Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Appearance based object modeling using texture database: Acquisition, compression and rendering

R. Furukawa [1] , H. Kawasaki [2], K. Ikeuchi [2] and M. Sakauchi [2]

[1] Faculty of Information Sciences, Hiroshima City University, Hiroshima, Japan
[2] Institute of Industrial Science, The University of Tokyo, Tokyo, Japan

Abstract

Image-based object modeling can be used to compose photorealistic images of modeled objects for various rendering conditions, such as viewpoint, light directions, etc. However, it is challenging to acquire the large number of object images required for all combinations of capturing parameters and to then handle the resulting huge data sets for the model. This paper presents a novel modeling method for acquiring and preserving appearances of objects. Using a specialized capturing platform, we first acquire objects' geometrical information and their complete 4D indexed texture sets, or bi-directional texture functions (BTF) in a highly automated manner. Then we compress the acquired texture database using tensor product expansion. The compressed texture database facilitates rendering objects with arbitrary viewpoints, illumination, and deformation.

1. Introduction

Making renderable models from actual objects is a challenging problem. Modeling manually using a CG modeler is time consuming and the cost is high. The models can be made more easily by acquiring the shapes of the objects using a range finder and making simple textures from several photographs. However, observing the objects from an arbitrary viewpoint or changing the directions of the light sources requires models that are more detailed.

One possible method is estimating detailed reflection models for all the surfaces using physics-based computer vision techniques [9]. The advantage of this method is that the object can be rendered very realistically if the object's surface model is correct. The drawback is that sometimes the surface property is so complicated that it is intractable to describe with simple mathematical forms (for example, complicated surface textures such as fur). Even if it is possible to represent reflection formulaically, estimation of parameters for all the surfaces is a difficult problem, especially if the formula has many parameters.

One alternative method is to model the surface properties of the object from the sampled photographic data, as it is. Objects can be rendered realistically using a combination of geometrical data and textures sampled from photographs [12, 3, 18]. For example, the surface light field rendering technique captures light rays emitted from the object's surface instead of from the camera plane. Originally, these light rays could be defined by a 4D parameter called the bi-directional reflectance distribution function (BRDF). Since BRDF is a point-wise function, for efficient handling of mesh models, bi-directional texture function (BTF), defined as a six dimensional function with 2D texture associated with 4D light and view direction[5], is often captured for each polygon patch. Using this BTF data, we could render an object by IBR more freely as in conventional model-based 3D CG applications; however, data for 6D parameterized textures is huge and difficult to acquire and handle. Recent approaches [12, 3, 18]. use 2D or 3D subsets of the whole BTF or BRDF.

In this paper, we present a BTF-based object modeling technique for rendering with arbitrary viewpoints, illumination and deformation. To realize this goal, we acquire and manage a full BTF dataset with 4D lighting/viewing parameterization. To enable commonly used PCs to handle the enormous amount of data, we have developed a new method to compress the BTF data. Our method simultaneously utilizes several independent data correlations, which are inherent in high-dimensionally parameterized BTF data. By exploiting more than one correlation, our compression algorithm achieves more efficient compression rates than does

singular value decomposition (SVD), which is often used for texture compression. Objects with unaltered geometry can be rendered as an approximation of the original appearance. With deformation, rendering results are correct with respect to shading effects and specular locations, although effects of interreflection and shadows become erroneous as the geometry is altered.

2. Background and Related Works

In IBR research, dealing with changes in rendering conditions (the viewpoint/light direction) has been a difficult problem because the original idea of IBR was to "replay" the light information of the scene as is [7] [8]. One possible solution for realizing arbitrary view/light direction is to use geometric information. In an actual implementation, the surface light field, a term coined by Miller et al. [10], is a function that assigns an RGB value to every ray emanating from every point on a surface. Using this function and geometric data makes it possible to compose scenes from an arbitrary viewpoint. Since the appearance data of surface light fields can be regarded as an approximation of BTF with added effects of interreflection and shadows, it can be expressed by the same parameterization and value type as those of BTF. Since the dimension of BTF is 6D, obtaining BTFs of a 3D object is a challenging problem[5].

The most closely related work to ours is that of Nishino et al.[12], Wood et al.[18], Chen et al.[3] and Furukawa et al. [6]. Nishino et al. realize image synthesis with an arbitrary light direction, and a view direction with one degree of freedom (rotation about a fixed axis). Wood et al. achieve image composition from an arbitrary viewpoint and a fixed lighting condition. Chen et al. propose a method for exploiting graphics hardware acceleration, realizing real-time rendering, and composing images from arbitrary viewpoints and zoom-ups with a fixed lighting direction. Furukawa et al. capture BTF database by using a specialized capturing platform and render objects using the database.

With regard to data acquisition and compression, Nishino et al. acquire a BTF subset with 3D parameterized light and view (1D for view and 2D for light direction) by using a turntable and a light dome. They compress the data using an SVD-based compression algorithm called the eigentexture method. Wood et al. acquire a subset of BRDF with a 2D parameterized view and fixed lighting direction, generated from photographs taken from every direction with a gantry. They treat the 2D light field on each surface point as a unit (lumisphere) and propose compression methods similar to PCA and vector quantization. In the work of Chen et al., a 2D subset of BRDF is taken by capturing object images from various directions. Their data compression is done with an SVD-based method. The capturing platform Furukawa et al. proposed can acquire a BTF subset with 3D view/light parameterization. (See Table 1)

In all of the above research methods that acquire the BTF of jagged objects, only subsets with 2D or 3D view/light parameters are constructed, and the freedom of the rendering conditions (deformation/view/lighting) is inherently restricted. On the other hand, our research, which fully captures the BTF with 4D view/light parameters, has no limitations on either the view or the light directions. We can synthesize the object's image under arbitrary view and lighting conditions. In addition, allowing distorted effects of interreflection and shadows, we can render 3D objects deformed from their original shapes.

