PATCH-BASED BTF SYNTHESIS FOR REAL-TIME RENDERING

Hiroshi Kawasaki, Kyoung-Dae Seo, Yutaka Ohsawa

Saitama University, Japan
Faculty of Engineering
kawasaki@mm.ics.saitama-u.ac.jp

Ryo Furukawa

Hiroshima City University, Japan
Faculty of Information Science
ryo-f@cs.hiroshima-cu.ac.jp

ABSTRACT

In this paper, we propose a novel synthesis technique for BTFs. A BTF (bidirectional texture function) is a 6D function which can represent appearances of a texture under arbitrary view and lighting conditions. Until now, several approaches of BTF synthesis have been researched. For ordinary textures, patch based methods are promising techniques for texture synthesis. However, it has not been effectively tried to BTFs yet. This is mainly because data size of BTFs is so large and it is not easy to apply the techniques to BTFs. Further, efficient rendering of BTFs is still under research. In this paper, we extend a patch based synthesis technique for BTFs by utilizing compact representation of BTFs. In addition, we propose a novel technique to render the synthesized BTFs efficiently. With our proposed method, we can successfully and effectively render objects with complicated surfaces under arbitrary sizes.

1. INTRODUCTION

A BTF(bidirectional texture function) [1] is a 6D function which can describe the changes of reflections and appearances of a texture under arbitrary lighting and view conditions. By using BTFs, rendering of objects with higher level of reality than before with less computational cost is possible. Thus, many intensive researches of BTFs have been conducted so far. For actual use of BTFs, seamless synthesis of BTFs is necessary to cover large surfaces with small BTFs. To date, several approaches of BTF synthesis have been proposed. For example, Liu et al. proposed a method to reconstruct a meso-scopic geometry by using a shape-from-shading technique [2], and Tong et al. proposed a method of synthesizing BTFs using surface textons [3].

Patch based methods [4] are effective techniques for ordinary texture synthesis. They are also considered to be applicable for synthesis of BTFs. However, few techniques which adopt the patch based method for BTFs have been proposed. A reason for this is that huge data sizes of BTFs make it difficult to simply apply the techniques to BTFs. Another reason is difficulty of efficient sampling and representation of BTFs.

In this paper, we propose a patch based synthesis technique for BTFs. To achieve this, we used a compact BTF representation which utilizes compression by spherical harmonics (SH) for real-time rendering. In addition, we propose a novel technique to render the synthesized BTF efficiently by using programmable shaders which function on hardware. By using the proposed techniques, we can render complicated shaped objects under arbitrary lighting conditions and view directions in real-time and reduce quantity of storage.

2. REPRESENTATION OF BTFS

A BTF is expressed as a function with arguments of a light direction, a view direction, and texture coordinates. A BTF with a discretized view direction parameter is expressed as

$$T(i_v, \mathbf{l}, s, t, c),$$

where i_v is index of view, $\mathbf{l}$ is light direction, (s, t) is image coordinates of appearance and c is an index for designating three primal colors (i.e., red, green, blue).

In this paper, we approximate light-dependent variation of object appearances by SH expansion, whereas view-dependent variation are kept discretized. Interpolation of view is realized by view dependent texture mapping(VDTM).

Normally, light-dependent variations of appearances with fixed views are relatively smooth (although there are exceptions such as self-shadows or speculars) and can be easily interpolated and approximated. We use SH expansion for this purpose, which is similar to the technique of PRT[5].

Approximation of light-dependent variation of appearances are processed for each view samples as the following. First, pixel values for fixed view index and fixed pixel coordinates are extracted from the appearance samples, which represents a light-dependent variation of appearances. Then, these pixel values are aligned as the light directions in the object coordinates, and rendered to a cubic texture-map. Using smooth shadings, the sample values are interpolated for light directions.

