Level-of-Detail Representation of Bidirectional Texture Functions for Real-Time Rendering

Wan-Chun Ma[1] Sung-Hsiang Chao[1] Yu-Ting Tseng[1]

Yung-Yu Chuang[1] Chun-Fa Chang[2] Bing-Yu Chen[1] Ming Ouhyoung[1]

[1]National Taiwan University [2]National Tsing Hua University

Abstract

This paper presents a new technique for rendering bidirectional texture functions (BTFs) at different levels of detail (LODs). Our method first decomposes each BTF image into multiple subbands with a Laplacian pyramid. Each vector of Laplacian coefficients of a texel at the same level is regarded as a Laplacian bidirectional reflectance distribution function (BRDF). These vectors are then further compressed by applying principal components analysis (PCA). At the rendering stage, the LOD parameter for each pixel is calculated according to the distance from the viewpoint to the surface. Our rendering algorithm uses this parameter to determine how many levels of BTF Laplacian pyramid are required for rendering. Under the same sampling resolution, a BTF gradually transits to a BRDF as the camera moves away from the surface. Our method precomputes this transition and uses it for multiresolution BTF rendering. Our Laplacian pyramid representation allows real-time anti-aliased rendering of BTFs using graphics hardware. In addition to provide visually satisfactory multiresolution rendering for BTFs, our method has a comparable compression rate to the available single-resolution BTF compression techniques.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: levels of detail, bidirectional texture function, anti-aliasing, real-time rendering

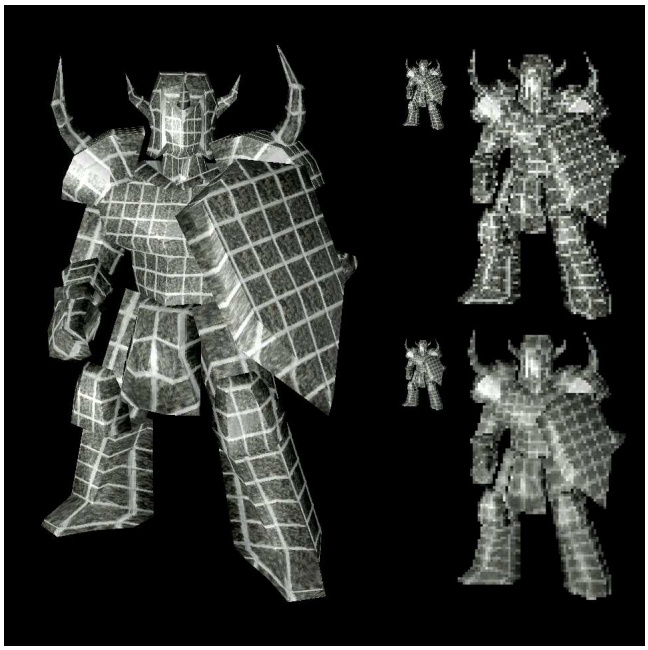

Figure 1: Comparisons between the BTF renderings with and without LOD representation. On the left, we show a model mapped with a BTF. On the right, the top portion shows the rendering result without LOD and the bottom shows the results with LOD. The smaller images are the original image displayed on the screen and the larger ones are their magnifications. Without LOD, there are the artifacts of jaggy lines on the model. With LOD, these artifacts are removed.

1 Introduction

Over the past few years, we have seen an impressive improvement in the capabilities of graphics processing units (GPUs). Among many revolutionary features, the most fascinating achievement is the realization of GPU programmability for real-time realistic rendering. Today, real-time rendering techniques have been widely utilized in video games. However, even with the help of today's graphics hardware, the visual quality of most games is still not as photorealistic as we might expect (although not every computer game needs this characteristic). One of the reasons behind is that most materials of 3D objects are still only modelled by a combination of multiple textures, such as the combination of bump, shininess, specular and diffuse maps. Such a material synthesis technique is widely adopted by games because it is suitable for GPU implementation. For example, in Xbox's Halo 2 [Bungie 2004], most of the objects are bump mapped to create depth illusion. However, bump mapping has its limitations. It only captures the shading caused by normal variation, but not the visually important effects such as self-shadowing, masking and complex non-diffuse reflection. To render more realistic materials interactively, there is a need for the general and efficient representations of complex materials suitable for real-time rendering.

BTF is an extension to textures dependent on the lighting and viewing direction and is suitable for modeling complex materials for real-time rendering. While rendering of BTF has been extensively studied for years, to best of our knowledge, there is no multiresolution technique for anti-aliased BTF rendering. Multiresolution technique for texturing is important as it allows anti-aliasing and efficient rendering. The goal of this paper is to bridge this gap by providing a multiresolution representation for real-time rendering of BTF at different LODs.

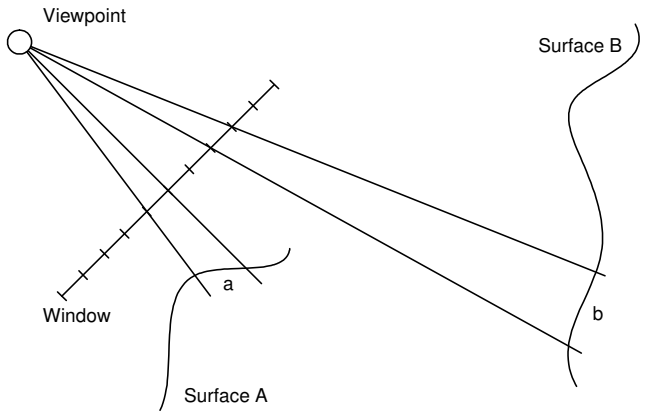

Figure 2: Illustration of the LOD issue in pixel shading.