Table 1: *Dimension of light field*

	θ_{light}	ϕ_{light}	θ_{view}	ϕ_{view}
Wood et al.			○	○
Debevec etal.	○	○		
Nishino et al.	○	○	○	
Chen et al.			○	○
Furukawa et al.		○	○	○
Our method	○	○	○	○

3. Data Acquisition

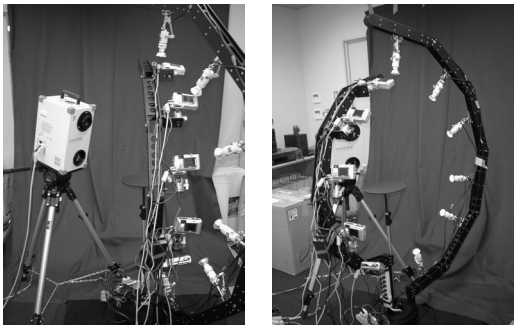
3.1. BTF capturing system

In order to model objects with textures for arbitrary view/lighting directions, we have to provide a texture database with multi-dimensional indices. To make this collection of textures, we designed and constructed an original data acquisition platform. As shown in Fig.1(a), this system consists of two large concentric circles. One circle is equipped with multiple CCD cameras and the other is equipped with multiple halogen lamps. A PC controls these CCD cameras and halogen lamps. At the center of this circle, there is a platform on which we place an object to be captured; this platform is rotated around the vertical axis by a stepping motor. In addition, the large circle equipped with multiple lamps is rotated horizontally by another motor(Fig.1(b)). These stepping motors are individually controlled by the PC. Therefore, this system can capture complete BTF information by changing the parameters of turn table angle (1D "α"), selection of camera (1D "β"), rotation angle of lamp circle (1D"γ"), and selection of lamps (1D "δ").

Using the capturing platform, we capture the images of the object, changing the parameters by

$$\alpha = \Delta_\alpha\, i_\alpha,\ \beta = \Delta_\beta\, i_\beta,\ \gamma = \Delta_\gamma\, i_\gamma,\ \delta = \Delta_\delta\, i_\delta,$$

where notations Δ_X are the capturing intervals and i_X is the index for each parameter X. Normally, we set $\Delta_\alpha = \Delta_\gamma$. The captured images are indexed by tuple $(i_\alpha, i_\beta, i_\gamma, i_\delta)$.



(a) (b)

Figure 1: *Image capturing platform (a)equipped with range sensor (b) concentric arc system*

From the images, we construct a 3D shape with a visual hull technique using voxel carving [13], [16]. First, we extract the object silhouette by background subtraction (this process is quite robust and accurate because we fix the camera and light source position for the capturing process) and then project this silhouette onto the voxel space to construct the 3D shape. In this paper, we express acquired polygons by $P(i_p)(0 \leq i_p < N_p)$ where i_p is the polygon index, N_p is the number of polygons, $P()$ is the polygon specified by polygon index.

Shape acquisition based on image silhouettes theoretically cannot reconstruct concavities of objects. For such objects, we capture their correct 3D geometrical data using laser range scanners that allow concavities, and align the captured geometries to those retrieved by voxel carving. The alignment procedure is based on Wheeler's iterative method[17].

Textures are acquired by mapping images into fixed triangles for each polygon. The acquired textures are specified by polygon index i_p and the indices for capturing parameters $(i_\alpha, i_\beta, i_\gamma, i_\delta)$. We reparameterize these textures so that the indices are separated into two pairs, the first representing view directions and the second representing light directions. Both view and light directions are represented by certain local coordinates which are fixed onto the object. The process is done in the following form:

$$i_{v\theta} \equiv i_\alpha,\ i_{v\phi} \equiv i_\beta,\ i_{l\theta} \equiv i_\gamma - i_\alpha,\ i_{l\phi} \equiv i_\delta.$$

$$\Delta_{v\theta} \equiv \Delta_\alpha,\ \Delta_{v\phi} \equiv \Delta_\beta,\ \Delta_{l\theta} \equiv \Delta_\alpha,\ \Delta_{l\phi} \equiv \Delta_\delta.$$

Here, $i_{v\theta}$ and $i_{v\phi}$ represents view directions, and $i_{l\theta}$ and $i_{l\phi}$ represents light directions. $\Delta_{v\theta}$, $\Delta_{v\phi}$, $\Delta_{l\theta}$ and $\Delta_{l\phi}$ denotes sampling intervals for all the indices. From the re-indexed texture set, each texture is specified by tuple $(i_p, i_{v\theta}, i_{v\phi}, i_{l\theta}, i_{l\phi})$. To express a texture specified by the index, we use a notation of $T(i_p, i_{v\theta}, i_{v\phi}, i_{l\theta}, i_{l\phi})$ in this paper. We denote domains of these indices by

$$0 \leq i_{v\theta} < N_{v\theta}, 0 \leq i_{v\phi} < N_{v\phi}, 0 \leq i_{l\theta} < N_{l\theta}, 0 \leq i_{l\phi} < N_{l\phi}$$

Although there are more sophisticated parameterization[14], we use the simple parameter space described above in order to avoid resampling of textures. Since we sampled the images by relatively coarse intervals in parameter space (30°) for the experiments described later, degradation of quality of textures due to resampling was not neglectable.

An example of a re-indexed set of textures is shown in Fig. 4. Fig. 4 (a) is the modeled object (a toy post) and (b) visualizes a subset of the texture data which originates from a single polygon, which can be considered to be a set of textures $\{T(i_p, *, *, *, *)\}$, where $*$ means "don't care". Since the shape of the object is roughly convex, we used only voxel carving to generate a 3D model. In Fig. 4(b), each column (vertical sequence) of textures is a subset of a certain view direction, which is $\{T(i_p, i_{v\theta}, i_{v\phi}, *, *)\}$, and each row (horizontal sequence) of textures is a subset of a certain light direction, which is $\{T(i_p, *, *, i_{l\theta}, i_{l\phi})\}$. There is some change in the appearance of the textures in Fig. 4(b) due to erroneous polygon localization in images (some textures have blue colored areas while others don't). If the errors of polygon location have large discontinuities in parameter space $(i_{v\theta}, i_{v\phi}, i_{l\theta}, i_{l\phi}))$, they sometimes cause "double image" artifacts. If the errors are continuous along the parameter space, they have a relatively small impact on rendering results.

