

Wan-Chun Ma · Chun-Tse Hsiao · Ken-Yi Lee · Yung-Yu Chuang · Bing-Yu Chen

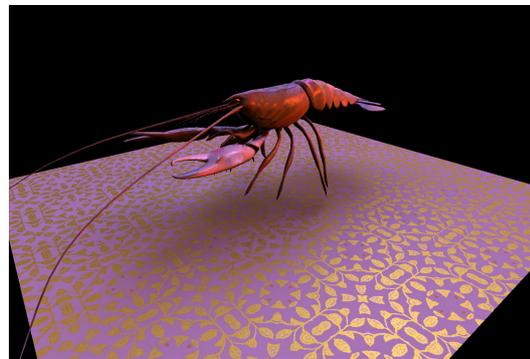
# Real-Time Triple Product Relighting Using Spherical Local-Frame Parameterization

**Abstract** This paper addresses the problem of real-time rendering for objects with complex materials under varying all-frequency illumination and changing view. Our approach extends the triple product algorithm by using local-frame parameterization, spherical wavelets, per-pixel shading and visibility textures. Storing BRDFs with local-frame parameterization allows us to handle complex BRDFs and incorporate bump mapping more easily. In addition, it greatly reduces the data size compared to storing BRDFs with respect to the global frame. The use of spherical wavelets avoids uneven sampling and energy normalization of cubical parameterization. Finally, we use per-pixel shading and visibility textures to remove the need for fine tessellations of meshes and shift most computation from vertex shaders to more powerful pixel shaders. The resulting system can render scenes with realistic shadow effects, complex BRDFs, bump mapping and spatially-varying BRDFs under varying complex illumination and changing view at real-time frame rates on modern graphics hardware.

**Keywords** All-frequency relighting · Precomputed radiance transfer · Local frame · Spherical wavelets · Real-time rendering

## 1 Introduction

Realistic rendering of objects with complex materials, complex natural lighting and intricate shadowing effects has many applications. Conventional approaches could take minutes per frame to render these effects. Recently, with the advancement of graphics hardware, several methods based on pre-computation have been proposed to enable interactive ren-



**Fig. 1** A typical result with the techniques of this paper. Our method can handle complex materials. Here, both the *Lobster* model and the floor are mapped with SBRDFs. Illumination and viewpoint can be changed in real-time. These images are of  $1024 \times 768$  resolution and rendered at 15fps to 45fps using an ATI X1900 XTX.

dering for rich lighting effects, such as precomputed radiance transfer using spherical harmonics by Sloan *et al.* [15, 16] and all-frequency relighting using wavelets by Ng *et al.* [8, 9]. The former allows changing lighting and viewpoint in real-time, but limited to low-frequency illumination. The latter captures all-frequency lighting effects, but requires a few seconds for manipulation of lighting and viewpoint. This paper proposes a method for real-time rendering of objects with complex materials under varying all-frequency illumination and changing view. Our approach extends the triple product approach proposed by Ng *et al.* [9] in the following ways:

- **The use of local-frame parameterization for shading.** We change the coordinate system for shading from the global frame used in previous wavelet-based relighting papers into a local frame. Since both of the visibility function and BRDF stay in the local frame, it is not necessary to recompute their wavelet coefficients when changing viewing or lighting condition. Storing BRDF in a local frame also enables us to easily handle complex BRDFs and incorporate bump mapping. In addition, it greatly reduces the data size compared to storing BRDF

in the global frame. Local-frame parameterization has been used in spherical harmonics-based relighting [15]. For spherical harmonics, it is an obvious choice since spherical harmonics can be easily rotated and every function should just be parameterized in its natural domain, global for lighting and local for BRDFs. However, for wavelet-based relighting, a trade-off has to be made when choosing between global and local frames since rotation is not easy for a wavelet basis. Previous wavelet-based work [8, 9, 19–21] uses global-frame parameterization. In this paper, we propose to use local-frame parameterization and show its advantages.

- **The use of spherical wavelets for avoiding uneven sampling.** Since functions involved in the rendering equation, illumination, visibility and BRDF, are all spherical functions, it makes sense to represent them using spherical wavelets [12] to avoid uneven sampling and energy normalization in cubical parameterization.
- **The use of per-pixel shading and visibility textures for efficient GPU implementation.** Previous relighting approaches require a fine tessellation to capture material and visibility variations over a surface even if the surface is flat. It is because rendering is performed on a per-vertex basis. Instead, we use per-pixel shading to shift computation from vertex shaders to more powerful pixel shaders for a more efficient GPU implementation. In addition, we sample the visibility functions over a surface and store them in a *visibility texture*. It allows a fine tessellated mesh to be replaced with a coarse mesh along with a visibility texture.

With these changes, our system can render scenes with realistic shadowing effects, complex BRDFs, bump mapping and spatially-varying BRDFs (SBRDFs) under varying complex illumination and changing viewpoint in real-time on modern graphics hardware. Figure 1 demonstrates rendering of a scene with SBRDFs under complex illumination. Note how the rich lighting effects on the ground depends on the lighting conditions.

We have carefully designed our method so that it is suitable for GPU implementation. The resulting system achieves comparable rendering quality with the triple product wavelet relighting [9] but it is much faster than previous methods. In addition, our method is capable of rendering some effects that have not been demonstrated before, such as bump mapping and SBRDFs for all-frequency relighting. Furthermore, our method also allows all-frequency material editing such as changing the glossiness number in Phong model in a few seconds.

## 2 Related Work

In this section, We briefly review related work in three categories, precomputed radiance transfer, wavelet-based relighting and spherical wavelets.

**Precomputed radiance transfer (PRT).** PRT is an efficient precomputation method for realistic image synthe-

sis. Most PRT papers use spherical harmonics as the basis function [5, 6, 11, 15–17]. Using a local frame for shading is widely adopted in spherical-harmonics-based PRT techniques [5, 6, 15, 17] because spherical harmonics basis has a good rotational property. Bump mapping [14] and anisotropic BRDFs [5] can be integrated into spherical harmonics PRT framework. However, the biggest problem of spherical harmonics is that it can only represent low-frequency lighting effects and requires a very large number of coefficients when modeling high-frequency lighting. Thus, this restricts the use of spherical harmonics PRT to mostly diffuse-like BRDFs and low-frequency lighting environments.

**Wavelet-based and the other all-frequency relighting techniques.** The use of wavelets as basis enables adaptive all-frequency relighting of complex scenes [7–9, 19–21]. However, most of the up-to-date techniques project spherical functions onto cubemaps. Such an uneven sampling, especially occurring at corners, needs normalization to keep the energy of the functions consistent. In addition, transport functions (or simply BRDFs) in a global frame require more space for storage, and are more difficult to be coupled with local-frame shading techniques such as bump mapping. Recently, new techniques have been proposed to model transport and lighting functions for more glossy materials [3] or better performance [18].

