Analytic Double Product Integrals for All-Frequency Relighting

Rui Wang, Minghao Pan, Weifeng Chen, Zhong Ren, Kun Zhou, Wei Hua, Hujun Bao*

Abstract—This paper presents a new technique for real-time relighting of static scenes with all-frequency shadows from complex lighting and highly specular reflections from spatially-varying BRDFs. The key idea is to depict the boundaries of visible regions using piecewise linear functions, and convert the shading computation into double product integrals – the integral of the product of lighting and BRDF on visible regions. By representing lighting and BRDF with spherical Gaussians and approximating their product using Legendre polynomials locally in visible regions, we show that such double product integrals can be evaluated in an analytic form. Given the precomputed visibility, our technique computes the visibility boundaries on the fly at each shading point, and performs the analytic integral to evaluate the shading color. The result is a real-time all-frequency relighting technique for static scenes with dynamic, spatially-varying BRDFs, which can generate more accurate shadows than the state-of-the-art real-time PRT methods.

Index Terms—analytic double product integral, all frequency relighting.

1 Introduction

All-frequency direct illumination effects, such as sharp shadows from environmental lighting and highly specular reflections from spatially-varying BRDFs, are very important for realistic image synthesis. These effects, however, are difficult to achieve in interactive applications due to the high computational cost caused by the interactions of complex lights, occluders and materials.

During the past few decades, Precomputed Radiance Transfer (PRT) [1], [2] has achieved great success in interactive rendering with natural lighting and shadowing for static scenes by precomputing the objects' response to lighting represented in basis functions. In particular, the *triple product integral* technique proposed by Ng et al. [3] provides a framework for fast rendering of sharp shadows and specularities with changing view. In this framework, the lighting, visibility and BRDF are all represented in wavelets, and the outgoing radiance is computed as a triple product integral of the three terms. Wang et al. [4] generalized this framework to render dynamic, spatially-varying BRDFs by using spherical Gaussians as the basis functions. Real-time frame rates are achieved by approximating the triple product (*i.e.*, attenuating the product of lighting and specular BRDF by the ambient term of visibility), leading to inaccurate shadows cast on glossy surfaces (see Fig. 9).

In this paper, we aim at real-time relighting of static scenes with all-frequency shadows from complex lighting and highly specular reflections from spatially-varying BRDFs. Our main contribution is an *analytical double product integral* technique for computing the shading integral of the product of lighting, visibility and BRDF. Unlike previous PRT methods that represent visibility in basis functions, our technique depicts the boundaries of visible regions using piecewise linear functions, and computes the shading as an integral of the product of lighting and BRDF on visible regions described by the boundary functions. By representing lighting and BRDF with spherical Gaussians and approximating their product using Legendre polynomials locally in visible regions, we show that these double product integrals can be evaluated analytically in a closed form of the visibility boundary parameters and the coefficients of lighting and BRDF approximations. Given the precomputed visibility, our technique computes the visibility boundaries on the fly at each shading point, and performs the analytic integral to evaluate the shading color. The result is a real-time all-frequency relighting technique for static scenes with spatially-varying BRDFs, which can produce more faithful shadows than the state-of-the-art all-frequency PRT method [4].

In the rest of the paper, we first briefly review related work. In Section 3, we introduce the analytical double product integral formulation, followed by the description of the visibility evaluation algorithms in Section 4. Section 5 details the rendering algorithm and Section 6 gives experimental results. The paper concludes with some discussions about limitations and future work in Section 7.

2 Related Work

This paper focuses on interactive rendering of spatially-varying BRDFs under environmental lighting. Here we briefly review techniques that are most relevant to our work.

• *The authors are all with the State Key Lab of CAD&CG, Zhejiang University, Hangzhou, 310058, P.R.China.*
Corresponding author: Hujun Bao, bao@cad.zju.edu.cn

(a) (b) (c) (d)

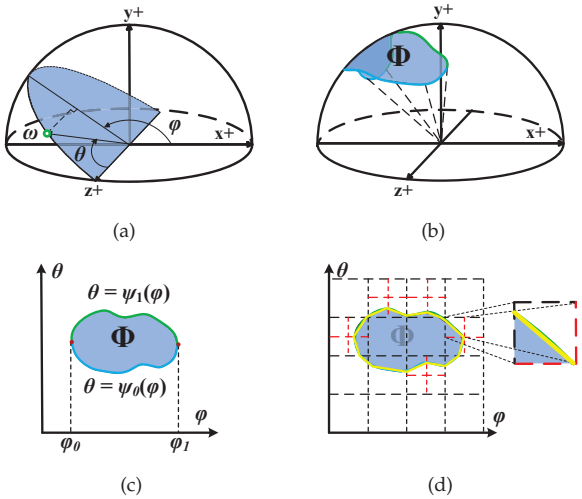
Fig. 1: Results rendered at 10-23 fps using our technique under complex lighting. Interactive relighting of static scenes with dynamic, spatially varying BRDFs are supported, and the shadowing effect is more accurate than the start-of-the-art interactive techniques.

PRT achieves realtime rendering under dynamic environment lighting by precomputing and storing object's response to lighting in a low order spherical harmonics (SH) basis [1] or a wavelet basis [2]. A lot of follow-on techniques have been proposed to render glossy effects with changing view [3], [5], [6], dynamic BRDFs [7], [8], [9], and soft shadows in dynamic scenes with moving or deformable objects [10], [11], [12]. See the recent survey by Ramamoorthi [13] for a comprehensive review.

The triple product wavelet integral technique proposed by Ng et al. [3] is able to render all-frequency shadows and highly-specular reflections at interactive frame rates without any restrictions on viewing directions. It represents the lighting, visibility and BRDF in a wavelet basis. Due to the huge data size of wavelet-represented BRDFs, it is not practical to handle dynamic and spatially varying reflectance. Recently, inspired by the Lightcuts approach [14], [15], Cheslack-Postava et al. [16] proposed visibility cuts to compute the triple product integral of lighting, visibility and dynamic BRDF samples to achieve interactive per-pixel shading. This method, however, cannot handle highly-specular surfaces.

More recently, Wang et al. [4] proposed a real-time all-frequency rendering algorithm for spatially-varying BRDFs. They also employ the triple product integral framework, but used different basis functions (*i.e.*, spherical Gaussians), which are more compact, allow detailed textures, and are closed under products and rotations. To achieve real-time frame rates, the triple product is approximated by attenuating the product of lighting and specular BRDF by the ambient term of visibility, leading to inaccurate shadows cast on glossy surfaces.

Xu et al. [17] and Wang et al. [4] represent spatially-varying visibility with a nonlinear representation called *spherical signed distance function* (SSDF), which is stored at each mesh vertex and is interpolated per-pixel over triangles. The benefit of this representation is that it provides ghost-free interpolation. We also use SSDFs, but only as an intermediate representation. At runtime, a boundary extraction method is performed to extract the piecewise-linear boundary functions of visibility boundaries from SSDFs.



