Radiance Transfer Biclustering for Real-Time All-Frequency Biscale Rendering

Xin Sun, Qiming Hou, Zhong Ren, Kun Zhou, and Baining Guo, *Fellow*, *IEEE*

Abstract—We present a real-time algorithm to render all-frequency radiance transfer at both macroscale and mesoscale. At a mesoscale, the shading is computed on a per-pixel basis by integrating the product of the local incident radiance and a bidirectional texture function. While at a macroscale, the precomputed transfer matrix, which transfers the global incident radiance to the local incident radiance at each vertex, is losslessly compressed by a novel biclustering technique. The biclustering is directly applied on the radiance transfer represented in a pixel basis, on which the BTF is naturally defined. It exploits the coherence in the transfer matrix and a property of matrix element values to reduce both storage and runtime computation cost. Our new algorithm renders at real-time frame rates realistic materials and shadows under all-frequency direct environment lighting. Comparisons show that our algorithm is able to generate images that compare favorably with reference ray tracing results, and has obvious advantages over alternative methods in storage and preprocessing time.

Index Terms—Illumination, rendering, shadow algorithm, graphics hardware.

1 Introduction

Synthesizing realistic images requires faithful simulation of the interactions of light and matter at all relevant scales, which is very challenging due to the large representation gaps between these scales. Many existing rendering techniques are limited to effects at a single scale.

Based on *precomupted radiance transfer* [2], or PRT, Sloan et al. propose biscale radiance transfer [3] to model radiance transfer at two scales. The main idea is to decompose the radiance transfer function into a global part, which is represented in a macroscale and coarsely sampled at each vertex, and a local part, which is represented in a mesoscale, densely sampled at each pixel and mapped over the object. The biscale transfer is then combined at run time to render self-shadowing and interreflection effects at both scales. Due to the low-order spherical harmonics (SH) used to represent lighting and transfer, the method is limited to very blurry shadows and matte materials. Sometimes it is not sufficient to reveal the fine details of the object surface and the relationship between the lighting and different parts of the object, as Fig. 1a shows.

- *X. Sun is with the State Key Laboratory of CAD&CG, Zhejiang University, and with Microsoft Research Asia, 5081, Sigma Building, Zhichun Road 49, Haidian District, Beijing 100190, China. E-mail: sunxin@microsoft.com.*
- *Q. Hou is with the Tsinghua University, 5036, Sigma Building, Zhichun Road 49, Haidian District, Beijing 100190, China. E-mail: hqm03ster@gmail.com.*
- *Z. Ren is with the Microsoft Research Asia, 5084, Sigma Building, Zhichun Road 49, Haidian District, Beijing 100190, China. E-mail: renzhong@microsoft.com.*
- *K. Zhou is with the State Key Laboratory of CAD&CG, Zhejiang University, ZiJinGang Campus, HangZhou 310058, China. E-mail: kunzhou@acm.org.*
- *B. Guo is with the Microsoft Research Asia, and with the Tsinghua University, 5142, Sigma Building, Zhichun Road 49, Haidian District, Beijing 100190, China. E-mail: bainguo@microsoft.com.*

Manuscript received 12 Feb. 2009; revised 6 July 2009; accepted 12 Nov. 2009; published online 7 Apr. 2010.
Recommended for acceptance by K. Bala.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2009-02-0028.
Digital Object Identifier no. 10.1109/TVCG.2010.58.

PRT has also been extended to all-frequency illumination, by representing the lighting and transfer with wavelets [4], [5], [6], [7], or spherical radial basis functions [8]. These techniques, however, compute per-vertex shading and would be impractically expensive if the fine details such as in Fig. 1c are to be rendered.

We propose a real-time algorithm for all-frequency biscale radiance transfer rendering. The algorithm renders hard and soft shadows at both macroscale and mesoscale (Fig. 1b), and supports arbitrary materials.

The main challenge of all-frequency biscale radiance transfer is the precomputation and storage of the macroscale transfer function, which is defined in a 6D space formed by global incident direction, local incident direction, and surface position. To capture all-frequency occlusion and reflection effects, each dimension has to be sampled at a reasonable rate. This makes the transfer function very costly to be precomputed, stored or manipulated at runtime.

We represent the light and transfer in a pixel basis, and compress the macroscale transfer matrix by a novel biclustering technique, which reduces both the storage and runtime computation cost and enables real-time rendering of all-frequency biscale radiance transfer.

A pixel basis is poor in representing smooth functions, as pointed out by Ng et al. [4]. In our case, however, the transfer function is not a smooth one, because the reflectance term is decoupled from the macroscale transfer function, and the transfer function is determined by visibility, rotation and down-sampling. As a result, the the transfer function is a coarsely discretized spherical function that tends to form blocks (Section 4.1). We exploit this property in biclustering (Sections 4.2 and 4.3) to losslessly compress the transfer function.

Compared with existing transfer matrix compression techniques such as clustered principal component analysis(CPCA) [1], [6], biclustering takes advantage of specific properties of the transfer matrix, and it also better adapts to different levels of available coherence. As a result, a more efficient compression is achieved, as will be demonstrated in Section 6.

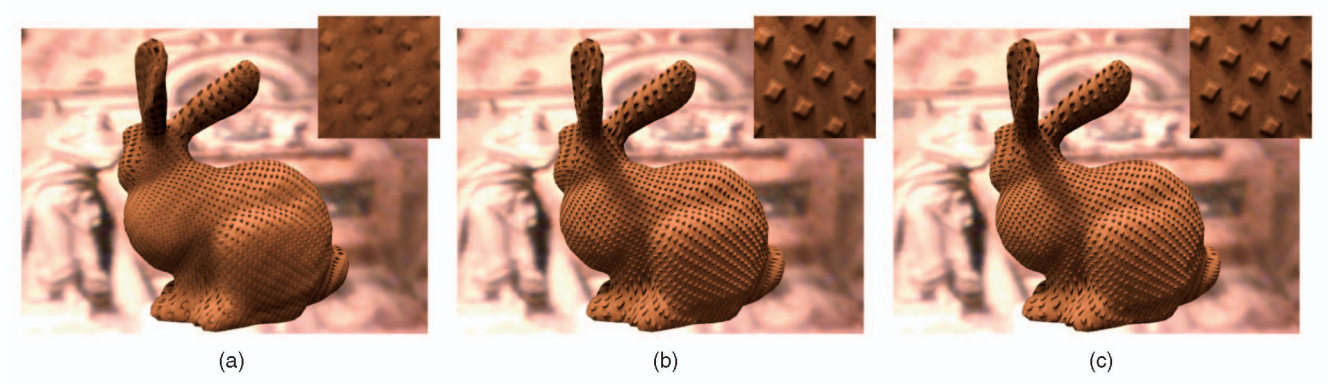


Fig. 1. A Stanford bunny with a bumpy surface illuminated by environment lighting. Note how our all-frequency algorithm more faithfully captures the shadowing effects at both global and local scales, as compared with the low-frequency biscale algorithm [1]. (a) SH [1]. (b) Biclustering. (c) Reference.

2 Related Work

Local Effects: In computer graphics, local lighting effects had been represented by textures, bidirectional reflectance distribution functions(BRDFs), height fields [9], or bump maps. The bidirectional texture function (BTF) [10], which is a 6D function that encapsulates appearance that varies spatially and with light and view direction, is introduced to represent more general local effects. Hardware based techniques have been developed [11], [12] for full 6D BTFs rendering. Lower dimensional alternatives have also been introduced [9], [13], [14] to reduce the storage and rendering cost. BTF capturing [10], [15], synthesis [12], [16], and editing tools [17], [18] have also been proposed, and a wide variety of BTFs are available for representing different kinds of material appearance. See the survey by Müller et al. for a good overview [19].