**Spherical wavelets.** Spherical wavelets have been used for BRDF representation [12], texture processing [13], analysis of fluid flow [10], image-based relighting [22] and other applications. Schörder *et al.* [12] are the first to introduce the concept of spherical wavelets to graphics community. They have demonstrated the potential of using spherical wavelets for rendering by representing a partial BRDF defined on a single hemisphere by fixing the incoming direction. Our spherical wavelets are derived from the studies on the orthogonality of triangular Haar spherical wavelet basis by Bonneau [1] and Nielson *et al.* [10].

## 3 Algorithm

We develop our algorithm based on the triple product formulation introduced by Ng *et al.* [9] which we describe below for completeness and clarity for symbols. This framework is based on the rendering equation for direct illumination.

$$B(\mathbf{x}, \omega_o) = \int_{\Omega} L(\mathbf{x}, \omega_i) V(\mathbf{x}, \omega_i) \rho(\mathbf{x}, \omega_i, \omega_o) (\omega_i \cdot \mathbf{n}(\mathbf{x})) d\omega_i, \quad (1)$$

where  $B$  is the radiance function of position  $\mathbf{x}$  and outgoing direction<sup>1</sup>  $\omega_o$ ,  $L$  and  $V$  are the lighting and visibility functions respectively,  $\rho$  is the BRDF and  $\mathbf{n}$  is the surface normal. By assuming the lighting  $L$  is a distant illumination function (environment map) and incorporating the term  $\omega_i \cdot \mathbf{n}(\mathbf{x})$  into the BRDF function as the function  $\tilde{\rho}$ , Equation 1 becomes

$$B(\mathbf{x}, \omega_o) = \int_{\Omega} L(\omega_i) V(\mathbf{x}, \omega_i) \tilde{\rho}(\mathbf{x}, \omega_i, \omega_o) d\omega_i. \quad (2)$$

<sup>1</sup> Note that  $\omega_o$  is a function of  $\mathbf{x}$  for a given view. Hence,  $\omega_o$  is actually an abbreviation for  $\omega_o(\mathbf{x})$ .

For a given view, for each position  $\mathbf{x}$ , we can infer the corresponding outgoing directions  $\omega_o$  and calculate the reflected color  $B^{\mathbf{x},\omega_o}$  by the following triple product integration,

$$B^{\mathbf{x},\omega_o} = \int_{\Omega} L(\omega_i) V^{\mathbf{x}}(\omega_i) \tilde{\rho}^{\mathbf{x},\omega_o}(\omega_i) d\omega_i, \quad (3)$$

where  $V^{\mathbf{x}}$  is the visibility function for the given position  $\mathbf{x}$ ; similarly,  $\tilde{\rho}^{\mathbf{x},\omega_o}$  is  $\tilde{\rho}$  for the given position  $\mathbf{x}$  and the given direction  $\omega_o$ . As shown by Ng *et al.* [9], by expanding the spherical functions,  $L$ ,  $V^{\mathbf{x}}$  and  $\tilde{\rho}^{\mathbf{x},\omega_o}$ , with some appropriate basis functions  $\Psi(\omega)$ ,

$$\begin{aligned} L(\omega) &= \sum_i L_i \Psi_i(\omega), \\ V^{\mathbf{x}}(\omega) &= \sum_j V_j^{\mathbf{x}} \Psi_j(\omega), \\ \tilde{\rho}^{\mathbf{x},\omega_o}(\omega) &= \sum_k \tilde{\rho}_k^{\mathbf{x},\omega_o} \Psi_k(\omega), \end{aligned} \quad (4)$$

where  $L_i$ ,  $V_j^{\mathbf{x}}$  and  $\tilde{\rho}_k^{\mathbf{x},\omega_o}$  are coefficients for the spherical illumination, visibility and BRDF functions respectively, Equation 3 can be written in terms of these basis functions,

$$B^{\mathbf{x},\omega_o} = \sum_i \sum_j \sum_k L_i V_j^{\mathbf{x}} \tilde{\rho}_k^{\mathbf{x},\omega_o} \int_{\Omega} \Psi_i(\omega) \Psi_j(\omega) \Psi_k(\omega) d\omega. \quad (5)$$

The above equation is complicated to evaluate because of the triple product integrals. Ng *et al.* call these integrals the *tripling coefficients*  $C_{ijk}$  as

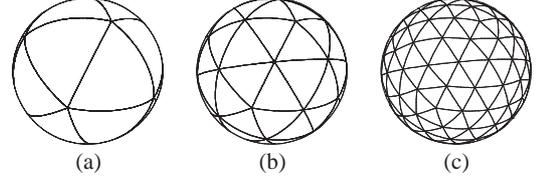
$$C_{ijk} = \int_{\Omega} \Psi_i(\omega) \Psi_j(\omega) \Psi_k(\omega) d\omega. \quad (6)$$

They have devised and analyzed procedures to evaluate  $C_{ijk}$  for different basis functions including points, 2D Fourier series, spherical harmonics and 2D Haar wavelets [9]. For the relighting application in Equation 5, they use cubemap Haar wavelets as the basis functions and parameterize the spherical lighting, visibility and BRDF functions using a global frame. In the following, we introduce our extensions of spherical wavelets, local-frame parameterization, per-pixel shading and visibility textures, to make the original triple product algorithm more flexible and efficient.

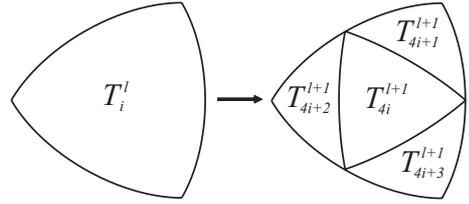
### 3.1 Spherical wavelets

Since the lighting, visibility and BRDF functions in Equation 3 are all spherical functions, we choose to use spherical wavelets [12] as the basis functions to avoid uneven sampling and energy normalization of cubical parameterization used in most previous work [7–9, 19]. We believe that the spherical parameterization is a natural and better choice than cubical one. Actually, spherical wavelets have been used for representing illumination and BRDFs by Schröder *et al.* [12].

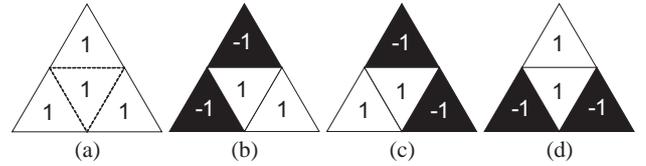
Many spherical wavelet bases have been proposed, such as lifted Butterfly basis and Bio-Haar basis. We have derived



**Fig. 2** Geodesic sphere construction. Starting with the icosahedron ((a) subdivision level 0), successive levels (b,c) are generated by subdividing triangles into four sub-triangles, accomplished by adding geodesic edges connecting midpoints of original edges.



**Fig. 3** Subdivision of a geodesic triangle  $T_i^l$  into four sub-triangles  $T_{4i+1}^{l+1}$ ,  $T_{4i+2}^{l+1}$ ,  $T_{4i+3}^{l+1}$  and  $T_{4i+4}^{l+1}$ .