(a) (b) (c) (d)

Fig. 2: (a) The hemispherical coordinate of ω, *i.e.* $\omega = (\cos\varphi\sin\theta, \sin\varphi\sin\theta, \cos\theta)$. The y^+ axis is taken along the normal. (b) An example visible region on the hemisphere. (c) The visible region in the (θ, φ) space. (d) The subdivided visible region.

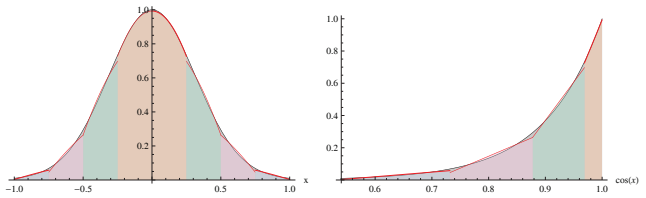
3 Analytic Double Product Integrals

In this section, we formulate the shading computation as a sum of analytical double product integrals.

The direct lighting at point $\mathbf{x}$ in viewing direction ω_o can be computed by the following integral:

$$L_o(\mathbf{x}, \omega_o) = \int_{\Omega_{2\pi}} L_i(\mathbf{x}, \omega) f(\mathbf{x}, \omega, \omega_o) V(\mathbf{x}, \omega) (\mathbf{n} \cdot \omega) d\omega, \quad (1)$$

where L_o is the outgoing radiance, L_i is the incident lighting, $f(\mathbf{x}, \omega, \omega_o)$ is the spatially-varying BRDF, $V(\mathbf{x}, \omega)$ is the visibility function, and $(\mathbf{n} \cdot \omega)$ is the cosine term. The visibility is a binary function that indicates whether the incident direction ω is blocked by scene geometry or not. For notation simplicity, we will omit the dependency on the shading point $\mathbf{x}$ hereafter.



(a) (b)

Fig. 3: The linear approximation of an 1D Gaussian. The example Gaussian is $G(x) = e^{10(\cos(x)-1)}$, where $\cos(x) = p \cdot \omega$. We uniformly partition the Gaussian in the angular space, i.e. x space, and use linear functions defined in $\cos(x)$ space, i.e. $\alpha_i + \beta_i \cos(x)$, to approximate the Gaussian. (a) The range $[-1, 1]$ in the x space is uniformly partitioned into eight regions. In each region, the Gaussian is approximated by a linear function, depicted as a red curve. (b) The approximation is a piecewise linear function in the $\cos(x)$ space.

Suppose that (θ, φ) is the hemispherical coordinate of direction ω, *i.e.* $\omega = (\cos\varphi\sin\theta, \sin\varphi\sin\theta, \cos\theta)$, where the normal is taken along the y-axis, Fig. 2(a). Let us consider the integral on a visible region Φ, which is enclosed by two boundary curves, $\theta = \psi_0(\varphi)$ and $\theta = \psi_1(\varphi)$, defined within the range of φ_0 and φ_1. See Fig. 2 (b) and (c) for an illustration. The integral can thus be computed as

$$L_o^{\Phi}(\omega_o) = \int_{\varphi_0}^{\varphi_1} \int_{\psi_0(\varphi)}^{\psi_1(\varphi)} L_i(\omega) f(\omega, \omega_o)(\mathbf{n} \cdot \omega) \sin\theta d\theta d\varphi. \quad (2)$$

This is a double product integral of the lighting and BRDF since the visibility function is no longer in the integral. We can compute the outgoing radiance L_o by adding up the integrals on all visible regions. In the following, we first introduce a cell-based visibility boundary representation, and then give the analytic form of the double product integral.

Visibility Boundary Representation We subdivide the complex visible regions into simpler cells within which double product integrals are computed. As shown in Fig. 2(d), the 2D space spanned by (θ, φ) is recursively subdivided into cells until the boundary of the visible region within each cell can be accurately approximated by a linear function, *i.e.*, $\theta = \psi^j(\varphi) = k^j\varphi + b^j$, for the j-th cell. We will detail the subdividing algorithm in Section 4. For now let us assume the visible regions are readily subdivided.

As illustrated in Fig. 2(d), the visible region within a cell is bounded by the cell boundaries and the linear function. Therefore, Eq. (2) can be written as

$$L_o^j = \int_{\varphi_0^j}^{\varphi_1^j} \int_{\theta_0^j}^{k^j\varphi + b^j} L_i(\omega) f(\omega, \omega_o)(\mathbf{n} \cdot \omega) \sin\theta d\theta d\varphi, \quad (3)$$

where φ_0^j and φ_1^j are the minimum and maximum values of φ of the cell respectively, and θ_0 is the minimum or maximum of θ of the cell, depending on the relative position between the visible region and the linear function. The overall shading integral on visible regions is thus the sum of the double product integrals on all cells. To make the notation clearer, we will drop the j superscript in the following discussion.

Analytic Double Product Integral Following [4], we represent the incident lighting and BRDF in spherical Gaussians

$$\begin{array}{rcl} L_i(\omega) & = & \sum_l G_l(\omega; \mathbf{p}_l, \lambda_l, \mu_l), \\ f(\omega, \omega_o) & = & \sum_m G_m(\omega; \mathbf{p}_m^{\omega_o}, \lambda_m^{\omega_o}, \mu_m^{\omega_o}). \end{array} \quad (4)$$

G has the following form

$$G(\omega; \mathbf{p}, \lambda, \mu) = \mu e^{\lambda(\mathbf{p} \cdot \omega - 1)},$$

where $\mathbf{p}$ is the lobe direction, λ is the lobe sharpness and μ is the lobe amplitude.

Substituting Eq. (4) into Eq. (3), we get

$$L_o = \int_{\varphi_0}^{\varphi_1} \int_{\theta_0}^{k\varphi+b} \sum_{l,m} G_l(\omega) G_m(\omega)(\mathbf{n} \cdot \omega) \sin\theta d\theta d\varphi. \quad (5)$$

According to the derivation in [4], the vector product of two spherical Gaussians yields another spherical Gaussian

$$G(\omega; \mathbf{p}_1, \lambda_1, \mu_1) G(\omega; \mathbf{p}_2, \lambda_2, \mu_2) = G(\omega; \frac{\mathbf{p}'}{||\mathbf{p}'||}; \lambda'||\mathbf{p}'||; \mu'),$$

where $\mathbf{p}' = (\lambda_1\mathbf{p}_1 + \lambda_2\mathbf{p}_2)/(\lambda_1 + \lambda_2)$, $\lambda' = \lambda_1 + \lambda_2$ and $\mu' = \mu_1\mu_2 e^{\lambda'(\|\mathbf{p}\|-1)}$.

