Submitted to JOURNAL OF INFORMATION SCIENCE AND ENGINEERING

Physically-Based Analysis and Rendering of Bidirectional Texture Functions Data

YING-CHIEH CHEN, SIN-JHEN CHIU, HSIANG-TING CHEN, CHUN-FA CHANG
Department of Computer Science
National Tsing Hua University
Hsinchu, 300 Taiwan

In order to draw a photorealistic surface, Bidirectional Texture Function (BTF), a 6D texture function which extends Bidirectional Reflectance Distribution Function (BRDF) to include the self-shadowing, self-occlusion and inter-reflection effects, has been used frequently in recent years. Its main drawback is its massive data size. To solve this, the Spatial Bidirectional Reflectance Function (SBRDF) techniques compress BTFs into reflectance model parameters. However, SBRDF cannot produce the self-shadowing and self-occlusion effects in real-world surface geometry. This work is aimed to this drawback. We investigate how self-shadowing and self-occlusion affect the surface appearance by additional physically-based analysis and rely on two physical phenomena to separate self-shadowing and self-occlusion into two independent effects. First, self-shadowing is view independent. Second, self-occlusion is independent of lighting direction changes. After these analyses, we add self-shadowing and self-occlusion to SBRDF to achieve rendering quality that is much closer to the original uncompressed BTF data.

Keywords: real-time rendering, GPU, Bidirectional Texture Functions, self-shadows, self-occlusion

1. INTRODUCTION

In recent years, we have seen continuing improvement of visual realism in computer graphics based feature movies and video games. As the visual realism in feature movies and video games keeps improving, the importance of advanced surface materials and appearance models also increases. Another driving force of advanced surface and appearance models is the increasing power of modern graphics hardware which supports programmable shading stages. Simply scanning the shape of a 3D object and then applying uniform colors, simple Phong shading model, or static textures to the surfaces no longer suffices.

In addition, good surface appearance models can even hide the deficiency in the geometric shape, especially if advanced texturing techniques such as bump mapping [13], displacement mapping, or view-dependent texture mapping [5] is used. Imagine that a sphere model with a golf ball texture on it can be easily recognized as a golf ball even though we do not explicitly model the geometric shape of each dimple. In fact, using view-dependent texturing effects to replace fine geometry should be no surprise if we treat those textures as *surface light fields* [14][9][3] or *lumigraph* [2][6]. From a different point of view, we could also consider advanced surface appearance models such as the surface light fields and bidirectional texture functions as a way to avoid the daunting task of modeling the geometric shapes of micro-level or meso-level structures of complex materials.

There are many surface models that are extended from the well-known Bidirectional Reflectance Distribution Function (BRDF). If the incident light position is different from its exit position, then we have the Bidirectional Surface Scattering Distribution Function (BSSRDF). This extension allows BSSRDF to model sub-surface scattering effects of translucent materials, such as human skin or a lighting candle [7]. Furthermore, if different surface points contain different BRDFs, then the Spatial Bidirectional Reflectance Distribution Function (SBRDF) [11] and the Bidirectional Texture Function (BTF) [4] may be applied. The Bidirectional Texture Function (BTF) is a 6D function which describes the spatially-variant reflectance and the mesostructure effects, such as self-shadowing, self-occlusion, and self-reflectance. The 6D function contains two parameters for surface position, two for viewing direction and the other two for illumination direction. Therefore, during rendering, we can determine the color from BTF by the given surface position, light direction and viewing direction. Because BTF captures the self-shadowing, self-occlusion, and self-reflectance effects that are exhibited by the mesostructures of complex materials, it can reproduce the surface appearance of real-world objects with extremely high fidelity. A drawback of BTF, however, is its requirement of huge storage space since it is a 6D function. Many works have been focusing on the compression of the BTF datasets. A good survey can be found in [12]. Most of those works are based on the fitting of analytical BRDF models [11][10] or the matrix factorization methods.

In this paper, we demonstrate a physically-based analysis of BTF model and its rendering in real-time. In the first part of our work, we present a method to detect the data points under self-shadowing by treating them as outliers during the SBRDF [11] fitting process. The outliers are first considered as a 6D shadow function and will be reduced into a 4D function as described in Section 2.2.

In the second part of our work, we use two phenomena to separate self-shadowing and self- occlusion into two independent effects. First, when the self-shadowing occurs is often a view independent problem (Figure 5). Second, the surface point we look at is independent of lighting direction changes (Figure 8). We use the first phenomenon to reduce the dimension of 6D shadow function into 4D and use the second phenomenon to analyze the self-occlusion effect.

Finally, we use SBRDF parameters with additional self-shadowing and self-occlusion representations to represent the original BTF efficiently. And we also apply this representation onto programmable graphics hardware. So a complex mesostructure surface can be rendered and relighted in real time.

2. PRECOMPUTATION OF SURFACE GEOMETRY EFFECTS

Geometric effects can influence function fitting of a reflectance model significantly. For example, we use the Lafortune reflectance model which consists of several cosine lobes for function fitting. Fewer specular lobes of a reflectance model may be used if the geometry effect can be separated from the fitting data.