Appearance of real materials has different LOD properties. Reflectance of homogeneous materials, which can be represented by a single BRDF, has no LOD issue: because the BRDF is only the ratio between incoming and outgoing flux. The ratio is always the same despite the distance to the camera. However, complex materials represented by BTF are different: under the same sampling resolution, the BTF ratio could be different as the distance from the camera to the object changes. For example, if we take a look at a weave cloth closely, the knitting patterns are visible and distinct. On the other hand, if the cloth is located far away from us, we can hardly see these details.

Take Figure 2 as an example. Two pixels are projected on areas a and b, which are on surfaces A and B respectively. Without loss of generality, the area of b is larger than a because surface B is farther away from the viewpoint. If the sampling rates at a and b are equal, more samples are required to precisely calculate the color of b. The implication is that it actually takes more time to correctly shade the area which is far from the viewpoint. This is not reasonable and run-time inefficient, since usually we pay more attention to the shading of the objects near the camera, not the far background. Hardware multisampling cannot solve this issue because the number of samples is not proportional to the pixel projection area (it is based upon fixed subpixel samples). Instead of trying to integrate the samples on-the-fly, we propose an idea called *precomputed LOD reflectance* to solve this problem: the appearance (reflectance) of different LODs should be precomputed and stored for later use. In this paper, we propose an LOD representation of BTF based on Laplacian pyramid. Our specific contributions are:

1. **LOD Representation for BTF Rendering:** At the rendering stage, the number of texture access depends on which LOD is chosen. If the surface is far away from the viewpoint, fewer levels are required to reconstruct its BTF. Therefore, the number of texture accesses is reduced and the performance is enhanced. Thus, the proposed method can achieve the anti-aliased BTF rendering in real-time.

2. **Good Compression Rate of BTF:** Although our initial goal is the anti-aliased and efficient rendering of BTFs, our representation also has a comparable compression rate to the other available BTF compression methods. It is because our BTF Laplacian pyramid not only provides a multiresolution representation but also removes much of the texel-to-texel BRDF correlation. Hence, the norms of most of the Laplacian BRDF vectors are around zero. Therefore, the variance and entropy are reduced and a good compression rate is achieved.

2 Related Work

Our method builds upon prior work in the fields of BTF and LOD. This section gives a brief overview of the related work and background knowledge relevant to the problem we try to address.

2.1 BTF

Dana *et al.* [1999] proposed the BTF representation to represent the appearances of real-world surfaces. A BTF is a six dimensional reflectance field:

$$F(p, \omega_i, \omega_o), \tag{1}$$

which connects for each texture coordinate p the outgoing direction ω_o to the incident direction ω_i. The raw data of a BTF is a series of images taken from different viewpoints under various incident lighting conditions. Usually there are two ways to arrange the BTF data: one is to treat it as a set of BTF images $F^{\omega_i,\omega_o}(p)$, which is the image for the lighting direction ω_i and the view direction ω_o; another is to take it as a set of per-texel BRDFs $F^p(\omega_i, \omega_o)$, which is the set of colors at a pixel p under different viewing and lighting directions.

Due to its high dimensionality, a BTF requires gigantic memory space for storage (usually more than hundreds of mega-bytes). Hence, similar to most image-based rendering approaches, how to efficiently compress and manipulate the BTF becomes an important issue. Methods such as PCA [Sattler et al. 2003; Müller et al. 2004a], chained factorization [Suykens et al. 2003], reflectance field [McAllister 2002; Meseth et al. 2004], and vector quantization [Leung and Malik 2001] have already been adopted to deal with the BTF in order to get better run-time efficiency. Recently, multilinear decomposition is also used for BTF analysis [Vasilescu and Terzopoulos 2004]. Ma *et al.* [2004] proposed to split the BTF into two bands: a lowpass (an average BRDF of the material) and a highpass band (shading effects of surface mesostructures). The lowpass band is fitted by the Phong shading model. This operation compresses the lowpass band to only a few of coefficients (shading parameters). For the highpass band, a polynomial-based compression method is applied to it. However, the highpass band still comprise shading signals with various frequencies. Direct compression of the intermixed highpass band may result in a less accurate reconstruction, in other words, the loss of visible shading variations. It is also hard to compute the coefficients of high power terms with a polynomial-based method because of the restrictions on numerical precision.

How to render BTF in real-time becomes another important issue. Many recent methods demonstrate the ability of rendering a single BTF mapped object in real-time. These methods usually take advantages of the graphics hardware to achieve real-time performance. For the readers who are interested in BTF processing and rendering, we recommend the excellent survey paper by Müller *et al.* [2004b].

2.2 LOD

LOD has many applications in graphics such as mesh simplification and terrain rendering. Here, we only discuss the LOD methods related to material shading and rendering.

Rendering in different LODs is very important in computer graphics. In a dynamic scene, a complex-shaded object could be viewed at different distances from the viewpoint. As the object moves further away from the viewpoint, one may notice some obvious

