Eurographics Symposium on Rendering 2003
Per Christensen and Daniel Cohen-Or (Editors)

Efficient and Realistic Visualization of Cloth

Mirko Sattler, Ralf Sarlette and Reinhard Klein

Institute of Computer Science II, University of Bonn, Germany

Abstract
Efficient and realistic rendering of cloth is of great interest especially in the context of e-commerce. Aside from the simulation of cloth draping, the rendering has to provide the "look and feel" of the fabric itself. In this paper we present a novel interactive rendering algorithm to preserve this "look and feel" of different fabrics. This is done by using the bidirectional texture function (BTF) of the fabric, which is acquired from a rectangular probe and after synthesis, mapped onto the simulated geometry. Instead of fitting a special type of bidirectional reflection distribution function (BRDF) model to each texel of our BTF, we generate view-dependent texture-maps using a principal component analysis of the original data. These view-dependent texture maps are then illuminated and rendered using either point-light sources or high dynamic range environment maps by exploiting current graphics hardware. In both cases, self-shadowing caused by geometry is taken into account. For point light sources, we also present a novel method to generate smooth shadow boundaries on the geometry. Depending on the geometrical complexity and the sampling density of the environment map, the illumination can be changed interactively. To ensure interactive frame rates for denser samplings or more complex objects, we introduce a principal component based decomposition of the illumination of the geometry. The high quality of the results is demonstrated by several examples. The algorithm is also suitable for materials other than cloth, as far as these materials have a similar reflectance behavior.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Bitmap and framebuffer operations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism; Color, shading, shadowing, and texture

1. Introduction

In addition to the microstructure, the mesostructure of a fabric is of great importance for the reflectance behavior of cloth. The mesostructure is responsible for fine-scale shadows, occlusions, specularities and subsurface scattering effects. Altogether these effects are responsible for the "look and feel" of cloth. There are essentially two techniques of cloth rendering according to the way in which mesostructure is captured. The first approach explicitly models the mesostructure of the fabric in detail and renders it using different lighting models and rendering techniques [21, 13] Although these algorithms produce impressive results and some of them are already applicable at interactive frame rates, using these methods, it is difficult to reproduce the special appearance of a given fabric. In the second approach the reflectance properties of a given real fabric are measured and then used to generate realistic images[11, 43]. As shown by[11],

Figure 1: *Wool shirt rendered under natural illumination (Uffizi street scene).*

the most important optical parameters of opaque materials including their mesostructure can be described by the bidirectional texture function (BTF). This six-dimensional function describes how a planar texture probe changes its appearance when illuminated and viewed from different directions, as shown on the bottom row of figure 4. The resulting texture probes capture all effects caused by the mesostructure like roughness, self-shadowing, occlusion, inter-reflections and subsurface scattering. Furthermore, the BTF describes how the texture has to be filtered when viewed from different directions. Therefore, in order to achieve the most realistic visualization of a given cloth, we follow the second approach based on measured BTF data. For the illumination we provide two different methods: first by point or directional light sources. Second, illumination by utilizing high dynamic range environment maps. Both techniques are of interest, since on one hand, illuminating the material by point light sources allows the user to inspect the material under a controlled lighting situation and reveals the mesostructure nicely. Here, we also introduce a new method, to generate smooth shadow boundaries on polygonal meshes. On the other hand, people can judge and recognize the material more easily under natural illumination than under the simplified and artificial one provided by point light sources. Our algorithm uses a decomposition of the illumination of the geometry, to ensure the change of the environment maps at interactive frame rates. In addition to the mesostructure captured by the BTF, a further essential ingredient for the realistic rendering of cloth are macroscopic shadows caused by self-shadowing of the object. These shadows enhance especially the draping of the fabrics. The main contribution of this paper is a new algorithm for the accurate realistic real-time visualization of a wide variety of cloth, including highly structured materials like corduroy or knitwear based on measured reflection properties. Special features of this algorithm are

- Preserving the "look and feel" of the real cloth.
- Support of point and directional light sources as well as image based lighting at interactive frame rates.
- A simple, but efficient technique to calculate dynamic shadows caused by point or directional light sources with smooth shadow boundaries on polygonal meshes
- A new efficient decomposition technique for illumination of geometry with BTF data, including self-shadowing.

The rest of the paper is organized as follows: in section 2 we briefly describe related and previous work. Section 3 describes our measurement setting and discusses the postprocessing of raw image data. Section 4 describes our BTF-Renderer for point and directional light sources including a special method to include macroscopic shadows due to self-shadowing on the cloth and to generate smooth shadow boundaries. In section 5 we extend the methods in order to illuminate the clothes using high dynamic range environment maps. Here we describe also a decomposition method for the illumination of a geometry. Section 6 presents some result images and reports on storage requirements and frame rates before concluding in section 7.

2. Related Work

2.1. Modelling Mesostructure

Previous work in cloth rendering falls into two main categories. The first is the explicit modelling of the underlying mesostructure and rendering it using volumetric techniques. Modelling has the general advantage of being able to create complete artificial results for non-existing materials. While certain approaches are not real-time capable[21, 22, 55], some interactive methods exist which use special shading models[13, 12]. Up to now, these algorithms are mainly used for knitwear and cannot handle materials like e.g. corduroy. Image based lighting and macroscopic self-shadowing are neglected.

2.2. Measuring reflection properties

Using measured reflection properties of real world surfaces naturally implies higher realism. Effects, which give important visual clues for material identification, like microstructure self-shadowing or scattering are preserved. On the other hand careful measuring is required.

Light fields

Capturing images of models under different lighting conditions and from different viewing angles automatically captures the reflection properties and yields very realistic renderings of the objects, although using these so called light field approaches[14, 7, 40, 19], it is not possible to change the lighting conditions. A general drawback of these approaches is that the measured material properties are coupled with a fixed geometry, thus not allowing to change the geometry or the material without remeasuring the object.
Malzbender et al.[42] introduced polynomial texture maps, where the coefficients of a biquadratic polynomial are stored per texel, and used to reconstruct the surface color under varying lighting conditions. Lensch et al.[39] proposed a method to capture spatially varying materials on known geometry, by finding basis BRDFs for reconstruction on a per-pixel level. These approaches can also be applied for cloth.

BRDFs

BRDFs are four dimensional functions and were introduced by Nicodemus[46]. These functions describe the reflection distribution at a surface point depending on incoming and outgoing light directions. BRDFs overcome the limitations of geometry coupling, fixed lighting and viewing directions. Early results approximated a single BRDF by a Ward[37] or Lafortune[36] model. Ashikhmin[2] e.g. produces good results for velvet by incorporating a special shadowing term. Kautz and McCool[31] approximate the four-dimensional BRDF by a product of two two-dimensional functions splitting viewing and light direction, which are stored as textures and combined during the rendering step. McCool et al.[44] improved the above method by employing homomorphic factorization, leading to approximations with user controllable quality features. The above approaches were further improved[50, 51, 38], which all enable the BRDF to be lit by image based illumination while relying on different approximation functions. Unfortunately, their representations cannot easily be applied for realtime rendering of spatially varying materials.