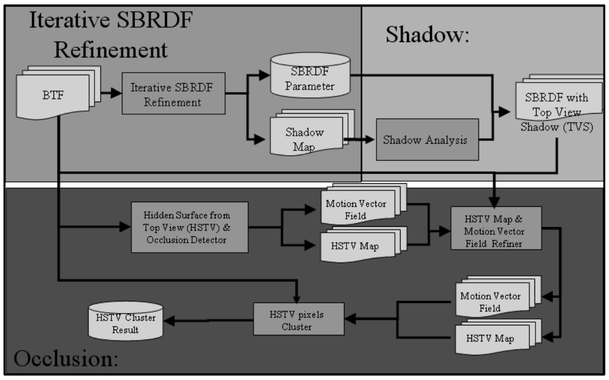

Iterative SBRDF Refinement
Shadow:
BTF
Iterative SBRDF Refinement
SBRDF Parameter
Shadow Map
Shadow Analysis
SBRDF with Top View Shadow (TVS)
Hidden Surface from Top View (HSTV) & Occlusion Detector
Motion Vector Field
HSTV Map
HSTV Map & Motion Vector Field Refiner
Motion Vector Field
HSTV Map
HSTV pixels Cluster
HSTV Cluster Result
Occlusion:

Figure 1 The framework of my preprocess system

The geometry effects can be roughly divided into self-shadowing, self-occlusion, and inter-reflection parts. However, determining when they occur in the surface is very difficult. Inter-reflection exists all the time, and detect self-occlusion automatically is still an open problem in computer vision.

SBRDF is a 6D function which stores all values at different views, lights and surface points. In this section, we propose a method to separate self-shadowing and self-occlusion into independent effects by physical phenomenon. This physically-based analysis will reduce the 6D shadow function into two 4D functions. One of the 4D functions is a shadow map, and the other is a viewing shift map which does not change with the lighting. **Figure 1** shows the framework of our preprocessing system.

The iterative SBRDF refinement method is introduced in section 2.1. Why and how to analyze self-shadow effect are explained in section 2.2 and self-occlusion in section 2.3.

2.1 Iterative SBRDF Refinement

Lafortune el al. [8] proposed a reflectance model that uses several cosine lobes to represent the surface appearance. As done in [11], we first fit the BTF data into spatially-varying BRDF (SBRDF) using the Lafortune model. This section will describe how to find the outliers (data affected by self-shadowing and self-occlusion).

It is reasonable to assume that self-occlusion is the material property when we detect the self- shadowing only. Although detecting self-shadowing correctly is also difficult, it can be approximated by using the following observation.

A reflectance model can represent a BRDF which changes rather regularly as shown in **Figure 2(b).** However, self-shadowing makes BRDF exhibit discontinuity as shown in **Figure 2(a).** So the shadow can be detected by comparing the color difference between the original pixel-wise BTF data and the BRDF reconstructed from the reflectance model. The data will be decided as shadow if its fitting error is above a threshold (i.e., $Error(x,Li,Vi) < threshold_{error}$ where Li and Vi denote the lighting and viewing directions) and it is in a dark region (i.e., $BTF(x,Li,Vi) < threshold_{BTF}$). The Error function is defined as follows:

$$Error(x, L_i, V_i) = \frac{SBRDF(x, L_i, V_i) - BTF(x, L_i, V_i)}{BTF(x, L_i, V_i)}$$

After this step, the shadow data will be discarded from the original pixel-wise BRDF and we can iteratively refit the remainders to get more accurate parameters. In our experiment, we set both $threshold_{error}$ and $threshold_{BTF}$ to be 0.3 and repeat this process three times for each pixel. **Figure 3** show the results before and after the refinement, and **Figure 3(c)** is the approximate shadow map and the gray level is the shadow level which is the ratio of real data (BTF) to our refined SBRDF data. Shadow level is determined as follow:

$$ShadowMap(x, L_i, V_i) = \begin{cases} 1, & if\ BTF(x, L_i, V_i)\ is\ not\ shadow \\ BTF(x, L_i, V_i) / SBRDF(x, L_i, V_i), & otherwise \end{cases}$$

After this processing, the BTF reconstruction function will be:

$$BTF(x,L,V) = SBRDF(x,L,V) \times ShadowMap(x,L,V)$$

and **Figure 3(c)** shows the difference between the original pixel-wise BRDF and the SBRDF multiplied by shadow map. Compare **Figure 4 (b)** and **Figure 4 (c)**, the self-shadowing effect can be perceived after applying the approximate shadow map.

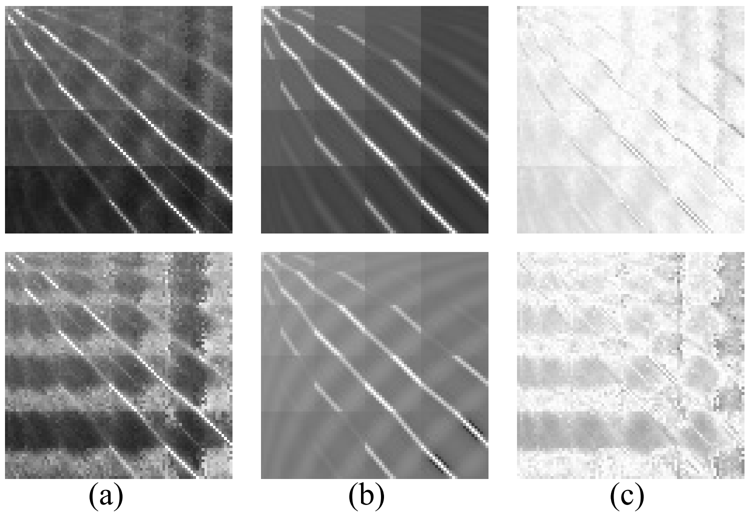


Figure 2 The original pixel-wise BRDF vs. SBRDF reconstructed BRDF and its error. The two rows show them at two different pixel locations. (a) is the original pixel-wise BRDF, (b) is the SBRDF, (c) is the error between the original and SBRDF where darker levels represent larger errors.