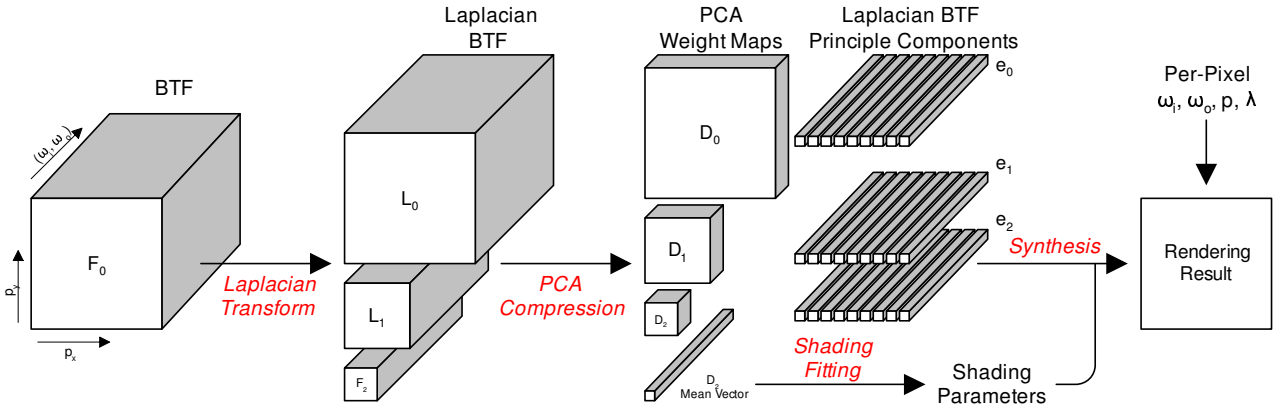

Figure 3: Illustration of the proposed algorithm. First, a Laplacian transform is applied to the BTF images. Then, the BTF subbands, which contain information of image-based appearance, are further compressed by PCA. The mean vector of the top level Laplacian pyramid is fitted by a parametric shading function. With the extracted eigen Laplacian BRDFs and shading parameters, we can reconstruct the BTF at the pixel shader stage.

aliasing artifacts such as shimmering, scintillation, and the Moiré patterns. The most popular anti-aliasing method for textures is *mipmapping* [Williams 1983]. Mipmapping works by creating lower resolution, prefiltered versions of the texture map. During rendering, the mipmap at the appropriate resolution is chosen. Hence, the texture pixels (texels) are already properly filtered when they are rendered on the screen. The intent of mipmapping is to keep the pixel-to-texel at least 1:1 in order to catch up the Nyquist rate, or in other words, to avoid aliasing. However, mipmapping can only be applied on color texture. Shading parameter maps such as shininess map cannot apply mipmapping because it is not physically correct.

For achieving LOD of arbitrary shading, Goldman [1997] created multiple shaders manually in order to render fur at different LODs. Meyer *et al.* [2000] proposed an analytical shader model based on micro-geometry reflection for rendering a forest of pine-trees. Self-shadowing and visibility are taken into account in the shaders without having to sample them. Adabala *et al.* [2003] demonstrated the ability of rendering woven clothes at any LOD. For LOD of normal maps, Fournier [1992] suggested to use a normal map pyramid, where each level stores distributions of surface normals. These distributions are represented as sums of a small number of Phong-like spreads of surface normals at a given resolution. Kautz *et al.* [2001] presented a method that automatically synthesizes bump maps at arbitrary LOD using a normal density function. Toksvig [2004] proposed a mipmapping technique of normal maps which uses shortening as a measure of normal variation to eliminate specular highlight aliasing.

Run-time efficiency is another important LOD topic, as pointed out by Olano *et al.*[2003]. They proposed an automatic shader simplification algorithm. The basic idea is to manually identify blocks of shader code that are candidates for reduction. And then, an LOD shader is created automatically by assembling these code blocks with appropriate conditionals. The LOD could be chosen by different viewing distances. They also mentioned the possibility of using different LOD selection criterions, such as hardware resource limits.

3 Algorithm

In this section, we describe our algorithm for generating and rendering the LOD representations of BTFs. Figure 3 shows an overview of our algorithm. The method for generating the LOD representation consists of three stages: band segmentation of BTF images with Laplacian pyramids (Section 3.1), compression of Laplacian BRDFs with PCA (Section 3.2), and data packing (Section 3.3). For rendering the LOD representations in real-time, we rely on the power of modern consumer graphics hardware (Section 3.4).

3.1 Band segmentation of BTF images with Laplacian pyramids

The input to our algorithm for generating LOD representations is a BTF, $F(p, \omega_i, \omega_o)$. We take BTFs from the BTF database provided by University of Bonn [1]. In this database, for each material, 6,561 images are taken (81 lighting and 81 viewing directions) to sample the variations of the appearance.

To reduce the color correlation for higher compression rate, we first convert each BTF image, $F^{\omega_i,\omega_o}(p)$, to the YCbCr color space. We then perform a multiresolution analysis for each channel separately. For each color channel, we decompose a BTF image into multiple subbands using Laplacian pyramids [Burt and Adelson 1983]. The resulted Laplacian pyramid represents an image as a series of bandpass-filtered images, each sampled at successively sparser spatial densities. The scale of the Laplacian operator doubles from a level to the above level of the pyramid, while the center frequency of the passband is reduced by an octave. The Laplacian pyramid is a versatile data structure with many attractive features for image processing. One essential property of Laplacian pyramid is that the first-order statistics of the bandpass filtered images are highly peaked around zero, which means these images are largely decorrelated and can be highly compressed. Another obvious option for band segmentation is the wavelet transform. We chose Laplacian transform because it requires less computation and is easier to be implemented on GPU.

[1] http://btf.cs.uni-bonn.de/

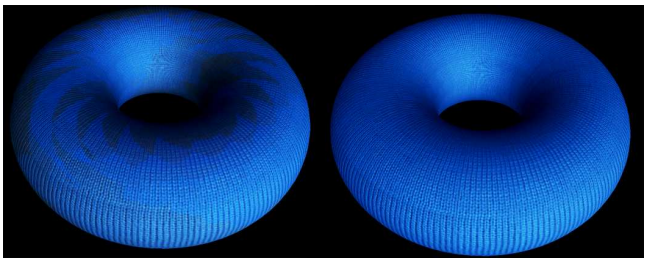

Figure 4: Elimination of shading discontinuity using a parametric shading function. On the left is the BTF rendering using the nearest lighting and viewing directions, which produces a distinct zigzag pattern. On the right, a parametric shading model is used to rectify the result.