BTFs

BTFs were introduced by Dana et al.[11]. A planar surface sample is lit by a directional light source and photographed from different directions. Thus the resulting images are a function of viewing and illumination direction, hence capturing effects caused by the mesostructure of a surface, like roughness, self-shadowing, occlusion, inter-reflections, subsurface scattering and color bleeding. Registering the different images of the BTF the data can be considered as a 6 dimensional reflectance field

$$L = L(x, y, \theta_i, \phi_i, \theta_o, \phi_o)$$

which connects for each surface point (x, y) of a flat sample the outgoing to the incoming radiance in the direction (θ_o, ϕ_o), (θ_i, ϕ_i) respectively. The measurement is done in RGB space, wavelength changes and time dependent effects like fluorescence are ignored. Due to the computational complexity of the 6 dimensional function only a few realtime rendering algorithms exist[32, 43]. To achieve interactive rates, Kautz et al. use an approximation to an anisotropic version of the Blinn-Phong model and to the Banks model. In his recent work McAllister et. al. represented the 6D-reflectance field as a spatially varying BRDF. At each discretizised surface position a Lafortune model is fitted and the parameters are stored in a texture map, which is called SBRDF. This representation can efficiently be evaluated in current graphics hardware. In addition to point and directional light sources their algorithm also supports image based illumination[4, 45, 20, 15]. Though their algorithm yields good results for materials with low depth range, it proves unsatisfactory for more structured materials with high depth, as even for a high number of lobes the Lafortune model is hardly capable of capturing the variation in the reflectance behavior caused by the mesostructure.

2.2.1. Measuring and synthesizing BTF data

In their pioneering work, Dana et. al. measured 61 samples of real-world surfaces and made them publicly available in the CUReT[26] database. Unfortunately, their data is not spatially registered. In order to demonstrate the enhancement over common texture mapping, we manually performed the registration for a small number of samples and mapped them

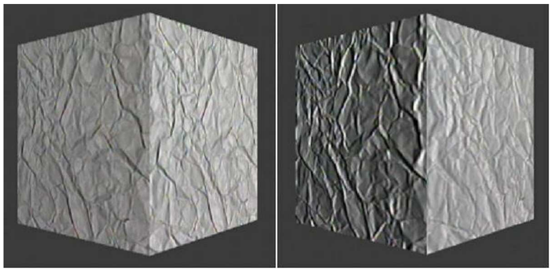

Figure 2: *The images show texture mapped cubes using the post-processed CUReT BTF data sample crumpled paper. In the left image only a frontal viewed texture is applied. The right image uses the BTF data set.*

onto a cube, as shown in figure 2. Self-shadowing and self-occlusion of the mesostructure on the surface are clearly visible. A drawback of the CUReT database is that it contains some graphical errors, caused by frame-grabber artifacts or reflections of the robot sample holder plate visible in the raw data. Our solution to these problems is described in section 3.1.

Synthesizing BTF data addresses two problems. If only a discrete set of BTF samples is available it allows to synthesize the continuous BTF and furthermore it allows to synthesize BTF data of arbitrary size. Liu et al.[41] registered some samples from the CUReT database using statistical properties and appearance preserving procedures. Further methods to synthesize BTF data on a surface is described in Tong et al.[53] using 3D textons or using histogram models[10]. The advantages of these methods are the low memory requirements and that the overall structure and appearance is preserved. On the other hand, by introducing statistical and random components these methods destroy certain mesostructures, hence changing the BTF significantly and are not suitable for all kinds of materials, see e.g.[53]. In order to preserve the mesostructure we use the measured image data, which is sampled dense enough to not require any synthesis and nevertheless stored in a compact form in memory. Because of the tileability of our fabrics the size of the measured probe is sufficient for our needs.

3. Measurement

This section describes the process of measuring and post-processing the bi-directional texture function.

3.1. Setup and Data acquisition

Our setup is designed to conduct an automatic measurement of a BTF that also allows the automatic alignment and post-processing of the captured data. We restrict ourselves to planar samples with the maximum size of $10 \times 10\ cm^2$. In spite of these restrictions we are able to measure a lot of different material types, e.g. fabrics, wallpapers, tiles and even car interior materials. As shown in figure 3, our laboratory consists of a HMI (Hydrargyrum Medium Arc Length Iodide) bulb (broncolor F575), a robot (intelitek SCORBOT-ER4u) holding the sample and a rail-mounted CCD camera (Kodak DCS 760). Table 1 shows two different samplings H_1 and H_2 of the halfspace of point X above the sample. According to the varying reflection properties of each sample, the sampling must be sparser or denser. We used a maximum of $n = 81$ unique directions for camera and light position as shown in table 1, resulting in an approximately equal sampling of the hemisphere. Figure 4 shows three measured samples: *CORDUROY*, *PROPOSTE* and *WOOL*. 6561 raw images were captured for each sample, each 6 megabytes in size (lossless compression) with a resolution of 3032×2008 pixels (Kodak DCR 12-bit RGB format). To ensure the correct correspondence of the measured reflection properties to a fixed surface position on the sample, we pay close attention to minimize positioning errors.

θ_1 [°]	$\Delta\phi$ [°]	θ_2 [°]	$\Delta\phi$ [°]	No. of images
0	–*	0	–*	1
17	60	15	60	6
34	30	30	30	12
51	20	45	20	18
68	18	60	18	20
85	15	75	15	24

Table 1: *Two different sampling densities H_1 and H_2 of viewing and illumination angles of the BTF database.* **= only one image taken at* $\phi = 0°$

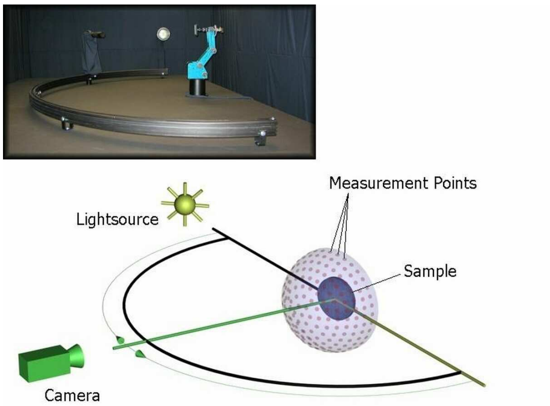

Figure 3: *Measurement setup consisting out of an HMI lamp, a CCD camera and a robot with a sample holder.*

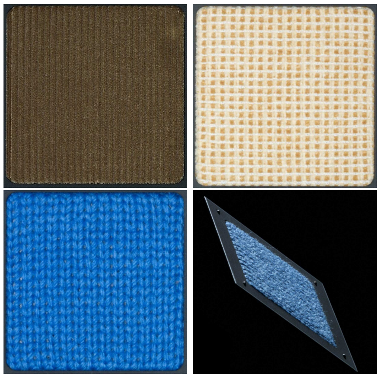

Figure 4: *Measured BTF samples; from left to right (top row): CORDUROY, PROPOSTE. Bottom row: WOOL frontal and perspective view.*