**Fig. 4** Scaling and wavelet functions. (a) scaling function  $\varphi_i^l$ . (b) type-0 wavelet function  $\psi_{i,0}^l$ . (c) type-1 wavelet function  $\psi_{i,1}^l$ . (d) type-2 wavelet function  $\psi_{i,2}^l$ .

our spherical wavelets based on the optimal triangular Haar bases [1]. The construction of spherical wavelets relies on the geodesic sphere construction shown in Figure 2. Starting with an icosahedron (subdivision level 0), for each subdivision step, each geodesic triangle is divided into four sub-triangles by bisecting the geodesic edges at their mid-points. Let  $T_i^l$  be the  $i$ -th triangle at the subdivision level  $l$ . For level  $l$ , we define the  $i$ -th scaling function  $\varphi_i^l(\omega)$  as

$$\varphi_i^l(\omega) = \begin{cases} 1 & \text{if } \omega \in T_i^l \\ 0 & \text{otherwise.} \end{cases}$$

Let  $T_{4i}^{l+1}$ ,  $T_{4i+1}^{l+1}$ ,  $T_{4i+2}^{l+1}$  and  $T_{4i+3}^{l+1}$  be the four sub-triangles of  $T_i^l$  (Figure 3) and the area for each of the twenty triangles at level 0 equal 1, denoting as  $A_0 = 1$ . According the subdivision rule, we have  $A_{l+1} = \frac{1}{4} A_l$ . Hence, the area  $A_l$  of a triangle  $T_i^l$  at level  $l$  is equal to  $4^{-l}$ . We then define three types of wavelet functions associated with the domain  $T_i^l$  at

level  $l$  as

$$\begin{aligned}\psi_{i,0}^l(\omega) &= \varphi_{4i}^{l+1}(\omega) - \varphi_{4i+1}^{l+1}(\omega) - \varphi_{4i+2}^{l+1}(\omega) + \varphi_{4i+3}^{l+1}(\omega) \\ \psi_{i,1}^l(\omega) &= \varphi_{4i}^{l+1}(\omega) - \varphi_{4i+1}^{l+1}(\omega) + \varphi_{4i+2}^{l+1}(\omega) - \varphi_{4i+3}^{l+1}(\omega) \\ \psi_{i,2}^l(\omega) &= \varphi_{4i}^{l+1}(\omega) + \varphi_{4i+1}^{l+1}(\omega) - \varphi_{4i+2}^{l+1}(\omega) - \varphi_{4i+3}^{l+1}(\omega)\end{aligned}$$

Figure 4 illustrates the scaling function and the wavelet functions. This set of level-0 scaling and wavelet functions  $\Psi = \{\varphi_{i_0}^0, \psi_{i_1,0}^1, \psi_{i_1,1}^1, \psi_{i_1,2}^1, \dots, \psi_{i_l,0}^l, \psi_{i_l,1}^l, \psi_{i_l,2}^l, \dots\}$  forms a basis for spherical functions, where  $i_l \in \{0, 1, 2, \dots, 20 \cdot 2^l - 1\}$ . The set  $\Psi$  is an orthogonal basis because

$$\begin{aligned}\int_{\Omega} \varphi_i^0(\omega) \varphi_{i'}^0(\omega) d\omega &= A_0 \delta_{ii'} = \delta_{ii'} \\ \int_{\Omega} \varphi_i^0(\omega) \psi_{i',t}^l(\omega) d\omega &= 0 \\ \int_{\Omega} \psi_{i,t}^l(\omega) \psi_{i',t'}^l(\omega) d\omega &= A_l \delta_{ll'} \delta_{ii'} \delta_{tt'},\end{aligned}$$

where  $\delta_{ij}$  is Dirac's delta function. Note that  $\Psi$  is not orthonormal. We could scale the wavelets properly to make the basis orthonormal, but we prefer to leave them unnormalized so that they reflect the underlying energy appropriately. We have derived the tripling coefficients for the spherical wavelets defined above:

*Tripling Coefficient Theorem for Spherical Wavelets.* Here, a tripling coefficient defined in Equation 3 for our spherical wavelets is non-zero only for the following three cases:

*case 1:* All three bases are the same scaling function.

$$\int_{\Omega} \varphi_i^0(\omega) \varphi_i^0(\omega) \varphi_i^0(\omega) = \int_{\Omega} \varphi_i^0(\omega) d\omega = 1.$$

*case 2:* All three are different types of wavelets at the same level with the same index.

$$\int_{\Omega} \psi_{i,0}^l(\omega) \psi_{i,1}^l(\omega) \psi_{i,2}^l(\omega) = \int_{\Omega} \varphi_i^l(\omega) d\omega = 4^{-l}.$$

*case 3:* Two are identical wavelets at level  $l$  and their domain is overlapped with the domain of the third one who is at a strictly coarser level  $l'$ , i.e.,  $l' < l$ ,

$$\begin{aligned}\int_{\Omega} \psi_{i,t}^l(\omega) \psi_{i,t}^l(\omega) \varphi_{i'}^{l'}(\omega) d\omega &= \int_{\Omega} \varphi_i^l(\omega) \varphi_{i'}^{l'}(\omega) d\omega = 4^{-l} \\ \int_{\Omega} \psi_{i,t}^l(\omega) \psi_{i,t}^l(\omega) \psi_{i',t'}^{l'}(\omega) d\omega &= \int_{\Omega} \varphi_i^l(\omega) \psi_{i',t'}^{l'}(\omega) d\omega = \pm 4^{-l},\end{aligned}$$

where the sign depends on which part the coarser-level basis overlaps with the other two's domain. It is not surprising that our spherical wavelet tripling coefficients are similar to 2D Haar wavelet tripling coefficients proven by Ng *et al.* [9] since spherical wavelets are isomorphic to 2D Haar wavelets.

symbol	meaning
$n_l^{\omega_i}$	sampling resolution of a lighting function
$r_l$	number of precomputed rotations of lighting
$n_v^{\omega_i}$	sampling resolution of a visibility function
$n_x$	number of vertices
$n_{\rho}^{\omega_o}$	number of samples for outgoing directions
$n_{\rho}^{\omega_i}$	sampling resolution of a BRDF function
$r_{\rho}$	number of precomputed rotations of normals
$n_{\rho}$	number of materials

**Table 2** Symbols for storage requirement calculation.

### 3.2 Local-frame parameterization

To the best of our knowledge, all relighting papers using wavelets [8,9,19] parameterize the lighting, visibility and BRDF functions with respect to the global frame. That is, the integration domain  $\Omega$  in Equation 3 is defined over the global frame and the same for all positions  $\mathbf{x}$  when calculating  $B^{\mathbf{x},\omega_o}$ . It is a natural choice to use the global frame since Equation 3 is defined with respect to the global frame. In such a setting, we only need to store a global lighting function. However, each position  $\mathbf{x}$  has to store its own BRDF function even if they are made of the same material. It is because BRDF is defined with respect to the local frame defined at  $\mathbf{x}$ . It requires a lot of storage to store a BRDF function per  $\mathbf{x}$  considering that BRDF is a 4D function. To save space, Ng *et al.* [9] precompute several rotated versions of the BRDF function and use  $\mathbf{n}(\mathbf{x})$  to look up the appropriate rotated BRDF to be used for  $\mathbf{x}$ . It is why  $\tilde{\rho}$  in Equation 3 depends on  $\mathbf{x}$ . In this setting, rotating lights is achieved by rotating and resampling the lighting function on the fly and changing views only involves using different  $\mathbf{x}$  and  $\omega_o$  to look up the BRDF table.