We obtain a Laplacian pyramid for a BTF image by repeatedly applying the following procedure:

$$F_{k+1}^{\omega_i,\omega_o}(p) = \downarrow_p g(p) * F_k^{\omega_i,\omega_o}(p), \tag{2}$$

$$L_k^{\omega_i,\omega_o}(p) = F_k^{\omega_i,\omega_o}(p) - \uparrow_p F_{k+1}^{\omega_i,\omega_o}(p), \tag{3}$$

where $F_0^{\omega_i,\omega_o}(p) = F^{\omega_i,\omega_o}(p)$ is the original BTF image and $F_k^{\omega_i,\omega_o}(p)$ is the result of applying an appropriate low-pass filter $g(p)$ and the downsampling operation $\downarrow_p$ on $F_{k+1}^{\omega_i,\omega_o}(p)$. The dimension of $F_k^{\omega_i,\omega_o}(p)$ is $1/2^k$ of the dimension of the original BTF image. The kth-level BTF Laplacian coefficients, $L_k^{\omega_i,\omega_o}$ is the bandpass filtered image obtained by subtracting the lowpass filtered $F_{k+1}^{\omega_i,\omega_o}$ from $F_k^{\omega_i,\omega_o}$ as shown in Equation (3), where $\uparrow_p$ donates the upsampling operation. We perform the above procedure λ_{max} times. The Laplacian coefficients $L_0^{\omega_i,\omega_o}(p), \cdots, L_{\lambda_{max}-1}^{\omega_i,\omega_o}(p)$ and the lowpass filtered image $F_{\lambda_{max}}^{\omega_i,\omega_o}(p)$ are kept to build a $(\lambda_{max}+1)$-level Laplacian pyramid. In our implementation, we chose a 3×3 box filter for downsampling and low-pass filtering, and used the nearest neighbor as the upsampling operation.

3.2 Compression of Laplacian BRDFs with PCA

We stack all the Laplacian coefficients at level k obtained in the previous section, $L_k^{\omega_i,\omega_o}(p)$, together to form the Laplacian BTF at level k, L_k, where $L_k(p,\omega_i,\omega_o) = L_k^{\omega_i,\omega_o}(p)$. Each Laplacian BTF L_k can be regarded as a set of per-texel Laplacian BRDFs, $L_k^p(\omega_i,\omega_o)$. We treat each Laplacian BRDFs L_k^p as a vector and perform PCA on these vectors to find the n most dominating principal components at level k, e_{kj}, $j = 1..n$. We call e_{kj} the j-th Laplacian BRDF principal component (or eigen Laplacian BRDF) at level k. We can then approximate each $L_k^p(\omega_i,\omega_o)$ using a linear combination of e_{kj}:

$$L_k^p(\omega_i,\omega_o) \approx \left(\sum_{j=1}^{n} d_{kj}(p) e_{kj}(\omega_i,\omega_o) \right) + \mu_k(\omega_i,\omega_o),$$

where $d_{kj}(p)$ stores the PCA weights for a pixel p and μ_k is the mean of Laplacian BRDFs. Due to the property of Laplacian transform, μ_k is actually equal to a zero vector, so we can approximate the whole Laplacian BTF at level k as

$$L_k(p,\omega_i,\omega_o) \approx \sum_{j=1}^{n} d_{kj}(p) e_{kj}(\omega_i,\omega_o). \tag{4}$$

Hence, for each Laplacian BTF, L_k, we only need to store the eigen Laplacian BRDFs e_{kj} and the corresponding weights d_{kj} for reconstruction.

We stop the construction of the pyramid when the number of Laplacian BRDFs is close to n and PCA is not necessary any more. For the top of the Laplacian pyramid, $F_{\lambda_{max}}$, we also approximate it by PCA. However, $F_{\lambda_{max}}$ is a lowpass filtered image instead of Laplacian coefficients, so its mean vector is not zero and has to be stored. As suggested by Ma *et al.* [2004], we fit the mean vector $\mu_{\lambda_{max}}(\omega_i,\omega_o)$ of $F_{\lambda_{max}}$ by a parametric shading function. It is because, after several passes of lowpass filtering, only the low frequency appearance of a BTF, such as the diffuse components, stays in $\mu_{\lambda_{max}}(\omega_i,\omega_o)$. Hence, it is usually sufficient to fit $\mu_{\lambda_{max}}(\omega_i,\omega_o)$ with a parametric shading model and only to store the fitted shading parameters. Ideally, a complex BRDF model could be used. In our implementation, we used the Phong shading model because it is a built-in shading model in GPU and we found it sufficient for the materials we have experimented. Hence, we approximate $F_{\lambda_{max}}$ by $\mu'_{\lambda_{max}}$:

$$\mu'_{\lambda_{max}}(\omega_i,\omega_o) = k_a + k_d(N \cdot L) + k_s(N \cdot H)^{\alpha}, \tag{5}$$

where k_a is the ambient contribution, k_d is the diffuse coefficient, k_s and α are the specular coefficients, $N = (0,0,1)$ is the normal, L depends on ω_i and H is a function of ω_i and ω_o. To find the shading parameters k_a, k_d, k_s and α, we used the Levenberg-Marguardt algorithm for the resulted non-linear optimization problem.

Another benefit of using a shading model instead of storing the original mean vector $\mu_{\lambda_{max}}(\omega_i,\omega_o)$ is that we can avoid the shading discontinuities without using expensive interpolation operations. Because BTF is only sampled from a set of viewing and lighting directions, to evaluate $\mu_{\lambda_{max}}(\omega_i,\omega_o)$ for an arbitrary set of ω_i and ω_o, one solution is to interpolate the nearby samples [Liu et al. 2004; Sattler et al. 2003] by assuming that $\mu_{\lambda_{max}}$ is a piecewise-linear function. However, the interpolation operation an expensive operation on GPU. Also, since $\mu_{\lambda_{max}}$ is an average of per-texel BRDFs, it should be better approximated by shading models. Figure 4 shows the comparisons between using the nearest neighbor and the shading model.