Figure 5: *Sample holder with the PROPOSTE sample. The left image shows the frontal view ($\theta = 0°$, $\phi = 0°$); the right image shows ($\theta = 60°$, $\phi = 342°$). White point and border markers are visible.*

3.2. Postprocessing

After the measurement the raw image data is converted into a BTF representation, i.e. the perspectively distorted images must be registered. In this representation a complete set of discrete reflectance values for all measured light and viewing directions is assigned to each texel of a 2D texture. Registration is done by projecting all sample images onto the plane which is defined by the frontal view ($\theta = 0, \phi = 0$). To be able to conduct an automatic registration we have attached point and borderline markers to our sample holder plate, see figure 5. After converting a copy of the raw data to black-and-white (8-bit TIFF), we use standard image processing tools, to detect the markers during the measurement process. We restrict ourselves to the common 8-bit RGB texture format. To take advantage of the linear part of the camera response curve, we choose the central 8-bit range of the 12-bit images. As we use a fixed focal length during one measurement, the maximum effective resolution of the sample holder in the image is 1100×1100 pixels. After all transformations are carried out, we rescale all images to an equal size of 1024×1024 pixels, which we call *normtextures* (N). After this postprocessing step, the data amount of 167 gigabytes captured by the camera CCD chip is reduced to roughly 20 gigabytes of uncompressed data. By measuring planar probes of a certain size, we rely on the tileablility of our fabrics. Therefore, a manually chosen region of interest (approximately 550×550 pixels) is cut out and resized. To create the final normtextures (256×256 pixels in size) linear edgeblending is applied, which reduces the usual tiling artifacts.

4. Illumination using point and directional light sources

To allow a closer inspection of the measured fabrics, single point or directional light sources can be used. The easiest way to texture an object with a BTF texture would be to store a complete database in memory and fetch the nearest measured BTF image to the current viewing and lighting direction. This way, the textures would approximately be viewed under the same angle they were acquired and therefore artifacts due to anisotropic sampling are avoided. The texturing can be done on a per face basis, introducing edge artifacts or on a per-vertex basis using blending, as described by Chen[7]. Unfortunately the size of the database of one sample at a resolution of 256×256 pixels exceeds 1230 megabytes, which is not practical on today's hardware. Next, we present our algorithm that overcomes this problem. The main idea is, to replace for each viewing direction the BTF defined by the normtextures by a series of basis textures obtained by using a principal component analysis. Utilizing only a few components (≤ 16) of this series, the texture can be reconstructed at runtime.

4.1. PCA

Principal component analysis[29, 33, 48] has been widely used to compress image data[47]. Ramamoorthi[49] showed by an analytic PCA construction, that using about five components is sufficient to reconstruct lighting variability in images of a lambertian object.

Our measured samples all have a certain three-dimensional mesostructure, which leads to significantly varying surface appearance for changing viewing directions. To ensure a pixel position coherence, thus coping with the varying height of a surface position on the sample, we do a principal component analysis for each of the n viewing directions separately. We call these directions view slots $S_j, j \in (1 \dots n)$. Thus in these slots the viewing direction is fixed so that only the light direction varies, therefore the analysis is done only on the effects caused by the changing illumination. The n normtextures $N_{ij}, i \in (1 \dots n)$ per view slot j are represented as vectors $X_{ij} = (r_{1,1}, g_{1,1}, b_{1,1}, \dots, r_{h,w}, g_{h,w}, b_{h,w})$ of dimension $3 \times h \times w$, where h and w are the height and width of the normtextures, respectively. We perform a PCA of these vectors, resulting in a series of eigenvalues $\lambda_{1j}, \dots, \lambda_{nj}$ and eigenvectors $E_{1j}, \dots, E_{nj}$ which corresponds to eigennormtextures $B_{1j}, \dots, B_{nj}$ for this slot. The first $c < n$ eigennormtextures approximate any of the original normtextures N_{ij} in such a way that the sum of the squares of the projection errors onto the affine subspace spanned by $\{B_{1j}, \dots, B_{cj}\}$ is

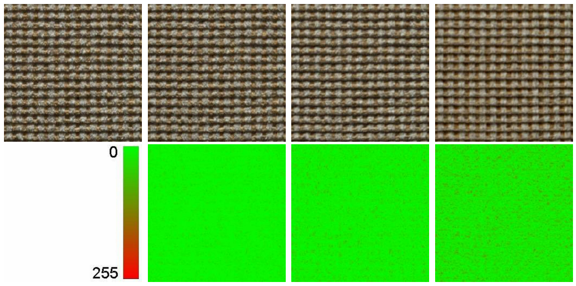

Figure 6: *Texture reconstruction using PCA. From left to right (top row): original normtexture, 16, 10, 5 components. Bottom row: difference images to the original, see text for details.*

minimized

$$N_{ij} \approx \sum_{k=1}^{c} p_{ikj} B_{kj}, \quad i = 1 \dots n. \tag{1}$$

The coefficients $p_{ikj} = N_{ij} \cdot B_{kj}$ are weights, where $\cdot$ denotes the standard scalar product in $\mathbb{R}^{3 \times h \times w}$. Figure 6 gives examples for reconstructed textures with a different number of eigennormtextures, and also shows difference images. Therefore, we calculated the length of the 8-bit RGB error vector between the original normtexture and the reconstructed images. Green color indicates a length of zero, whereas red indicates a length of 255 units. Figure 7 shows the absolute eigenvalues for all components of three different view slots. The decay of the absolute values indicate the statistical dimensionality of our given normtextures. As the eigenvalues decrease rapidly in all our examples, $c = 16$ components were sufficient to reproduce the look and feel of the sample materials. Note, that performing a principal component on the different view-slots reduces the size of our data set from about 1230 megabytes to 260 megabytes per sample for a 256×256 resolution.