Therefore, we can rewrite the integral in Eq. (5) as

$$L_o = \int_{\varphi_0}^{\varphi_1} \int_{\theta_0}^{k\varphi+b} \sum_{l,m} G_{l,m}(\omega)(\mathbf{n} \cdot \omega) \sin\theta d\theta d\varphi. \quad (6)$$

Now we approximate the spherical Gaussian *locally* within each cell using linear functions (an approximation in 1D case is visualized in Fig.3).

$$G(\omega; \mathbf{p}, \lambda, \mu) \approx \alpha + \beta(\mathbf{p} \cdot \omega). \quad (7)$$

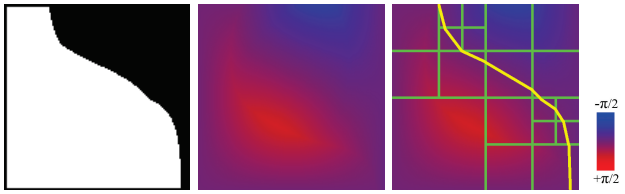
Substituting Eq. (7) into Eq. (6), we get

$$\begin{array}{l} L_o \approx \int_{\varphi_0}^{\varphi_1} \int_{\theta_0}^{k\varphi+b} \sum_{l,m} (\alpha_{l,m} + \beta_{l,m}(\mathbf{p}_{l,m} \cdot \omega)) \\ \qquad\qquad \cdot (\mathbf{n} \cdot \omega) \sin\theta d\theta d\varphi \\ = \int_{\varphi_0}^{\varphi_1} \int_{\theta_0}^{k\varphi+b} (\alpha^* + (\mathbf{p}^* \cdot \omega)) \cdot (\mathbf{n} \cdot \omega) \sin\theta d\theta d\varphi, \end{array} \quad (8)$$

where $\alpha^* = \sum_{l,m} \alpha_{l,m}$, $\mathbf{p}^* = \sum_{l,m} \beta_{l,m}\mathbf{p}_{l,m}$. It is true, of course, that a higher order approximations would be more accurate. However, the first order approximation leads to a simple analytic form for the integration in Eq. 8 that we can sum different lobe centers, $\mathbf{p}_{l,m}^*$, to a new direction $\mathbf{p}^*$ before taking the dot operation.

It can be easily proven that the integral in Eq. (8) has an analytical solution, and can be evaluated as

$$L_o \approx \alpha^* A(\varphi_0, \varphi_1, \theta_0, k, b, \mathbf{n}) + B(\varphi_0, \varphi_1, \theta_0, k, b, \mathbf{n}, \mathbf{p}^*), \quad (9)$$



(a) Visibility (b) Distance field (c) Cells and visibility boundary

Fig. 4: (a) The original sampled binary visibility function. (b) The SSDF of such a visibility. (c) The extracted adaptive boundaries from the SSDF. The SSDF is represented in (θ, φ) space. The red color indicates positive distance value and blue color indicates negative one.

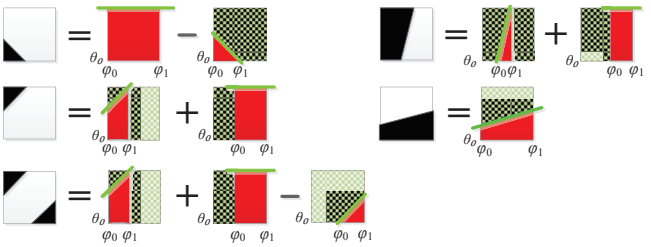


Fig. 5: In each of the above cases, black refers to occluded region. Each term on the right of a equal sign corresponds to one double product integral. The equation for the green line is given by $\theta = k\varphi + b$ and the integral on each red region can be computed according to Eq. (3).

where $A(\varphi_0, \varphi_1, \theta_0, k, b, \mathbf{n})$ and $B(\varphi_0, \varphi_1, \theta_0, k, b, \mathbf{n}, \mathbf{p}^*)$ are analytical functions of parameters of $\varphi_0, \varphi_1, \theta_0, k, b, \mathbf{n}$ and $\mathbf{p}$. Please refer to the appendices for the exact formulas and derivation.

4 Visibility Evaluation

In this section, we describe how to compute the visibility boundary for static scenes.

We use the spherical signed distance function (SSDF) [4] as an intermediate visibility representation. The SSDF at a surface point stores the signed angular distance to the closest visibility boundary at each direction ω (see Fig. 4). The function's sign encodes whether the direction ω is occluded or not. More precisely, the SSDF is defined as:

$$D(\omega) = \begin{cases} + \min\limits_{V(\omega')=0} \arccos(\omega' \cdot \omega), \text{if } V(\omega) = 1 \\ - \min\limits_{V(\omega')=1} \arccos(\omega' \cdot \omega), \text{if } V(\omega) = 0 \end{cases},$$

Compared to the piece-wise linear visibility representation, SSDFs are easier to interpolate and compress.

Given a static scene, we first precompute the binary visibility function at each surface vertex and convert it to an SSDF. The precomputed SSDFs are then compressed using PCA and stored at vertices as that in [4]. At runtime, the SSDF at each shading point is obtained via interpolation over the triangle which the point lies in, and is used to extract the visibility boundaries.

Visibility Boundary Extraction The visibility boundaries of the scene might be very complex. To extract the visibility boundaries from an SSDF, we employ an iterative subdivision scheme in the (θ, φ) space.Our approach is similar to that used in [18]. The (θ, φ) space is recursively subdivided into smaller cells until the linear boundary extracted in each cell is a sufficiently accurate approximation of the actual visibility boundary determined by the SSDF. The approximation accuracy is controlled by a user-defined threshold.

For each cell, we first extract an isoline from the SSDF using the marching squares algorithm, the 2D analog of the marching cubes algorithm [19], then evaluate the closeness between the extracted line and the actual visibility boundary. This is done by summing up at a set of sample points within the cell the squared difference between the distance reconstructed from the extracted line and the exact distance. In our current implementation, we use the squared differences at sample points in the cell – the cell center and the midpoints of cell edges – to determine further subdivision.

If the sum of the squared differences is greater than a threshold, indicating that the extracted line does not match the actual boundary well, we subdivide the current cell into four equally-sized sub-cells. This subdivision scheme will be described in details in Section 5.

Determine Integration Ranges: Given the linear boundary function of a cell, the integration ranges in Eq. (5) can be determined by the intersections as well as the relative position between the line and the cell. Five basic cases of visibility regions in a cell are depicted in Fig. 5. In each case, the double product integral can be decomposed into integrals formulated in Eq. (3) and computed respectively. All cases of the isoline and cell can be derived from one of the above cases.