Note that the above approximation is only valid for the Y channel. For the Cr and Cb channels, we found that their mean vectors for $F_{\lambda_{max}}$ are nearly constant with respect to the change of lighting and viewing. Hence, for mean vectors in Cb (and Cr) channels, we only approximate it as a constant function whose value is the average of all the Cb (or Cr) values, $\overline{c}$, that is,

$$\mu'_{\lambda_{max}}(\omega_i,\omega_o) = \overline{c}. \tag{6}$$

3.3 Data packing

We assume that the original BTF is consisted of $p \times q$ 24-bit RGB images of size $r \times r$, where p and q are the number of lighting and viewing sampling directions. To efficiently render the LOD representation using GPU, we have to compactly pack the LOD representation into several texture maps. For that purpose, we use eight eigen Laplacian BRDFs for Y channel and two for Cb and Cr channels at every level. The LOD representation can then be represented as two texture maps for efficient BTF rendering on GPUs:

1. *Eigen Laplacian BRDF map*. Each eigen Laplacian BRDF, as shown in the top of Figure 5, is tabulated and indexed by a light-view index $T_{pc} = (T_{\omega_i}, T_{\omega_o})$, where T_{ω_i} and T_{ω_o} are the sequence numbers corresponding respectively to the lighting and viewing directions that BTF images were sampled. This

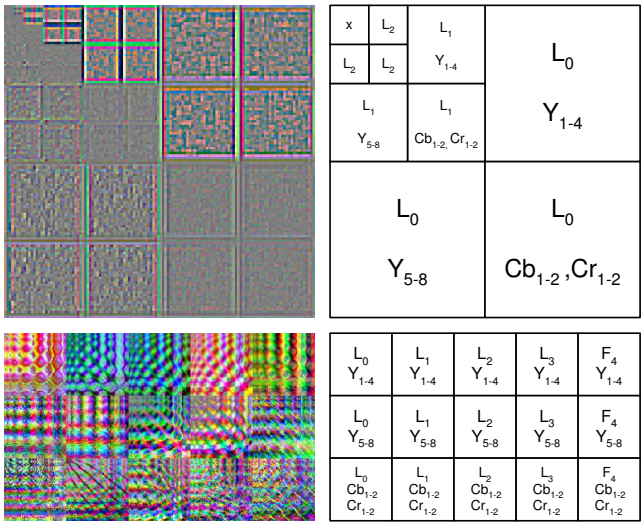

Figure 5: On the left, eigen Laplacian BRDF map (top) and PCA weight map (bottom) are shown. The map data is scaled and shifted, and only RGB channels are displayed. On the right is the corresponding arrangements of the maps. Eight eigenvectors are stored for Y channel (Y_{1-8}) and two for Cb and Cr (Cb_{1-2}, Cr_{1-2}).

map is stored as a $3p \times (\lambda_{max}+1)q$ 128-bit RGBA floating point texture, and the total size is $48pq(\lambda_{max}+1)$ bytes.

2. *PCA weight map*. We pack the PCA weight maps at different levels into a single square texture, as shown in the bottom of Figure 5. This map is stored as a $2r \times 2r$ 128-bit RGBA floating point texture, and the total size is $64r^2$ bytes.

In our experiments, $q = p = 81$ and $r = 64$. Hence, the size of the original BTF data is 80,621,568 bytes. We choose $\lambda_{max} = 4$. Therefore, the size of eigen Laplacian BRDF map and PCA weight map is $1,574,640$ and $262,144$ bytes respectively.

3.4 Rendering

The whole rendering procedure is executed by GPU. For each pixel, we use a pixel shader program to perform the following steps to calculate its pixel color.

1. *Obtain the surface texture coordinate and the light-view index*. For the pixel to be rendered, we first obtain its surface texture coordinate p as the index to access PCA weight map. Let ω_i and ω_o be the lighting and viewing directions with respect to the pixel to be rendered. First, we have to obtain the corresponding sequence numbers to index the eigen Laplacian BRDF map. For that purpose, we use the nearest neighbor approach. That is, we find the closest lighting(viewing) direction to $\omega_i(\omega_o)$ at which BTF was sampled, and use its corresponding sequence number to index the eigen Laplacian BRDF map. To speed up the nearest neighbor search, in the preprocessing stage, we prepare a lookup table to store the sequence number of the closest direction for many directions. We can then determine the light-view index T_{pc} for ω_i and ω_o by simply looking up this map.

2. *Determine the LOD parameter*. We use the range-based LOD selection method [Akenine-Möller and Haines 2002] to obtain a continuous LOD parameter λ between 0 and λ_{max}. More specifically, $\lambda = \min(\lambda_{max}, \max(\log_2(d), 0))$, where d is the distance from the viewpoint to the surface point corresponding to the pixel to be rendered.

3. *Obtain the eigen Laplacian BRDFs and PCA weights*. For each level k, $\lfloor\lambda\rfloor \le k \le \lambda_{max}$, we compose two n-dimensional vectors: (a) PCA weight vector D_k, in which $D_k[j] = d_{kj}(p), 1 \le j \le n$, and (b) eigen Laplacian BRDF vector E_k, in which $E_k[j] = e_{kj}(T_{pc}), 1 \le j \le n$. These vectors are composed by directly looking up the PCA weight map and the eigen Laplacian BRDF map using p and T_{pc}.

4. *Calculate the pixel color*. The pixel color is calculated as

$$\left(\sum_{k=\lfloor\lambda\rfloor}^{\lambda_{max}} w_k D_k^T E_k\right) + \mu'_{\lambda_{max}}(\omega_i, \omega_o), \tag{7}$$

where $w_k = \min(\max(1-\lambda+k, 0), 1)$ is the weight for the contribution of level k of λ and the vector $\mu'_{\lambda_{max}}(\omega_i, \omega_o)$ is the $F_{\lambda_{max}}$ mean vector approximation described in Equations (5) and (6). Finally, we convert the YCbCr pixel color back to the RGB color space.

The maximal level, λ_{max} affects the number of texture access in the rendering stage. The reconstruction of a single level requires six texture accesses.