More general representations include the shell texture function [20], view-dependent displacement map [21], and generalized displacement maps [22]. These representations, though being able to capture more general local effects such as subsurface scattering and fine-scale silhouettes, are also more expensive to store and/or render, and are not as widely used as BTFs.

Recently, Sloan et al. proposed to fit the precomputed transfer function of local features by zonal harmonics, which can then be rotated efficiently at runtime for multiplication with the global incident lighting [23], represented in spherical harmonics.

These methods focus on reflectance representation and rendering of lighting effects caused by local radiance transfer.

Wang et al. render shadows at both scales [24] by introducing a 4D mesostructure distance function (MDF) representation. It is limited to simple lighting sources like point lights, since rendering is based on shadow mapping and accumulation over complex light sources is not affordable for interactive applications.

Global Effects: Realistic rendering of global effects under complex lighting requires integration over the incident directions, which is expensive if conducted directly. Sloan et al. solve the light integration problem by an offline tabulation of an object's response to low-frequency lighting [2], represented by SH, and turn the runtime integration into a linear combination of the responses to each SH basis lighting. Exploiting the coherence among vertices, compression methods are introduced using PCA [25] or CPCA [1]. We take a different approach for both transfer representation and compression and achieve higher rendering quality at the same storage cost.

Traditional PRT has been extended to fine, repetitive local details. By combining the local and global radiance transfer [3], self-shadowing and interreflection effects at both macroscale and mesoscale can be rendered in realtime, yet only under low-frequency environment lighting. Our method extends the low-frequency biscale transfer to all-frequency, and supports both accurate shadows and complex materials.

PRT has also been extended to handle all-frequency shadows by using a nonlinear wavelet approximation [4], albeit with either fixed lighting or viewpoint. This limitation has been overcome by factoring the transfer function into a visibility term and a reflectance term and computing at runtime a wavelet triple product of lighting, visibility, and reflectance to yield the outgoing radiance [5]. A similar approach can be difficult to apply to our transfer matrix, since unlike the reflectance function, the rotation from a global coordinate frame to a local coordinate frame for each vertex is defined in 6D space, and is still too expensive to precompute.

Another dimensionality reduction technique is based on decomposition of the BRDF, which has been used in many all-frequency PRT algorithms [6], [7], [8], [26], [27]. It is not clear how this technique could be used in our case, since the reflectance function has already been decomposed to mesoscale and the rotation term in the transfer function depends on the surface location, making the decomposition infeasible.

Xu et al. [28] exploit the coherence in directions of both lighting, visibility, and BRDF, corresponding to the coherence in the rows of our transfer matrix, and enables object rotation in all frequency rendering of dynamic scenes and on-the-fly BRDF editing. Our method exploits coherence in both rows and columns and can tackle transfer matrix of higher dimensionality, which enables biscale radiance transfer.

Garg et al. [29] leverage the symmetry and data-sparseness of the transfer matrix and represent it using hierarchical tensors. The fourth-order tensor representing the light transport is recursively divided into 16 children until the node can be represented using rank-1 approximation. The nodes in that hierarchy are subblocks formed by neighboring rows and columns. The characteristic of the transfer matrix in our case is different from theirs, which will be shown in Section 4.1, due to the asymmetry of the input and output of the transfer. Thus we take a different approach to compress the transfer matrix.

Recently, an all-frequency shadow algorithm for dynamic scenes has been proposed [30]. Global incident radiance is approximated by a small number of area light sources, and extended convolution shadow maps [31] are used to render the shadows generated by each of these light sources. No precompuatation is needed for the visibility and dynamic scene objects can be supported. The algorithm, however, focuses on shadow rendering instead of local reflectance. Integration of the BRDF across the light source domain is not supported, and the BRDF in the direction of the center of each area light is evaluated to weight the contribution. It is not clear if complex materials such as shown in this paper can be supported.

Integration of mesoscale surface details into the PRT framework has recently been shown possible [32] by exploiting a clustered piecewise constant representation for the visibility and lighting. Precomputation of this representation for the 4D BRDF, however, is already prohibitively expensive so that it is evaluated dynamically at the cluster centers. In our approach, the global radiance transfer matrix are compressed instead of approximated, and the local response encoded in the 6D BTF is precisely rendered.

Biclustering: Biclustering is a technique which has been extensively used in biological data analysis to find submatrices where the genes exhibit highly correlated activities for every condition, For more details, see a recent survey by Madeira and Oliveira [33]. We apply the idea to transfer matrix compression and develop a novel algorithm for bicluster construction and real-time rendering.

3 Overview and Terminology

We use math italics for scalars, 3d points or vectors (e.g., x, p), boldface italics for higher dimensional vectors (e.g., $\boldsymbol{L}$), and sans serif for matrices or biclusters (e.g., T, B). A submatrix is noted as $\mathrm{T}[j_1, \ldots, j_r; k_1, \ldots, k_l]$, where $j_1, \ldots, j_r$ are the row indices and $k_1, \ldots, k_l$ are the column indices.

As in low-frequency biscale rendering [3], we decouple the coarse (PRT) and fine (BTF) radiance transfer in precomputation and combine the transfer at both scales at runtime. In a preprocess, we compute and store the matrix that transfers the source lighting, represented in the global coordinate frame, into the local frame of each vertex. Both visibility and rotation are incorporated into the transfer matrix, and macroscale shadowing effects are captured.

As depicted in Fig. 2, the global incident radiance function $\boldsymbol{L}$ is transferred to each vertex by a multiplication with the transfer matrix:

$$\boldsymbol{L}_p^* = \mathrm{T}_p \boldsymbol{L}. \tag{1}$$

We use the vector form (such as $\boldsymbol{L}$) to indicate a spherical function (such as $L(\omega_i)$) represented in any basis defined on the sphere. And "*" is used to emphasize that $\boldsymbol{L}_p^*$ is defined in the local coordinate frame of p. T_p is the transfer matrix sampled at p, which will be explained in more detail below.

It is sometimes more convenient to use the compact form of (1), which simply packs the $\boldsymbol{L}_p^*$ vectors for all vertices in the scene into a single vector $\boldsymbol{L}^*$:

$$\boldsymbol{L}^* = \mathrm{T}\boldsymbol{L}. \tag{2}$$

The mesoscale transfer is computed per pixel to capture the finer variations of the mesostructures on object's surface, as shown in Fig. 2. More precisely, for each pixel x, shading is computed as

$$B(x, e) = \sum_d b(u, e, d) L^*(x, d) = \boldsymbol{b}(u, e) \cdot \boldsymbol{L}^*(x), \tag{3}$$

where $B(x, e)$ is the outgoing radiance, b is the 6D BTF, u is a 2D texture coordinate and e is the view direction, d is the direction of the incident radiance. Note that e and d are both defined in the local coordinate frame of the point corresponding to x. $L^*(x, d)$ is the local incident radiance at pixel x. The last part of (3) is a reformulation in vector form. The per-pixel local incident radiance vector $\boldsymbol{L}^*(x)$ can be obtained by interpolating $\boldsymbol{L}_p^*$ defined at each vertex.