5 Rendering

During a preprocess, the environment lighting and BRDFs are approximated with a set of spherical Gaussians using nonlinear optimization [20]. And the SSDF is precomputed at each vertex of the mesh and compressed using PCA.

The runtime rendering pipeline consists of three parts: deep frame buffer generation, visibility boundary extraction and analytic double product integral accumulation.

Deep frame buffer generation. Shading computation is performed per pixel. The position, local coordinate frame, texture coordinate, material index and

PCA coefficients for the SSDF are first interpolated from the vertices of scene objects and rasterized into a deep frame buffer.

Visibility boundary extraction. The adaptive algorithm described in Section 4 is implemented in CUDA and the pseudo-code is given in Algorithm 1. The SSDF at the shading point is obtained by a dot product of the PCA coefficients and the eigenvector textures. The whole boundary extraction process is performed in several iterations. In each iteration, cells in $active\_cells$ list are processed in batch. Not all cells are subdivided. We only extract visibility boundaries for cells that are neither completely inside nor outside the visible region. To determine the relationship between cells and visible regions, we take a simple but conservative test. We compute the SSDF distance at the cell center which is compared with the spherical distances from the cell center to cell corners. If the center's absolute SSDF value is greater than all distances to cell corners, the closest visible region boundary must be completely outside the cell, which means we can compute the double product integral over the whole cell. Otherwise, the cell is considered to be subdivided into smaller cells which are stored in $next\_cells$ for the next iteration.

The subdivision process is determined by two factors. One is e_r, the closeness between the extracted line and the actual visibility boundary as described in Section 4. The other one is e_a, the approximation error of our linear approximation. While the nonlinearity of spherical Gaussians may lead to inaccurate approximations in large cells, they can be locally approximated very well in small cells. The analytic expression of the approximation error is given in Appendix A. For each vector product of the spherical Gaussians of the incident lighting and BRDF, we evaluate the approximation error in the cell, and subdivide the cell if the error exceeds a threshold. e_r and e_a are evaluated sequentially and subdivision will be triggered once one error test fails.

Analytic double product integral. The shading integral is computed according to Eq. (8). The spherical Gaussians used to approximate the environment lighting and BRDF are iterated, and their vector product Gaussians are determined and their parameters are substituted into Eq. (9) to compute the shading integral on each cell. The contributions of all cells are finally summed up to yield the overall shading integral at the shading point.

Algorithm 1 Adaptive Visibility Boundary Extraction

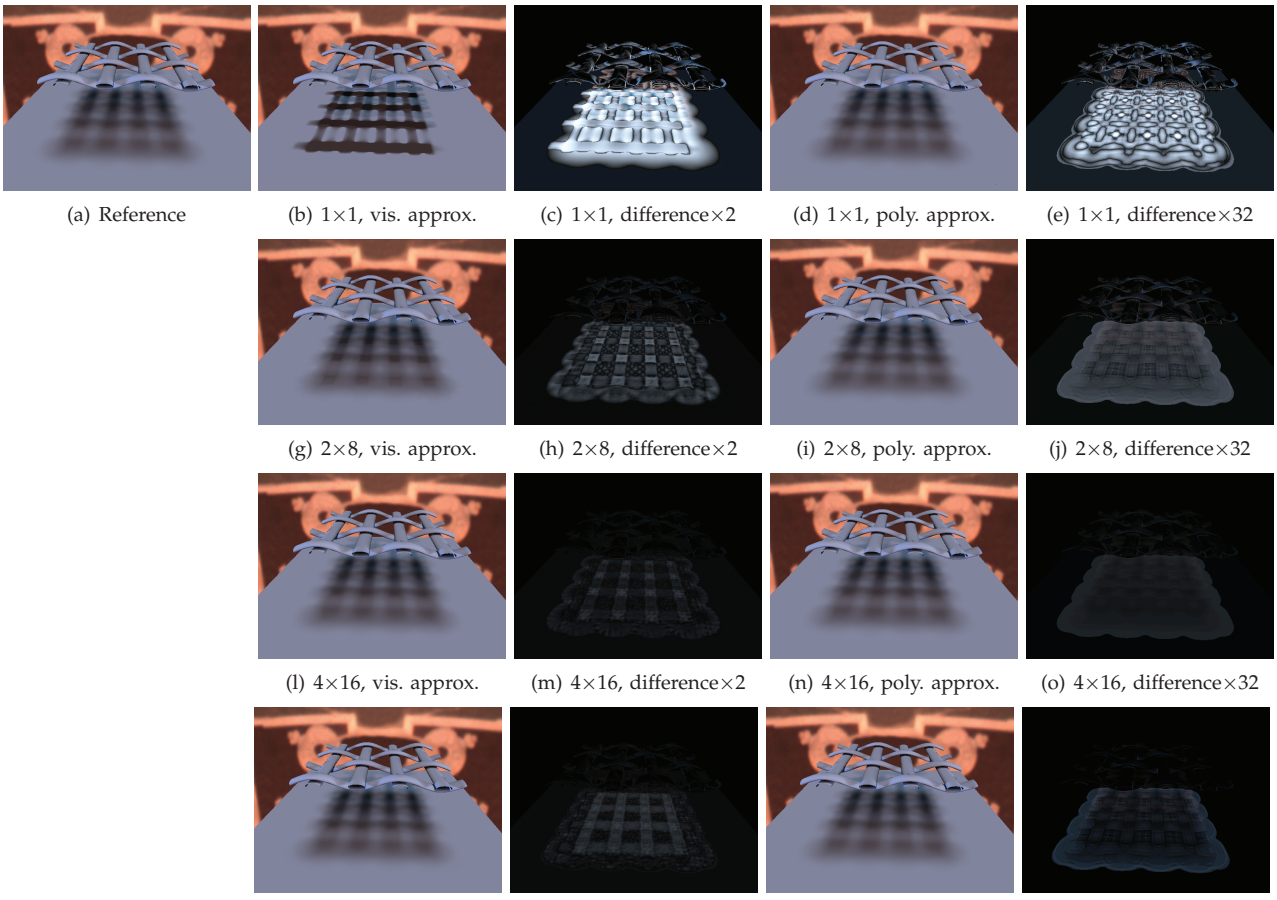
```
//SPLITCELL(parentCell, nφ, nθ): divides
//        parentCell into nφ × nθ cells uniformly
//        along the θ and φ axes.
//CELLRADIUS(cell): calculates the radius of cell.
//EVALUATEEr(cell,l_iso) computes the reconstruc-
//        tion error of isoline l_iso in cell.
//EVALUATEEa(cell,lobes) computes the approxi-
//        mation error of lobes in cell.

procedure VISIBILITYBOUNDARYEXTRACTION()
 in in_lobes,  //in_lobes list of all lobes of pixels.
 in in_cells,  //in_cells list of initial cells of pixels.
 out out_cells, //out_cells stores cells for taking
                //  analytic double product integral.
begin
  active_cells ← in_cells
  next_cells ← new list
  // adaptive cell partition
  for level j = 0 to maxLevel − 1
    next_cells.clear()
    for each cell i in active_cells in parallel
      // sample at cell center
      dis_c ← sample SSDF at center of cell_i
      radius ← CELLRADIUS(cell_i)
      if dis_c >= radius then
        // cell_i is entirely in visible region
        out_cells.add(cell_i,NULL)
      else if dis_c > −radius || dis_c < radius then
        if j==0 then
          //generate 2 × 4 top-level cells
          next_cells.add(SPLITCELL(cell_i, 2, 4))
        else
          l_iso = EXTRACTISOLINE(cell_i)
          e_r = EVALUATEEr(cell_i,l_iso)
          e_a = EVALUATEEa(cell_i,in_lobes)
          if e_r > t_r || e_a > t_a then
            //errors exceed thresholds
            next_cells.add(SPLITCELL(cell_i, 2, 2))
          else
            out_cells.add(cell_i, l_iso)
    active_cells ← next_cells
end
```