Coefficients of series expansion in terms of SH can be acquired by spherical integrals. The integrals can be approximately calculated by calculating products of texel values, values of spherical harmonics basis functions, and the solid angles of the texels for each texel in the cube map, and then summing up all the products. Using resulting coefficients, the function approximating light-dependent variation of T can be expressed as

$$\sum_{m=1}^{M} c_m(i_v, s, t, c) Y_m(\mathbf{l})$$

where Y_m is m-th SH basis function, $\mathbf{l}$ is light direction, i_v is view index, (s, t) is texture coordinates of sampled appearance and c is the color channel.

The above approximation process is repeated for all of the view samples, the texture coordinates, and color channels. The acquired SH coefficients, which is expressed as vectors, are approximated using primary component analysis (PCA). To reduce computational complexity for calculation of SVD, we sample the vectors of SH coefficients and use only a small number of them as inputs.

Weight vectors calculated with PCA are indexed by (i_v, s, t). Let us denote the k-th principal component vectors as $b_{k,m,c}$ $(1 \leq k \leq K)$ and their weights as $W_k(i_v, s, t)$. The light dependent variation of appearance at view direction i_v and texel location $(s, t))$ can be approximated as the following function:

$$T(i_v, \mathbf{l}, s, t, c) \approx \sum_{m=1}^{M} \sum_{k=1}^{K} W_k(i_v, s, t) b_{k,m,c} Y_m(\mathbf{l}) \quad (1)$$

The calculated function (1) is an approximation of appearance which is continuous for light direction $\mathbf{l}$ and discrete for view direction. By taking a weighted sum of the function (1) for three view directions, we approximate view-dependent variation, similarly with VDTM[6].

3. SYNTHESIS ALGORITHM

3.1. Quilting BTFs

3.1.1. Image quilting algorithm

Image quilting [4] is a simple method for creating arbitrary sizes of textures from small input textures. We apply this method for BTFs represented as section 2. This enables us to synthesize BTFs of arbitrary large sizes from source BTFs with finite sizes.

Let us explain the idea of image quilting which is shown in Fig. 1. The resulting texture is synthesized by aligning unit image patches, which are normally square blocks. The image patches are selected from the subimages of the input texture. The selection is performed randomly from the subimages which satisfy certain constraints. The purpose of the constraints is for selecting subimages such that adjacent image patches have "good continuities." To measure the

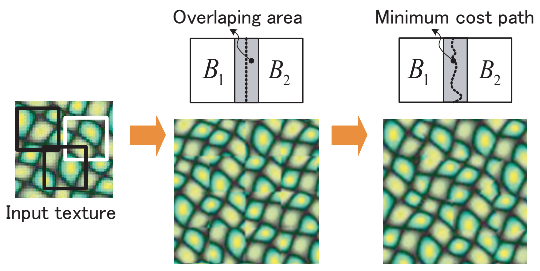

Fig. 1. Image Quilting.

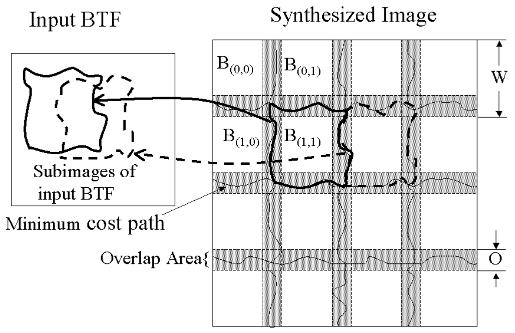

Fig. 2. Quilting of a BTF.

continuities, overlapping areas are introduced around the image patches. The constraints are that the differences of the image pixels of the adjacent patches on the overlapping areas are within some error tolerance.

The borders of the aligned image patches in the synthesized image is determined such that the borders go through the minimum cost paths. The minimum cost paths are paths that go through the overlapping areas, on which the differences between the pixels of the adjacent patches are minimum. They can be calculated by dynamic programming.

3.1.2. Extension for BTF

The coefficients $W_k(i_v, s, t)$ in the form (1) can be considered as textures indexed by i_v, whose texture coordinates are (s, t). Quilting of BTFs is applied for these images.