4 Results

We used a desktop PC with an Intel Pentium 4 3.0GHz CPU, 1GB memory and an NVIDIA GeForce 6800 GPU with 128MB video memory to demonstrate our system. Vertex and fragment shaders were implemented with the NVIDIA Cg shading language. The window size of the rendering system is 1024×768. Under this preset configuration, the filling rate of 30+ frames per second could be reached. However, because the algorithm is fill-limited, the runtime performance varied with the number of pixels on the projected image.

4.1 LOD

Figure 7 compares the images generated with and without LOD BTF rendering. Without LOD rendering, distinct Moiré patterns reveal when the object is rendered at a distance. The proposed method provides continuous LODs using linear interpolations between levels. Our LOD BTF representation successfully captures the progressive transition between BTF and BRDF. The performance of our rendering system varies with different selected LODs: less detailed BTF rendering results in a faster frame rate because fewer texture accesses are needed. (Notice that GeForce 6 Series support dynamic branching, which allows true loops and conditionals in shader programs. We used conditionals in the fragment shader to determine which level we need to access.)

4.2 Compression

Using the data packing schema described in Section 3.3, the total size of the compressed data is $1,836,784$ bytes and the compression rate is 43.9x. Table 1 lists the mean square errors (MSE) of the reconstructions for different materials. The measurement of error is the same as suggested by Meseth *et al.* [2004]. Note that most of the shading variations and reconstruction errors lie in the Y channel. Figure 6 shows the reconstruction errors in the Y channel with

Material		Average error	Minimum error	Maximum error
Wool	Y	0.0285	0.0161	0.0379
	Cb	0.0075	0.0037	0.0128
	Cr	0.0089	0.0048	0.0146
Impalla	Y	0.0359	0.0182	0.0925
	Cb	0.0046	0.0033	0.0083
	Cr	0.0040	0.0027	0.0064
Wallpaper	Y	0.0137	0.0069	0.0373
	Cb	0.0034	0.0025	0.0080
	Cr	0.0028	0.0019	0.0079
Proposte	Y	0.0302	0.0186	0.0404
	Cb	0.0104	0.0063	0.0200
	Cr	0.0081	0.0049	0.0133
Corduroy	Y	0.0254	0.0142	0.0354
	Cb	0.0068	0.0041	0.0085
	Cr	0.0043	0.0028	0.0059

Table 1: BTF reconstruction errors of different materials in the Y, Cb and Cr channels using the data packing schema described in Section 3.3. This table justifies why we allocate more space for the Y channel than Cb and Cr channels. Notice that eight principal components were used for the Y channel in this table.

respect to the number of principal components used. This figure justifies our choice of using eight eigen Laplacian BRDFs for the Y channel.

5 Conclusions and Future Work

We have presented a method which introduces Laplacian pyramid into the BTF rendering framework to achieve continuous LOD rendering. We take an arbitrary BTF as the input, build a multiresolution representation and use it for shading reconstruction at the rendering stage. The advantages of the proposed LOD BTF representation include:

1. Anti-aliasing rendering is achieved by our LOD BTF representation. Subbands of the BTF progressively contribute to the rendering result according to the viewing distance.

2. At the rendering stage, the real-time performance depends on which LOD is chosen. If the BTF surface is far away from the viewpoint, fewer BTF levels are required to reconstruct the appearance. Therefore, the number of texture accesses is reduced. and the frame rate could be increased.

3. The compression rate of our subband compression method for BTFs is comparable to the best available BTF compression methods.

BTF is an effective representation for complex materials and there are number of ways to make it more practical to be used in game or film industry. We would like to proceed a better analysis on BTF subband data. We also like to study and implement the wavelet-based BTF representation, and try to integrate it into the wavelet lighting framework [Ng et al. 2003; Ng et al. 2004]. In addition, how to synthesize the BTF on arbitrary surface with our representation is also an challenge. We also plan to study on other topics such as large-scale BTF acquisition and synthesis (such as grass field, sandy beach, forest and so on) and the importance sampling for faster BTF acquisition.

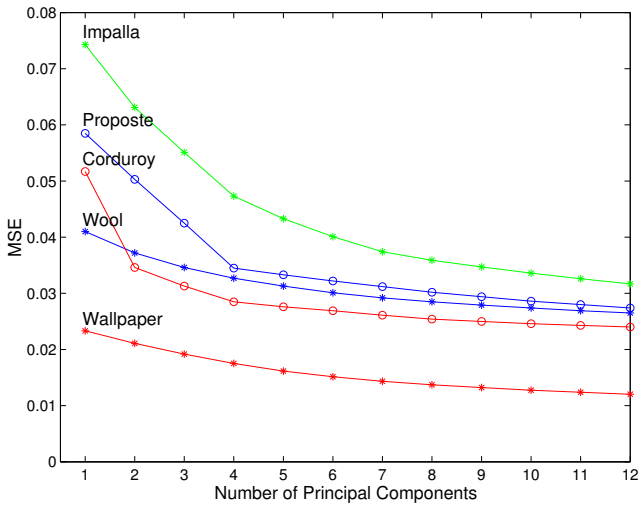

Figure 6: Plot of the reconstruction errors in the Y channel with respect to the number of principal components. This plot indicates that it is sufficient to use eight principal components for most materials we have tested.

Acknowledgement

Many thanks to the following people for their valuable suggestions: Tomoyuki Nishita, Ja-Ling Wu, I-Chen Lin, James Davis and Erika S. Chuang. We would also like to thank the reviewers for their pertinent comments. This work was partially supported by the CIET-NTU(MOE) and National Science Council of Taiwan under NSC93-2622-E-002-033, NSC93-2752-E-002-007-PAE, NSC93-2213-E-002-083, NSC93-2213-E-002-084 and NSC94-2213-E-002-051.