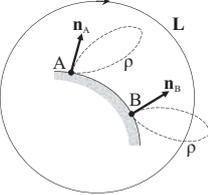
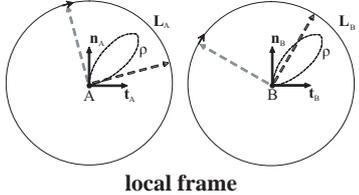
We propose to parameterize these functions using the local frame defined for each vertex. With the local-frame parameterization, when evaluating Equation 3 for a position  $\mathbf{x}$ , the integration domain becomes defined over the local frame, which is established by  $\mathbf{x}$ 's normal  $\mathbf{n}(\mathbf{x})$  and tangent  $\mathbf{t}(\mathbf{x})$ . Hence, in this setting, Equation 3 becomes

$$B^{\mathbf{x},\omega_o} = \int_{\Omega_{\mathbf{x}}} L^{\mathbf{x}}(\omega_i) V^{\mathbf{x}}(\omega_i) \tilde{\rho}^{\omega_o}(\omega_i) d\omega_i, \quad (7)$$

where  $\Omega_{\mathbf{x}}$  is the spherical domain defined by  $\mathbf{x}$ 's local frame and  $L^{\mathbf{x}}$  is the global lighting function reparameterized in  $\mathbf{x}$ 's local frame<sup>2</sup>. Using this parameterization, the object of the same material can share the same BRDF table. However, the global illumination function needs to be rotated into different local frames for different  $\mathbf{x}$ 's during rendering. Unfortunately, as cubical 2D wavelets, we are not aware of any efficient algorithm to rotate spherical wavelets. To solve this problem, similar to Ng *et al.*'s solution, we precompute a set of rotated versions of the lighting function by uniformly sampling Euler angles and use  $\mathbf{x}$ 's normal to look up the proper rotated lighting function  $L^{\mathbf{x}}$  when rendering  $\mathbf{x}$ .

<sup>2</sup> Note that, while using the same symbol,  $V^{\mathbf{x}}(\omega_i)$  in Equations 3 and 7 are parameterized with respect to different frames.

**Table 1** Comparisons between global-frame parameterization and local-frame parameterization.

	 global frame	 local frame
rendering equation	$B^{\mathbf{x}, \omega_o} = \int_{\Omega} L(\omega_i) V^{\mathbf{x}}(\omega_i) \tilde{\rho}^{\mathbf{x}, \omega_o}(\omega_i) d\omega_i$	$B^{\mathbf{x}, \omega_o} = \int_{\Omega_{\mathbf{x}}} L^{\mathbf{x}}(\omega_i) V^{\mathbf{x}}(\omega_i) \tilde{\rho}^{\omega_o}(\omega_i) d\omega_i$
lighting function $L$	one copy	multiple rotated versions indexed by normal $\mathbf{n}(\mathbf{x})$
visibility function $V$	one copy per $\mathbf{x}$	one copy per $\mathbf{x}$
BRDF function $\tilde{\rho}$	multiple rotated versions indexed by normal $\mathbf{n}(\mathbf{x})$	one copy per material
storage requirement*	$n_l^{\omega_i} + n_x n_v^{\omega_i} + n_\rho r_\rho n_\rho^{\omega_o} n_\rho^{\omega_i}$	$r_l n_l^{\omega_i} + n_x n_v^{\omega_i} + n_\rho n_\rho^{\omega_o} n_\rho^{\omega_i}$

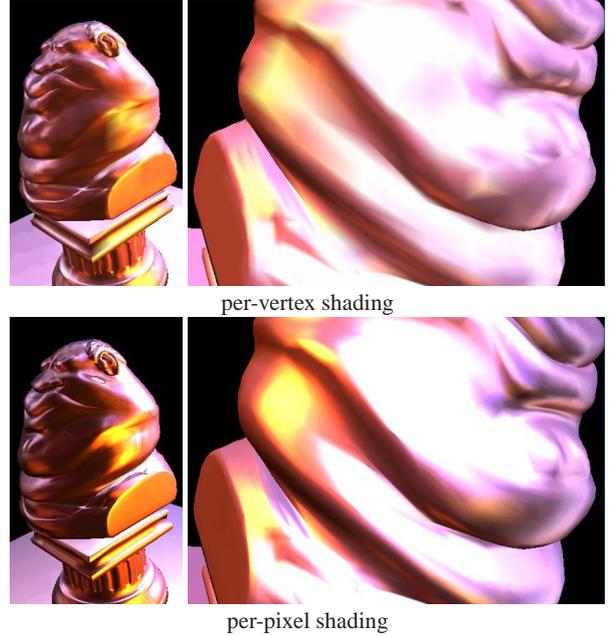
\* The storage requirement is estimated without other parameterization and data compression. Related symbols are defined in Table 2

In local-frame parameterization, only one BRDF table is required for a material, but multiple pre-rotated lightings are required. On the contrast, the global-frame parameterization only needs to store a lighting function but has to store multiple rotated BRDF functions. We argue that the local-frame approach is more storage efficient since it stores multiple 2D lighting functions while the global-frame approach would have to store multiple 4D BRDF functions, as indicates by Clarberg *et al.* [2]. There is only one lighting function at a time, but there could be several BRDF functions. Using the symbols defined in Table 2, the storage requirement is  $n_l^{\omega_i} + n_x n_v^{\omega_i} + n_\rho r_\rho n_\rho^{\omega_o} n_\rho^{\omega_i}$  for the global-frame approach and  $r_l n_l^{\omega_i} + n_x n_v^{\omega_i} + n_\rho n_\rho^{\omega_o} n_\rho^{\omega_i}$  for the local-frame approach without other parameterization and compression. Table 1 compares these two parameterization schemes. In general,  $(r_l - 1)n_l^{\omega_i}$  is less than  $n_\rho(r_\rho - 1)n_\rho^{\omega_o} n_\rho^{\omega_i}$ , justifying our choice of the local-frame parameterization. Hence, the local-frame parameterization greatly reduces the data size of BRDFs and allows us to use more materials. For example, we can use spatially-varying BRDF, a composition of multiple BRDFs. In addition, local-frame parameterization allows us to separate the precomputation of BRDFs from the knowledge of normals. Thus, the precomputation can be performed separately and it becomes straightforward to apply bump maps. On the other hand, the global-frame approach is better for applications which requires dynamic lighting.