4.2. Real-time algorithm

In this section we describe the algorithm to reconstruct the texture T for a vertex V of a given triangle mesh at runtime, while using a single point or directional light source. The emitted radiance $\mathbf{g}$ from the light source is stored as a three-component RGB float vector. We first compute the light and view vector $(\hat{l}, \hat{v})$ for the vertex V. Because of the memory requirements for storing the raw normtextures, we now use the representation of our textures as a series of basis normtextures B_{kj}. Choosing the nearest slot j corresponding to $\hat{v}$ and the weights p_{ikj} corresponding to $\hat{l}$ the texture T_j can be reconstructed.

$$T_j \approx \mathbf{g} \sum_{k=1}^{c} p_{ikj} B_{kj} \tag{2}$$

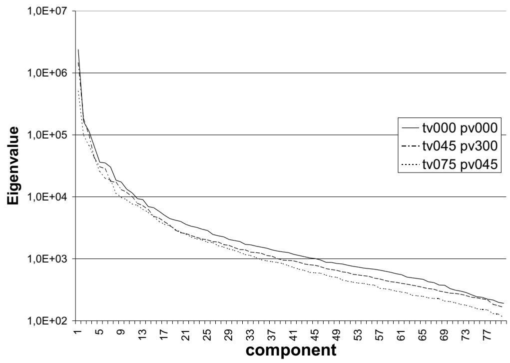

Figure 7: *Eigenvalues for the PROPOSTE sample in three different view slots j ($\theta = 0°$, $\phi = 0°$), ($\theta = 45°$, $\phi = 300°$), ($\theta = 75°$, $\phi = 45°$).*

Because in general $\hat{l}$ does not match a measured direction exactly we use our known samplings H_1 or H_2 from the measurement to compute the four nearest measured light directions $i_m, m \in (1\ldots4)$ from our texture database for bilinear interpolation with the interpolation weights τ_m and with $N_{i_m j}$ denoting the reconstructed textures corresponding to i_m:

$$\begin{aligned} T_j &\approx \mathbf{g}\left(\tau_1 N_{i_1 j} + \tau_2 N_{i_2 j} + \tau_3 N_{i_3 j} + \tau_4 N_{i_4 j}\right) \\ &= \mathbf{g}\sum_{m=1}^{4} \tau_m N_{i_m j} \\ &= \mathbf{g}\sum_{m=1}^{4} \tau_m \sum_{k=1}^{c} p_{i_m k j} B_{kj} \\ &= \mathbf{g}\sum_{k=1}^{c} \left(\sum_{m=1}^{4} \tau_m p_{i_m k j}\right) B_{kj} \\ &= \mathbf{g}\sum_{k=1}^{c} \gamma_{kj} B_{kj} \qquad (3) \end{aligned}$$

This means, that the texture T_j is simply a weighted sum of basis textures.

$$T_j = \mathbf{g}\cdot(\gamma_{0j}B_{0j} + \gamma_{1j}B_{1j} + \ldots + \gamma_{cj}B_{cj}) \qquad (4)$$

We use a *fragment program* to accomplish the reconstruction of the texture with $c = 16$ components using a ATI Radeon 9700. Therefore, γ_{kj} is transferred to the GPU for each vertex and when blending the three resulting textures per triangle in a three pass rendering, a smooth transition is ensured[7]. If also view interpolation is desired, denote the four nearest view slots as $j_m, m \in (1\ldots4)$ with the corresponding interpolation weights ω_m. Following (3) we obtain:

$$\begin{aligned} T &= \omega_1 T_{j_1} + \omega_2 T_{j_2} + \omega_3 T_{j_3} + \omega_4 T_{j_4} \\ &= \omega_1 \sum_{k=1}^{c_1} \gamma_{kj_1} B_{kj_1} + \omega_2 \sum_{k=1}^{c_2} \gamma_{kj_2} B_{kj_2} \\ &\quad + \omega_3 \sum_{k=1}^{c_3} \gamma_{kj_3} B_{kj_3} + \omega_4 \sum_{k=1}^{c_4} \gamma_{kj_4} B_{kj_4} \qquad (5) \end{aligned}$$

Note, that in the case of $j_1 \neq j_2 \neq j_3 \neq j_4$ four different eigennormtexture sets B_{kj_m} are needed.

4.3. Incorporating Shadows

In the context of cloth rendering, incorporating shadows and geometry self-shadowing is crucial for realistic rendering. Using point and directional light sources implies rendering hard shadow boundaries. An efficient method for this purpose are the well known shadow maps[54]. The calculation is hardware accelerated, e.g. through several OpenGL extensions[25]. Nevertheless, a common problem with shadow mapping is projection aliasing[1]. Increasing depth buffer size and precision, as well as polygon offsets[35] reduce these artifacts. Further improvements could be made using perspective shadow maps as introduced by Stamminger et. al[52]. Unfortunately, in spite of self-shadowing these artifacts are still visible, and destroy the realistic appearance of cloth. Therefore, we use volumetric shadows, as proposed by Crow[9]. A lot of work was done in this fieldm including hardware acceleration[24, 17, 3, 34, 18, 6, 23, 5, 6, 23, 5] Nevertheless, there is a further problem that leads to disturbing artifacts in cloth rendering: the shadow boundary always coincides with the silhouette of the mesh as seen from the light source. This silhouette is defined by those edges in the mesh which are incident to one front-facing and one back-facing triangle with respect to the light source position, respectively, see figure 8 left side. Therefore, in an arbitrary triangle mesh the silhouette edges do not define a smooth path but instead show a zigzag pattern. Note, that this is independent of the accuracy of the shadow computation and is worse for low-resolution meshes which are common in cloth modelling. Furthermore, if the light source moves, the silhouette edge jumps between adjacent triangles leading to disturbing artifacts. One way to cope with these problems is to consider the mesh as a smooth surface. This is actually also assumed during rendering, when interpolating the vertex normal vectors for lighting calculations. Using this observation leads to a simple solution to the problem. If the sign of the scalar product between the normalized vertex normal $\hat{n}_1$ and the normalized light-vector $\hat{l}_1$ of V_1 and $\hat{n}_2, \hat{l}_2$, respectively, changes along an edge V_1V_2 of a triangle , the shadow boundary lies between these two vertices and the position of this boundary (P) on an assumed smooth surface can be estimated by the proportion of the angles at the two vertices V_1 and V_2, see figure 9. Therefore, for each vertex V of a triangle we compute one-dimensional texture coordinates

$$(u)_V = (1.0 + cos(\alpha + \angle(\hat{n}_V, \hat{l}_V)))/2.0, \qquad (6)$$

into a one-dimensional 1D half black and half white texture of 1024 pixels size. Using this texture leads to the smooth shadow boundary. In order to generate soft boundaries this texture can be blurred. α is an offset, to compensate the pop-

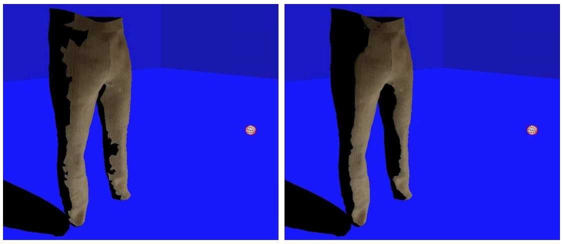

Figure 8: *Shadow boundaries. Left image shows the zigzag behaviour, which is gone in the right image using our technique.*

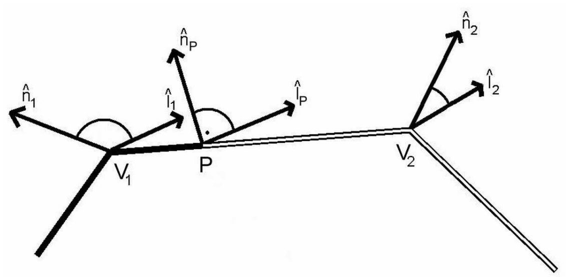

Figure 9: *Computing position P of the shadow boundary between V_1 and V_2.*

ping artifacts, caused by the silhouette edge jumps. We have chosen $\alpha = 15°$ for our high resolution pair of trousers. The combination of this simple texturing method with the shadow volume algorithm in the form of blending, delivers nice results as shown in figure 8 right side. Note, that the presented algorithm is also well suited for rendering dynamic meshes with macroscopic shadows, since the needed computations and the shadow volumes can be carried out each frame easily.