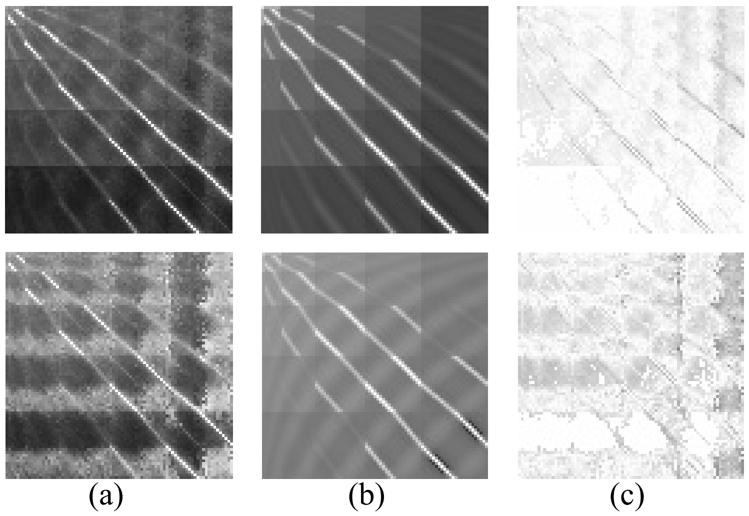


Figure 3 Original pixel-wise BRDF vs. refined SBRDF reconstructed BRDF and its error. The two rows show them at two different pixel locations. (a) is original pixel-wise BRDF, (b) is refined SBRDF, and (c) is the error between them where darker levels represent larger errors

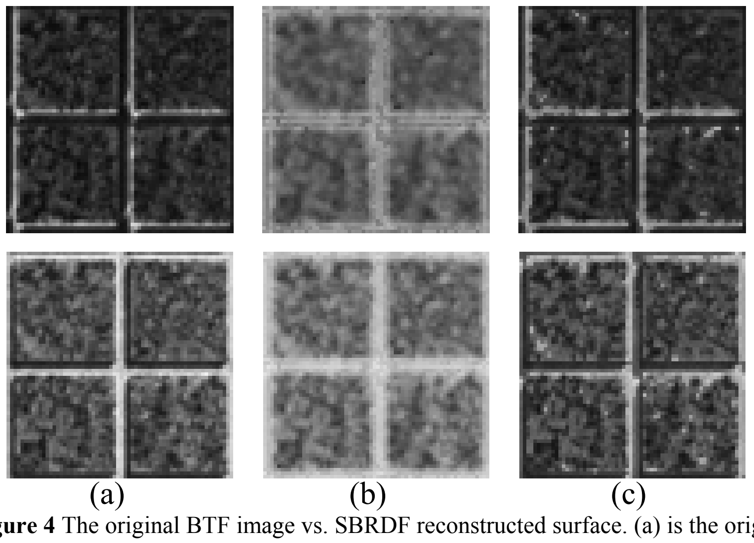


Figure 4 The original BTF image vs. SBRDF reconstructed surface. (a) is the original [1], (b) is the SBRDF, (c) is the result of refined SBRDF multiplied by shadow map.

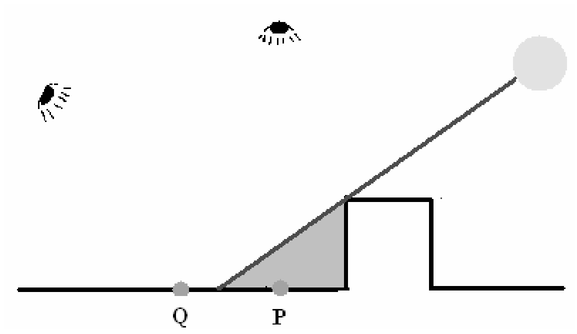


Figure 5 Shadow regions are view independent, meaning they do not vary with the viewing position when the lighting is fixed.

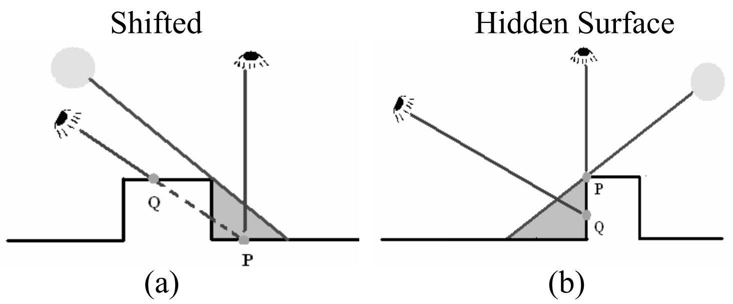


Figure 6 Occlusion results in error of applying top view shadow to other views. (a) is P occluded by Q, and (b) is Q is a invisible point at top view.

2.2 Shadow Analysis

In Section 2.1, we present a method to generate the approximate shadow map, but there is still a problem, that is, those shadow maps are too irregular to find an effective representation. The reason why shadow map is irregular is that surface geometry effects such as self-shadowing and self-occlusion can affect each other in a rather complex manner, so the whole shadow maps can only be represented as a 6D function. In our observation there is a lot of redundant data in the 6D representation, and it is possible to reduce the 6D function down to 4D if there is no occlusion effect. Take **Figure 5** for an example. When the lighting direction is fixed, *P* is always in the shadow side and *Q* is always in the bright side no matter how the viewing direction could be changed if there is no self-occlusion. In other words, without self-occlusion, the shadow regions will not change with the viewing direction. So we only need to store the surface shadow regions once for each lighting direction regardless of the viewing direction. The representative view we choose is the *Top View* because it is the symmetrical center of all sampled viewing directions, and the *Top View Shadow* (TVS) is called the shadow regions. After this step, the dimensionality of shadow function is reduced from a 6D function *ShadowMap(x,L,V)* to a 4D function *TopViewShadowMap(x,L)* and BTF reconstruction function can be modified as follow:

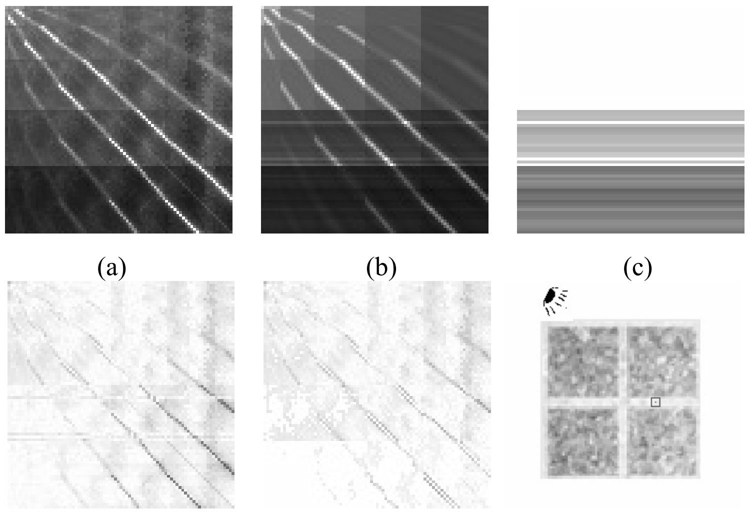


Figure 7 The original pixel-wise BRDF vs. STVS (SBRDF multiplied by Top View Shadow). (a) is the original pixel-wise BRDF, (b) is STVS reconstructed result, (c) is top view shadow, (d) is the error between them, and (e) is the error by 6D shadow function where darker levels represent large errors. (f) shows where the sample point is on original surface.

$$\begin{aligned} &BTF(x,L,V) \doteqdot STVS(x,L,V) \\ &\qquad = SBTDF(x,L,V) \times TopviewShadowMap(x,L), \end{aligned}$$

and **Figure 7** shows the STVS reconstructed result, and *STVS* is the acronym of SBRDF multiplied by TVS. The error of STVS in **Figure 7(d)** and SBRDF multiplied by 6D shadow function in **Figure 7(e)** are almost the same. So using 4D shadow function TVS to approximate 6D shadow function is enough for lots of viewing directions. However, the more we lower the viewing direction toward the surface the more occlusion occurs. It can be seen that **Figure 7(d)** shows larger error than **Figure 7(e)** at lower viewing direction. Take **Figure 6(a)** for an example. *P* is in the shadow region at V_{top}, but is occluded by *Q* at V_{target} and *Q* is in the bright region. If TVS is applied to V_{target}, *Q* will use *P*'s shadow level and make a critical error. The other larger error will occurr as **Figure 6(b)**. It shows that *Q* is a invisible surface at V_{top} but not at V_{target}. Such this case the shadow map will be very different from TVS as shown in **Figure 7(f).**

2.3 Occlusion Analysis

In this section, we focus on self-occlusion of BTF. By the similar analysis, there is a simple property of self-occlusion. The property is that the surface point we look at is independent of lighting direction changes as shown in **Figure 8**. According to this property, we can discard the lighting effect and only analyze when the occlusion effect occurs for each view direction. In the other words, the self-occlusion effect can be represented by a 4D function.

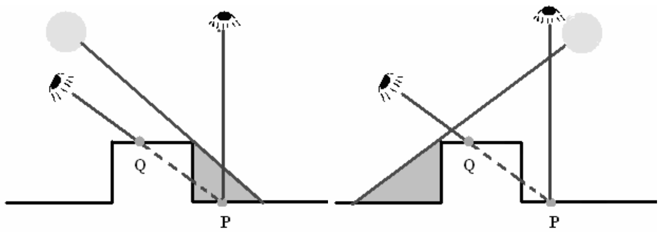


Figure 8 Surface point we look at is independent of lighting direction changes.

The occlusion effect can be separated into two cases. One is shifted, and the other is hidden surface as shown in **Figure 6**. The shifted effect is shown in **Figure 6(a)**. *P* is occluded by *Q* at V_{target}, but *Q* is still a visible point at V_{top}. For this case, we only need a texture shift to move *P* to the correct point *Q*. The other case is hidden surface as shown in **Figure 6(b)**. For this case, we cannot find the relative texture shift between V_{top} and V_{target}. So It need additional process to deal with it.

In subsection 2.3.1, we represented a method, called *vote-based optical flow*, to recognize if a surface point at V_{target} is a shifted point or a hidden surface point. In subsection 2.3.2, we represent a method to refine the distinguished result more correctly. In subsection 2.3.3, we present the additional process to deal with hidden surface points.

2.3.1 Vote-based Optical Flow

Optical flow is a method which finds the motion between two similar pictures. But it does not work well when applied to any two images in BTF data. It is because that the color of two images with different view direction and the same lighting direction may change significantly, so two images are not enough.

As we mention that when the view setting is fixed, the surface point we look at will not be changed no matter where the light is located. In the ideal case, every image pair with different views and the same lighting direction should have the same occlusion shift and hidden surface map.