To allow for anisotropic materials, we actually sample the lighting function with both normal and tangent, making the lighting function a 3D array. Again, it would cause a larger storage increase to extend global-frame parameterization to support anisotropic materials.

### 3.3 Per-pixel shading and visibility textures

Most existing all-frequency relighting approaches evaluate Equation 7 for each vertex  $\mathbf{x}$  and then interpolate vertices' colors to rasterize visible triangles. This per-vertex shading



**Fig. 5** Comparisons between per-vertex shading and per-pixel shading. The left column shows rendering of a statue model with different shading methods. Closeup renderings of the same model from other view are shown on the right. Per-pixel shading clearly yields substantial improvement over per-vertex shading.

approach can be described more precisely in pseudocode:

```

for each visible triangle  $T$  do
  for each of  $T$ 's vertices  $\mathbf{x}_1$ ,  $\mathbf{x}_2$  and  $\mathbf{x}_3$  do
    evaluate Equation 7 for  $\mathbf{x}_i$  to obtain its color  $B^{\mathbf{x}_i}$ 
  end for
  for each pixel  $\mathbf{p}$  within  $T$  do
    evaluate  $B^{\mathbf{p}}$  by interpolating  $B^{\mathbf{x}_1}$ ,  $B^{\mathbf{x}_2}$  and  $B^{\mathbf{x}_3}$ 
  end for
end for

```

Instead of interpolating colors, we interpolate functions and use the interpolated functions to evaluate color for each visible pixel  $\mathbf{p}$ . Here is a more precise description of per-pixel shading method:

```

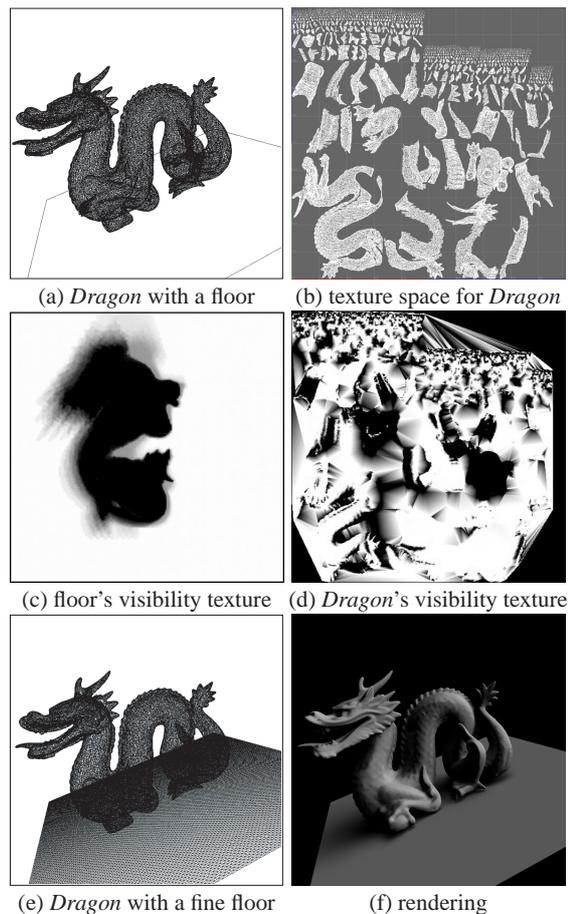
for each visible triangle  $T$  with vertices  $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$  do
  for each pixel  $\mathbf{p}$  within  $T$  do
    obtain  $L^{\mathbf{p}}$  by interpolating  $L^{\mathbf{x}_1}, L^{\mathbf{x}_2}$  and  $L^{\mathbf{x}_3}$ 
    obtain  $V^{\mathbf{p}}$  by interpolating  $V^{\mathbf{x}_1}, V^{\mathbf{x}_2}$  and  $V^{\mathbf{x}_3}$ 
    obtain  $\tilde{\rho}^{\mathbf{p}}$  by interpolating  $\tilde{\rho}^{\omega_o(\mathbf{x}_1)}, \tilde{\rho}^{\omega_o(\mathbf{x}_2)}$  and  $\tilde{\rho}^{\omega_o(\mathbf{x}_3)}$ 
    evaluate Equation 7 using  $L^{\mathbf{p}}, V^{\mathbf{p}}, \tilde{\rho}^{\mathbf{p}}$  to obtain  $B^{\mathbf{p}}$ 
  end for
end for

```

Analogous to the comparison between Phong shading and Gouraud shading, per-pixel shading renders more accurate results than per-vertex shading with less vertices. Figure 5 clearly shows the advantages of per-pixel shading over per-vertex shading. In addition, per-pixel shading shifts most computation (Equation 7) from vertex shaders to more powerful pixel shaders. Per-pixel shading also allows us to better render spatially varying material with mapping techniques such as SBRDFs, texture mapping and bump mapping.

Another problem with per-vertex shading is that, to obtain fine shadows, the models often have to be finely tessellated. For example, to render very simple geometry such as a floor (simply a quadrangle), we usually have to subdivide it into tens of thousands of triangles to have a very high vertex density on the floor for better rendering. This, however, takes too much time for vertex processing. For faster rendering, we propose to use visibility textures to reduce vertex count required for fine shadows. To create a visibility texture for a model, we first parameterize the model into a 2D  $(u, v)$ -map. Figure 6(b) shows the mapping between triangles and the resulting map for the dragon model (Figure 6(a)). We used Maya to generate this parameterization. Better approaches such as geometry image [4] could improve the utilization of texture maps for better results. For each texel of this map, we find its corresponding position  $\mathbf{x}$  on surface of the 3D mesh and compute  $\mathbf{x}$ 's visibility function  $V^{\mathbf{x}}$ . The result is a visibility texture map for the original model. In this map, each "texel" represents a visibility function for some position over the original surface by storing wavelet coefficients of the associated visibility function. Figure 6(c) is the visibility texture associated with the floor in Figure 6(a). At first sight, this approach seems not too different from highly tessellated meshes since we just shift fine tessellation from vertices to texels. However, mapping visibility as a texture over the surface speeds up rendering by 300% along with per-pixel shading. One concern about visibility texture is about the possible aliasing problem as most mapping techniques have to face. Our argument is that one can just use a high-resolution visibility map to avoid aliasing. In addition, mesh tessellation faces similar aliasing problem as well.

In sum, to accurately rendering spatially varying materials, it requires highly tessellated meshes with per-vertex shading. On the contrast, per-pixel shading along with visibility textures renders similar results in a shorter time by using coarse meshes.