5. Image based illumination

In this section we show how to illuminate the geometry using high dynamic range environment maps[16, 15, 27]. The next four subsections describe the numerical integration of the rendering equation, the computation of visibility maps, the runtime algorithm for using image based illumination and a method to decompose the illumination of the geometry, to allow a faster change of the environment map.

5.1. Numerical integration of the Rendering Equation

Following the rendering equation [30, 28] at a surface point x the outgoing radiance L_o is given by

$$L_o(x,\mathbf{w}) = L_e(x,\mathbf{w}) + \int_S f_r(x,\mathbf{w}',\mathbf{w}) L_i(\mathbf{w}') V(x,x') G(x,x') dA', \quad (7)$$

where $\mathbf{w}$ is the outgoing direction, L_e the emitted radiance, S the hemisphere domain over x, f_r the BRDF, x' another surface point, $\mathbf{w}'$ the direction from x' to x, L_i the incident radiance, $V(x,x')$ the visibility between the two surface points and $G(x,x')$ a geometrical term defined as

$$G(x,x') = -\frac{(\vec{\mathbf{w}}' \cdot \vec{\mathbf{n}}')(\vec{\mathbf{w}}' \cdot \vec{\mathbf{n}})}{||x' - x||^2} \quad (8)$$

with the normal $\mathbf{n}$ at x and $\mathbf{n}'$ at x', respectively. For our purposes we set the emitted radiance $L_e(x,\mathbf{w}) = 0$ and do not compute any inter-reflections. We discretize the hemisphere domain using a hemicube[8], which leads to

$$L_o(x,\mathbf{w}) \approx \sum_\alpha f_r(x, p_\alpha \to x, \mathbf{w}) L_i(p_\alpha \to x) V(x, p_\alpha) G(x, p_\alpha), \quad (9)$$

where p_α is a pixel of the hemicube.

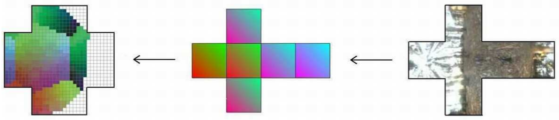

Figure 10: *Visibility map computation. Visibility map (left) with rendered color-coded lookup environment map (middle). White color in the visibility map stands for occlusion caused by the mesh. On the right side a HDR environment is shown, which is mapped onto the color-coded one.*

5.2. Visibility map pre-computation

Because we use image based illumination and store our radiance values in an environment map we have to provide a lookup into this map. Therefore, we precalculate visibility maps M for each vertex. These maps store a discretization of the hemisphere of the vertex V, which is a hemicube with its top side perpendicular to the vertex normal. Figure 10 (left) shows an unfolded hemicube. Using a color-coded environment map (figure 10 middle) a look-up table into a high dynamic range map (figure 10 right) is created. This allows easy exchange of the environment map. By also rendering the geometry itself macroscopic self-shadowing is included. Because a pixel p_α represents a certain direction $(V \to p_\alpha)$ and does not necessarily match one of the measured directions, we subdivide our visibility map into n direction patterns, as seen in the figure 11, and assign the four nearest measured directions in respect to $(V \to p_\alpha)$ to p_α. This allows us to do a bilinear interpolation with the interpolation weights $h_{d_k}, k \in (1 \ldots 4)$ for all four directions d_k. A visibility map pixel now stores the following information:

- visibility of a pixel of the environment map and if it is visible, the position of this pixel in the map
- four nearest measured directions in respect to the direction represented by this pixel
- corresponding interpolation weights

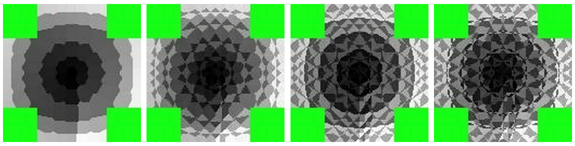

Figure 11: *Visibility map (64 × 64 resolution) direction encoding with n grey levels. From left to right: Nearest, second nearest, third and fourth nearest direction.*

5.3. Real-time algorithm using image based illumination

It would require $n = 81$ multi-texturing passes for each triangle to incorporate all measured light directions, which cannot be done in real-time. Therefore, we now show how to use the visibility maps and our representation as a series of basis normtextures, to illuminate a triangle mesh using high dynamic range images. First the view vector $\hat{v}$ for each vertex V is calculated and the nearest view slot j is chosen. At this point we have to evaluate the radiance $\mathbf{g_i}$ coming out of our n measured directions at each vertex. Similar to equation 2 we now calculate the texture T_j as follows:

$$\begin{aligned} T_j &\approx \sum_{i=1}^{n} \mathbf{g_i} N_{ij} \\ &= \sum_{i=1}^{n} \mathbf{g_i} \sum_{k=1}^{c} p_{ikj} B_{kj} \\ &= \sum_{k=1}^{c} \left(\sum_{i=1}^{n} \mathbf{g_i} p_{ikj} \right) B_{kj} \\ &= \sum_{k=1}^{c} \gamma^*_{\mathbf{kj}} B_{kj} \qquad (10) \end{aligned}$$

Introducing a multiplication factor f, denoting the exposure level of the high dynamic range map, the texture T_j is reconstructed very similar to 4:

$$T_j = f \cdot (\gamma^*_{\mathbf{0j}} B_{0j} + \gamma^*_{\mathbf{1j}} B_{1j} + \ldots + \gamma^*_{\mathbf{cj}} B_{cj}) \qquad (11)$$

Note, that now $\gamma^*_{\mathbf{kj}}$ is also a three component float vector.
In order to compute $\mathbf{g_i}$ for a vertex V, for all $p_\alpha \in M_V$ a lookup into the environment map at the position stored in p_α is performed. The radiance $\mathbf{r}$ stored at that position is assigned to $\mathbf{g_{d_k}}$ and weighted with h_{d_k}. Here $d_k, k \in (1 \ldots 4)$ denotes the four directions stored with p_α as described above. For view interpolation the same calculations as in (5) have to be applied.
$\gamma^*_{\mathbf{kj}}$ is computed for all vertices $V_\zeta, \zeta \in (1 \ldots N)$ where N is the number of vertices of the geometry. Thereby we introduce a new vector U holding all $\gamma^*_{\mathbf{kj}\zeta}$:

$$U = (\gamma^*_{111} \ldots \gamma^*_{c11}, \ldots, \gamma^*_{1nN} \ldots \gamma^*_{cnN}) \qquad (12)$$

with the dimension $3 \times c \times n \times N$. This vector has to be calculated once per environment map and allows the real-time change of the viewing position and of the exposure f.