According to this assumption, all the images with the same view direction can be considered as a group and to replace the role of the two 2D images in the original optical flow algorithm with the corresponding two image sets. Then the new result of the two image sets is several optical flow results. These several optical flow results can vote a best occlusion shift map from all occlusion shift maps and determine whether the pixels are hidden surface points or not by all hidden surface maps. The condition to determine that a pixel *x* is a hidden surface point is that $HiddenRatio(x) > Threshold_{hidden}$. *HiddenRatio(x)* is defined as follow.

$$HiddenRatio(x) = avg(HiddenMap(x))$$

If *x* is not a hidden surface point, then the motion vector of x should be calculated. The method of how to vote its motion vector is as follow. *VotedOcclusionShift(x)* is a function to query the motion vector of point *x*. It is decided by majority from motion vectors of all sample lighting pairs between two views. And this is why we call it *vote-based optical flow*.

Figure 9 shows vote-based optical flow results. The hidden surface map is detected at the fillisters of the brick as **Figure 9(d)** as we expect. But the detected result as **Figure 9(e)** is not well enough. There are some parts like the four upstairs squares on the surface in **Figure 9(a)(b)**. They should not be the hidden surface but we consider them as hidden one after the threshold determined. So we want to correct those parts by a refinement process which will be introduced in the next subsection.

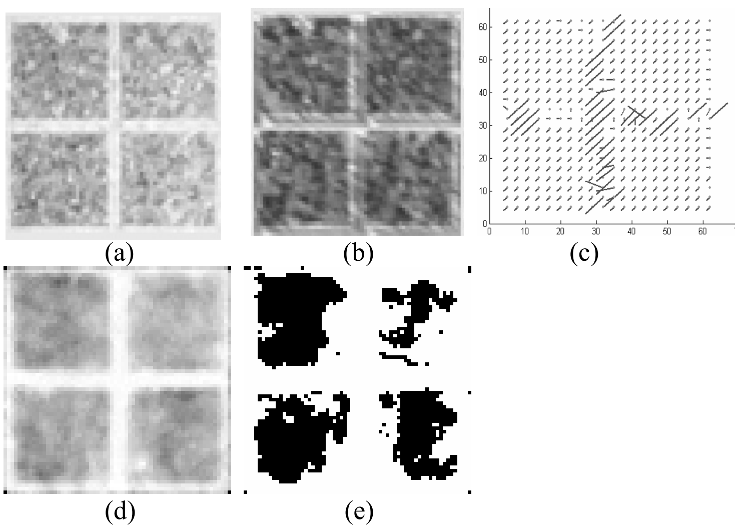


Figure 9 Vote-based optical flow result. (a) is the top view image, (b) is the target view image, (c) is the motion vector field after voted, (d) is hidden surface map ratio where darker levels represent low hidden probability, (e) is determined hidden surface map which threshold is 0.7, and white parts are detected hidden surface.

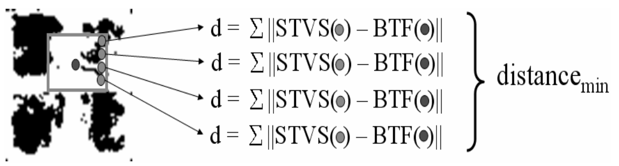

d = Σ||STVS(●) – BTF(●)||
d = Σ||STVS(●) – BTF(●)||
d = Σ||STVS(●) – BTF(●)||
d = Σ||STVS(●) – BTF(●)||
distance_min

Figure 10 Illustration of searching the most similar reconstructed point (green points) to replace hidden surface point (red point), orange window is the searching region.

2.3.2 Refinement of Hidden Surface Map

The idea of the refinement process is very simple. The method is to minimize the color difference between the reconstructed point color and the hidden point color. The reconstructed color is generated by the function STVS which is defined in section 2.2. The distance function is as follow:

$$\begin{aligned} &\mathit{dis}\text{tan}\mathit{ce}(x,V) \\ &= \sum \left\| BTF(x, L_i, V) - STVS(x + motionVector(\mathrm{x}), L_i, V) \right\|, \\ &L_i \in \text{all lighting directions} \end{aligned}$$

If this error is below the threshold, then this hidden surface point can be elevated to a non-hidden one and its motion vector will direct to the reconstructed surface.

The reconstructed point is searched in a window relative to the hidden surface point as shown in **Figure 10**. The hidden surface point is replaced by the green point which has the minimum distance, $distance_{min}$. In our experiment, the window size is 256×256.

As of the threshold for elevating a hidden surface point to non-hidden one, instead of determining by the user himself, we present a reasonable and automatic system. As the result of vote-based optical flow, we distinguish hidden surface points and non-hidden ones roughly. We assume the non-hidden parts that the vote-based optical flow found are correct. So, we can find the distances of all non-hidden surface points at the target view by the distance function which is defined in subsection 2.3.2, and the *threshold* will be the average distance of them as follow:

$$threshold = average(distance(x, V_{target})),\ x \text{ is all non-hidden surface points}$$

If a hidden surface point matches some reconstructed surface point where the $\text{distance}_{\text{min}}$ is lower than the threshold, then it means that this hidden surface point can match a surface point which has more similar behaviors than non-hidden surface points.

So it is perfectly reasonable to be considered a non-hidden surface point. In the contrary, if $\text{distance}_{\text{min}}$ is above the threshold, it can be determined as a hidden surface point confidently. **Figure 11** show the result of the refinement process. The four elevated squares on the surface are all determined as non-hidden surface parts after the refinement process. **Table 1** demonstrates the refinement process can reduce a lot of the hidden surface points.