The steps of the algorithm are as the following. Let i and j be row and column indices of grids of the synthesized image. Also let $B_{(i,j)}$ denote the patch at row i and column j. Fig. 2 shows the concept for quilting a BTF.

1. Randomly choose a $W \times W$ texture patch for the grid $B_{(0,0)}$ from the subimages of $W_k(i_v, \mathbf{p})$. The patch size W is defined previously by the user.

2. For every grid $B_{(i,j)}$, randomly choose a patch from some candidate subimages that satisfy the overlap constraints, in which that the error d calculated for the overlap area is within the error tolerance d_{max}. The overlap size O is defined previously by the user.

3. Find the minimum cost path along the overlapping area by dynamic programming.

4. Paste the patch $B_{(i,j)}$ cut along the minimum cost path onto the output texture.

5. Repeat the steps 2-4 until the output texture is fully covered.

The error d and The error tolerance d_{max} are defined as

$$d = \{\frac{1}{N}\sum_{v=1}^{V}\alpha_v\sum_{m=1}^{K}\beta_m\sum_{k=1}^{N}(p_k^1 - p_k^2)^2\}^{\frac{1}{2}} \quad (2)$$

$$d_{max} = \{\frac{1}{N}\sum_{v=1}^{V}\alpha_v\sum_{m=1}^{K}\beta_m\sum_{k=1}^{N}(\epsilon p_k^1)^2\}^{\frac{1}{2}} \quad (3)$$

where N is the number of pixels in the overlap domain, K is the number of principal components used, β_m is the weight of the SH coefficient designated by index m, V is the number of view directions and α_v is a weight for view direction designated by index v. p_k^1 and p_k^2 are the coefficients of the BTF at the k-th pixel on the overlap domain of adjacent patches. ϵ is the error sensitive parameter by which the user can adjust the error tolerance.

The minimum cost path is defined by dynamic programming. The cost for a vertical path is defined as

$$e_p(i_v) = \sum_{v=1}^{V}\alpha_v\sum_{m=1}^{K}\beta_m(O_p^1 - O_p^2)^2 \quad (4)$$

$$E_{k,l} = e_{k,l}(i_v) + min(E_{k-1,l-1}, E_{k-1,l}, E_{k-1,l+1}) \quad (5)$$

where $p = (k, l)$ are coordinates of pixels, e_p is the error at the coordinates p, O_p^1 and O_p^2 are the coefficients of the BTF at the coordinates p. To compute the horizontal minimum cost path, the similar method is applied.

Regarding $W_k(i_v, s, t)$ as a k-channel, values for each pixel represent light-dependent variations of the BTF. All the light-dependent variations are integrated into the error function by just summing up the errors for all k channels.

The view-dependent variations of the BTF for a fixed pixel coordinates includes shifts of the geometrical locations on the surface. So, preparing a different border for each view direction might be desirable if each view direction were synthesized independently with other views. However, if textures for different view directions have discontinuities, interpolations between views will not work properly and serious flickering will occur with view changes. Thus, our method produces only one set of borders for all BTFs.

To acquire satisfying synthesized BTF, it is necessary that the input BTF is large enough for synthesis algorithm. It means that the search area of a similar patch in the input image is large, and thus, computational cost is high. For a solution of this, we adopt pyramid technique. In addition, we learn that a low frequency of SH coefficient largely influences the synthesis of BTFs from an experimental experience, therefore, we can reduce applied SH coefficient to 2-dimensions.

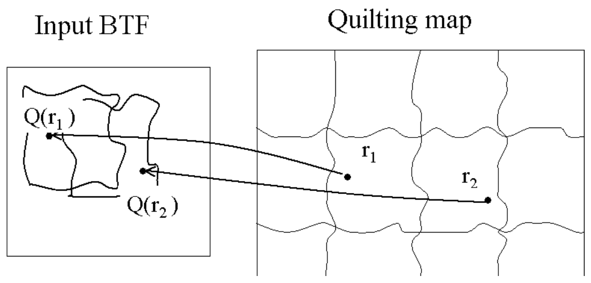

Fig. 3. Quilting Map.