A drawback of this method is, that changing the environment map implies a complete new calculation of $\mathbf{g_i}$ for all vertices. This heavily depends on the visibility map resolution and the number of vertices and therefore on the hardware rendering speed. Reducing the visibility map resolution adaptively to achieve interactive changing rates introduces under-sampling artifacts of the environment map during motion, which can be compensated if the change stops, by using an adaptively higher resolution for the visibility map.

5.4. Decomposition of the Environment Map

To overcome the problems mentioned in the last section we propose a decomposition method. For this we again use a principal component analysis. As aforementioned, we have to evaluate all incoming radiance, if the object is rotated or the environment is changed. Daily observation shows that e.g. rotation of an object under natural illumination leads only to slight irradiance changes on the object surface, if the rotation angle is small.
The key idea is that we now compute a set of vectors $U_a, a \in (1 \ldots A)$, where A denotes the number of different environment maps used (see also subsection 5.3). Performing a PCA on these vectors, results in a series of new eigenvalues and eigenvectors. The latter correspond to eigenweightsets $W_1, \ldots, W_A$. The first $e < A$ eigenweightsets can be used to approximate any of the original weightsets U_a:

$$U_a \approx \sum_{k=1}^{e} o_{ak} W_k, \quad a = 1 \ldots A. \qquad (13)$$

The coefficients $o_{ak} = U_a \cdot W_k$ are weights, where $\cdot$ denotes the standard scalar product in $\mathbb{R}^{3 \times c \times n \times N}$.
To test our method we rotate an object relative to the environment and compute the vector U for each rotation step. This is equal to using several different environment maps. By using $\nu = 12^\circ$ degree steps we obtain $A = 30$ weight sets. We use a high resolution environment and visibility map (256×256 pixels). A comparison between reconstructed ($e = 5$ eigenvectors) and original images is shown in figure 13. We calculated the length of the RGB error vector between the original and the reconstructed images. Green color indicates a length of zero, whereas red indicates a length of 255 units. Increasing the number e of used eigenweightsets clearly minimizes the error. Figure 12 shows the weights o_{ak} for all $A = 30$ sets for all eigenvectors. Note the oscillation denoting the rotation around the object axis. As a result, we now can rotate the object or the environment, hence changing the lighting situation and the view at interactive frame rates, by reconstructing the complete weight set U for a desired rotation angle $\delta \in (0 \ldots 360^\circ)$ at runtime.

6. Results

We implemented our method within an interactive hardware-accelerated OpenGL system. The results were obtained under Windows 2000 on a 1.5GHz Athlon with a ATI Radeon

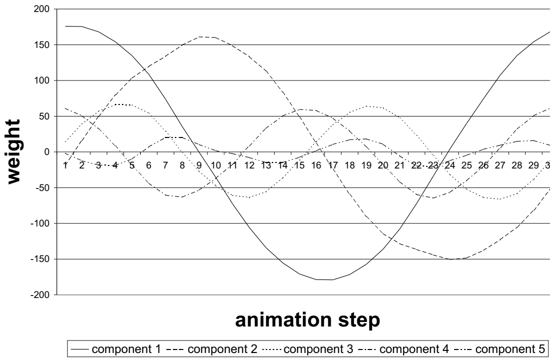

Figure 12: *Change of the weights for each component during the animation. Note the oscillations, denoting the rotation of the object.*

9700 graphics accelerator. For the texture reconstruction we used a *fragment program* (GL_ARB_fragment_program) and multi-texturing. The Eigentextures for all samples have a resolution of 256×256 and need about 260 megabytes memory per sample. Figure 1 and all images in figure 14 were rendered at a 1280×960 resolution. Our used meshes range from nearly 800 vertices to 9200 vertices. Table 2 gives an overview over the frame rates achieved and preprocessing times needed. While enabling the decomposition, we get about 2.0 frames per second, including the dynamic change of illumination and camera position, instead of the recalculation time of 3700 milliseconds per change for the shirt mesh (900 vertices). The algorithm is capable of rendering several BTF sets onto one geometry, if proper texture coordinates and material id's per vertex are supplied and sufficient system memory is available (see also the avatar in figure 14). The algorithm handles non-fabric materials (*WALLPAPER, STONE*) as well, as can be seen in figure 14 in the second row.

mesh name	vertices	illum. method	average frame rate [FPS]	HEM update time [msec]	VPT [msec]
shirt	900	PLS	9.3		
shirt	900	HEM	9.5	3.8k	8.0k
shirt (high)	9208	PLS	1.3		
shirt (high)	9208	HEM	1.1	38.5k	
pair of trousers	833	PLS	9.5		
pair of trousers	833	HEM	10.1	3.7k	7.1k
pair of trousers (high)	5222	PLS	2.1		
pair of trousers (high)	5222	HEM	2.1	23.2k	205.5 k

Table 2: *Results for different meshes and illumination methods. The frame rates were obtained using four times view blending with a total of 16 PCA components. PLS=point light source, HEM=high dynamic environment map, VPT=visibility preprocessing time.*

7. Conclusions

We have presented a method to capture and visualize reflection properties of cloth at interactive frame rates. Our approach decouples reflection properties from geometry while preserving the "look and feel" of a fabric, including important mesostructural features. The image based illumination allow the further usage in a desired clothing shop environment. With the presented decomposition method, interactive change of viewing and illumination is possible for static objects. While using single point or directional light sources we introduced a simple but effective method to compute smooth shadow boundaries. With the emersion of new graphic hardware in the near future, which supports more multi-texturing operations per pass we are able to do the four times view blending in one pass and/or increase the number of used PCA components. We are confident, that this will allow real-time frame rates. Future work will include the handling of deformable objects and dynamic meshes.

Acknowledgements

We would like to thank Jan Kautz from the MPI Saarbruecken and the anonymous reviewers for helpful comments, Markus Wacker from the University of Tuebingen for some of the used meshes. We also are grateful for the discussions with our colleagues Jan Meseth, Ferenc Kahlesz and Gabriel Zachmann. The HDR images were obtained from Paul Debevecs web page. The avatar was provided by human solutions.