References

ADABALA, N., MAGNENAT-THALMANN, N., AND FEI, G. 2003. Real-time rendering of woven clothes. In *Proceedings of ACM Virtual Reality Software and Technology 2003*, 41–47.

AKENINE-MÖLLER, T., AND HAINES, E. 2002. *Real-Time Rendering, 2nd Ed.* A. K. Peters.

BUNGIE, 2004. HALO 2. http://www.bungie.net/Games/Halo2/.

BURT, P. J., AND ADELSON, E. H. 1983. The Laplacian pyramid as a compact image code. *IEEE Transaction on Communications COM-31*, 4, 532–540.

DANA, K. J., VAN GINNEKEN, B., NAYAR, S. K., AND KOENDERINK, J. J. 1999. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics 18*, 1, 1–34.

FOURNIER, A. 1992. Filtering normal maps and creating multiple surfaces. *TR-92-41, Department of Computer Science, University of British Columbia*.

GOLDMAN, D. B. 1997. Fake fur rendering. In *Proceedings of ACM SIGGRAPH 1997*, 127–134.

KAUTZ, J., HEIDRICH, W., AND SEIDEL, H.-P. 2001. Real-time bump map synthesis. In *Proceedings of Graphics Hardware 2001*, 109–114.

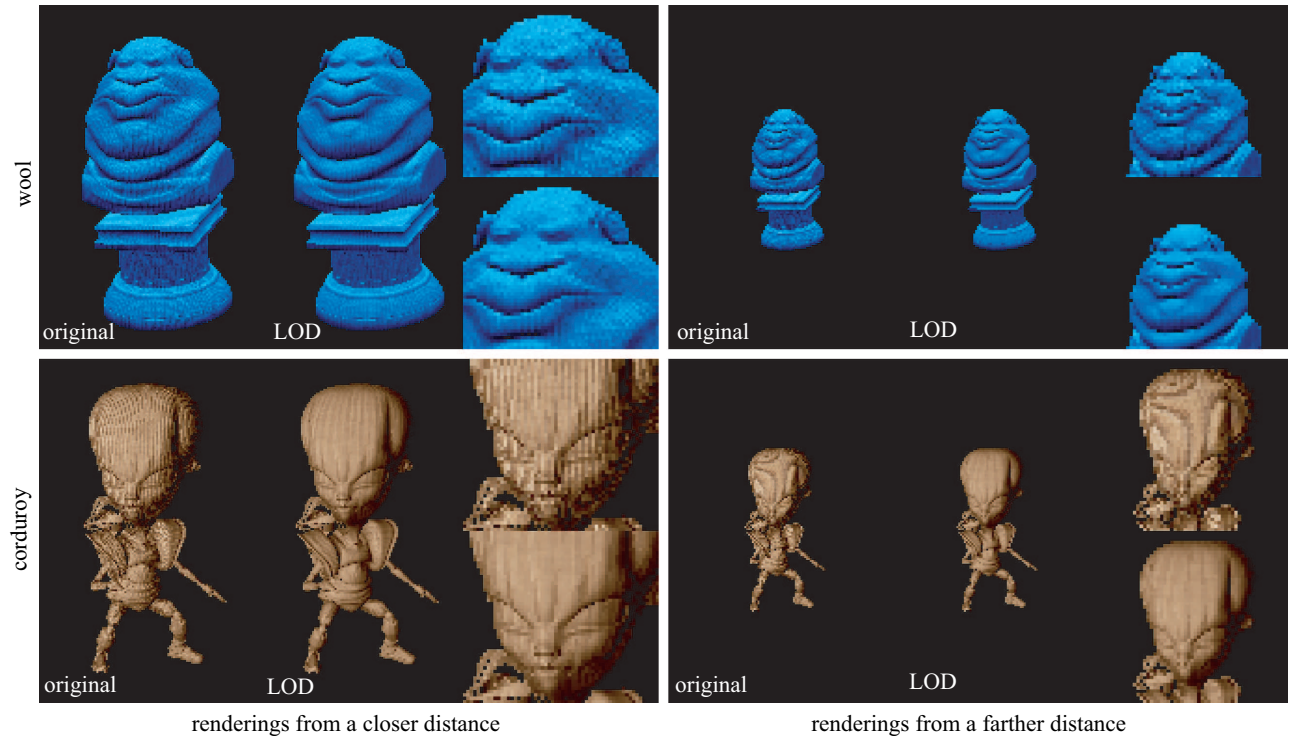

Figure 7: Comparisons between the original BTF and LOD BTF renderings for the materials wool (top) and corduroy (bottom) at closer (left) and farther (right) distances. For each set of images, original BTF rendering is shown on the left, LOD BTF rendering is shown in the middle, and a closeup view for parts of these two renderings is shown on the right (the top one is the closeup for the original BTF rendering and the bottom one is for the LOD BTF rendering.).

LEUNG, T. K., AND MALIK, J. 2001. Representing and recognizing the visual appearance of materials using 3D textons. *International Journal of Computer Vision 43*, 1, 29–44.

LIU, X., HU, Y., ZHANG, J., TONG, X., GUO, B., AND SHUM, H.-Y. 2004. Synthesis and rendering of bidirectional texture functions on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics 10*, 3, 278–289.

MA, W.-C., CHAO, S.-H., CHEN, B.-Y., CHANG, C.-F., OUHYOUNG, M., AND NISHITA, T. 2004. An efficient representation of complex materials for real-time rendering. In *Proceedings of ACM Virtual Reality Software and Technology 2004*, 150–153.

MCALLISTER, D. K. 2002. *A Generalized Surface Appearance Representation For Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill.

MESETH, J., MÜLLER, G., AND KLEIN, R. 2004. Reflectance field based real-time, high-quality rendering of bidirectional texture functions. *Computers and Graphics 28*, 1, 103–112.

MEYER, A., AND NEYRET, F. 2000. Multiscale shaders for the efficient realistic rendering of pine-trees. In *Proceedings of Graphics Interface 2000*, 137–144.