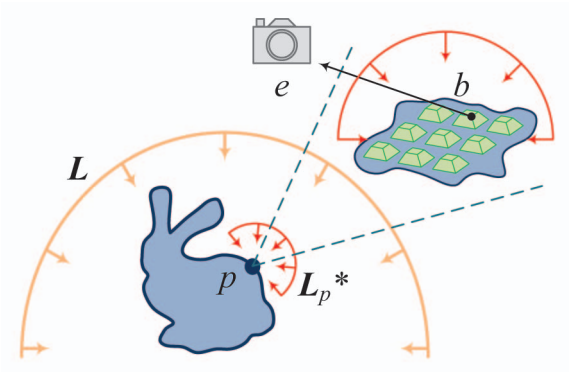


Fig. 2. Transfer the global incident radiance L to the local coordinate frame of p, yielding the local incident radiance L_p^*. Per-pixel shading is then computed by integrating the product of BTF and L_p^*.

We choose the pixel basis to represent both lighting and transfer. In other words, each component of $\boldsymbol{L}$ is simply the sampled incident radiance value of the corresponding pixel on the direction cubemap.

Note that (3) requires $L^*(x, d)$ to be sampled at the same set of directions as $b(u, e, d)$. Assuming the BTF is sampled at M incident radiance directions, $\boldsymbol{L}^*(x)$ is M-dimensional, and so is $\boldsymbol{L}_p^*$. The light source $L(\omega_i)$, on the other hand, is often discretized in a higher resolution (N sample directions, $N > M$). The vertex transfer matrix T_p is a $M \times N$ transfer matrix, with its element (j, k) representing the contribution of the global incident radiance at direction k to the local incident radiance at direction j. For a scene with n vertices, the T is $Mn \times N$.

The transfer matrix T is very large, due to the high-dimensional nature of the transfer function. For example, for a moderate scene configuration with 15k vertices, $6 \times 32 \times 32$ source lighting directions and 8×8 BTF lighting sample directions, T has nearly 6 billion entries. Though sparse, it is still quite cumbersome to store or relight with.

To compress T and accelerate runtime multiplication, we decompose it into a number of submatrices, in a way that minimizes the computation cost of (2). The decomposition, which will be described in detail in the next section, is conducted as a preprocess and is referred to as the biclustering of the transfer matrix.

4 Precomputation

In a preprocess, we construct the transfer matrix T_p at each vertex to obtain the overall transfer matrix T, on which a

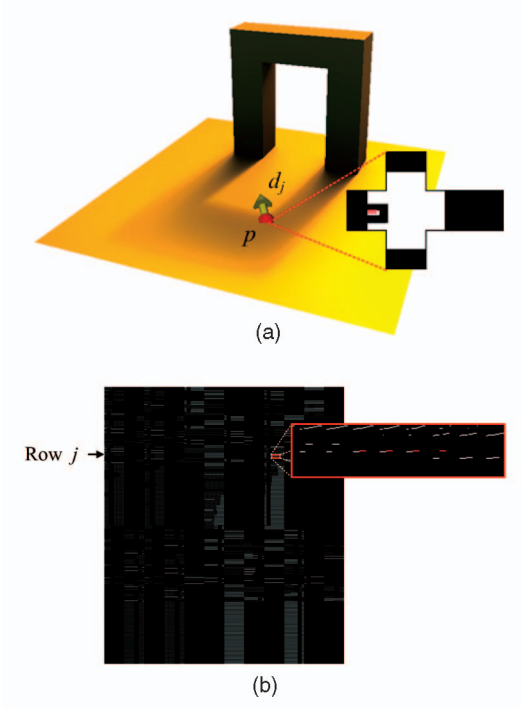


Fig. 3. An example of transfer matrix construction. A small patch of 76 vertices is used to show the process. (a) A small patch and a sample vertex, together with a particular local incident direction. (b) Transfer matrix and the corresponding row.

greedy iterative search is performed to yield a compact bicluster representation.

4.1 Transfer Matrix Construction

For a given local incident direction d_j at vertex p, the transfer matrix element $(\mathrm{T}_p)_{jk}$ represents the contribution of the global incident direction d_k. In other words, the element indicates how much of the global incident radiance at d_k would reach the local incident radiance direction d_j.

The construction of the transfer matrix is depicted in Fig. 3. We rasterize a high resolution visibility hemicube at each vertex p, and down-sample it to the same resolution as the environment lighting. Each pixel of the cubemap corresponds to an incident radiance direction. These directions are iterated to evaluate their contribution to each local incident radiance direction d_j.

At each incident direction d_k, we first check the visibility, and totally blocked directions are bypassed. We find the nearest local incident radiance direction d_j and set $(\mathrm{T}_p)_{jk}$ to the visibility value at d_k.

Note that the local incident radiance directions d_j are defined in p's local coordinate frame and to find the nearest d_j for global incident radiance direction d_k we need to rotate all the local incident radiance directions to the global coordinate frame.

Fig. 3a shows a particular local incident radiance direction d_j at vertex p with the contributing cubemap directions shown in red. These contributing directions correspond to elements in the jth row of the transfer matrix. After all vertices are processed, we pack the transfer matrices T_p to obtain the overall transfer matrix. We show the transfer matrix for this example in Fig. 3b, and the corresponding row for the direction in Fig. 3a is highlighted in red.

Due to the down-sampling, the visibility at each pixel can have values other than 0 and 1. Generally, if we perform $2^m \times 2^m$ down-sampling, the number of possible visibility values is $2^{2m} + 1$.

One alternative representation for the radiance transfer would be to store the down-sampled visibility vectors and the rotation matrix as reshuffling vectors. This, however, swaps the order of down-sampling and rotation, and the low sampling rate of the rotation will introduce noticeable rendering artifacts.

From the small example in Fig. 3, we see two properties of the transfer matrix. First, the transfer matrix is very sparse. Actually, only 0.69 percent of the elements are nonzero. This can be easily understood: for a given local incident radiance direction, only a small number of directions on the source lighting cubemap may contribute. The sparsity alone, however, still does not make the transfer matrix tractable, since there are still tens of millions of entries to be stored and multiplied with the incident radiance at runtime.

Second, we see regularity in the distribution of the nonzero entries—they tends to form blocks. This regularity is due to the structure in visibility and the distribution of normal directions of vertices. Note that unlike in the transfer matrix between symmetric incident and exitant light field, where the matrix entries tend to form continuous subblocks that can be well approximated using rank-1 factors [29], the "blocks" here might be discontinuous, since the transfer is between the incident 2D environment lighting and exitant 4D surface light field and we do not take interreflections into account. This kind of regularity inspires us to use a biclustering technique to exploit the coherence in arbitrary rows and columns. Unlike CPCA, which exploits the coherence between elements of fixed size (the rows), we use biclusters, whose size can adapt to the available coherence. This, joined with exploiting the limited number of possible matrix entries, makes it possible for us to develop a technique that provides higher compression rates than existing techniques such as CPCA and wavelet.

4.2 Bicluster

The most straightforward way to exploit the coherence in the transfer matrix would be to extract submatrices having constant entries. This simple clustering approach however tends to yield a large number of very small clusters and we instead seek a more general representation for the coherence, and we found the biclustering technique that is extensively used in biological data analysis is very suitable for our purpose.

We base our algorithm on a very simple observation. Suppose we have a submatrix of T formed by selecting the $j_1, j_2, \ldots, j_r$ rows and the $k_1, k_2, \ldots, k_l$ columns from T, noted as $\mathrm{T}[j_1, j_2, \ldots, j_r; k_1, k_2, \ldots, k_l]$. If each column of the submatrix is composed of elements of a constant value:

$$\mathrm{T}_{j_1 k_m} = \mathrm{T}_{j_2 k_m} = \cdots = \mathrm{T}_{j_r k_m} \triangleq t_m, \quad \forall m \in \{1, 2, \ldots, l\} \tag{4}$$

then we have

$$\begin{aligned} &\mathrm{T}[j_1, \ldots, j_r; k_1, \ldots, k_l]\boldsymbol{L}[k_1, \ldots, k_l] = [\overbrace{a \ldots a}^{r \text{ scalars}}]^T, \\ &a = t_1 L_{k_1} + t_2 L_{k_2} + \cdots + t_l L_{k_l}. \end{aligned} \tag{5}$$

The submatrix that satisfies (4) can be represented as a *constant column bicluster*, or simply *bicluster*. And we call such a submatrix a *bicluster submatrix*. A bicluster is determined by the row indices and the column indices of the submatrix, plus the values of each column $t_1, t_2, \ldots, t_l$.