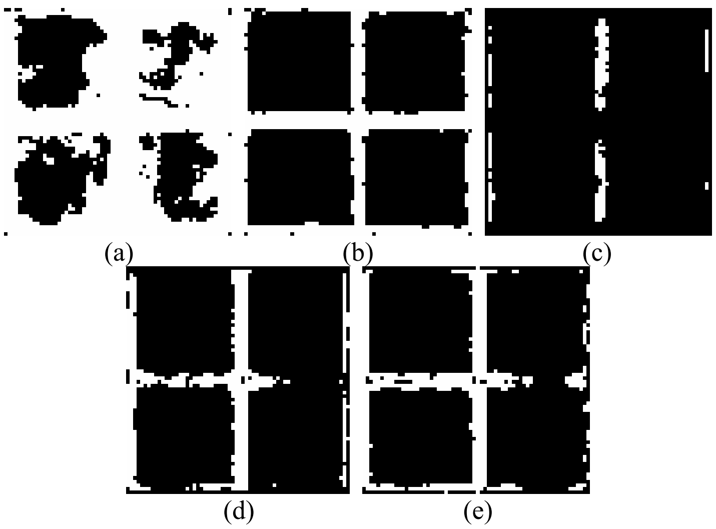


Figure 11 After hidden surface map refinement. And hidden surface maps at different depressive angle. (a) is before refinement process, (b) is after refinement process (c) is 30°, (d) is 45°,(e) is 60°,and horizontal angle is 0°

	Before Refinement		After Refinement	
	Non-hidden Surface Points	Hidden Surface Points	Non-hidden Surface Points	Hidden Surface Points
IMPALLA	209542	122234	274269	57507

Table 1 The number of hidden surface points and non-hidden surface points before and after the hidden surface map refinement. Test cases' resolutions are 64 x 64 and number of simple view directions is 81, so total surface points are 64 x 64 x 81= 331776

2.3.3 Hidden Surface Points Clustering

Each non-hidden surface point can be represented as STVS(x, L, V) which is mentioned in section 2.2, but all hidden surface points cannot. So those points need another representation. The most intuitive and simplest representation method is to store them all and extract them while rendering. But as shown in **Table 1**, the number of hidden points is about one-sixth of whole BTF size, it is impossible to do so. Fortunately, the surface or BTF is consisted of similar patches that mean hidden points are also similar, so instead of saving them all, we save some presentational points which is chosen by using a clustering approach.

The clustering method we choose is *k*-means cluster. *K*-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume *k* clusters) fixed a priori. The main idea is to define *k* centroids, one for each cluster. These centroids should be placed in a clever way because of different location causes different result. So, a better choice is to place them as far away as possible from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early grouping is done. At this point we need to re-calculate *k* new centroids as barycenters of the clusters resulting from the previous step. After we have these *k* new centroids, a new binding has to be done between the same data set points and the nearest new centroid. The same iteration continues until it converges. Finally, we can use these *k* centroids or *k* data points which are closest to individual centroids to represent whole data set.

The total number of centroids *k* we choose is 1984 (this will be explained in section 3.1). Because the number of hidden surface points is different at different view as shown in **Figure 11**, and so are their behaviors, we apply *k*-means cluster to each view and the centroids *k*' of each view is shared from *k* according to the proportion of their hidden surface points ratio. So the final reconstruction method is as follow:

$$
\begin{aligned}
&BTF(x,L,V)\\
&\cong \begin{cases}
SBRDF(x',L,V)\times ShadowMap(x',L)\\
\qquad \text{where } x' = x + motionVector(x,V),\\
\qquad \text{if } x \text{ is not hidden surface point}\\
HiddenSurfacePointsCluster(x,L,C)\\
\qquad \text{where } C \text{ is the center } x \text{ used},\\
\qquad \text{if } x \text{ is a hidden surface point}
\end{cases}
\end{aligned}
$$

3. RENDERING

All the work mentioned in previous sections is to make an efficient representation of BTF for real-time rendering. In this section, we will discuss how it works in real-time rendering systems with programmable graphics processors (GPU).

3.1 Converting Data to Texture

In recent years, Graphics processors(GPU) have showed how powerful they are by their parallel arithmetic ability. In order to use GPU, the fitting results must be converted to the textures which GPUs support. These fitting results include SBRDF parameters, top view shadow maps, occlusion shifted maps, hidden surface maps, and cluster centers.

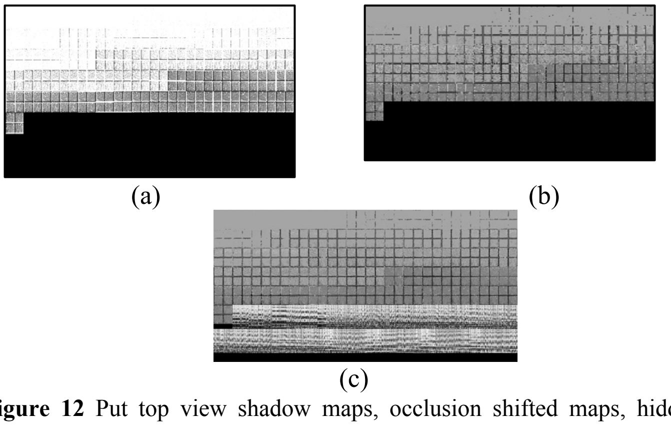
SBRDF parameters can be converted to the texture form easily, because the sample surface is a rectangle of a two-to-the-power size. So each SBRDF parameters can be stored into 32-bit floating-point textures, where the texture size equals the surface resolution. All SBRDF parameters contain a diffuse parameter ρ_d, four parameters C_x, C_y, C_z for each specular lobe, and three for RGB colors C_x, C_y, C_z, n. The modern GPUs can handle four channel floating texture at once, so we can store C_x, C_y, C_z, σ_d in one 128-bit texture and use another one for each specular lobe. In our experiment, we use three specular lobs, so there are four 128-bit textures to express the SBRDF parameters.

