Real-Time All-Frequency Relighting in Local Frame

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

National Taiwan University

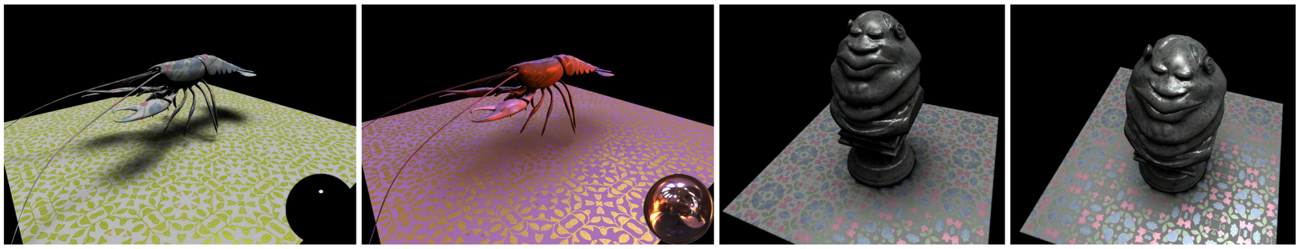
Figure 1 Sample results using our technique. Our method can handle complex materials, varying illumination and changing view. Here, models and the floors are mapped with SBRDFs. Illumination and viewpoint can be changed in real-time. From left to right, we demonstrate the effects of illumination change using an area light and the Grace Cathedral for the lobster model, and effects of changing view using two different views for the statue model.

1 Introduction

We address the problem of real-time rendering for objects with complex materials under varying all-frequency illumination and changing view. Our approach is based on the triple product algorithm proposed by Ng *et al.* [2004] with the following extensions:

The use of local frame for shading. We change the coordinate system for shading computation from the global frame used in previous papers [Ng et al. 2004] 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 viewing or lighting condition is changed. Storing BRDF in a local frame 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.

The use of spherical wavelets. Since functions involved in the rendering equation, illumination, visibility and BRDF, are all spherical functions, it makes more sense to represent them using spherical wavelets [Schröder and Sweldens 1995] to avoid uneven sampling and energy normalization for cubical parameterization.

The use of per-pixel shading and visibility textures for efficient GPU implementation. A fine tessellation is typically required to capture the visibility variation over a surface even if the surface is flat since rendering is performed on a per-vertex base. 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*. By doing so, a fine tessellation is replaced with a coarse mesh plus a visibility texture, along with per-pixel shading, resulting in an efficient GPU implementation.

With these extensions, 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 in real-time on modern graphics hardware.

2 Algorithm and Results

As shown by Ng *et al.* [2004], the reflection equation can be written in terms of basis functions $\Psi(\omega)$,

$$B = \sum_i \sum_j \sum_k L_i V_j \rho_k \int_\Omega \Psi_i(\omega)\Psi_j(\omega)\Psi_k(\omega)d\omega, \tag{1}$$

where L_i, V_j and ρ_k are coefficients for illumination, visibility and BRDF respectively.

Pre-computation. We choose to use spherical wavelets as the basis functions Ψ in Equation 1. An n-subdivided icosahedron is used for spherical function projection and a discrete spherical wavelet transform is applied to generate wavelet coefficients. Note that the BRDF and visibility functions are sampled in the local frame. Hence, the global illumination function needs to be rotated into different local frames for different vertices during rendering. Unfortunately, we are not aware of any efficient algorithm to rotate spherical wavelet coefficients. Instead, we generate many rotated versions of the illumination function and their spherical wavelet coefficients by uniformly sampling Euler angles. In order to convert per-vertex shading into per-pixel shading, the key is to store the visibility coefficients in a texture. Hence, for each texel p_t in the texture space, we find its corresponding point p_m in the model space. Then, we use p_m and its local frame to sample the visibility. After applying spherical wavelet transform on the visibility function, we store the coefficients back to p_t.

Rendering. The pre-computation stage outputs three textures for illumination, visibility, and BRDF functions in wavelet space. During the rendering pass, a pixel shader simply retrieves coefficients L_i, V_j and ρ_k from the above three textures, and applies the triple product algorithm to evaluate Equation 1. Spatially-variant BRDF can be done easily by having different basis BRDF textures and interpolating BRDF coefficients by weighted summation. For bump mapping effect, we rotate the lighting function using the perturbation indicated by the bump map.

All the rendering computation is done by GPU. We use 80 scaling wavelet coefficients for each function. The final texture size for BRDF and illumination are 1MB and 6MB respectively. The size of visibility texture depends on the resolution of texture (current setting produces around 20MB data). The rendering frame rates are between 50 to 100 FPS on an ATIX1900 GPU. Figure 1 shows results using our technique.

References

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple product wavelet integrals for all-frequency relighting. *ACM Transaction on Graphics 23*, 3, 477–487.

SCHRÖDER, P., AND SWELDENS, W. 1995. Spherical wavelets: efficiently representing functions on the sphere. In *Proceedings of ACM SIGGRAPH 1995*, 161–172.