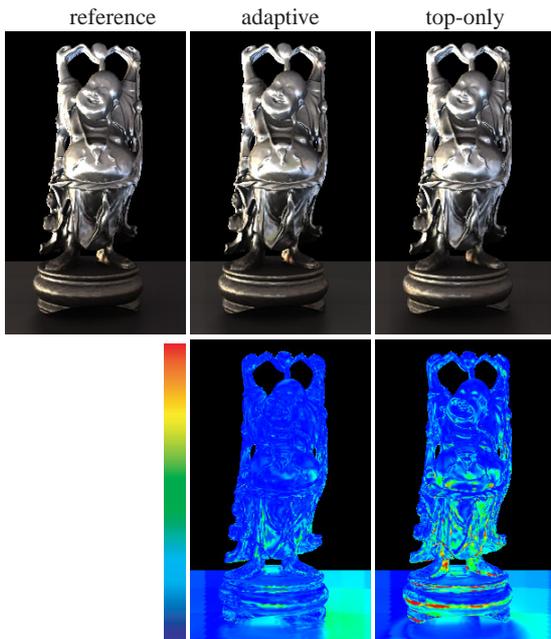
**Fig. 6** Illustrations and comparisons for visibility textures. The models (a) are first parameterized in a 2D texture space ((b) for *Dragon*). Visibility is sampled for each texel of the texture space to synthesize a visibility texture ((c) for the floor and (d) for the *Dragon* model). Without visibility textures, models have to be finely tessellated (e) to have comparable rendering (f), but about three times slower.

## 4 Implementation

In this section, we discuss implementation details of our real-time triple product relighting system.

### 4.1 Precomputation

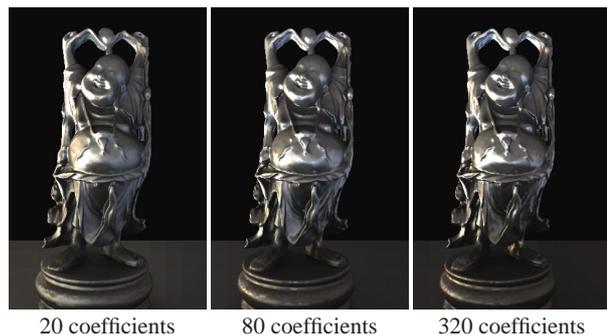
At the precomputation stage, we need to compute and store spherical wavelet coefficients for  $L^{\omega_i}, V^{\omega_i}$  and  $\tilde{\rho}^{\omega_o(\omega_i)}$  in Equation 7. For each spherical function, we sample its values at 5120 different  $\omega_i$ -directions. These directions are generated by subdividing an icosahedron for five levels. Discrete spherical wavelet transform is then applied to these sampled values to generate wavelet coefficients for the spherical function. Thus, in our implementation,  $n_i^{\omega_i} = n_v^{\omega_i} = n_\rho^{\omega_i} = 5120$  before data compression. For a lighting function, we precompute  $32 \times 16 \times 32$  rotations for  $32 \times 16$



**Fig. 7** Comparisons between adaptive and top-only strategies. Top row shows reference image and renderings, and the bottom row shows error for each pixel.

different normals and 32 tangent angles by uniform sampling, thus,  $r_l = 16384$ . For each material, we sample  $128 \times 64$   $\omega_o$ -directions, i.e.  $n_{\rho}^{\omega_o} = 8192$ . Hence, excluding storage requirement for visibility, local-frame parameterization would require roughly 170MB (without anisotropic materials) and 480MB (with anisotropic materials) before data compression. On the contrast, global-frame parameterization requires 82GB without anisotropic materials using formulas in Table 1.

Fortunately, not all coefficients need to be stored. To determine a strategy of selecting coefficients to store, we have experimented with two different strategies. The first one is adaptive, storing the  $n$  largest area-weighted coefficients. The second one is called top-only strategy, storing only the  $n$  coefficients in the top level. Figure 7 shows a comparison for them in terms of mean square errors. We have found that, in general, the adaptive approach is better than the top-only approach, but not by much as long as  $n$  is sufficiently large. The reason why both have similar performances could be explained by the tripling coefficient theorem for spherical wavelet in Section 3.1. It implies that, for a triple product to contribute, at least two of lighting, visibility and BRDF functions must carry coefficients at the same level of details. Since there is no way to predict where high-frequency parts will coincide during precomputation, most high-frequency coefficients are actually wasted because the other two functions do not necessarily store high frequency there. Hence, other than few occasional matches, storing top-level coefficients first is a reasonable approach since they represents more energy and are more likely to contribute. The top-only



**Fig. 8** A comparison of rendering with different number of coefficients, 20, 80 and 320.

approach is only worse than the adaptive approach for hard shadows (where  $V$  and  $L$  happens to have high-frequency coefficients of the same level at the same place) and specularly (where  $\rho$  and  $L$  matches). But, the top-only approach is better suited for GPU rendering because of its regularity. Thus, users have to make a trade-off between speed and quality here.

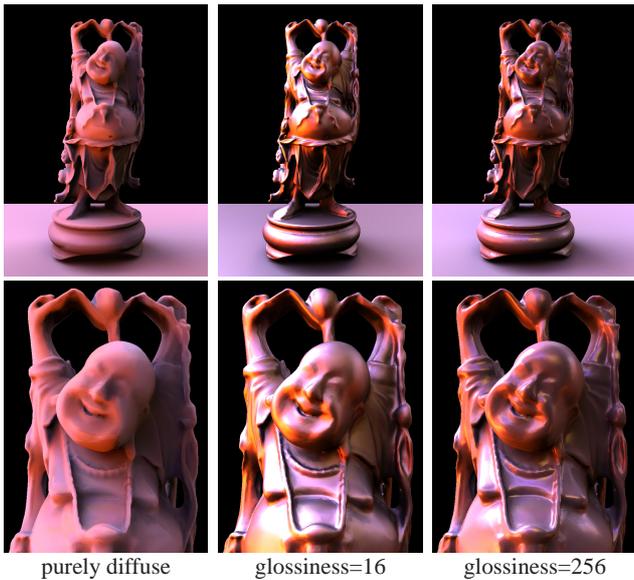
To decide the number of coefficients to store, we have performed experiments to determine the impact of number on rendering quality. In general, we found that rendering with 80 or 320 coefficients gives sufficient quality with real-time performance. Figure 8 shows a comparison. After such a reduction, typical storage usages for lighting and BRDF are 11.5MB/46.1MB and 7.2MB/28.6MB (80/320) respectively.

Since all the rendering computation is performed in GPU, in order to accelerate the rendering pipeline, all the textures are quantized to 8-bit RGBA, and tiled so that their sizes are powers of two in order to enable hardware-based linear interpolation. For visibility texture, the texture size of per-vertex interpolated visibility depends on the number of vertices, a  $4096 \times 4096$  texture (maximum size) can support up to 204K vertices. The size of the visibility texture depends on the UV resolution of the model. We use a  $1024 \times 1024 \times 128$  3D texture for all results in this paper.