The set of textures acquired from raw images is not the BTF in the rigorous meaning since it includes effects of interreflection and shadows. However, we use the acquired texture set as a BTF dataset for rendering bacause objects with unaltered geometry can be rendered as an approximation of the original appearance. With deformation, rendering results are correct with respect to shading effects and specular locations, although effects of interreflection and shadows become erroneous as the geometry is altered from the original shape.

4. Compression of Parameterized Texture using Tensor Products expansion

Now, we have constructed an indexed set of textures, or BTF. Before describing our compression method for the huge amount of data, let us briefly overview the compression methods of existing surface light field research.

One well known method to compress texture images uses PCA/SVD-based compression [12][3]. In that research, texture data is rearranged into 2D indexed elements, or a matrix. The matrix is approximated as a sum of the outer products of vectors using SVD. The approximation is more efficient if the row vectors (or column vectors) of the matrix have strong correlations. In the eigentexture method [12], texture images for each polygon are rasterized into vectors specified by a 1D view index. Matrices are formed by aligning these vectors according to view indices. In the work of Chen et al. [3], compression is done in a way similar to the eigentexture

method, except textures are first re-sampled for uniformly distributed 2D view directions and then compressed. The re-sampling of textures prevents uneven approximation, and 2D indexing of the view enables efficient hardware-accelerated rendering.

Let us examine our indexed texture set for a single polygon shown in Fig. 4, in which each column of textures corresponds to a certain view direction, and each row corresponds to a certain light direction. We can see that the textures in each row tend to have similar average intensities. This is because diffuse reflection, which amounts to most of the reflection, tends to depend only on light direction. For each column, texture patterns are most similar because they are captured from fixed view directions. Thus, the changes in intensity and texture pattern are strongly correlated between columns and rows of textures. If we compress the textures by PCA/SVD-based techniques such as the eigentexture method, and arrange the coefficients of eigentextures (principal components) by view and light indices, it is expected that there still remain strong correlations along these indices. To utilize these correlations for efficient compression, we pack the texture database into tensors and approximate the tensor using tensor product expansion (TPE).

Tensor and TPE are generalizations of matrices and SVD. As matrices are expanded into sums of products of vectors, tensors can be expanded into sums of tensor products[2]. Let A be a 3D tensor of size $L \times M \times N$. A can be expressed as

$$A = \sum_r \alpha_r u_r \otimes v_r \otimes w_r, \tag{1}$$

where r is an index of terms, α_r is a coefficient of term r, u_r, v_r and w_r are unit vectors, and the operator $\otimes$ means tensor product. Thus, the form above means

$$A_{i,j,k} = \sum_r \alpha_r u_{r,i} v_{r,j} w_{r,k}. \tag{2}$$

$$|u_r| = 1, |v_r| = 1, |w_r| = 1,$$

$$\alpha_r \geq \alpha_s \quad \text{if} \quad r < s$$

where $A_{i,j,k}$ is an element of tensor A with indices of i, j, k, $u_{r,i}$ is ith element of vector u_r. We can approximate tensor A by neglecting terms with small significance (i.e. terms with small α_r). Truncating the form into a sum of K terms, we achieve a compression rate of $K(L+M+N)/LMN$.

There are several different ways to pack texture set information into tensors. One of them is to pack a tensor set from a polygon $\{T(i_p, *, *, *, *)\}$ (here, symbols "*" mean "don't care") into a 3D tensor, using the first tensor index for indicating texel, the second for view direction, the third for light direction. This is done by constructing tensors $A(i_p)$ of size $N_t \times (N_{v\theta} N_{v\phi}) \times (N_{l\theta} N_{l\phi})$ (N_t is the number of texels in a texture) for each polygon $P(i_p)$ by

$$A(i_p)_{i,\,(i_{v\theta}\,N_{v\phi}+i_{v\phi}),\,(i_{l\theta}\,N_{l\phi}+i_{l\phi})}$$

$$= Texel(i, T(i_p, i_{v\theta}, i_{v\phi}, i_{l\theta}, i_{l\phi})) \tag{3}$$

where $Texel(i, \cdot)$ denotes ith texel value of a texture. 2D arrangement of textures by view and light indices (i and j in the form above) is the same as the one shown in Fig 4(b).

One drawback of this packing is that textures which have strong specular reflection do not align into columns in the arrangement of Fig 4(b). Examining the figure, we can see some bright textures aligned in a diagonal direction. Those bright textures include strong specular components. There are also some blank textures aligned in the same way. These textures cannot be captured because the lights for the halogen lamps on the capturing platform are occluded by the circle equipped with cameras for the view/light condition. These diagonally aligned elements are difficult to approximate by the form (2), and we have found them to be harmful for TPE compression. Since these textures are almost uniform textures, we subtract DC components from all the textures and approximate only AC components by TPE. DC components are stored separately.

As opposed to SVD, for which there exists a robust algorithm to calculate optimal solution, an algorithm to obtain the optimal solution for TPE is still an open area of research. Murakami et. al proposed a fast calculation method for TPE, applying the power method which was originally used for calculating SVD [11]. Although their algorithm is not guaranteed to produce the optimal solution, we use this method because it is fast and its solution is sufficient for the purpose of compression. A brief description of their algorithm to calculate the expansion of a 3D tensor A is as follows:

- By iterating the following procedure, obtain α_s, u_s, v_s, w_s of equation (1) for $s = 1, 2, \cdots$.
  - Initialize u_s, v_s, w_s as arbitrary unit vectors.
  - Obtain the residual tensor R through the operation $R \leftarrow A - \sum_{r=1}^{s-1} \alpha_s u_s \otimes v_s \otimes w_s$.
  - If $(\|R\|/\|A\|)^2$ is less than ϵ, stop the calculation, where $\|\cdot\|$ means the 2-norm of a tensor (the root of the sum of squared elements in a tensor), and ϵ is the tolerable squared error rate.
  - Iteratively update u_r, v_r, w_r until these vectors converge by applying the following steps.
    - Obtain $\tilde{u}_r, \tilde{v}_r, \tilde{w}_r$ by the following contract operation:

$$\tilde{u}_{r,i} \leftarrow \sum_{j=1}^{M} \sum_{k=1}^{N} R_{i,j,k} v_{r,j}, w_{r,k}$$

$$\tilde{v}_{r,j} \leftarrow \sum_{k=1}^{N} \sum_{i=1}^{L} R_{i,j,k} w_{r,k}, u_{r,i}$$

$$\tilde{w}_{r,k} \leftarrow \sum_{i=1}^{L} \sum_{j=1}^{M} R_{i,j,k} u_{r,i}, v_{r,j}$$

- Update u_r, v_r, w_r as the normalized $\tilde{u}_r, \tilde{v}_r, \tilde{w}_r$: $u_r \leftarrow \frac{\tilde{u}_r}{|\tilde{u}_r|}$, $v_r \leftarrow \frac{\tilde{v}_r}{|\tilde{v}_r|}$, $w_r \leftarrow \frac{\tilde{w}_r}{|\tilde{w}_r|}$.

To test the efficiency of the compression method, we compressed textures using an SVD technique (eigentexture method) and TPE-based compression with two different dimensions. The sample object was a toy post shown in Fig. 4. Each texture has 136 pixels. There exist 2592 ($12 \times 3 \times 12 \times 6$) textures for each polygon. Intervals of azimuth and elevation angles are 30°. For SVD we packed the pixel values into matrices $B(i_p)$ with size of 136×2592, which can be expressed as

$$B(i_p)_{i,(i_{v\theta} N_{v\phi} N_{l\theta} N_{l\phi} + i_{v\phi} N_{l\theta} N_{l\phi} + i_{l\theta} N_{l\phi} + i_{l\phi})} = Texel(i, T(i_p, i_{v\theta}, i_{v\phi}, i_{l\theta}, i_{l\phi})).$$

We tried two different packing methods for TPE based compression. One method consists of packing textures into 3D-tensors A whose size is $136 \times 36 \times 72$, where the three tensor indices correspond to texel location, view direction ($i_{v\theta}$ and $i_{v\phi}$) and light direction ($i_{l\theta}$ and $i_{l\phi}$) respectively, using the form 3. The other method is packing textures into 4D-tensors C whose size is $136 \times 36 \times 12 \times 6$, where the four tensor indices correspond to texel location, view direction ($i_{v\theta}$ and $i_{v\phi}$), and light direction indices ($i_{l\theta}$ and $i_{l\phi}$). The packing is done by the form

$$A_2(i_p)_{i,(i_{v\theta} N_{v\phi} + i_{v\phi}), i_{l\theta}, i_{l\phi}} = Texel(i, T(i_p, i_{v\theta}, i_{v\phi}, i_{l\theta}, i_{l\phi}))$$

We approximated the matrices and tensors by SVD/TPE so that each matrix/tensor had a root of sum of squared error less than 15.0 (Range of pixel value is from 0 to 255). The threshold was chosen so that the average approximation rate became about 0.01. The experiment was done for 100 polygons. Table 2 shows the result, which describes data sizes needed to store each term, average numbers of terms needed to approximate textures for each polygon, and average data sizes for each polygon including stored AC components. It was assumed that the compressed data were expressed as a collection of 2 byte short numbers. Because the freedom of the approximation model decreases in order of SVD, 3D TPE, and 4D TPE, the number of terms needed for approximation increases in the same order. TPE-based compression uses less size to store 1 term of the expanded data, but it needs more terms for approximation. As a result, data size 3D TPE compression was about 2.4 times less than SVD. Although the data size for 1 term of 4D TPE was smaller than that of 3D TPE, the average data size of 4D TPE compression was larger than 3D TPE because of the increased number of terms. Figure 2 plots the data sizes of compressed textures for each polygon, compressed using SVD and 3D TPE method. The horizontal axis represents polygon index i_p, and the vertical represents the compressed size of the texture data.

Fig. 3 shows how TPE approximates textures. Fig. 3(b) is the AC components of the original texture (a). (b) is approximated by term 1 (c) and the residual of term 1 is (d). The texture is approximated by terms including term 3, 5 and 9 (shown in (e)-(g)). The resulting approximation is shown in (h).

Table 2: *Compression result*

	Data size (1 term)	Average number of term	Average data size
SVD	5456	8.56	46703
3D TPE	488	23.22	19107
4D TPE	380	34.99	21072

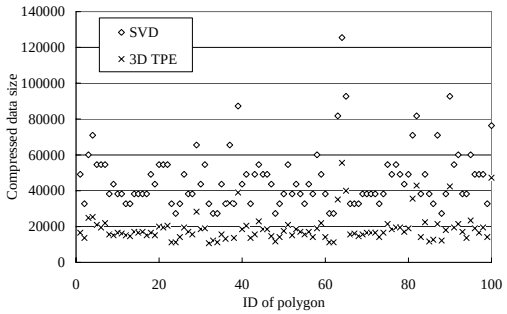


Figure 2: ***Data sizes of compressed textures (SVD and TPE)***

5. Rendering

5.1. Texture synthesis

To render modeled objects we have to generate textures for each polygon. Let us assume that we have vertices $V(i_v)(0 \leq i_v < N_v)$ which form polygons $P(i_p)(0 \leq i_p < N_p)$, where i_v represents the index of vertices, $V(i_v)$ is the vertex specified by index i_v and N_v is the number of vertices. Polygon $P(i_p)$ consists of three vertices $V(t(i_p, j))(0 \leq j \leq 2)$, where $t(i_p, j)$ is a table that enables us to look up vertex indices by polygon indices.

For the rendering process, the user can specify virtual camera position, light direction, object position (translation and rotation) and geometrical deformation. From the camera position and object position, rotation from the camera coordinate system to the object coordinate system, which we express as R_o, is calculated. Let us assume that view direction and light direction can be expressed by v_c and l_c in the camera coordinate system. Normally, v_c is a fixed vector (for example $[0, 0, -1]^t$). Thus, view and light directions expressed by the object coordinate system are $R_o \mathrm{v}_c$, $R_o \mathrm{l}_c$.