Given a bicluster $\mathrm{B}[j_1, \ldots, j_r; k_1, \ldots, k_l; t_1, \ldots, t_l]$ belongs to a $Mn \times N$ transfer matrix, we can define its multiplication with a N vector $\boldsymbol{L}$

$$(\mathrm{B}\boldsymbol{L})_j = \begin{cases} \sum_{m=1}^{l} t_m L_{k_m} & \text{if } j \in \{j_1, j_2, \ldots, j_r\} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$
$$j = 1, 2, \ldots, Mn,$$

yielding a $M \times n$ vector. Note we only need l floating point multiplications and $l-1$ additions, as well as r additions to accumulate the resulting vector of (6). We define the *cost* of a bicluster as

$$W(\mathrm{B}) = r + 2l - 1, \quad (7)$$

reflecting the number of FLOPs involved in the bicluster multiplication with B.

By decomposing the transfer matrix into bicluster submatrices, we can obtain its *bicluster representation*. Notice that an 1×1 submatrix of T can be represented by a smallest bicluster described by three scalars. This guarantees the existence of a bicluster representation for any given T. What we need though is an optimal biclustering of the transfer matrix that minimizes the runtime computation cost.

4.3 Transfer Matrix Biclustering

Given the transfer matrix T, the biclustering of T is a set of biclusters B_i that satisfies

$$\mathrm{T}\boldsymbol{L} \equiv \sum_i \mathrm{B}_i \boldsymbol{L}, \forall \boldsymbol{L}. \quad (8)$$

We want to find the biclustering that minimizes $\sum_i W(\mathrm{B}_i)$.

It has been shown that such a biclustering of a matrix is NP-complete [33]. In this section, we describe a greedy iterative algorithm for transfer matrix biclustering.

Qualitatively, when possible, we want to look for larger biclusters, which provide higher reduction in computation and storage. However, looking for submatrices that strictly satisfy (4) tends to yield very small biclusters, weakening the effectiveness of biclustering. An important observation is that we can decompose any submatrix into a bicluster submatrix and a residual submatrix. Below is an example.

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{3}{4} & 1 \\ 1 & \frac{1}{2} & \frac{3}{4} & 1 \\ 1 & \frac{1}{2} & \frac{3}{4} & 0 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{2} & \frac{3}{4} & 1 \\ 1 & \frac{1}{2} & \frac{3}{4} & 1 \\ 1 & \frac{1}{2} & \frac{3}{4} & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

This allows us to find a larger bicluster submatrix, paying the price that the additional nonzero elements in the residual submatrix should be handled in later iterations. But that is a reasonable trade-off as long as we can keep the number of nonzero elements in the residual submatrix small.

We start from a randomly chosen element, which is a 1×1 bicluster submatrix itself, then iteratively try to insert (delete) rows or columns into (from) the current bicluster submatrix. Every insert or delete operation produces a new bicluster submatrix that changes the total computation required by the matrix-vector multiplication (8). We evaluate the changes and select the operation that reduces the computation most.

Operations are performed iteratively to improve the current bicluster submatrix until no more computation reduction is possible. As a bicluster is produced, the corresponding submatrix is subtracted from the transfer matrix. The algorithm terminates when all the remaining elements in the transfer matrix become zeroes.

The most important issue is how to evaluate the computational change that is related with a particular submatrix and its bicluster representation. For a given $r \times l$ submatrix M and its bicluster representation B, the floating point operations related to the submatrix is $2\mathrm{nnz}(\mathrm{M})$, where the $\mathrm{nnz}(\mathrm{M})$ is the number of nonzero elements of M. This accounts for a per-element multiplication and addition that are involved in the matrix-vector multiplication. The operations involved in the corresponding bicluster multiplication is $W(\mathrm{B}) = r + 2l - 1$. If a residual submatrix exists, then every nonzero element in the residual submatrix M_R requires another addition and multiplication. Thus, the change in computation cost can be evaluated as

$$\Delta W(\mathrm{M}) = 2\mathrm{nnz}(\mathrm{M}) - (r + 2l - 1) - 2\mathrm{nnz}(\mathrm{M}_R) \quad (9)$$

$\Delta W(\mathrm{M})$ is a measurement of the change of the required FLOPs due to representation transfer. It can be either negative or positive.

We list the pseudo code for transfer matrix biclustering in Algorithm 1. Note that for "possible operation" in line 8, we refer to the insertion of a new row/column to the current bicluster matrix or the deletion of an existing row/column from it. For deletions, the corresponding nonzero elements in the residual submatrix are absorbed. Note that T_b is usually smaller than T_c and the matrix subtraction in line 16 means subtracting the elements of T_b from the corresponding elements of T_c.

Algorithm 1. Pseudo code for transfer matrix biclustering
Input: transfer matrix T
Output: $\{\mathrm{B}_i\}$, a biclustering of T

```
1: T_c ← T.
2: if nnz(T_c) = 0 then
3:         terminate
4: end if
5: Randomly choose a nonzero element (T_c)_jk
6: T_b ← T_c[j; k] {initialed as an 1 × 1 matrix}
7: ΔW_c ← ΔW(T_b)
8: for all possible operation f do
9:         if ΔW(f(T_b)) > ΔW_c then
10:            ΔW_c ← ΔW(f(T_b))
11:            f_c ← f
12:        end if
13: end for
14: if ΔW_c = ΔW(T_b) then
15:        Output T_b's corresponding bicluster
16:        T_c ← T_c − T_b
17:        go to 2
18: else
19:        T_b ← f_c(T_b)
20:        go to 7
21: end if
```

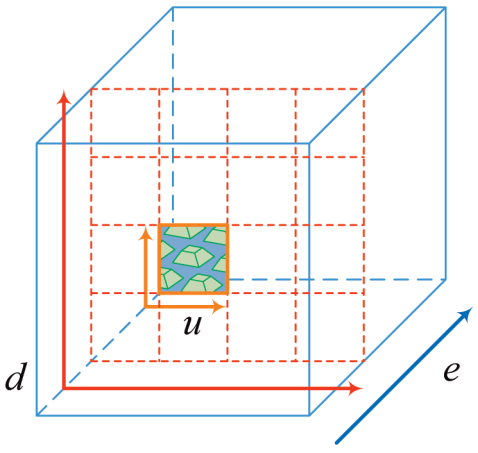


Fig. 4. Packing a 6D BTF into a 3D RGBA texture.

5 Rendering

Rendering is divided into per vertex computation and per pixel computation, corresponding to macroscale transfer and mesoscale transfer, respectively.

We take the cubemap representation of the environment lighting as the input light vector $\boldsymbol{L}$, perform the bicluster multiplications and accumulate the results to calculate $\sum_i \mathrm{B}_i \boldsymbol{L}$. Each multiplication is computed via (6). The resulting incident radiance vector, $\boldsymbol{L}^*$, is packed into the vertex buffer.

Per pixel computation is conducted on the GPU according to (3). In a pixel shader, we take the interpolated vertex data as the per-pixel incident radiance vector $\boldsymbol{L}_i^*$. We also obtain the view direction and texture coordinate, which are then used to lookup the BTF vector $\boldsymbol{b}(u, e)$. Then a simple dot product $\boldsymbol{b}(u, e) \cdot \boldsymbol{L}_i^*$ is performed to yield the shading at the corresponding pixel.