## 4.2 Rendering

Given the textures generated at the precomputation stage, to render at  $\mathbf{x}$ , we use  $\mathbf{x}$ 's normal  $\mathbf{n}(\mathbf{x})$  and tangent  $\mathbf{t}(\mathbf{x})$  to look up nearby lighting functions and obtain coefficients  $L_i^{\mathbf{x}}$  by interpolating coefficients of nearby lighting functions. Similarly, we can retrieve coefficients  $V_j^{\mathbf{x}}$  and  $\tilde{\rho}_k^{\omega_o}$  from the the corresponding textures, and then apply the triple product algorithm to evaluate Equation 7.

In our framework, spatially-variant BRDF can be rendered easily by a linear combination of coefficients from two different BRDF textures. It is also very simple to implement bump mapping with local-frame parameterization by rotating the lighting function the same amount as the bump map



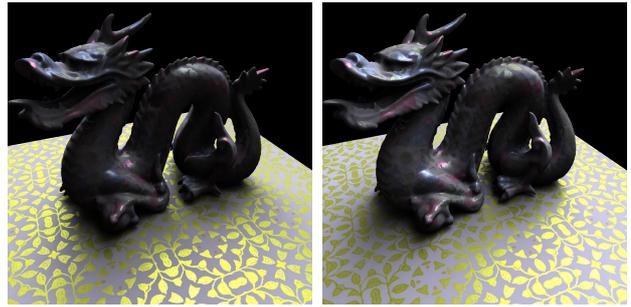
**Fig. 9** Material editing of a glossy *Buddha* in *Grace Cathedral*. The *Buddha* model is rendered using Phong model with different glossiness numbers. The bottom row shows closeup views of the top row. Our system allows changing parameters of BRDFs by sampling new BRDFs on the fly in a few seconds.

indicates. Note that this is not a physically correct solution. The visibility function should rotate accordingly but can't get rotated easily. However, in practice, we found this approximation works well visually.

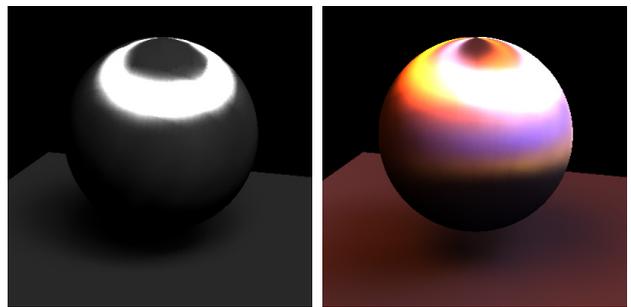
## 5 Results

Rendering was performed and timings were collected on a machine with Intel dual core Pentium D 3.2GHz with 2GB memory and an ATI X1900 XTX with 512MB graphics memory. The rendering resolution is  $1024 \times 768$ . Table 3 lists the average rendering speed under different settings for wavelet coefficients and visibility estimation. The 'visibility texture' column, reports frame rates using visibility textures, while the 'interpolated visibility' column reports frame rates of rendering with estimated visibility by interpolating visibility functions of nearby vertices. In the case of interpolated visibility, a floor of 26K triangles must be used to provide comparable shadow effects.

Our system allows real-time rendering of objects with complex materials. Figure 9 shows the results of a glossy Buddha with different glossiness settings. Note that our system could support interactive material editing. After adjusting parameters of BRDFs, it typically takes 2 to 3 seconds to resample the edited BRDF. Figure 10 demonstrates the effects for spatially-varying BRDFs under two different lighting conditions. Figure 11 shows the rendering of Ward's anisotropic model. We use a sphere as the model to clearly demonstrate the effects of anisotropic models. Figure 12 displays bump mapping effects under two different lighting con-



**Fig. 10** Rendering with spatially-varying BRDF. Both the *Dragon* model and the floor are mapped with SBRDFs and the scene is illuminated by the *Uffizi Gallery* environment map with different orientations.



**Fig. 11** A sphere rendered with Ward's anisotropic BRDF with  $\alpha_x = 0.1$  and  $\alpha_y = 1.0$ . The image on the left is illuminated by an area light source while the image on the right is illuminated by the *Grace Cathedral* environment map.

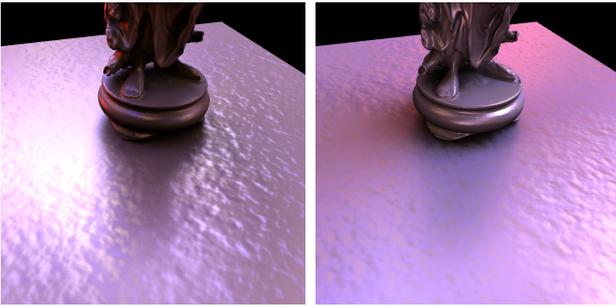
Model	number of vertices	visibility texture		interpolated visibility	
		80	320	80	320
Dragon	25k	97.4	38.4	21.5	8.5
XYZRGB	50k	46.6	16.9	14.7	6.9
Buddha	50k	43.5	17.6	15.1	6.8

**Table 3** Frame rates (in fps) of rendering models in this section with different numbers of coefficients and visibility estimation approaches.

ditions. Figure 13 shows complex shadowing effects from multiple area light sources. Finally, Figure 14 demonstrates the combination of these effects.

## 6 Conclusions and Future Work

We present extensions to the triple product all-frequency relighting method based on spherical wavelets, local-frame parameterization and per-pixel shading. The resulting rendering algorithm, which is implemented purely on GPUs, has real-time performance and per-pixel rendering quality. With this configuration, we achieve comparable quality to the previous triple product work [9], but two orders of magnitude faster. There are several interesting research directions we want to explore:



**Fig. 12** Bump mapping. The floor is mapped with a wave bump map and illuminated by the *Grace Cathedral* environment map with different orientations.



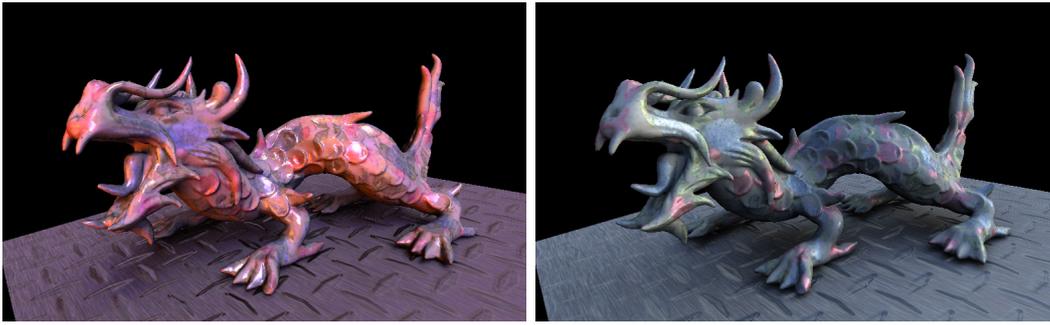
**Fig. 13** The *Buddha* model is illuminated by a set of multiple area light sources from three different orientations and casts shadows on the ground.