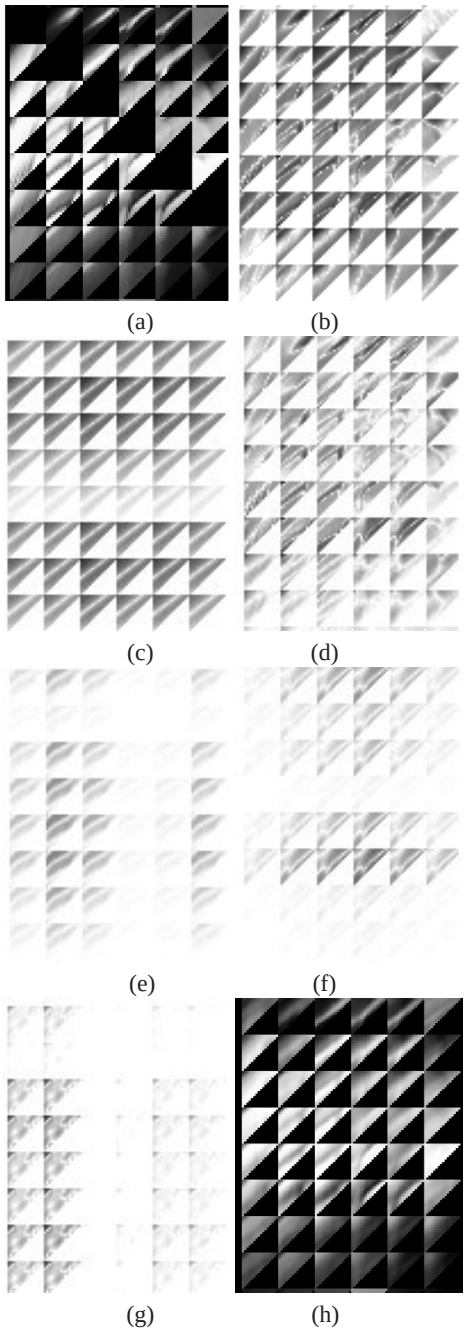


Figure 3: ***Approximation by TPE: (a) Original image, (b) AC components of textures, (c) Term 1 of TPE, (d) Residual image after subtraction of term 1, (e) Term 3, (f) Term 5, (g) Term 9, (h) Result image***

From the given deformation, we calculate the 3D rotation of surface point $V(i_v)$ relative to the object coordinate system, which can be expressed as $R_d(i_v)$. If there is no geometrical deformation, $R_d(i_v) = I$ (I is an identity rotation) for all vertices. Relative rotation of each vertex $V(i_v)$ from the camera coordinate system can be expressed as $R_d(i_v) \circ R_o$.

$R_d(i_v)$ may be directly calculated if a mathematical model of the deformation is given. If it is not available, we can calculate $R_d(i_v)$ from a geometrical transformation caused by deformation. To do so, we calculate the normal vector at $V(i_v)$ with and without deformation, which we describe as $\mathrm{n}'(i_v)$ and $\mathrm{n}(i_v)$, respectively. Further, we calculate normalized direction vectors of edges connected to vertex $V(i_v)$ with and without deformation, which we describe as $\mathrm{e}'(i_v, j)(j = 0, 1, 2, \cdots, E(i_v) - 1)$ $\mathrm{e}(i_v, j)(j = 0, 1, 2, \cdots, E(i_v) - 1)$, where $E(i_v)$ denotes the number of edges connected to vertex $V(i_v)$. Then we obtain the rotation $R_d(i_v)$ such that $\mathrm{n}'(i_v) \approx R_d(i_v)\mathrm{n}(i_v), \mathrm{e}'(i_v, j) \approx R_d(i_v)\mathrm{e}(i_v, j)(j = 0, 1, 2, \cdots, E(i_v) - 1)$. We used the method of Arun et. al [1] for the calculation.

Now, we get

$$\mathrm{v}_r(i_v) \equiv (R_d(i_v) \circ R_o)\mathrm{v}_c$$

$$\mathrm{l}_r(i_v) \equiv (R_d(i_v) \circ R_o)\mathrm{l}_c,$$

where $\mathrm{v}_r(i_v)$ and $\mathrm{l}_r(i_v)$ are view and light directions at vertex $V(i_v)$ expressed by object coordinate system. Describing azimuth and elevation angles of the direction vector by $azm(\cdot)$ and $elv(\cdot)$, four angles $azm(\mathrm{v}_r(i_v)), elv(\mathrm{v}_r(i_v)), azm(\mathrm{l}_r(i_v))$ and $elv(\mathrm{l}_r(i_v))$ have direct correspondence to indices of the texture database, $i_{v\theta}, i_{v\phi}, i_{l\theta}$ and $i_{l\phi}$. Since the pair of view and light directions $[\mathrm{v}_r(i_v)), \mathrm{l}_r(i_v)]$ represent conditions of the texture used for rendering, we call it "rendering condition." Also, we call the view direction of the pair "rendering view condition," and the light direction "rendering light condition."

Textures in the BTF database are sampled at discrete directions of view and light, so we interpolate the sample textures to generate textures needed for rendering. We call the view/light direction pairs in the database "sample conditions," and we use the terms "sample view conditions" and "sample light conditions" in a similar way as rendering conditions. If we plot sample view conditions or sample light conditions, regarding their azimuth and elevation angles as 2D orthogonal coordinates, the plots form lattice points aligned by fixed intervals for each axis.