Current graphics hardware has a limitation on the number of available pixel input registers, and we cannot process all the vertex data in a single pass. Multipass rendering has to be conducted and alpha-blending is used to add up the results of each pass.

6 Implementation and Result

In this section, we will first discuss some implementation details, and then present the experimental results.

6.1 Implementation Details

6.1.1 Data Organization

It is important to guarantee that the local incident radiance directions are distributed evenly on the hemisphere. We use a low distortion area preserving parameterization [34] for the incident radiance direction and view direction hemispheres.

As depicted in Fig. 4, we pack BTFs into 3D RGBA textures, each layer of which corresponds to a sample view direction. For each layer, the data is organized with a number of blocks, each of which corresponds to a sample incident radiance direction. Thus, each of these blocks is a sample of the patch rendered under a particular incident directional light and view direction combination. The interpolation across the block uses built-in hardware, and the interpolation in the view direction is performed manually in the pixel shader. Instead of using simple bilinear interpolation, which leads to ghosting artifacts, we map the view direction back to the direction sphere and retrieve the four adjacent samples for the interpolation. No interpolation is needed for the incident radiance directions, since we directly transfer the global incident radiance to the local sample incident directions.

6.1.2 Precomputation Optimization

Biclustering is the main bottleneck of precomputation. As a simple optimization for Algorithm 1, we maintain an influence value for each row and each column, defined as the influence of the operations on the change of cost in (9). We organize all row and column operations into a priority queue according to the influence value. At every step we select the operation at the front of the operation queue. And after an operation is conducted, we update the affected operations in the operation queue, and refresh the queue accordingly.

This select-conduct-update process is more efficient than the straightforward implementation, since we do not need to iterate all the operations to select the one that reduces the cost most. This is achieved at the cost of updating the queue after each operation. Fortunately, the updating process is highly local, since each operation only affects a very small number of operations in the queue.

The above process is repeated until the operation at the front of the queue has negative influence on ΔW, when all operations in the queue cannot reduce the overall cost anymore. Note that during the whole process the size of the queue remains as $Mn + N$—the sum of column number and row number.

For example, for a new column k_0 which is not in the current submatrix $\mathrm{T}_b[j_1, j_2, \ldots, j_r; k_1, k_2, \ldots, k_l]$. The constant value t_0 is chosen to be the nonzero value that most frequently appears in the rows $j_1, j_2, \ldots, j_r$ of column k_0. Then the influence of the new column to ΔW consists of the cost of the column itself, which corresponds to the $r + 2l - 1$ term in (9), plus the sum of the influence of each of the elements in the rows $j_1, j_2, \ldots, j_r$, which correspond to the $2\mathrm{nnz}(\mathrm{T_b}) - 2\mathrm{nnz}((\mathrm{T}_b)_R)$ term. For this new column, the cost of the column is -2, since it causes l to increase by 1. To compute the influence of a particular element, three different cases need to be considered:

- if the element's value is t_0, it causes $\mathrm{nnz}(\mathrm{T_b})$ to increase by 1, and the influence is 2;
- if the element's value is 0, it causes $\mathrm{nnz}((\mathrm{T}_b)_R)$ to increase by 1, and the influence is -2;
- if the element's value is a nonzero value other than t_0, it causes both $\mathrm{nnz}(\mathrm{T_b})$ and $\mathrm{nnz}((\mathrm{T}_b)_R)$ to increase by 1, and the influence is 0.

If the insertion of this new column is conducted, first, the influence value of the column itself should be changed, since the corresponding operation with the column has changed from insertion to deletion. Second, influence values of all the rows that has a nonzero element in this column, plus the rows that the residuals in the column are located should be changed, according to the above element influence evaluation.

For other operations, including the deletion of an existing column from the current submatrix and the

deletion/insertion of rows, the influence value can be derived similarly.

6.1.3 Rendering Optimization

Biclustering multiplication is mostly bounded by memory access, since the computation itself is very simple. To improve the memory access coherence, we first perform all the bicluster multiplications $\mathrm{B}_i\boldsymbol{L}$. A runtime bicluster buffer P_B is maintained, each slot of which corresponds to a bicluster. We iterate all the biclusters and calculate a single scalar $\sum_{m=1}^{l} t_m L_{k_m}$ (6) for each of them. At the end of this iteration we have the result of all required bicluster multiplications in the buffer.

Then we need to accumulate the products to appropriate local incident radiance directions. For each direction(row) j, we store the indices of all the related biclusters in I_j, which are then used to access the bicluster buffer and sum up the products. The pseudo code is listed in Algorithm 2.

Algorithm 2. Pseudo code for computing $\sum_i \mathrm{B}_i\boldsymbol{L}$
Input:
$\{\mathrm{B}_i\}$, transfer matrix biclustering
$\boldsymbol{L}$, lighting vector
Output:
$\boldsymbol{L}^*$, local incident radiance vector

```
for all bicluster index i do
    (P_B)_i ← Σ_{m=1}^{l} t_m L_{k_m} {{t_m}, {k_m} in B_i}
end for
for j = 1 to Mn do
    L*_j = 0
    for k = 1 to K_j do
        {K_j: number of biclusters for row j}
        i ← (I_j)_k {I_j: related bicluster's indices}
        L*_j ← L*_j + (P_B)_i
    end for
end for
```

We implement the above computation by CUDA [35], and pass the result to the final BTF shading by CUDA-OpenGL interoperatability routines.

Another easy optimization is to reuse the local incident radiance vector if only the view direction is changed. In the results presented in Section 6, we will list the fps numbers for lighting and view changes separately.

6.2 Results and Comparisons

6.2.1 Platform

We implemented our system on a workstation with a 3.00 GHz Quad Xeon CPU, 4 GB RAM, and a NVidia GTX 280 graphics card. We use CUDA to compute the incident radiance of all vertices and render the shading of BTF with OpenGL.

6.2.2 Comparison with Low-frequency Biscale Rendering

In Fig. 1, we compare our rendering result with low-frequency biscale rendering [3]. In the global scale, we can see a more accurate shadowing effect, especially for the bunny's ears, while in the local scale, a better perception of the bumpy bunny surface is achieved by our result, thanks to the more accurate capturing of the self-shadowing effects of the surface details.

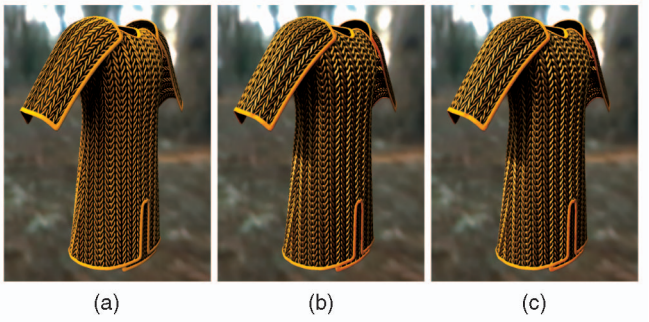


Fig. 5. Our result better captures the appearance of glossy materials than its low-frequency counterpart. (a) SH. (b) Biclustering. (c) Reference.

Fig. 5 shows the advantages of our algorithm over low-frequency biscale rendering in rendering glossy materials. The material of the chains is modeled by an anisotropic analytic representation [36], [37]. The specularity of the material is better captured in our result, producing a higher shading contrast. In addition, our algorithm better captures the thickness of the armor, thanks to the more faithful rendering of the self-shadowing effects at the mesoscale.