5.1 Optimizations and Implementation Details

Narrow spherical Gaussians often require intensive subdivision and generate a large number of cells, leading to high rendering cost. We separately process the narrow Gaussians with $\lambda > 400$ from the result of the nonlinear optimization [20]. This set of narrow spherical Gaussians is referred to as Λ.

Let us first consider the diffuse component of the BRDF. The vector product of a spherical Gaussian with the diffuse component of the BRDF only changes the lobe amplitude of the original Gaussian. For each Gaussian in Λ, we define the local coordinate system by aligning the z axis with its lobe direction. The energy of the Gaussian in Λ is centralized in a narrow band of $\theta < \theta_0$. In practice, we set the cut-off inclination θ_0 to preserve 95% of the Gaussian energy and initialize the subdivision in this narrow band of θ.

This optimization requires that a separate visibility



(a) Reference (b) 1×1, vis. approx. (c) 1×1, difference×2 (d) 1×1, poly. approx. (e) 1×1, difference×32

(g) 2×8, vis. approx. (h) 2×8, difference×2 (i) 2×8, poly. approx. (j) 2×8, difference×32

(l) 4×16, vis. approx. (m) 4×16, difference×2 (n) 4×16, poly. approx. (o) 4×16, difference×32

(q) Adaptive,vis. approx. 17.5 Ave. cells. (r) Adaptive,difference×2 (s) Adaptive,poly. approx. (t) Adaptive, difference×32

Fig. 6: Error analysis of the piecewise linear visibility boundary approximation ("vis. approx.") and the linear approximation of spherical Gaussians ("poly. approx.").

Scene	BRDF Type	#Pixels	#Verts	t_I	#Int.	FPS
Fig. 9(b)	Blinn-Phong (1 SG)	162486	46k	42ms	19.98	22.7
Fig. 1(b)	Ward (1 SG) and measured card (1 SG)	184364	28k	45ms	22.8	21.2
Fig. 1(a) / Fig. 9(c)	Ward (1 SG) and Cook-Torrance (1 SG)	173674	32k	58ms / 62ms	24.1 / 26.6	16.1 / 15.5
Fig. 1(c)	Measured card (1 SG) and stain (5 SGs)	111147	21k	63ms / 69ms	35.7 / 38.1	14.9 / 13.7
Fig. 1(d)	Ashikhmin-Shirley (7 SGs) and Phong (1 SG)	137799	30k	89ms	40.4	10.7

TABLE 1: Statistics of the test scenes. t_I is the total time to extract the visibility boundary and compute the analytic double product integrals. #Int. is the average integrals computed per pixel. We test our scenes under under environment lighting and local lights. Performances with local lighting are listed after the slash.

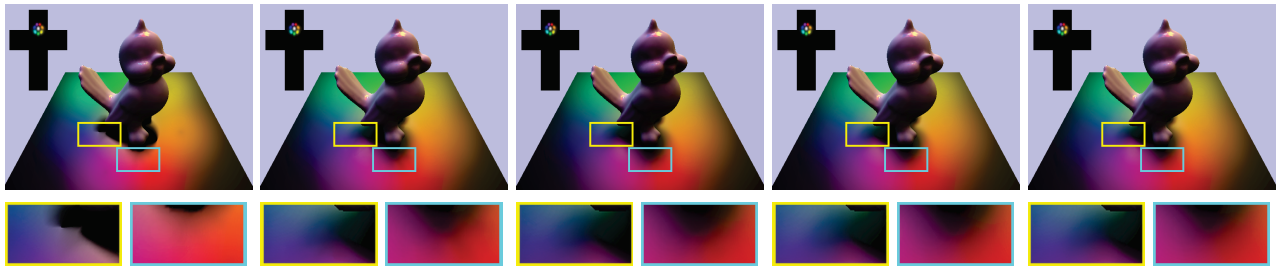
boundary extraction pass for each narrow Gaussian, and might not be a good trade-off if the support of the Gaussian is wide. Therefore, for all the Gaussians not in Λ, we still align the z axis with the normal of the sample point, and initialize the subdivision from a 1×4 grid of the entire $[0, \frac{\pi}{2}] \times [0, 2\pi]$ space. Since the local coordinate is independent of the spherical Gaussians, only one visibility boundary pass is needed for all these Gaussians.

For the specular component of the BRDF, we always align the z axis with the lobe direction of the Gaussian lobe of the BRDF. This is because the vector product of the lighting Gaussians and the BRDF Gaussians yields Gaussians whose lobe direction is near the lobe direction of the BRDF Gaussian and lobe sharpness λ is higher than the BRDF Guassian. We can also limit the subdivision of cells within this narrow band of the BRDF Guassian and cut off θ to preserve 95% of the lobe energy.