3.2. Quilting map

To model a quilted BTF, we use a texture image which maps texture coordinates of the synthesized BTF into texture coordinates of the source BTF. In this paper, the texture which consists of the texture coordinates is called a "quilting map." Each texel in the quilting map represents texture coordinates which point a spatial location in the source BTF. Fig.3 shows the relationships between the quilting map, the synthesized BTF and the source BTF. In this paper, the quilting map is denoted as $Q(\mathbf{r})$, where $\mathbf{r}$ represents the texture coordinates of the synthesized BTF and Q represents the mapping from the texture coordinates of the synthesized BTF into those of the source BTF. In fig.3, $\mathbf{r_1}$ and $\mathbf{r_2}$ are the texture coordinates of the synthesized BTF and $Q(\mathbf{r_1})$ and $Q(\mathbf{r_2})$ are those of the input BTF. Regarding the source BTF as a palette, a quilting map is a kind of index image representing the synthesized BTF. So, we can use the quilting map as the synthesized BTF in the process of quilting of BTFs described in subsection 3.1. As a result, a complete quilting map is acquired.

4. REAL-TIME RENDERING OF BTFS

Real-time rendering with quilted BTFs can be performed using programmable shaders.

First, we load encoded data as textures onto the video hardware for acquisition actual values for rendering. Encode data $Q(r)$ is obtained from the value r of the quilting map to the input BTF by referring. Then using $Q(r)$, which is a coefficient of SH, original RGB value can be efficiently recovered by using programmable shaders [5, 7]. If we can use graphic hardwares which support programmable shaders, real-time rendering can be achieved.

In [5], SH coefficients is used as input BTF, $Q(r)$ is SH coefficient c_m. However, it is not necessary that $Q(r)$ have to be SH coefficient. If it is encode data to which $Q(r)$ is interrelated with SH coefficient, $Q(r)$ may be data different from the input data of the synthesis. In this paper, we use the principal ingredient which approximated by primary component analysis (PCA) as for the SH coefficient.