- **Solution for rotating coefficients, continuous spherical wavelets and other basis functions.** To solve the rotation of lighting coefficients would be the most useful. We could use repeatable property of icosahedrons or seek for some continuous spherical wavelets which have the same rotational property as the spherical harmonics do.
- **Integration with bi-directional texture functions.** A bi-directional texture function (BTF) describes spatial variation of different 4D transport functions; it captures effects such as self-shadowing and self-occlusion. To the best of our knowledge, currently there is no truly all-frequency relighting algorithm for BTFs. We would like to investigate the possibility to integrate the 6D BTF into our rendering framework.
- **Interactive material editing.** Our framework is capable of changing BRDF parameters on the fly. Currently, it takes a couple of seconds to do so. With proper approximation and help of GPUs, it is possible to make material editing more interactive.

**Acknowledgements** The authors would like to thank Peter Pon for his suggestions on GPU programming. This research was done while Wan-Chun was a visiting scholar in Graphics Lab, Institute of Creative Technologies, University of Southern California. This research was supported by National Science Council of Taiwan under NSC 94-2213-E-002-096.

## References

1. Bonneau, G.P.: Optimal triangular Haar bases for spherical data. In: IEEE Visualization 1999, pp. 279–284 (1999)
2. Clarberg, P., Jarosz, W., Akenine-Moller, T., Jensen, H.W.: Wavelet importance sampling: Efficiently evaluating products of complex functions. *ACM Transactions on Graphics* **24**(3), 1166–1175 (2005)
3. Green, P., Kautz, J., Matusik, W., Durand, F.: View-dependent precomputed light transport using nonlinear gaussian function approximations. In: Proceedings of I3D 2006, pp. 7–14 (2006)
4. Gu, X., Gortler, S.J., Hoppe, H.: Geometry images. In: Proceedings of SIGGRAPH 2002, pp. 355–361 (2002)
5. Kautz, J., Sloan, P.P., Snyder, J.: Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In: Proceedings of EGWR 2002, pp. 291–296 (2002)
6. Lehtinen, J., Kautz, J.: Matrix radiance transfer. In: Proceedings of I3D 2003, pp. 59–64 (2003)
7. Liu, X., Sloan, P.P., Shum, H.Y., Snyder, J.: All-frequency pre-computed radiance transfer for glossy objects. In: Proceedings of the EGSR 2004, pp. 337–344 (2004)
8. Ng, R., Ramamoorthi, R., Hanrahan, P.: All-frequency shadows using non-linear wavelet lighting approximation. *ACM Transactions on Graphics* **22**(3), 376–381 (2003)
9. Ng, R., Ramamoorthi, R., Hanrahan, P.: Triple product wavelet integrals for all-frequency relighting. *ACM Transaction on Graphics* **23**(3), 477–487 (2004)
10. Nielson, G.M., Jung, I.H., Sung, J.: Haar wavelets over triangular domains with applications to multiresolution models for flow over a sphere. In: Proceedings of the 8th conference on Visualization 1997, pp. 143–149 (1997)
11. Ramamoorthi, R., Hanrahan, P.: An efficient representation for irradiance environment maps. In: Proceedings of SIGGRAPH 2001, pp. 497–500 (2001)
12. Schröder, P., Sweldens, W.: Spherical wavelets: efficiently representing functions on the sphere. In: Proceedings of SIGGRAPH 1995, pp. 161–172 (1995)
13. Schröder, P., Sweldens, W.: Spherical wavelets: texture processing. In: Proceedings of EGWR 1995, pp. 252–263 (1995)
14. Sloan, P.P.: Normal mapping for precomputed radiance transfer. In: Proceedings of I3D 2006, pp. 23–26 (2006)
15. Sloan, P.P., Hall, J., Hart, J., Snyder, J.: Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics* **22**(3), 382–391 (2003)
16. Sloan, P.P., Kautz, J., Snyder, J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In: Proceedings of SIGGRAPH 2002, pp. 527–536 (2002)
17. Sloan, P.P., Luna, B., Snyder, J.: Local, deformable precomputed radiance transfer. *ACM Transactions on Graphics* **24**(3), 1216–1224 (2005)
18. Tsai, Y.T., Shih, Z.C.: All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. In: Proceedings of SIGGRAPH 2006 (to appear)
19. Wang, R., Tran, J., Luebke, D.: All-frequency relighting of non-diffuse objects using separable BRDF approximation. In: Proceedings of EGSR 2004, pp. 345–354 (2004)
20. Wang, R., Tran, J., Luebke, D.: All-frequency interactive relighting of translucent objects with single and multiple scattering. *ACM Transactions on Graphics* **24**(3), 1202–1207 (2005)
21. Wang, R., Tran, J., Luebke, D.: All-frequency relighting of glossy objects. *ACM Transactions on Graphics* (to appear)
22. Wang, Z., Leung, C.S., Zhu, Y.S., Wong, T.T.: Data compression with spherical wavelets and wavelets for the image-based relighting. *Computer Vision and Image Understanding* **96**(3), 327–344 (2004)



**Fig. 14** Rendering results of objects with complex materials under different viewing and lighting conditions. Both the *XYZRGB Dragon* model and the floor are mapped with SBRDFs. In addition, the floor is bump mapped.



**Wan-Chun Ma** is a Ph.D. student in Communication and Multimedia Laboratory, Department of Computer Science and Information Engineering at National Taiwan University. He received his B.S. degree in Computer Science and Information Engineering from National Taiwan University in 2000. His research interests include real-time rendering, image-based rendering and modeling, GPU programming.



**Yung-Yu Chuang** is an Assistant Professor in the Department of Computer Science and Information Engineering at National Taiwan University. He received his B.S. and M.S. from National Taiwan University in 1993 and 1995 respectively, Ph.D. from University of Washington in 2004, all in Computer Science. His research interests includes real-time and realistic rendering, digital photography and computer vision.



**Chun-Tse Hsiao** is a Ph.D. student in Communication and Multimedia Laboratory, Department of Computer Science and Information Engineering at National Taiwan University. He received his B.S. in Computer Science from National Chung Cheng University in 2004. His research interests include computer vision and graphics. He is a student member of ACM SIGGRAPH.



**Bing-Yu Chen** received the B.S. and M.S. degrees in Computer Science and Information Engineering from the National Taiwan University, Taipei, in 1995 and 1997, respectively, and received the Ph.D. degree in Information Science from the University of Tokyo, Japan, in 2003. He is currently an assistant professor in the Department of Information Management and the Graduate Institute of Networking and Multimedia of the National Taiwan University since 2003. His research interest are mainly for computer graphics, geometric modeling, computer animation, web and mobile graphics. He is a member of ACM, ACM SIGGRAPH, Eurographics, IEEE, IEICE, and IICM.



**Ken-Yi Lee** is a M.S. student in Communication and Multimedia Laboratory, Department of Computer Science and Information Engineering at National Taiwan University. He received his B.S. degree in Computer Science and Information Engineering from National Taiwan University in 2005. His research interests include real-time rendering and image processing.