Next a 32-bit texture which contain four channels is used to store the top view shadow maps, occlusion shifted maps, hidden surface maps, and cluster centers.

First, because the number of top view shadow maps is related to the number of BTF lighting direction samples, and the texture size of each shadow map is equal to the sample surface resolution in our experiment. So we can put them from top to low and left to right to the texture by their sample lighting direction in the order of counter clockwise horizontal angle and increasing depressive angle like **Figure 12(a)**. For shadow maps is used one 8-bit channel to store it.

Second, for each sample view direction, there is a corresponding hidden surface map and an occlusion shifted map. Hidden surface map is a Boolean texture which represents whether the surface point at this view direction is hidden or not. It can be stored in one 8-bit channel. As for occlusion shifted map, It is stored in two 8-bit channels. If the surface point is non-hidden, one of these two channels stores the *x* shift and the other stores the *y* shift. If the surface point is hidden, we use these two channels to encode the cluster center index in our implementation. So in our experiment, the maximum number of cluster center index is 256 × 256 = 65536. Fortunately, the number of sample lighting direction and sample viewing direction is just the same, so those maps can be put into texture just like top view shadow map as shown in **Figure 12 (b).**

Because the texture size which GPU supports is 2 to the power, there may have some unused texture spaces. **Figure 12 (b)** show the unused area as black. In order to achieve maximum utility of this texture, we use these free spaces to store the cluster centers. In our experiment, the resolution of the sample surface is 64 x 64 and number of sampling directions is 81. The texture which is used to store these data is 1024 × 512. So 80 maps take up 1024 × 320 texture spaces. The remainder texture spaces are 1024 × 192. The cluster center is a 1 × 81 vector, so the number of cluster centers which can be filled in is (1024 × floor(192/81)) = 2048. In this case, because there is one shadow/occlusion shifted/hidden surface map, the valid space for cluster should take off the space which used by that map. So the maximum number of center should be 2048 - 64 = 1984.


(a) (b)

(c)

Figure 12 Put top view shadow maps, occlusion shifted maps, hidden surface maps, and cluster centers into a four channel 32-bit texture.

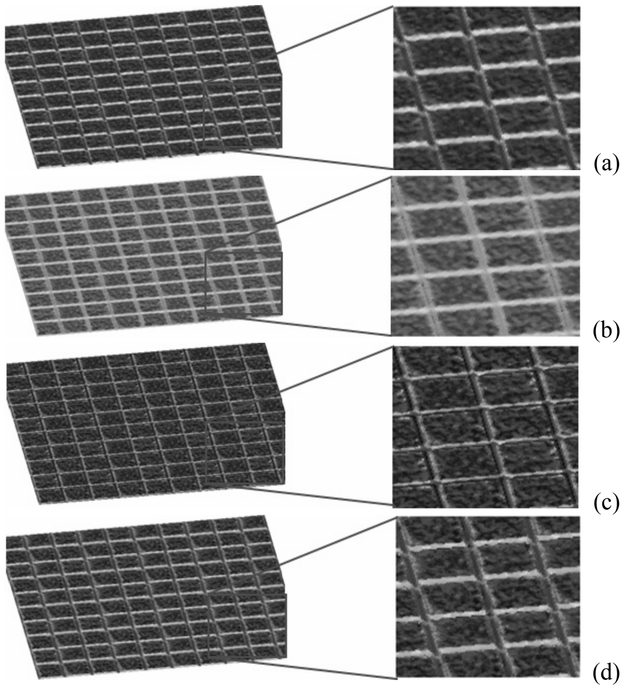
4. RESULTS AND COMPARISON

Our experiments are performed on a PC containing a 2.8GHz Pentium4 CPU and an nVidia GeForce 6600 graphics card. The output image size is 300×300. We achieve real time rendering of the BTF data at above 24 frames per second (FPS) and the compression ratios are shown in **Table 2**.

Figure 13 and **Figure 14** show the rendering results. Compared with the original BTF (**Figure 13 (a)**), our rendering results (**Figure 13 (d)**) present more surface detail than both SBRDF and STVS. Those textures can also be applied to any 3D model which only has rough geometry to represent more complex mesostructure, relighting, and render them in real-time. **Figure 14** is the results of applying our method on several 3D models with different lighting and viewing directions.

	BTF	SBRDF	STVS	Our method
Storage Size (MB)	80.621	0.596	1.008	2.58
Compression Ratio	0%	99.26%	98.75%	96.80%

Table 2 Compression ratio of all methods in this paper.



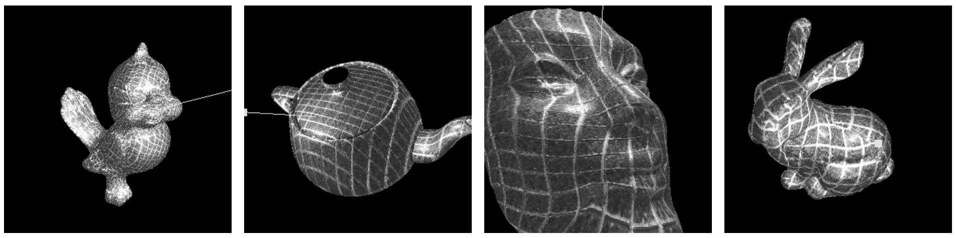


Figure 13 Rendering results and comparisons. (a) is rendered by original BTF. (b) is by SBRDF. (c) is STVS, (d) is our final method.

Figure 14 the results of applying our method to several 3D models at different lighting directions.