6.2.3 Comparisons with Alternative Methods

One important advantage of our algorithm is the storage efficiency. By exploiting coherence in both rows and columns of the transfer matrix, as well as exploiting the property that the nonzero matrix entries only evaluate to a limited number of values, our method provide much higher quality than alternative methods at comparable storage cost.

In Fig. 6, we compared the result generated by our method and CPCA [1] and nonlinear wavelet approximation [4], [5]. We choose the parameters of these alternative methods in a way that their storage costs are roughly the same as ours (36.4 MB). Severe rendering artifacts can be observed for both of these methods, as seen in Fig. 6c, 6d, while ours (Fig. 6c) is almost visually indistinguishable from the reference (Fig. 6d).

A comparison of the storage costs of these methods at the same quality would make the comparison more complete. The high cost of CPCA compression (more than 10 hours), however, made such a search of parameters infeasible.

6.2.4 Parameters and Performance

Environment source lighting is represented as a cubemap of resolution $6 \times 32 \times 32$ and the dimension of the light vector is 6,144. For visibility sampling, 2×2 down sampling is performed to yield $6 \times 32 \times 32$ cubemaps.

The resolution of incident directions of BTFs is 8×8, except for the "Armor" scene, where the resolution of 12×12 is used since the BTF used for that scene has a larger high frequency component. The resolution of viewing directions is 12×12. The patch sizes used for the BTFs of all our demos are 64×64 or 32×32. Other detailed settings of parameters and performance are included in Table 1 (Note that two performance numbers are measured for each scene, the first one is for fixed lighting and dynamic view, from which the cost of global transfer is excluded, and the second is for dynamic lighting and view). The scene named "Temple" (Fig. 7) is constructed to show our algorithm's ability to render multiple BTFs. Five BTFs are used for

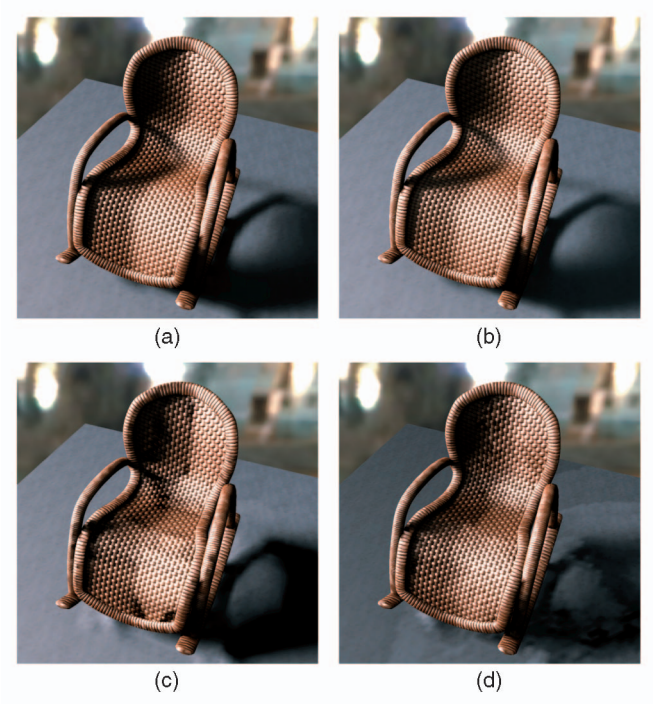


Fig. 6. Comparisons of different compression methods of global radiance transfer. The original transfer matrix is obtained by precomputing for each of the 16 K vertices a transfer matrix from the 6,144 global incident directions to the 64 BTF incident directions. For CPCA, the rows of the transfer matrix are divided into 32 clusters, and 8 eigen-vectors are generated for each cluster, yielding 36.2 MB final data. For wavelet, each row of the matrix is nonlinearly approximated by area-weighted selecting the first 12 wavelet basis lights, yielding 37.1 MB final data. (a) Biclustering. (b) Reference. (c) Clustered PCA. (d) Haar wavelet.

TABLE 1
Test Scene Statistics

Scene	Bunny	Armor	Chair	Temple
Figure	Fig. 1	Fig. 5	Fig. 6	Fig. 7
Resolution	640×480	480×640	640×640	640×480
#Vertices	3,286	10,652	16,101	32,409
#BTFs	1	1	3	5
BTFs size(MB)	113.2	255.2	169.9	226.5
$\mathrm{nnz}(\mathrm{T})$ **(M)**	10.6	21.9	40.5	62.8
MFLOPs	21.2	43.9	81.0	125.6
Reduced to	5.8	14.9	9.0	7.9
Computation%	27.4	33.9	11.1	6.3
Data size(MB)	1,400	9,500	6,300	12,600
Reduced to(MB)	24.2	59.0	36.4	32.9
Data size%	1.7	0.62	0.58	0.26
Precomp.(min)	12	46	33	59
Avg. FPS	97.0/56.1	42.8/24.9	51.8/33.7	61.1/34.3

* Statistics for the 4 test scenes used in our paper. For rendering performance, we give the average frame rates for view/lighting changes, respectively.

different parts of that scene, while the transfer matrix is defined and compressed for the entire scene.

From the numbers listed in Table 1, we can see that the efficiency of biclustering is dependent on the properties of the scene geometry. For scenes mainly consisting of "flat" geometries, such as "Temple," the coherence in the transfer matrix is very high, leading to more efficient compression. For curved geometry, as the "Bunny," the compression is less effective. The cost of transfer matrix multiplication and BTF shading are roughly comparable, and the former is dominated by the gathering phase, in which the products in the runtime bicluster buffer are accumulated to the corresponding incident radiance directions. For BTF shading, the important factors are the number of pixels to be shaded and the coherence of the BTF fetch. Denser BTF tiling leads to lower coherence between adjacent pixels, and in turn lower shading performance.

The biclustering algorithm scales well with the resolution of local incident directions, as shown in Fig. 8. The storage cost of the compressed transfer matrix at the resolution of 32×32 is 58.5 MB, which is less than 2.5 times larger than the storage at 8×8. While this is partially due to the sparsity increase of the transfer matrix, the biclustering algorithm did achieve practical storage cost at higher resolution. As for the rendering performance, the interpolation and BTF shading cost scales roughly quadratically with the local incident direction resolution, and dominates the rendering cost at higher resolution. This can be seen from the diminishing gap between the performance of fixed and moving lighting. Note that from (a) to (b) we observe superquadratic performance drop, mainly due to the large BTF size which lead to worse cache hit rate in BTF data fetches.

We have also tested the scalability with regard to scene complexity by using the same bunny model with different vertex number, and the result are shown in Table 2. We obtain steady compression ratio for different vertex numbers. The rendering performance, on the other hand, scales roughly linearly with the vertex number.

It should be noted that our technique is orthogonal to BTF compression techniques [38], [39], [40], since we focus on the compression of the global transfer matrix.

7 Conclusion and Future Work

In this paper, we address the problem of rendering all-frequency biscale radiance transfer. The main challenge is the large transfer matrix that needs to be stored and manipulated. We propose a lossless compression algorithm based on transfer matrix biclustering. We introduce a novel algorithm for finding the optimized biclustering of the transfer matrix and minimizing runtime computation cost. Our algorithm is

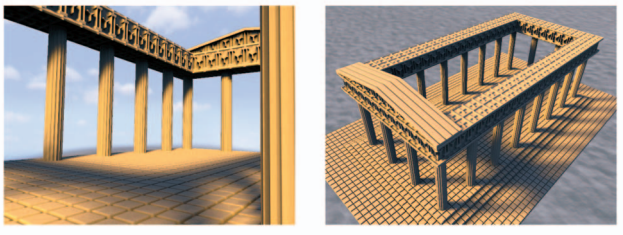


Fig. 7. A temple scene with a five different BTFs. The eaves molding uses a patch size of 64×64, while all other parts use patches of 32×32.