References

1. T. Akenine-Moeller and E. Haines. *Real-Time Rendering 2nd edition*. A K Peters, 2002.
2. Michael Ashikhmin, Simon Premoze, and Peter Shirley. A microfacet-based brdf generator. *ACM Siggraph 2000 Conference Proceedings*, 2000.
3. Bill Bilodeau and Mike Songy. Real time shadows. *Creativity 1999*, 1999.
4. J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19, 1976.
5. Stefan Brabec and Hans-Peter Seidel. Shadow volumes on programmable graphics hardware. *Proc. EUROGRAPHICS 2003*, 2003.
6. Chris Brennan. Shadow volume extrusion using a vertex shader. *in Wolfgang Engel ShaderX*.
7. W-C. Chen, J-Y. Bouguet, M. H. Chu, and R. Grzeszczuk. Light field mapping: Efficient representation and hardware rendering of surface light fields. *Proceedings of ACM SIGGRAPH 2002*, 2002.
8. M. F. Cohen and D. P. Greenberg. The hemicube: A radiosity solution for complex environments. *Symposium on Computational Geometry*, 19(3):31–40, 22–26 July 1985.
9. Franklin C. Crow. Shadow algorithms for computer graphics. *j-COMP-GRAPHICS*, 11(2):242–248, July 1977.
10. Kristian J. Dana, Shree K. Nayar, Bram van Ginneken, and Jan J. Koenderink. 3D Textured Surface Modeling. *WIAGMOR Workshop CVPR '99*, 1999.
11. Kristian J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and Texture of Real-World Surfaces. *ACM Transactions on Graphics*, 1999.
12. K. Daubert, H. P. A. Lensch, Wolfgang Heidrich, and H.-P. Seidel. Efficient cloth modeling and rendering. *Rendering Techniques '01 - Proceedings of the 12th*, 2001.
13. K. Daubert and H.-P. Seidel. Hardware-based volumetric knit-wear. *Computer Graphics Forum 21(3) - Proceedings of EUROGRAPHICS*, 63(2):314–325, 2002.

14. P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. *Proceedings of ACM SIGGRAPH 2000*, 2000.

15. Paul Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

16. Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 369–378. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

17. Mark A. DeLoura, editor. *Game Programming Gems 2*. Charles River Media Inc., 2001.

18. Cass Everitt and Mark J. Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. *NVIDIA White paper*, 2002.

19. Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Computer Graphics*, 30(Annual Conference Series):43–54, 1996.

20. Ned Greene. Environment mapping and other applications of world projection. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

21. E. Gröller, R. Rau, and W. Straßer. Modeling and visualization of knitwear. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):302–310, 1995.

22. E. Gröller, R. Rau, and W. Straßer. Modeling textiles as three dimensional textures. *submitted for publication*, 1996.

23. Evan Hart, Dave Gosselin, and John Isidoro. Vertex shading with direct3d and opengl. *Game Developers Conference 2001*, 2001.

24. Tim Heidmann. Real shadows, real time. *Iris Universe*, 1991.

25. http://oss.sgi.com/projects/ogl sample/registry/. Opengl extension registry.

26. http://www.cs.columbia.edu/CAVE/curet/.

27. http://www.debevec.org/Probes/.

28. Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Natick, MA, 2001.

29. I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

30. J. T. Kajiya. The rendering equation. volume 20, pages 143–150, August 1986.

31. Jan Kautz and M.D. McCool. Interactive rendering with arbitrary brdfs using separable approximations. In *10th Eurographics Workshop on Rendering*, 1999.

32. Jan Kautz and Hans-Peter Seidel. Towards interactive bump mapping with anisotropic shift-variant brdfs. In *Proceedings 2000 SIGGRAPH/EUROGRAPHICS workshop on on Graphics hardware*, pages 51–58. ACM Press, 2000.

33. M. Kendall. *Multivariate Analysis*. Charles Griffin Co, 1975.

34. Mark J. Kilgard. More advanced hardware rendering techniques. *Game Developers Conference 2001*, 2001.

35. Mark J. Kilgard. Shadow mapping with todays opengl hardware. *SIGGRAPH 2002 Course*, 2002.

36. Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-linear approximation of reflectance functions. *Proc. SIGGRAPH 97*, pages 117–126, 1997.

37. Gregory J. Ward Larson. Measuring and modeling anisotropic reflection. *Proc. SIGGRAPH 92*, pages 265–272, 1992.

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

39. Hendrik P.A. Lensch, J. Kautz, Michael Goesele, Wolfgang Heidrich2, and Hans-Peter Seidel. Image-based reconstruction of spatially varying materials. *Proceedings of Eurographics Rendering Workshop t'01*, 2001.

40. Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, 1996.

41. Xinguo Liu, Yizhou Yu, and Heung-Yeung Shum. Synthesizing bidirectional texture functions for real-world surfaces. pages 97–106, 2001.

42. Tom Malzbender, Dan Gelb, and Hans Wolters. Polynomial texture maps. pages 519–528, 2001.

43. D. McAllister, A. Lastra, and W. Heidrich. Efficient rendering of spatial bidirectional reflectance distribution functions. *Graphics Hardware 2002, Eurographics / SIGGRAPH Workshop Proceedings*, 2002.

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

45. Gene S. Miller and C. Robert Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. In *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*. July 1984.

46. F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Reflectance nomenclature and directional reflectance and emissivity. *Applied Optics*, pages 1474–1475, 1970.

47. K. Nishino, Y. Sato, and K. Ikeuchi. Eigen-texture method: Appearance compression and synthesis based on a 3d model. 2001.

48. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical recipes in C - The art of scientific computation*. ISBN 0-521-43108-5. Cambridge University Press, 2nd edition, 1992.

49. Ravi Ramamoorthi. Analytic pca construction for theoretical analysis of lighting variability in images of a lambertian object. *PAMI Oct 2002*, 2002.

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

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

52. M. Stamminger and G. Drettakis. Perspective shadow maps. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002.

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

54. Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274. ACM Press, 1978.

55. Ying-Qing Xu, Yanyun Chen, Stephen Lin, Hua Zhong, Enhua Wu, Baining Guo, and Heung-Yeung Shum. Photorealistic rendering of knitwear using the lumislice. pages 391–398, 2001.