MÜLLER, G., MESETH, J., AND KLEIN, R. 2004. Fast environmental lighting for local-PCA encoded BTFs. In *Proceedings of Computer Graphics International 2004*, 198–205.

MÜLLER, G., MESETH, J., SATTLER, M., SARLETTE, R., AND KLEIN, R. 2004. Acquisition, synthesis and rendering of bidirectional texture functions. In *Eurographics 2004 State of The Art Report*, 69–94.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2003. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Transactions on Graphics 22*, 3, 376–381. (Proceedings of SIGGRAPH 2003).

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics 23*, 3, 477–487. (Proceedings of SIGGRAPH 2004).

OLANO, M., KUEHNE, B., AND SIMMONS, M. 2003. Automatic shader level of detail. In *Proceedings of Graphics Hardware 2003*, 7–14.

SATTLER, M., SARLETTE, R., AND KLEIN, R. 2003. Efficient and realistic visualization of cloth. In *Proceedings of Eurographics Symposium on Rendering 2003*, 167–177.

SUYKENS, F., VOM BERGE, K., LAGAE, A., AND DUTRÉ, P. 2003. Interactive rendering with bidirectional texture functions. *Computer Graphics Forum 22*, 3, 463–472. (Proceedings of Eurographics 2003).

TOKSVIG, M. 2004. *NVIDIA Technical Brief*.

VASILESCU, M. A. O., AND TERZOPOULOS, D. 2004. Tensortextures: Multilinear image-based rendering. *ACM Transactions on Graphics 23*, 3, 336–342. (Proceedings of SIGGRAPH 2004).

WILLIAMS, L. 1983. Pyramidal parametrics. In *Proceedings of ACM SIGGRAPH 1983*, 1–11.

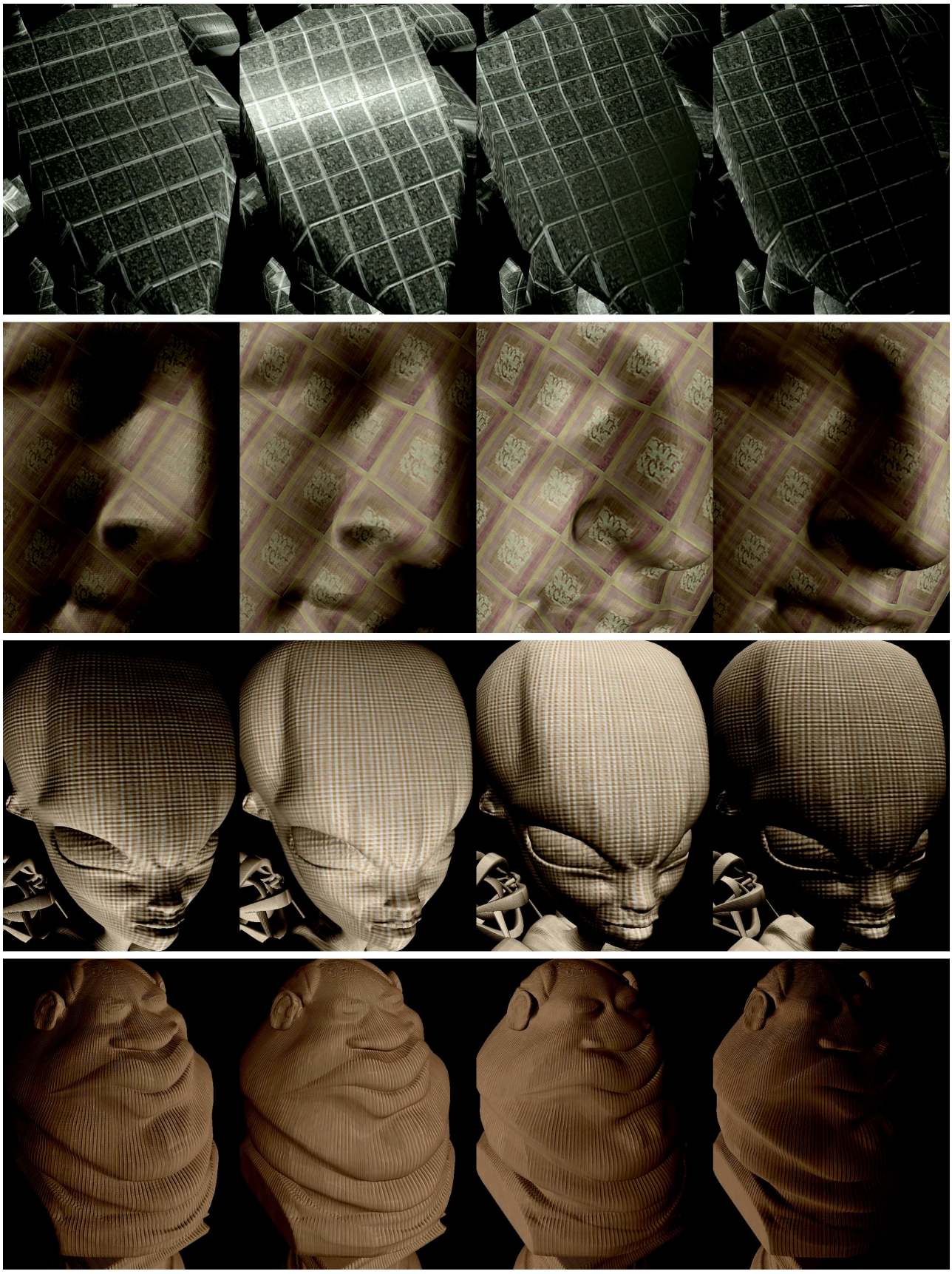

Figure 8: Rendering result of different materials under various lighting conditions. From top to bottom: impalla, wallpaper, proposte and corduroy. *(Models courtesy of NVIDIA Corp.)*