6 Experimental Results

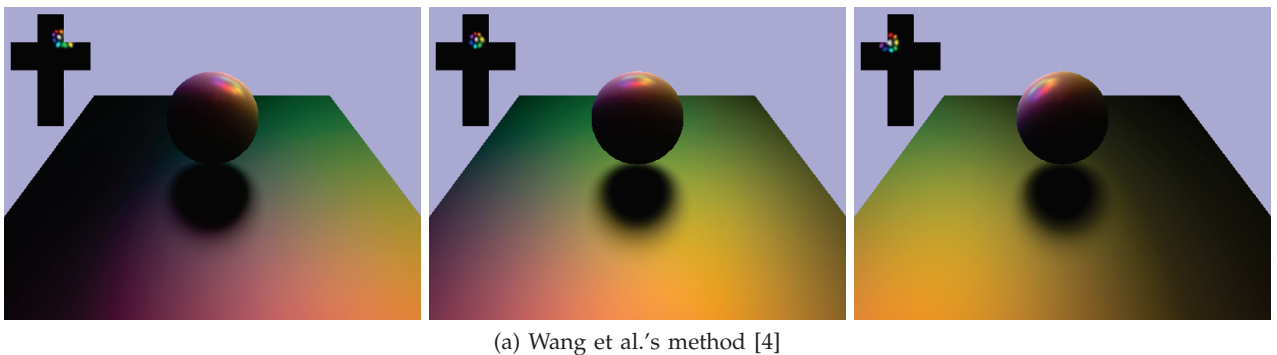
We implemented the described algorithm on a PC with Intel Core^{TM}2 Duo 2.8 GHz CPU and an NVIDIA GeForce 280 GTX graphics card. All images are generated at a 800×600 resolution.

The statistics of our test scenes are reported in Table 1. The spatial varying BRDF textures are obtained from the websites of the authors of [21], [22].

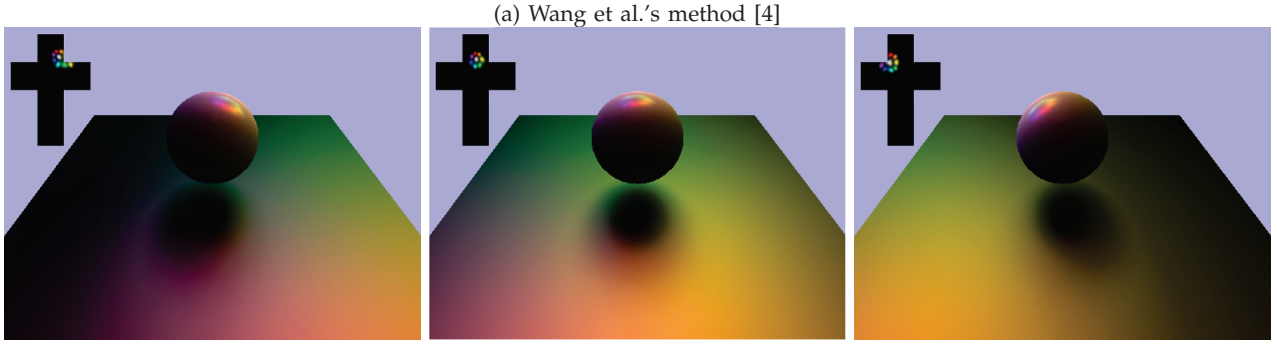


(a) 1×1, 4.23ms (b) 2×8, 21.1ms (c) 4×16, 65.1ms (d) Adaptive, 24.1ms (e) Reference

Fig. 7: Comparing our rendering results (d) with uniform partitioning at similar performance (b) and similar quality (c). The ray tracing reference (e) and the result obtained with only 1 cell are also provided. Note that the (a), (b), (c), (d) uses different ways to extract visibility boundary only for the plane under the tweety, and the reported timings only include the rendering of that plane. The average number of cells generated by the adaptive partitioning is 13.8 in this case.



(a) Wang et al.'s method [4]

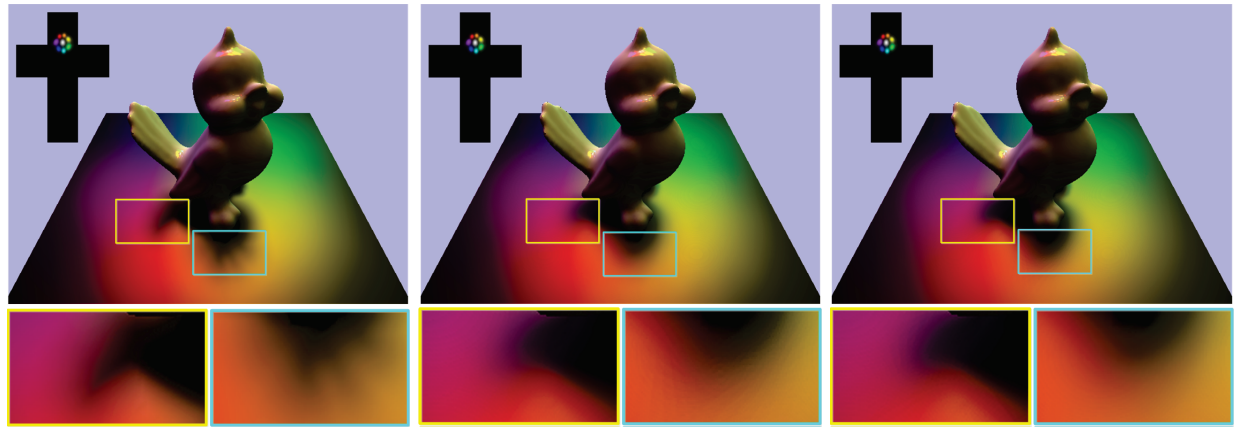


(b) Our method

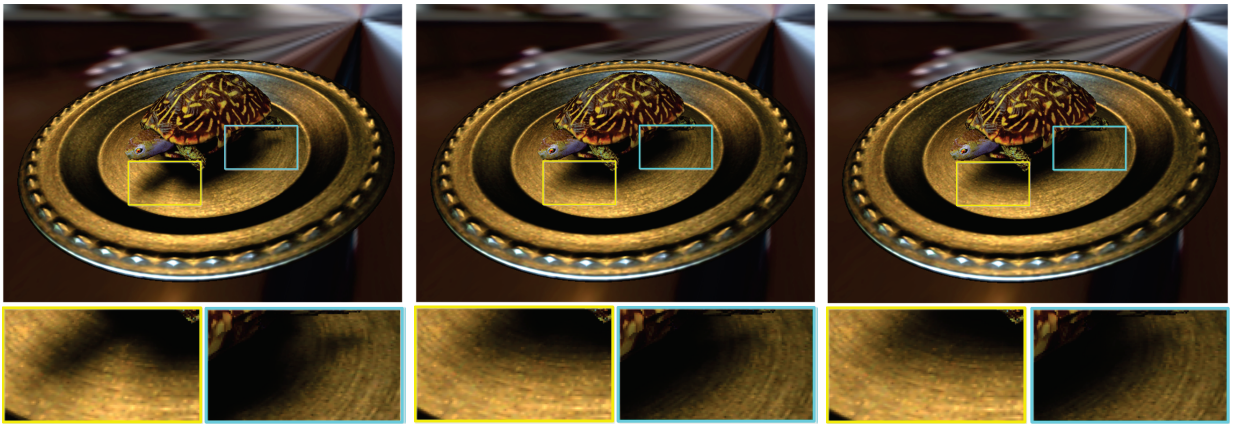
Fig. 8: Comparison of the shadow quality of Wang et al.'s method [4] and ours in a scene with a rotating environment light. The light rotates from right to left. Our method is able to capture the color transitions in the penumbra region, which are missed in the results of Wang et al.'s method.