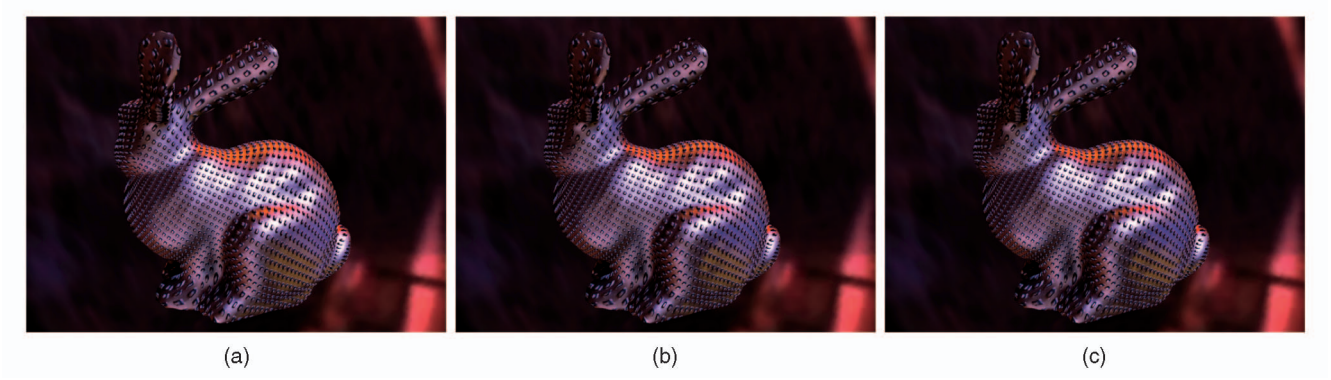


Fig. 8. Glossy bumpy Stanford bunny with different resolution of local incident direction. The material used for the bunny is a Phong model with the exponential equals to 128. As in Table 1, performance for both fixed lighting and moving lighting is shown. The transfer matrices are compressed to 24.2 MB, 38.3 MB, and 58.5 MB, at the cost of 12, 32, and 93 minutes of precomputation, respectively. The cost of computing macroscale transfer is 7.9 ms, 22.4 ms and 37.0 ms, respectively. The patch size of the BTF is 16×16. (a) 8×8 directions @ 277.9/86.8 fps. (b) 16×16 directions @ 31.3/18.4 fps. (c) 32×32 directions @ 5.4/4.5 fps.

able to reduce storage and computational complexity down to 5-30 percent, enabling real-time rendering.

For the mesoscale representation of the surface details, though we only focus on BTFs in this paper, our algorithm is not limited to BTF rendering. Other surface detail representations, such as VDM, normal map, spatially invariant or variant BRDF, can all be used. For simpler representations such as normal map or spatially invariant BRDF, our algorithm can handle dynamic surface details, since they are not involved in the precomputation. Fully dynamic environment lighting can also be supported, since we directly use the cubemap representation for the lighting.

As a limitation, our algorithm only handles distant lighting. Also, it is difficult to incorporate global scale interreflections, since many possible values will be introduced into the transfer matrix, breaking the prerequisite of our biclustering algorithm. Another limitation is that, as other per-vertex PRT algorithms, we require the models to be reasonably tessellated since the per-pixel local incident lighting is interpolated linearly from vertex data. Significant error can also be introduced by interpolating across vertices that pointing in very different directions, which could be alleviated by applying a proper cease angle as in shading interpolation.

In future work, we are interested in developing mechanisms that enable trade-offs between quality and storage/performance. Currently our method is based on a compression of the radiance transfer matrix, instead of an approximation. A possible way of approximating the transfer matrix is to ignore the residual matrices according to a carefully designed error metric. It is also interesting to improve the algorithm by making use of the knowledge of lightings and BTFs.

TABLE 2
Scalability Test Results

#Vertices	3,286	12,380	47,960
$\mathrm{nnz}(\mathsf{T})$ **(M)**	10.6	39.2	151.3
MFLOPs	21.2	78.4	302.7
Reduced to	5.8	21.2	80.7
Computation%	27.4	27.0	26.7
Data size(MB)	1,400	4,868	18,859
Reduced to(MB)	24.2	84.1	316.9
Data size%	1.7	1.7	1.7
Precomp.(min)	12	49	279
Avg. FPS	277.9/86.8	115.8/32.3	48.5/11.9

* Bunny models with different numbers of vertices are used to test our algorithm. Note the consistent compression ratios. The rendering performance scales linearly with the vertex number.

REFERENCES

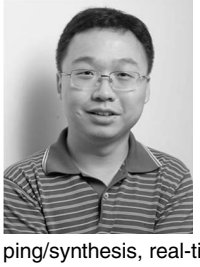
[1] P.-P. Sloan, J. Hall, J. Hart, and J. Snyder, "Clustered Principal Components for Precomputed Radiance Transfer," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 382-391, 2003.
[2] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments," *Proc. ACM SIGGRAPH*, pp. 527-536, 2002.
[3] P.-P. Sloan, X. Liu, H.-Y. Shum, and J. Snyder, "Bi-scale Radiance Transfer," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 370-375, 2003.
[4] R. Ng, R. Ramamoorthi, and P. Hanrahan, "All-Frequency Shadows Using Non-Linear Wavelet Lighting Approximation," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 376-381, 2003.
[5] R. Ng, R. Ramamoorthi, and P. Hanrahan, "Triple Product Wavelet Integrals for All-Frequency Relighting," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 477-487, 2004.
[6] X. Liu, P.-P. Sloan, H.-Y. Shum, and J. Snyder, "All-Frequency Precomputed Radiance Transfer for Glossy Objects," *Proc. Eurographics Symp. Rendering*, pp. 337-344, 2004.
[7] R. Wang, J. Tran, and D. Luebke, "All-Frequency Relighting of Glossy Objects," *ACM Trans. Graphics*, vol. 25, no. 2, pp. 293-318, 2006.
[8] Y.-T. Tsai and Z.-C. Shih, "All-Frequency Precomputed Radiance Transfer Using Spherical Radial Basis Functions and Clustered Tensor Approximation," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 967-976, 2006.
[9] W. Heidrich, K. Daubert, J. Kautz, and H.-P. Seidel, "Illuminating Micro Geometry Based on Precomputed Visibility," *Proc. ACM SIGGRAPH*, pp. 455-464, 2000.
[10] K.J. Dana, B. van Ginneken, S.K. Nayar, and J.J. Koenderink, "Reflectance and Texture of Real-World Surfaces," *ACM Trans. Graphics*, vol. 18, no. 1, pp. 1-34, 1999.
[11] K. Daubert, H.P.A. Lensch, W. Heidrich, and H.-P. Seidel, "Efficient Cloth Modeling and Rendering," *Proc. Eurographics Workshop Rendering*, pp. 63-70, 2001.
[12] X. Liu, Y. Hu, J. Zhang, X. Tong, B. Guo, and H.-Y. Shum, "Synthesis and Rendering of Bidirectional Texture Functions on Arbitrary Surfaces," *IEEE Trans. Visualization and Computer Graphics*, vol. 10, no. 3, pp. 278-289, May 2004.