We generate textures corresponding to calculated rendering conditions $[\mathrm{v}_r(i_v), \mathrm{l}_r(i_v)]]$ using the weighted sum of neighbor samples. Let $\tilde{\mathrm{v}}_0(i_v)$, $\tilde{\mathrm{v}}_1(i_v)$, $\tilde{\mathrm{v}}_2(i_v)$ be three neighbor sample view conditions of a rendering view condition $\mathrm{v}_r(i_v)$. In addition, let $W_0^v(i_v), W_1^v(i_v), W_2^v(i_v)$ be weights for the neighbor sample view conditions, fulfilling

the constraint $W_0^v(i_v)+W_1^v(i_v)+W_2^v(i_v)=1$. Selection of the neighbor sample view conditions is done by the following process: Let us define

$$
\begin{aligned}
i^-_{v\theta} &\equiv \left\lfloor \frac{azm(\mathrm{v}_r(i_v))}{\Delta_{v\theta}} \right\rfloor,\\
i^-_{v\phi} &\equiv \left\lfloor \frac{elv(\mathrm{v}_r(i_v))}{\Delta_{v\phi}} \right\rfloor,\\
r_{v\theta} &\equiv azm(\mathrm{v}_r(i_v)) - i^-_{v\theta}\Delta_{v\theta},\\
r_{v\phi} &\equiv elv(\mathrm{v}_r(i_v)) - i^-_{v\phi}\Delta_{v\phi}
\end{aligned}
$$

where $\lfloor\cdot\rfloor$ denotes floor function. Since $\Delta_{v\theta}$ and $\Delta_{v\phi}$ are intervals for azimuth and elevation angle of sample view conditions, $(azm(\mathrm{v}_r(i_v)), elv(\mathrm{v}_r(i_v)))$ exists in the region surrounded by $(i^-_{v\theta}\Delta_{v\theta}, i^-_{v\phi}\Delta_{v\phi})$, $((i^-_{v\theta}+1)\Delta_{v\theta}, i^-_{v\phi}\Delta_{v\phi})$, $(i^-_{v\theta}\Delta_{v\theta}, (i^-_{v\theta}+1)\Delta_{v\phi})$ and $((i^-_{v\theta}+1)\Delta_{v\theta}, (i^-_{v\theta}+1)\Delta_{v\phi})$. Then sample view conditions and their weights are defined as

$$
\begin{bmatrix} azm(\tilde{\mathrm{v}}_0(i_v)) \\ elv(\tilde{\mathrm{v}}_0(i_v)) \end{bmatrix}
\equiv \begin{cases} \begin{bmatrix} i^-_{v\theta}\Delta_{v\theta} \\ i^-_{v\phi}\Delta_{v\phi} \end{bmatrix} & \text{if } (r_{v\theta}+r_{v\phi}) \le 1 \\ \begin{bmatrix} (i^-_{v\theta}+1)\Delta_{v\theta} \\ (i^-_{v\phi}+1)\Delta_{v\phi} \end{bmatrix} & \text{otherwise,} \end{cases}
$$

$$
\begin{bmatrix} azm(\tilde{\mathrm{v}}_1(i_v)) \\ elv(\tilde{\mathrm{v}}_1(i_v)) \end{bmatrix}
\equiv \begin{bmatrix} (i^-_{v\theta}+1)\Delta_{v\theta} \\ i^-_{v\phi}\Delta_{v\phi} \end{bmatrix},
$$

$$
\begin{bmatrix} azm(\tilde{\mathrm{v}}_2(i_v)) \\ elv(\tilde{\mathrm{v}}_2(i_v)) \end{bmatrix}
\equiv \begin{bmatrix} i^-_{v\theta}\Delta_{v\theta} \\ (i^-_{v\phi}+1)\Delta_{v\phi} \end{bmatrix},
$$

$$
\begin{bmatrix} W_0^v(i_v) \\ W_1^v(i_v) \\ W_2^v(i_v) \end{bmatrix}
\equiv \begin{cases} \begin{bmatrix} 1-(r_{v\theta}+r_{v\phi}) \\ r_{v\theta} \\ r_{v\phi} \end{bmatrix} & \text{if } (r_{v\theta}+r_{v\phi}) \le 1 \\ \begin{bmatrix} (r_{v\theta}+r_{v\phi})-1 \\ 1-r_{v\phi} \\ 1-r_{v\theta} \end{bmatrix} & \text{otherwise.} \end{cases}
$$

By the definition above, three sample view conditions $\tilde{\mathrm{v}}_m(i_v)(m=0,1,2)$ are selected so that the triangle they form includes the rendering view condition $\mathrm{v}_r(i_v)$ in the orthogonal coordinate plane of azimuth and elevation angles, and we can regard the triple of weights $[W_0^v(i_v), W_1^v(i_v), W_2^v(i_v)]^t$ as barycentric coordinates for the view condition in the azimuth-elevation coordinate space. If the rendering view condition $\mathrm{v}_r(i_v)$ is placed on the sample view condition $\tilde{\mathrm{v}}_0(i_v)$, the weight $W_0^v(i_v)$ is 1 and linearly decreases to 0 as $\mathrm{v}_r(i_v)$ moves toward the opposite side of the triangle formed by the three sample view conditions.

For the light direction, let $\tilde{\mathrm{l}}_0(i_v)$, $\tilde{\mathrm{l}}_1(i_v)$, $\tilde{\mathrm{l}}_2(i_v)$ be the three neighbor sampling light conditions of the rendering light condition $\mathrm{l}_r(i_v)$, and let $W_0^l(i_v), W_1^l(i_v), W_2^l(i_v)(W_0^l(i_v)+W_1^l(i_v)+W_2^l(i_v)=1)$. be the weights for the sampling light conditions. Weights can be seen as barycentric coordinates for the light condition. Neighbor sampling light conditions and their weights are calculated in similar way as that of sampling view conditions and their weights which is described above.