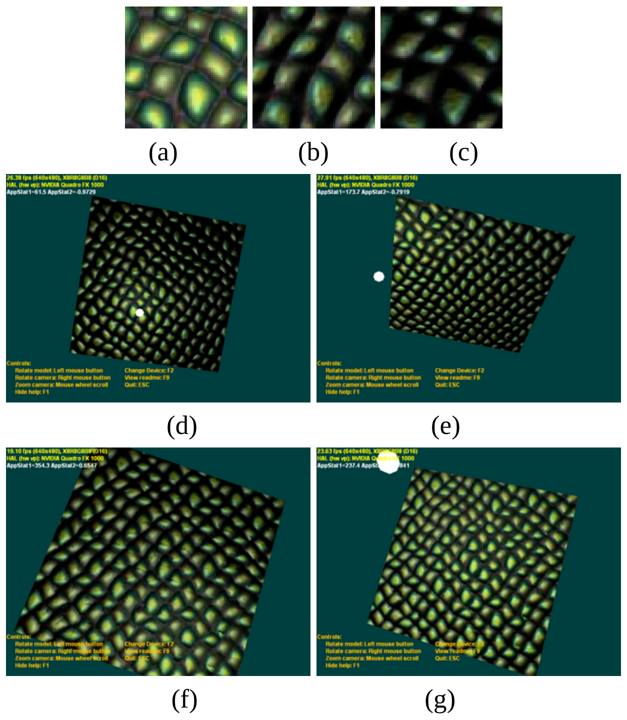

Fig. 4. Synthesized texture under arbitrary light and view directions

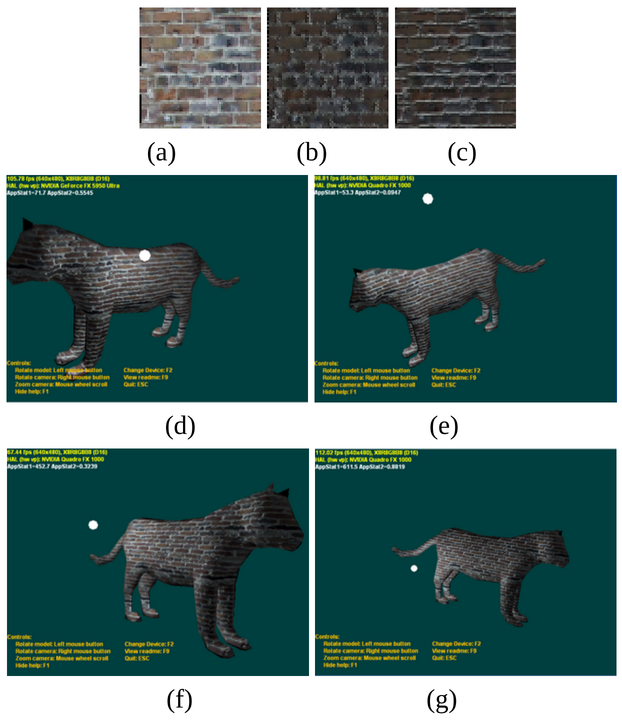

Fig. 5. Rendering complicated shaped object under arbitrary lighting conditions

5. EXPERIMENTS

Fig. 4 shows the result of experiment. Fig. 4 (a)-(c) are original samples of BTF and size is 64x64 pixel. Fig.4 (d)-(g) are rendered object under arbitrary lighting and view conditions using synthesized BTF. Size of synthesized BTF is 256x256 pixel and rendering frame rate is more then 30 fps. In these figures, we can hardly find the boundary between patches, besides lighting and view-dependent effects are successfully rendered.

Our proposed method can synthesize complete set of BTF, therefore, we can render arbitrary shaped object under arbitrary lighting and view conditions. Fig.5 shows the result of our proposed method. We can see that shading effects and specular effects derived from mezo-scopic geometry are successfully rendered without any boundary in the textures by our proposed method.

6. CONCLUSIONS

In this paper, we have proposed a patch-based synthesis technique for BTF. At first, the technique compress a large amount of data to produce several representative textures using sphrical harmonics. Then, we apply simple patch-based synthesis technique to the representative textures; this time, our technique does not create actual large size of BTF data, but “quilting map” for efficient and compact data representation. Finally, we render the object with “quilting map” using programmable shaders which realize real-time rendering of synthesized BTF.

7. REFERENCES

[1] K. Dana, B.V. Ginneken, and J. Koenderink, “Reflectance and texture of real-world surfaces,” in *ACM Transactions on Graphics*, Jan. 1999, pp. 1–34.

[2] X. Liu, Y. Yu, and H.-Y. Shum, “Synthesizing bidirectional texture functions for real-world surfaces,” in *Proc. SIGGRAPH 2001, Computer Graphics Proceedings*, Aug. 2001, pp. 97–106.

[3] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum, “Synthesis of bidirectional texture functions on arbitrary surfaces,” in *Proc. SIGGRAPH 2002*, July 2002, pp. 665–672.

[4] A.A. Efros and W.T. Freeman, “Image quilting for texture synthesis and transfer,” in *Proc. SIGGRAPH 2001*, Aug. 2001, pp. 341–346.

[5] Sloan, Peter-Pike, Jan Kautz, and John Snyder, “Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments,” *ACM Transactions on Graphics (TOG)*, pp. 527–536, 2002.

[6] Paul E. Debevec, George Borshukov, and Yizhou Yu, “Efficient view-dependent image-based rendering with projective texture-mapping,” in *9th Eurographics Rendering Workshop*, 1998, pp. 105–116.

[7] K. Jyo, H. Kawasaki, F. Ryo, and Y. Ohsawa, “Btf synthesis for hardware accelatered real-time rendering,” in *FIT2004*, Sept. 2004, pp. 247–248.