5. CONCLUSION AND FUTURE WORK

In this paper we present a physically-based analysis of BTF data. This analysis is based on the traditional SBRDF method first represented by McAllister [11]. We find that the traditional SBRDF cannot represent the self-shadowing and self-occlusion effects very well because they assume each sample points on the surface has an individual BRDF and fit it into a reflectance model. As we know that all reflectance models are smooth curve functions. Both self-shadowing and self- occlusion will make the shape of BRDF vary sharply, so they cannot be reconstructed very well by SBRDF.

One possible future improvement is to find the outliers more correctly. Currently, we only use the color differenc to find them. And the occlusion effects can be detected by different methods to find a more efficient and reliable shifted maps and hidden surface maps and to make the rendering result more appealing.

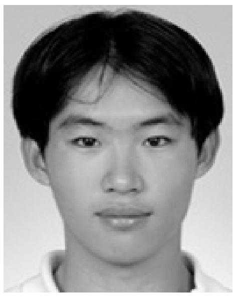
REFERENCES

1. BTF Database of University of Bonn. http://btf.cs.uni-bonn.de
2. Buehler C., Bosse M., McMillan L., Gortler S, and Cohen M. "Unstructured Lumigraph Rendering". In *Proceedings of ACM SIGGRAPH 2001*. pages 425-432
3. Chen W. C., Bouguet J. Y., Chu M. H., and Grzeszczuk R. "Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields". In *Proceedings of ACM SIGGRAPH 2002*, pages 447–456.
4. Dana K. J., Ginneken B. V., Nayar S. K., and Koenderink J. J. "Reflectance and Texture of Real-World Surfaces". In *Proceeding of ACM Transactions on Graphics (TOG)*, January 1999, Volume 18 Issue 1, pages 1-34.
5. Debevec P., Yu Y., and Borshukov G. D. "Efficient View-Dependent Image-Based

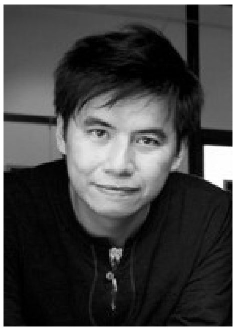
Rendering with Projective Texture-Mapping". In *Proceeding of Eurographics Rendering Workshop 1998*, pages 105-116

6. Gortler S., Grzeszczuk R., Szeliski R., and Cohen M. "The Lumigraph". In *Proceedings of ACMSIGGRAPH 1996*, pages 43–54.
7. Jensen H. W., Marschner S. R., Levoy M., and Hanrahan P. „A Practical Model for Subsurface Light Transport". In *Proceedings of ACM SIGGRAPH 2001*, pages 511-518
8. Lafortune E. P. F., Foo S.C., Torrance K.E., and GreenBerg D.P. "Non-linear Approximation of Reflectance Functions". In *Proceeding of ACM SIGGRAPH 1997*, pages 117-126.
9. Levoy M., and Hanrahan P. "Light field rendering". In *Proceedings ACM SIGGRAPH 1996*, pages 31–42.
10. Wan-Chun Ma, Sung-Hsiang Chao, Bing-Yu Chen, Chun-Fa Chang, Ming Ouhyoung, and Tomoyuki Nishita. "An Efficient Representation of Complex Materials for Real-Time Rendering". In *Proceedings of ACM Symposium on Virtual Reality Software and Technology (VRST) 2004*
11. McAllister D., Lastra A., Heidrich W. "Efficient Rendering of Spatial Bi-directional Reflectance Distribution Functions". In *Proceeding of Graphics Hardware 2002, Eurographics/SIGGRAPH Workshop Proceedings*.
12. Muller G., Meseth J., Sattler M., Sarlette R.,and Klein R. "Acquisition, Synthesis and Rendering of Bidirectional Texture Functions". In *Proceeding of Eurographics State of The Art Reports 2004*, pages 69-94
13. Alan Watt. 3D Computer Graphics (3rd Edition). ISBN: 0-201-39855-9, Published by Addison-Wesley
14. Wood D. N., Azuma D.I., Aldinger K., Curless B., Duchamp T., Salesin D.H., and Stuetzle W. Surface light fields for 3D photography. In *Proceedings of ACM SIGGRAPH 2000* , pages 287–296.

Ying-Chieh Chen (陳英傑) received his M.S. and B.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2005 and 2003. He is currently a Ph.D student of Computer Science Department, National Tsing Hua University, Hsinchu, Taiwan. His research interests in computer graphics include image based rendering and modeling, and photo-realistic image synthesis.



Sin-Jhen Chiu (邱信臻) received the B.S. degree from National Tsing Hua University, Hsinchu, Taiwan in 2004, the M.S. degree from National Tsing Hua University, Hsinchu, Taiwan in 2006. He is currently a R&D at Digimax Corporation. His work is to develop 3D animation tools.



Hsiang-Ting Chen (陳相廷) received his B.S in 2002 and M.S in 2004 from department of computer sceience, National Tsing Hua University, Hsinchu, Taiwan. He is now a Ph.D student in the Image-Based Rendering lab in National Tsing Hua University. His research interests lie in computer graphics and computer vision, especially texture synthesis, BTF rendering and augmented reality.

Chun-Fa Chang (張鈞法) received the B.S. degree from National Taiwan University, Taipei, Taiwan, in 1988, the M.S. from the Univeristy of Texas at Austin, U.S.A. in 1992, and the Ph.D. from the University of Noth Carolina at Chapel Hill, U.S.A. in 2001. He was a software engineer at Intel Corporation from 1992 to 1995. He is currently an assistant professor in the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. His research interests in computer graphics include 3D photography, image-based modeling and rendering, and photo-realistic image synthesis.