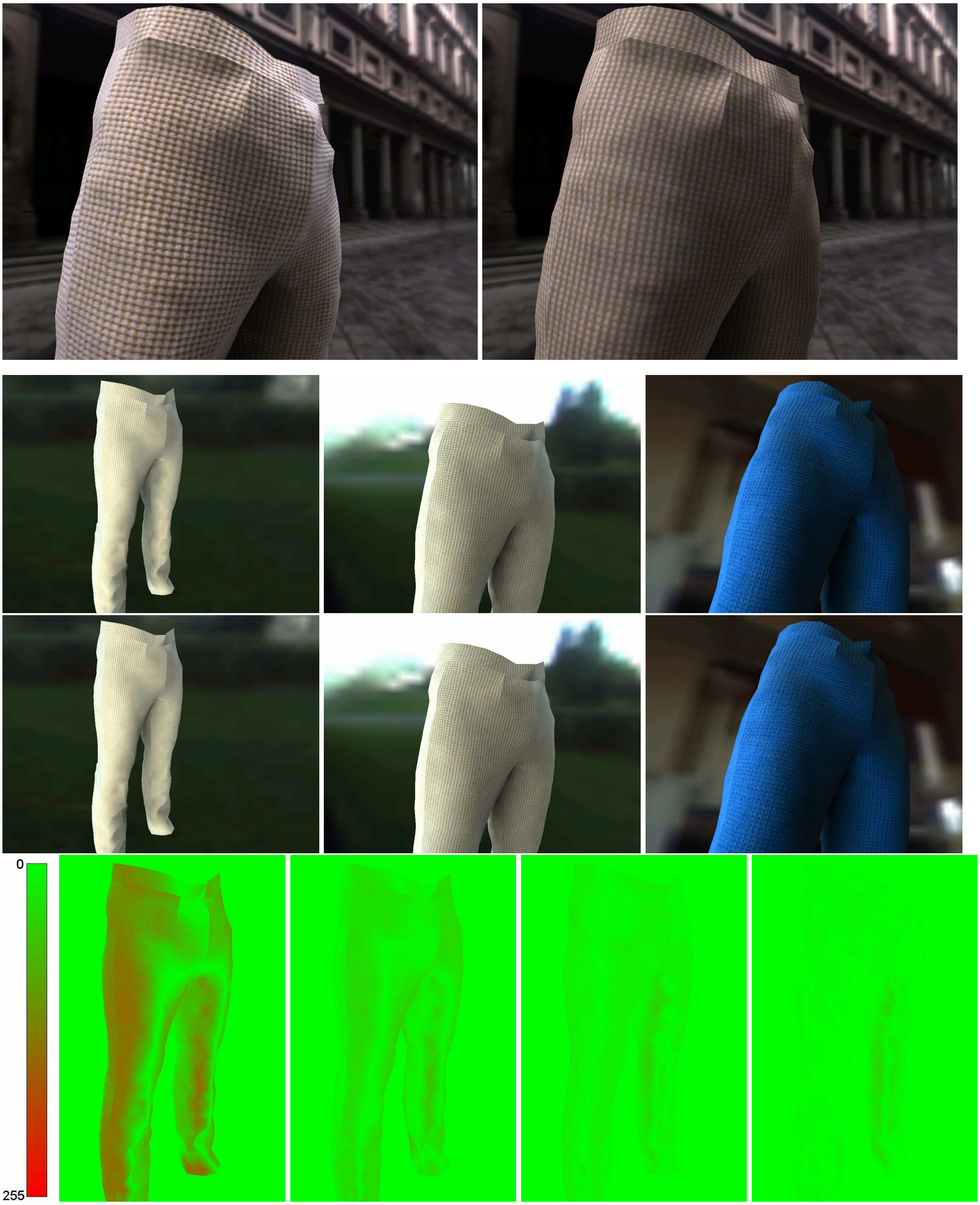

Figure 13: *Top row: PROPOSTE sample in Uffizi environment, left using BTF data, right normal texturing. In the right image the mesostructure is gone. Next rows: decomposition of the illumination of the geometry (from second to bottom row) original, reconstructed and difference error images. In the error image green denotes no error, while red denotes maximum error, see text for details. From left to right: 2,3,4 and 5 PCA components were used. The reconstruction was done with $e = 5$ eigenweight sets.*

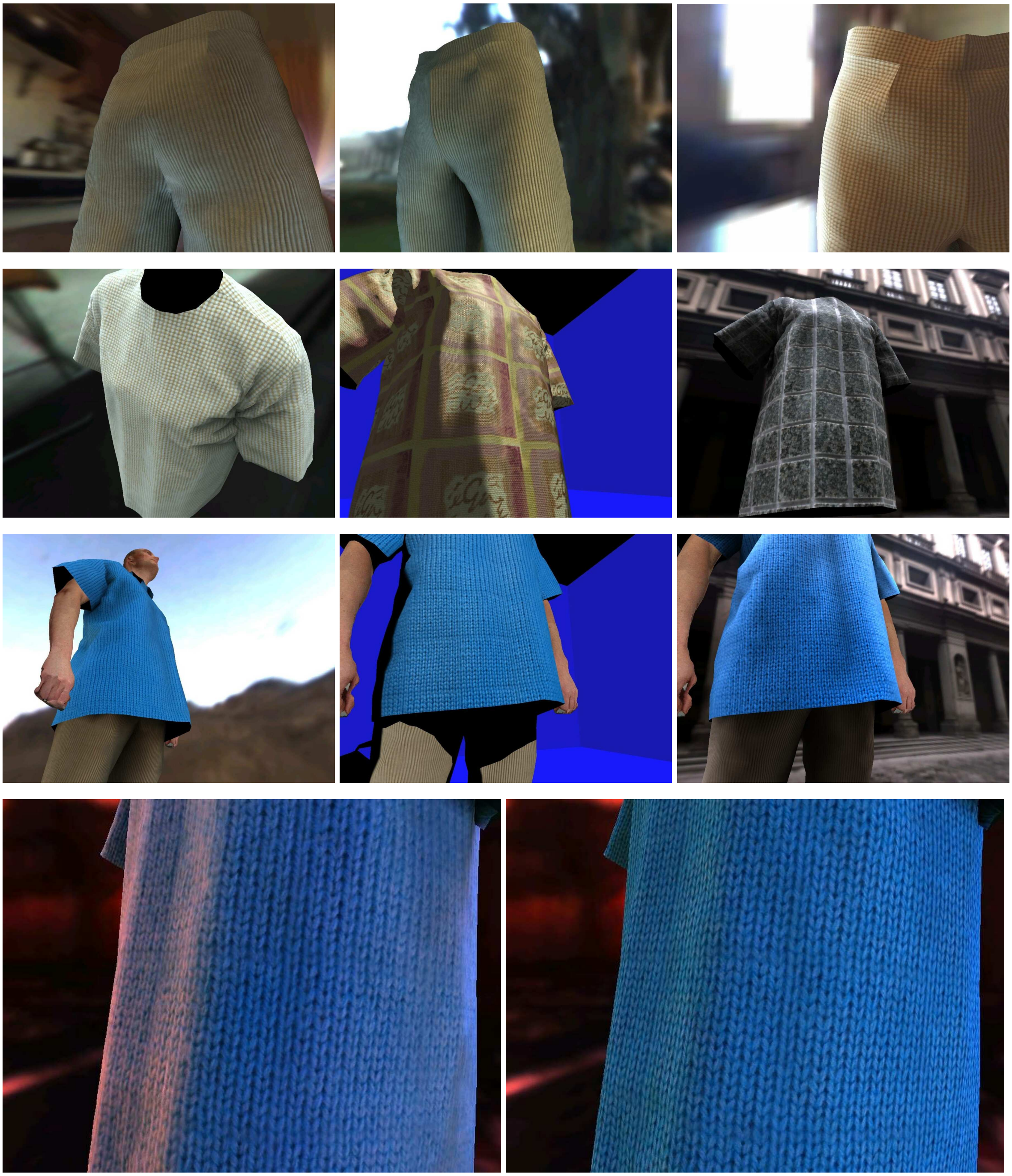

Figure 14: *Result images. Top row (from left to right): CORDUROY sample in Kitchen and RNL environment, PROPOSTE in Kitchen; next row: PROPOSTE in Building, WALLPAPER with point light source and STONE in Uffizi. Next row: WOOL and CORDUROY with avatar at Beach, with point light source and in Uffizi. Bottom row: WOOL sample in Grace environment, left using BTF data, right normal texturing. Notice the angular illumination dependence.*