Based on our analytic double product integral, we achieve real-time frame rates while handling dynamic viewing, lighting, reflectance and high-frequency, per-pixel shading. We precompute the SSDF at each vertex at a resolution of 128×128 and compress them by PCA of 10-30 eigenvector images. The precomputed data size is 0.5MB to 5MB, comparable with that of [4].

As shown in Fig. 6, we conducted an experiment to analyze the errors introduced by the two approximations taken in our approach, i.e. the piecewise linear approximation of visibility boundaries and the linear approximation of spherical Gaussians. We intentionally use diffuse objects to simplify the comparison – only the spherical Gaussians of lighting are approximated by polynomials. Note that as both approximations are done *locally* within the cells of visible regions, the errors heavily depend on the cell size. In our experiment, the integration domain (the (θ, φ) space) is divided into cells of three regular sizes (1×1, 2×8 and 4×16), and cells of adaptive sizes generated by our method (17.5 cells per pixel on average). In each case, two images are rendered which correspond to the two approximations. Note that when one approximation is taken, the other approximation is disabled. According to the results, both approximation errors decrease with the size of cells - the smaller the cell, the more accurate the approximations. Furthermore, it can be seen that the visibility boundary approximation produces



(a) Wang et al.'s method [4] (b) Our method (c) Reference



(d) Wang et al.'s method [4] (e) Our method (f) Reference

Fig. 9: Comparison of shadow quality of the method [4], ours and the reference generated by ray tracing.

(a) (b) (c) (d)

Fig. 10: Interactively changing the shininess of BRDFs of (a), (b)Ward model or (c), (d)Ashikhmin-Shirley model.

much larger errors than the linear approximation. This is consistent with our observation that the runtime adaptive cell subdivision process is mainly driven by visibility boundary errors.

The performance comparison of our adaptive visibility boundary extraction algorithm and that obtained by uniform partitioning is shown in Fig. 7. When almost the same number of cells are used, our result using adaptive partitioning (d), with 13.8 cells per pixel on average, better captures the shadowing effect than that obtained by uniform partitioning of 2×8 cells (b). The performance is almost identical because the additional cost introduced by adaptive partitioning is negligible in the overall rendering cost. To achieve similar quality as our method, 4×16 uniform cells are needed (c) and the performance is 2.7 times lower.

We compare our method with Wang et al. [4] in different scenes. In the first scene, a sphere is positioned above a specular plane under a moving environment light, which rotates from right to left. Shadows generated by [4] and ours under three different lighting positions are shown in Fig. 8. There are two main differences. First, it can be observed that our method captures the color transition of the shadows in the penumbra region, where there are smooth color

(a) (b) (c) (d)

Fig. 11: More rendering results of bump map editing ((a), (b)) and local light sources ((c), (d)).

changes from green, yellow to red. By contrast, no color transition in shadows is produced by [4]. Second, while the environment light rotates, the shadows generated by our method faithfully reflect the light changes, whereas that of [4] does not. The main reason is that in [4] the triple product integral is approximated by attenuating the product of lighting and specular BRDF by an ambient visibility term, which is not able to reliably capture the occlusion of all-frequency lighting. Our method better captures these effects by more accurately accounting for the visibility of the shading integral. More comparisons of shadow quality are shown in the accompanying video. Differences are more apparent in the video under continuous changes of the lighting.

In Fig. 9, we show more comparisons on the quality of shadows. In Fig. 9(a-c), we replace the sphere by a bird and use a plane with more specular reflectance. Fig. 9(d-f) show the shadows generated by a turtle under the "kitchen" environment light. It can be seen that our method more faithfully captures the all-frequency shadowing effect in these scenes than [4]. In [4], the shadowing effect may be underestimated for concave occluders, as can be seen from the shadows cast by the mouth and cheek in Fig. 9(a), or overestimated for convex occluders, as can be seen from the shadows cast by the head of the turtle in Fig. 9(d). This is because [4] approximates the occluders by a hemisphere shape determined by the nearest spherical distance from the lobe axis to the visibility boundary, an approximation that is too coarse to capture the shadows even of simple objects. On the other hand, to achieve higher quality, our method requires more computations and usually runs 3-5 times slower than [4] in our experiments. For example, [4] achieves 95 FPS for the scene shown in Fig. 9(a) and 62 FPS in Fig. 9(d), compared to 22.7 FPS in Fig. 9(b) and 16.7 FPS in Fig. 9(e) using our method. If we adjust the error thresholds in the cell subdivision process and produce less cells, our method can achieve 86 FPS in Fig. 9(b) at the cost of lower quality, allowing users to trade off between shadow quality and performance. We show more results to demonstrate the speed/quality trade-off achieved by our algorithm in Appendix D.

Our method supports spatially-varying BRDFs with multiple-lobe Fig. 1(c), as well as anisotropic material Fig. 1(d), which uses Ashikhmin-Shirley BRDF approximated by 7 spherical Gaussians. For multiple-lobes BRDFs, instead of aligning the z axis with each spherical Gaussian and running separate visibility extraction passes, we use the average direction of spherical Gaussian lobes as z axis to compute the double product integral.

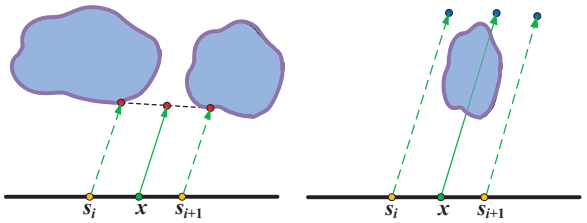
Material editing results are showin in Fig. 10 for objects with spatially varying Ward BRDF or Ashikhmin-Shirley BRDF. In (a), (b) the shadows on the ground becomes more blurry as the shininess decreases. And in (c), (d) we change the anisotropic degree from anisotropic to isotropic by moving multiple spherical Gaussian lobe directions. Please see the accompanying video for dynamic editing process.

More rendering results are shown in Fig.11. We can support interactive editing of the bump map ((a) and (b)). Local light source can also be naturally integrated as spherical Gaussians whose parameters varies with the shading point position ((c) and (d)), as in [4].

