parameters

Several LVSTs in the paper were generated using the *Lafortune* BRDF model. We have used the simplified version with non-zero values only along the diagonal of the lobe matrix $\mathbf{C}_l$ [LFTG97]. Here we show the explicit form for a one-lobe approximation (trichromatic color parameters are indicated in bold):

$$\boldsymbol{f}_r = \boldsymbol{K}_d + \boldsymbol{K}_s(\vec{\omega}^{\mathbf{T}}\boldsymbol{C}_l\vec{\omega}_i)^{k_l} \tag{15}$$

,where $\boldsymbol{K}_d$ and $\boldsymbol{K}_s$ are the diffuse and specular coefficients and k_l is the specular exponent for the lobe. When the specular coefficient is zero, BRDF reduces to the constant diffuse coefficient. There is a total of 15 unknown parameters, including the attenuation and the diffuse radiance term.

In order to minimize the objective function from Equation 6, we need to evaluate its gradient vector, as described in Section 4.2. The residual in Equation 6 has the form:

$$\boldsymbol{R}_n = \boldsymbol{f}_r\vec{N}_n \cdot \vec{\omega}_i \left[L_{i,n} + L_{d,n}\right] + a\boldsymbol{L}_{o,n+1} - \boldsymbol{L}_{o,n} \tag{16}$$

The gradient vector $\nabla\mathbf{x}$ is:

$$\frac{\partial\mathcal{O}}{\partial\vec{N}} = \vec{\omega}_i\left[\boldsymbol{K}_d + \boldsymbol{K}_s(\vec{\omega}^{\mathbf{T}}\boldsymbol{C}_l\vec{\omega}_i)^{k_l}\right][L_i + L_d]\cdot\boldsymbol{R}\,w \tag{17}$$

$$\frac{\partial\mathcal{O}}{\partial\boldsymbol{K}_d} = \vec{N}\cdot\vec{\omega}_i\,[L_i + L_d]\,\boldsymbol{R}\,w \tag{18}$$

$$\frac{\partial\mathcal{O}}{\partial\boldsymbol{K}_s} = (\vec{\omega}^{\mathbf{T}}\boldsymbol{C}_l\vec{\omega}_i)^{k_l}\,\vec{N}\cdot\vec{\omega}_i\,[L_i + L_d]\,\boldsymbol{R}\,w \tag{19}$$

$$\frac{\partial\mathcal{O}}{\partial\boldsymbol{C}_l} = \vec{\omega}^{\mathbf{T}}\vec{\omega}_i\,k_l\boldsymbol{K}_s(\vec{\omega}^{\mathbf{T}}\boldsymbol{C}_l\vec{\omega}_i)^{k_l-1}\,\vec{N}\cdot\vec{\omega}_i\,[L_i + L_d]\cdot\boldsymbol{R}\,w \tag{20}$$

$$\frac{\partial\mathcal{O}}{\partial k_l} = \ln(\vec{\omega}^{\mathbf{T}}\boldsymbol{C}_l\vec{\omega}_i)\,\mathbf{K}_s(\vec{\omega}^{\mathbf{T}}\boldsymbol{C}_l\vec{\omega}_i)^{k_l}\,\vec{N}\cdot\vec{\omega}_i\,[L_i + L_d]\cdot\boldsymbol{R}\,w \tag{21}$$

The layer subscript *n* has been dropped for better readability. Recall, that *w* is the weighting factor defined in Equation 7.