Using above notations, we can generate texture T_g of polygon $P(i_p)$ calculated from the rendering condition on vertex $V(i_v)$ as

$$
\begin{aligned}
&T_v(i_p, \mathrm{v}_r(i_v), \mathrm{l}_r(i_v))\\
&\quad\equiv \sum_{m=0}^{2}\sum_{n=0}^{2} W_m^v(i_v) W_n^l(i_v) T(i_p, \tilde{\mathrm{v}}_m(i_v), \tilde{\mathrm{l}}_n(i_v))\\
&T(i_p, \tilde{\mathrm{v}}_m(i_v), \tilde{\mathrm{l}}_n(i_v))\\
&\quad\equiv T(i_p, azm(\tilde{\mathrm{v}}_m(i_v))/\Delta_{v_\theta} elv(\tilde{\mathrm{v}}_m(i_v))/\Delta_{v_\phi},\\
&\qquad azm(\tilde{\mathrm{l}}_n(i_v))/\Delta_{l_\theta}, elv(\tilde{\mathrm{l}}_n(i_v))/\Delta_{l_\theta}).
\end{aligned}
$$

Note that $azm(\tilde{\mathrm{v}}_m(i_v))/\Delta_{v_\theta}$, $elv(\tilde{\mathrm{v}}_m(i_v))/\Delta_{v_\phi}$, $azm(\tilde{\mathrm{l}}_m(i_v))/\Delta_{l_\theta}$ and $elv(\tilde{\mathrm{l}}_m(i_v))/\Delta_{l_\phi}$ are all integers for $m=0,1,2$ because $[\tilde{v}_m, \tilde{l}_m]$ are sampling conditions where corresponding textures exist in the BTF database.

The final texture $T_p(i_p)$ of polygon $P(i_p)$ used for rendering is generated by blending three textures, and is calculated from the rendering conditions on three vertices forming the polygon, $V(t(i_p,j))(j=0,1,2)$. The blended textures are

$$
T_v(i_p, \mathrm{v}_r(t(i_p,m)), \mathrm{l}_r(t(i_p,m)), (m=0,1,2).
$$

The purpose of this process is to minimize the texture gap between polygons. This blending is done in the same way that pixel values of three vertices are blended when Gouraud shading is applied. Suppose that the texture coordinates $(0,0)$, $(1,0)$ and $(0,1)$ are mapped to the vertices $V(t(i_p,0))$, $V(t(i_p,1))$ and $V(t(i_p,2))$ respectively, and $(s_0(i), s_1(i))$ denote texture coordinates of ith texel. Then the texture $T_p(i_p)$ can be expressed as

$$
\begin{aligned}
&Texel(i, T_p(i_p))\\
&\quad= (1-s_0(i)-s_1(i))\\
&\qquad Texel(i, T_v(i_p, \mathrm{v}_r(t(i_p,0)), \mathrm{l}_r(t(i_p,0))))\\
&\qquad +s_0(i) Texel(i, T_v(i_p, \mathrm{v}_r(t(i_p,1)), \mathrm{l}_r(t(i_p,1))))\\
&\qquad +s_1(i) Texel(i, T_v(i_p, \mathrm{v}_r(t(i_p,2)), \mathrm{l}_r(t(i_p,2))))
\end{aligned}
$$

5.2. Alpha Estimation

When we acquire the textures, background images are removed by background subtraction. In the process, background pixels are often "mixed" into the resulting foreground images due to decision error of background pixels or

complicated object contours. To manage this error, we estimate the alpha values of textures at the contour of the object based on currently published techniques[4, 15].

To estimate the alpha value, we first detect the boundary region between foreground and background. This is nontrivial in general, and some fine algorithms have been proposed; chroma-key based technique is most well known. At this point, we already have the depth data of the object which is accurately matched to the image; therefore, boundary detection can be done automatically with good precision. The following is the actual process to detect the boundary and estimate the alpha value.

- Calculate the surface normal ($\mathrm{n}_p(i_p)$) for each polygon $P(i_p)$. If $arg(\mathrm{n}_p(i_p) \cdot \mathrm{v}_{viewdirection}) > \theta_{threshold}$ then consider the polygon to be located on the boundary.
- Divide the boundary area into small patches using the Delaunay Algorithm. Select one patch and search the nearest background and foreground areas by using a greedy algorithm
- Make color value clusters for background and foreground in RGB space by using the k-means algorithm. Then construct a network among background and foreground clusters
- Plot a pixel value from the selected patch into RGB space; then search the nearest link. Each node is estimated as background and foreground color. A ratio between foreground to pixel and background to pixel gives the alpha value.

Once the alpha values are estimated for the textures, we can use these values to prevent colors of the background image from appearing on the synthesized images. Texture images are synthesized by weighted average of original textures. In the averaging process, we multiply weight value by alpha value for each pixel. Since the sum of "modulated weight" may be less than 1, we divide the RGB color values by the sum.

6. Results

To demonstrate modeling of objects, we rendered a toy post shown in Fig. 4, a can wrapped with shiny paper (an example which has complicated surface attributes), and a stuffed animal with and without deformation. Geometrical data of the toy post was acquired only by voxel carving since the shape was roughly convex. The shape of the can was artificially generated as a cylinder. For the stuffed animal that has relatively complex shape, we used a range scanner to get surfaces and aligned them into the geometrical data obtained by voxel carving. The numbers of polygons forming the toy post, the can and the staffed animal were 5000, 1818 and 5000, respectively. Fig.5-7 show the post, the can, and the stuffed animal. To test the effectiveness of alpha value estimation, we rendered the stuffed animal without using the alpha estimation process, and fig.8 shows the results. For all experiments, capturing intervals of parameters (azimuth-elevation of light/view) are 30°. We can see that the lighting on the deformed objects' surfaces is correctly rendered. Two magnified parts of the synthesized images (the left ear and the right paw) are shown on the right side of Fig.7 and 8. We can see that artifacts due to the background colors of original images are much less severe in Fig.7 (with alpha estimation process) than in Fig.8 (without alpha estimation).

We also tried to merge the 3D CG object which was rendered by our image-based technique into conventional model-based 3D CG software. Fig.9 shows the result. The object in the center of the image is the image-based rendered object, while the rest of the scene was rendered with traditional 3D CG software. Since we set only one illumination in this situation, there are no soft shadows in the scene; the scene looks natural and the object was rendered photo-realistically.