7 Discussion and Conclusion

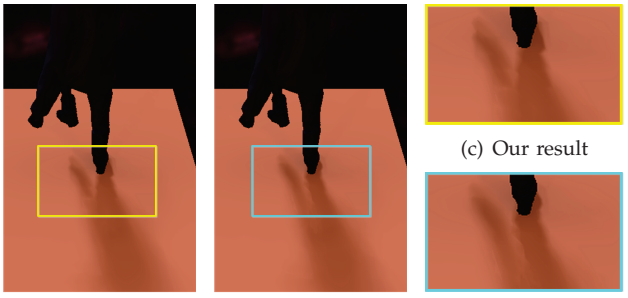
We have presented *analytical double product integrals*, a new technique for real-time relighting of static scenes with all-frequency shadows from complex lighting and highly specular reflections from spatially-varying BRDFs. Our technique represents visibility in the integration domain by depicting the boundaries of visible regions using piecewise linear functions, and convert the shading computation into an analytic form. The boundary representation of visibility is accurate and the analytic form of integrals is fast to compute. They make our technique generate more faithful shadows than the state-of-the-art PRT method while still allowing interactive performance. In contrast, previous PRT methods (e.g. [4], [17]) represent visibility in some basis functions instead of using boundaries, and need to compute triple product integrals when handling specular BRDFs.

Our method also bears some limitations. First, we compute and store visibilities at vertices and interpolate them for the shading at pixels. In this way, our method inherits the important limitation of the distance field representation in dealing with high-frequency visibility due to the interpolation of SSDFs.



(a) (b)

Fig. 12: Two failure cases that might be missed at interpolating the visibility are illustrated in (a) and (b).



(a) Our result (b) Reference (c) Our result (d) Reference

Fig. 13: The limitation of the visibility interpolation from SSDFs. (a)∼(d) show a comparison of results generated by our method and the reference ray tracing method. It can be seen that shadows generated by SSDF interpolation, shown in (a) with a close-up in (c), do not faithfully reflect the gap between legs.

In Fig. 12(a) and (b), we illustrate such two failure cases. The SSDF at shading point x is interpolated from two adjacent vertices, e.g. s_i and s_{i+1}. If the gap or the occluder is too small to be captured by the SSDFs at s_i and s_{i+1}, it will not be reflected in the interpolated SSDF at x. This results in incorrect shadows in the rendered images. We show an example in Fig. 13. The correct shadow cast from a walking man to the plane is shown in Fig. 13(b) with an enlarged detail image in Fig. 13(d). The shadow generated via the visibility interpolation are shown in Fig. 13(a) with an enlarged detail image in Fig. 13(c). Comparing these two images, it can be seen that the gap between legs is not correctly captured by the visibility interpolations. This limitation can be alleviated via more intensive sampling, which however results in more vertices and consumes more memory. Besides such interpolation errors, the adaptively sampled distance field technique used in our method may also lose some fine structures and details. In Fig. 6, we analyze the errors introduced by such approximations. It can be seen that smaller cell produce more accurate shadows. However, smaller cells will require more computation time both in extraction of visibility boundaries and integration of double product integrals.

We hope our analytical double product integral formulation would inspire further researches in all-frequency rendering, even for dynamic scenes. Another interesting direction is to apply our visibility representation to global illumination rendering.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their thoughtful comments. This work was supported in part by NSF of China (No. 60903037, No. 60825201 and No. 61003048) and the 973 program of China (No. 2009CB320803).

REFERENCES

[1] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 527–536, 2002.

[2] R. Ng, R. Ramamoorthi, and P. Hanrahan, "All-frequency shadows using non-linear wavelet lighting approximation," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 376–381, 2003.

[3] ——, "Triple product wavelet integrals for all-frequency relighting," *ACM Trans. Graph.*, vol. 25, no. 2, pp. 477–487, 2004.

[4] J. Wang, P. Ren, M. Gong, J. Snyder, and B. Guo, "All-frequency rendering of dynamic, spatially-varying reflectance," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 133:1–10, 2009.

[5] X. Liu, P. Sloan, H. Shum, and J. Snyder, "All-frequency precomputed radiance transfer for glossy objects," in *Eurographics Symposium on Rendering*, 2004, pp. 337–344.

[6] R. Wang, J. Tran, and D. Luebke, "All-frequency relighting of glossy objects," *ACM Trans. Graph.*, vol. 25, no. 2, pp. 293–318, 2006.

[7] A. Ben-Artzi, R. Overbeck, and R. Ramamoorthi, "Real-time brdf editing in complex lighting," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 945–954, 2006.

[8] X. Sun, K. Zhou, Y. Chen, S. Lin, J. Shi, and B. Guo, "Interactive relighting with dynamic brdfs," *ACM Trans. Graph.*, vol. 26, no. 3, p. 27, 2007.

[9] A. Ben-Artzi, K. Egan, F. Durand, and R. Ramamoorthi, "A precomputed polynomial representation for interactive brdf editing with global illumination," *ACM Trans. Graph.*, vol. 27, no. 2, pp. 1–13, 2008.

[10] K. Zhou, Y. Hu, S. Lin, B. Guo, and H.-Y. Shum, "Precomputed shadow fields for dynamic scenes," vol. 24, no. 3, 2005, pp. 1196–1201.

[11] P. Sloan, B. Luna, and J. Snyder, "Local, deformable precomputed radiance transfer," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1216–1224, 2005.

[12] Z. Ren, R. Wang, J. Snyder, K. Zhou, X. Liu, B. Sun, P.-P. Sloan, H. Bao, Q. Peng, and B. Guo, "Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 977–986, 2006.

[13] R. Ramamoorthi, "Precomputation-based rendering," *Found. Trends Comput. Graph. Vis.*, vol. 3, no. 4, pp. 281–369, 2009.

[14] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg, "Lightcuts: a scalable approach to illumination," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1098–1107, 2005.

[15] B. Walter, A. Arbree, K. Bala, and D. P. Greenberg, "Multidimensional lightcuts," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1081–1088, 2006.

[16] E. Cheslack-Postava, R. Wang, O. Akerlund, and F. Pellacini, "Fast, realistic lighting and material design using nonlinear cut approximation," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 128:1–10, 2008.

[17] 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. Vis. Comp. Graph.*, vol. 14, no. 2, pp. 454–467, 2008.

[18] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, "Adaptively sampled distance fields: a general representation of shape for computer graphics," in *ACM SIGGRAPH*, 2000, pp. 249–254.

[19] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *Comput. Graph. (SIGGRAPH)*, vol. 21, no. 4, pp. 163–169, 1987.
[20] Y.-T. Tsai and Z.-C. Shih, "All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 967–976, 2006.
[21] J. Lawrence, A. Ben-Artzi, C. DeCoro, W. Matusik, H. Pfister, R. Ramamoorthi, and S. Rusinkiewicz, "Inverse shade trees for non-parametric material representation and editing," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 735–745, 2006.
[22] J. Wang, S. Zhao, X. Tong, J. Snyder, and B. Guo, "Modeling anisotropic surface reflectance with example-based microfacet synthesis," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–9, 2008.