[13] T. Malzbender, D. Gelb, and H. Wolters, "Polynomial Texture Maps," *Proc. ACM SIGGRAPH,* pp. 519-528, 2001.
[14] M. Ashikhmin and P. Shirley, "Steerable Illumination Textures," *ACM Trans. Graphics,* vol. 21, no. 1, pp. 1-19, 2002.
[15] M.A.O. Vasilescu and D. Terzopoulos, "Tensortextures: Multilinear Image-Based Rendering," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 336-342, 2004.
[16] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H.-Y. Shum, "Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces," *Proc. ACM SIGGRAPH,* pp. 665-672, 2002.
[17] K. Zhou, P. Du, L. Wang, J. Shi, B. Guo, and H.-Y. Shum, "Decorating Surfaces with Bidirectional Texture Functions," *IEEE Trans. Visualization and Computer Graphics,* vol. 11, no. 5, pp. 519-528, Sept. 2005.
[18] J. Kautz, S. Boulos, and F. Durand, "Interactive Editing and Modeling of Bidirectional Texture Functions," *ACM Trans. Graphics,* vol. 26, no. 3, p. 53, 2007.
[19] G. Müller, J. Meseth, M. Sattler, R. Sarlette, and R. Klein, "Acquisition, Synthesis, and Rendering of Bidirectional Texture Functions," *Computer Graphics Forum,* vol. 24, no. 1, pp. 83-109, 2005.
[20] Y. Chen, X. Tong, J. Wang, S. Lin, B. Guo, and H.-Y. Shum, "Shell Texture Functions," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 343-353, 2004.
[21] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum, "View-Dependent Displacement Mapping," *ACM Trans. Graphics,* vol. 22, no. 3, pp. 334-339, 2003.
[22] X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum, "Generalized Displacement Maps," *Proc. Eurographics Symp. Rendering,* pp. 227-233, 2004.
[23] P.-P. Sloan, B. Luna, and J. Snyder, "Local, Deformable Precomputed Radiance Transfer," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 1216-1224, 2005.
[24] J. Wang, X. Tong, J. Snyder, Y. Chen, B. Guo, and H.-Y. Shum, "Capturing and Rendering Geometry Details for BTF-Mapped Surfaces," *The Visual Computer,* vol. 21, nos. 8-10, pp. 559-568, 2005.
[25] D.L. James and K. Fatahalian, "Precomputing Interactive Dynamic Deformable Scenes," *ACM Trans. Graphics,* vol. 22, no. 3, pp. 879-887, 2003.
[26] R. Wang, J. Tran, and D. Luebke, "All-Frequency Relighting of Non-Diffuse Objects using Separable BRDF Approximation," *Proc. Eurographics Symp. Rendering,* pp. 345-354, 2004.
[27] X. Sun, K. Zhou, Y. Chen, S. Lin, J. Shi, and B. Guo, "Interactive Relighting with Dynamic BRDFs," *ACM Trans. Graphics,* vol. 26, no. 3, 2007.
[28] K. Xu, Y.-T. Jia, H. Fu, S. Hu, and C.-L. Tai, "Spherical Piecewise Constant Basis Functions for All-Frequency Precomputed Radiance Transfer," *IEEE Trans. Visualization and Computer Graphics,* vol. 14, no. 2, pp. 454-467, Mar./Apr. 2008.
[29] G. Garg, E.-V. Talvala, M. Levoy, and H.P.A. Lensch, "Symmetric Photography: Exploiting Data-Sparseness in Reflectance Fields," *Proc. Eurographics Symp. Rendering,* pp. 251-262, 2006.
[30] T. Annen, Z. Dong, T. Mertens, P. Bekaert, H.-P. Seidel, and J. Kautz, "Real-Time, All-Frequency Shadows in Dynamic Scenes," *ACM Trans. Graphics,* vol. 27, no. 3, pp. 1-8, 2008.
[31] T. Annen, T. Mertens, P. Bekaert, H.-P. Seidel, and J. Kautz, "Convolution Shadow Maps," *Proc. Eurographics Symp. Rendering,* J. Kautz and S. Pattanaik, eds., vol. 18. pp. 51-60, 2007.
[32] E. Cheslack-Postava, R. Wang, O. Akerlund, and F. Pellacini, "Fast, Realistic Lighting and Material Design Using Nonlinear Cut Approximation," *ACM Trans. Graphics,* vol. 27, no. 5, pp. 1-10, 2008.
[33] S.C. Madeira and A.L. Oliveira, "Biclustering Algorithms for Biological Data Analysis: A Survey," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 1, no. 1, pp. 24-45, Jan.-Mar. 2004.
[34] P. Shirley and K. Chiu, "A Low Distortion Map Between Disk and Square," *J. Graphics Tools,* vol. 2, no. 3, pp. 45-52, 1997.
[35] NVIDIA, "CUDA Homepage,"http://developer.nvidia.com/object/cuda.html, 2007.
[36] M. Ashikhmin and P. Shirley, "An Anisotropic Phong BRDF Model," *J. Graphics Tools,* vol. 5, no. 2, pp. 25-32, 2000.
[37] M. Ashikhmin, S. Premože, and P. Shirley, "A Microfacet-Based BRDF Generator," *Proc. ACM SIGGRAPH,* pp. 65-74, 2000.
[38] F. Suykens, K. vom Berge, A. Lagae, and P. Dutr, "Interactive Rendering with Bidirectional Texture Functions," *Computer Graphics Forum,* vol. 22, no. 3, pp. 463-472, 2003.
[39] M. Sattler, R. Sarlette, and R. Klein, "Efficient and Realistic Visualization of Cloth," *Proc. Eurographics Symp. Rendering,* pp. 167-177, 2003.
[40] J. Meseth, G. Müller, and R. Klein, "Reflectance Field Based Real-Time, High-Quality Rendering of Bidirectional Texture Functions," *Computers & Graphics,* vol. 28, no. 1, pp. 105-112, 2004.

Xin Sun received the bachelor's degree and the PhD degree in computer science from Zhejiang University, Hangzhou, China in 2002 and 2008, respectively. After that, he joined Internet Graphics Group in Microsoft Research Asia as an associate researcher. His research interests include real-time global illumination rendering and GPU-based photorealistic rendering.

Qiming Hou received the BS degree in the academic talent program of Tsinghua University (Mainland China) in 2006 and is currently working toward the PhD degree in computer science at Tsinghua University (Mainland China). His research interests include general purpose processing using graphics processors,compiler techniques, realistic rendering, and interactive rendering. For the past three years, he has been an intern consultant in Microsoft Research Asia.

Zhong Ren received the bachelor's degree in information engineering, the master's degree in mechanical engineering, and the PhD degree in computer science in 2007, all from Zhejiang University, Hangzhou, China. In 2007, he joined Microsoft Research Asia, where he is currently an associate researcher in the Internet Graphics group. His research interests include real-time rendering of soft shadows and participating media, spherical harmonics for real-time rendering, and also GPU-based photorealistic rendering.



Kun Zhou received the BS and the PhD degrees in computer science from Zhejiang University in 1997 and 2002, respectively. He is currently a Cheung Kong distinguished professor in the computer science department of Zhejiang University, and a member of the State Key Lab of CAD&CG. Before joining Zhejiang University, he was a leader researcher of the graphics group at Microsoft Research Asia. His research interests include shape modeling/editing, texture mapping/synthesis, real-time rendering, and GPU parallel computing.

Baining Guo received the BS degree from Beijing University and the MS and PhD degrees from Cornell University. He is the assistant managing director of Microsoft Research Asia, where he also serves as the head of the graphics lab. Prior to joining Microsoft in 1999, he was a senior staff researcher with the Microcomputer Research Labs of Intel Corporation in Santa Clara, California. His research interests include computer graphics and visualization, in the areas of texture and reflectance modeling, texture mapping, translucent surface appearance, real-time rendering, and geometry modeling. He is a fellow of the IEEE.