7. Conclusion and Future Work

In this paper, we have proposed a modeling method based on actual textures. To construct models that can be rendered for arbitrary view/light directions, we captured 4D texture databases using a specialized platform. The platform has special facilities that consist of two concentric circles for the data acquisition process; these enabled us to easily capture sequential image data and a 3D model of the object, and to subsequently generate the texture data sets with 4D lighting/viewing parameters. To compress these 4D parameterized textures, we applied tensor products expansion and achieved higher compression rates than that of SVD-based compression.

To demonstrate the application of the captured models, we rendered several models with various deformations. With our proposed algorithm, we successfully rendered the deformed objects. For future work, we shall pursue applications of this method to CG animation and mixed reality systems.

References

1. K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets", *IEEE trans. PAMI*, **9**(5):698–700, 1987. 6

2. A. I. Borisenko and I. E. Tarapov. *Vector and tensor analysis with applications*. Dover Publications, N.Y., 1979. 4

3. W. C. Chen, R. Grzeszczuk, and J. Y. Bouguet, "Light Field Mapping: Hardware-Accelerated Visualization of Surface Light Fields", *SIGGRAPH 2001 Cource*. 2001. 1, 2, 3

4. Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski, "A Bayesian Approach to Digital Matting", *Proceedings of CVPR 2001*, pp. 264–271, 2001. 8

5. K. J. Dana, B.v. Ginneken, S. K. Nayar, and J. J. Koenderik, "Reflectance and texture of real-world surfaces", *ACM Transactions on Graphics*, **18**(1):1–34, 1999. 1, 2

6. R. Furukawa, H. Kawasaki, and K. Ikeuchi, "Acquiring Bidirectional Texture Function for Image Synthesis of Deformed Objects", *Proceedings of the Fifth Asian Conference on Computer Vision*, pp. 622–627, 2002. 2

7. S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph", *ACM SIGGRAPH*, pp. 43–54, 1996. 2

8. M. Levoy and P. Hanrahan, "Light field rendering", *ACM SIGGRAPH*, pp. 31–42, 1996. 2

9. T. Machida, H. Takeura, and N. Yokoya, "Dense Estimation of Surface Reflectance Properties for Merging Virtualized Objects into Real Images", *Proceedings of the Fifth Asian Conference on Computer Vision*, pp. 688–693, 2002. 1

10. G. Miller, S. Rubin, and D. Ponceleon, "Lazy decompression of surface light fields for precomputed global illumination", *Rendering Techniques (Eurographics'98 Proceedings)*, pp. 281–292, 1998. 2

11. J. Murakami, N. Yamamoto, and Y. Tadokoro, "High Speed Computation of 3-D Tensor Product Expansion by the Power Method", *IEICE Trans. (In Japanese)*, **J82-A**(8):1351–1359, 1999. 4

12. K. Nishino, Y. Sato, and K. Ikeuchi, "Eigen-texture method: Appearance compression based on 3D model", *Computer Vision and Pattern Recognition*, volume 1, pp. 618–624, 1999. 1, 2, 3

13. M. Potmesil, "Generating octree models of 3D objects from their silhouettes in a sequence of images", *Conputer Vision, Graphics, and Image Processing*, **40**(1):1–29, 1987. 3

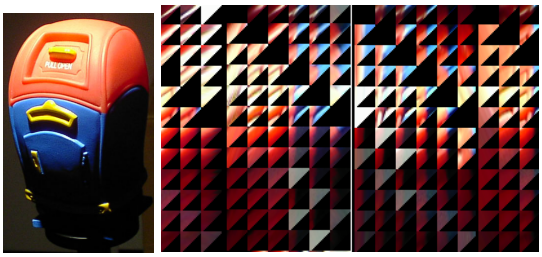
14. S. M. Rusinkiewicz, "A New Change of Variables for Efficient BRDF Representation", *Eurographics Rendering Workshop 1998*, pp. 11–22, 1998. 3

15. M. Ruzon and C. Tomasi, "Alpha Estimation in Natural Images", *Proceedings of CVPR 2000*, pp. 24–31, 2000. 8

16. R. Szeliski, "Rapid octree construction from image sequences", *Conputer Vision, Graphics, and Image Processing*, **58**(1):23–32, 1993. 3

17. M. D. Wheeler and K. Ikeuchi, "Sensor modeling, probabilistic hypothesis generation, and robust localization for object recognition", *IEEE Trans. on PAMI*, **17**(3):252–265, 1995. 3

18. D. Wood, D. Azuma, W. Aldinger, B. Curless, T. Duchamp, D. Salesin, and W. Steutzle, "Surface light fields for 3D photography", *ACM SIGGRAPH*, 2000. 1, 2



(a) (b)

Figure 4: *Model and texture database: (a) an original image of the modeled object, (b) visualization of parameterized textures, in which each row of textures is captured from a certain view direction and each column of textures is captured for a certain light direction.*

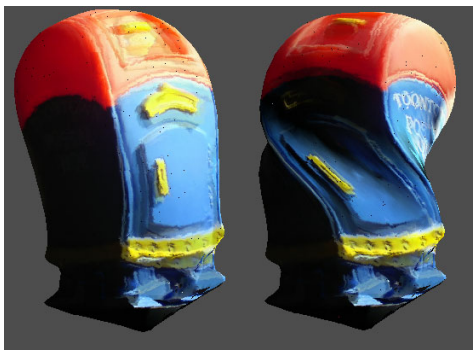


Figure 5: *Rendering examples: a toy post*

Figure 6: *Rendering examples: a can wrapped with shiny paper*

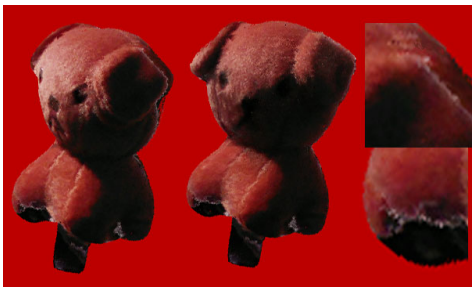


Figure 7: *Rendering examples: a stuffed animal with alpha estimation*

Figure 8: *Rendering examples: a stuffed animal without alpha estimation.*

Figure 9: *Rendering with model based 3D CG system*