Deferred Shadowing for Real-Time Rendering of Dynamic Scenes Under Environment Illumination

Naoki Tamura Henry Johan Tomoyuki Nishita
The University of Tokyo
{naoki, henry, nis}@nis-lab.is.s.u-tokyo.ac.jp

Abstract

Environment illumination, which is a complex and distant lighting environment represented by images is often applied to create photo-realistic images. However, creating photo-realistic animations under environment illumination is exceedingly compute intensive. The Precomputed Radiance Transfer (PRT) methods achieve real-time rendering under environment illumination, however, these methods only have a limited application in animation because the objects in the scene cannot be moved or rotated. In this paper, we propose a method for rendering photo-realistic animations of dynamic scenes under environment illumination in real time. We notice the fact that when objects are moved or rotated, changes of radiances occur mainly in the regions of shadows cast by other objects. Our method makes a distinction between self-shadow and shadows cast by other objects and computes these two kinds of shadows efficiently.

Keywords: deferred shadowing, photo-realistic animation, real-time rendering, environment illumination

1 Introduction

Generating photo-realistic animations in real time is one of the most challenging issues in computer graphics. Recently, complex and distant lighting environments represented by images (environment illumination) are often used to render photo-realistic images. Typical approaches such as ray tracing are generally applied to render images under environment illumination, however, rendering animations in real time is difficult to accomplish due to the high computational cost.

Sloan et al. [1] proposed the Precomputed Radiance Transfer (PRT) method for real-time rendering under low-frequency environment illumination. Afterwards, Ng et al. [2][3] proposed PRT methods for high-frequency environment illumination. However, these PRT methods have a serious limitation, that is, they can be applied only to static scenes.

In animations, we usually deal with dynamic scenes composed of several movable objects. Therefore, it is very important for PRT methods to deal with dynamic scenes and to render the scenes in real time in order to make PRT methods being widely applied to general applications such as virtual reality and games. In this paper, we propose an efficient method which is able to render dynamic scenes in real time under environment illumination. Our algorithm deals with dynamic scenes lit by direct illumination from an environment illumination. Each object in the scene has diffuse or glossy surface composed of triangle meshes and we assume that the shape of each object is not deformable. Each object is allowed to perform translation or rotation.

Based on the fact that when objects are translated or rotated, rapid changes of radiances tend to occur in the regions of shadows cast by other objects rather than self-shadow (see Figure 1), we first render the scene taking into account only self-shadow. Then, we subtract the radiance obscured by other objects afterwards, resulting in shadows cast by other objects. The proposed method uses some approximations when computing shadows, as a result, the resulting shadows are not completely physically correct. However, through experiments we verified that our method produces images with high quality shadows.

We compute shadows cast by other objects only in the visible regions of the scene. This approach is similar to the deferred shading techniques [4] which perform expensive rendering only in the visible regions, and therefore, we call our approach *deferred shadowing*. We also describe an implementation of our method on a modern GPU, and show that our method accomplishes real-time rendering. The advantages of our new method are as follows.

- Applicable to dynamic scenes.
- Real-time rendering using GPU.
- High quality rendering of soft shadows.

2 Related Work

In this section, we review two categories of computer graphics research that are closely related to our method, i.e. rendering of soft shadows and PRT. Soft shadows have important visual effects since our method is based on environment illumination. PRT is currently the most promising approach for real-time rendering under environment illumination.

2.1 Rendering soft shadows

Nishita et al. [5][6] proposed methods for rendering soft shadows for linear light source and area light source by extending the shadow volume method. They [7] also proposed a method for rendering soft shadows under skylight represented as a dome, which is similar to environment illumination. However, these methods require much computational time and are difficult to render soft shadows in real time.

Recently, there have been many approaches to generate soft shadows using GPU. Most of them utilize shadow mapping [8] or shadow volume [9]. Heckbert and Herf [10] combined a number of hard shadow images for each receiver surface to create soft shadow textures. Heidrich et al. [11] described an algorithm for generating soft shadows for linear light sources. The method samples each light source sparsely and generates separate shadow maps for each sample point. Soler and Sillion [12] used FFT convolution to compute approximate soft shadows. Agrawala et al. [13] computed soft shadows in image space, which was, however, not adequate for interactive rendering. Additionally, several extentions of the shadow mapping method for real-time rendering of soft shadows [14][15][16] were presented.

Akenine et al. [17] and Assarsson et al. [18] extended the shadow volume method to render soft shadows using GPU. However, the computational cost of their methods depend on the geometry of the objects, which makes them difficult to render complex objects rapidly. Keller [19] proposed an efficient and simple approach which can render realistic images and also capture soft shadows using GPU. The limitation of all of the approaches described above is that they do not consider or they are too slow for computing shadows under complex environment illumination.

2.2 Precomputed radiance transfer

Dobashi et al. [20] proposed a method for rendering scenes under skylight using basis functions. Ramamoorthi et al. [21] rendered scenes under environment illumination in real time using spherical harmonics. However, shadows were not taken into consideration. Extending their method, Sloan et al. [1] proposed PRT method for real-time rendering of various effects such as soft shadows, direct and indirect illumination and caustics. Then, Kautz et al. [22] proposed a method for rendering scenes with arbitrary BRDFs. Lehtinen et al. [23] proposed a method for rendering glossy objects efficiently. Sloan et al. [24][25] used clustered principle component analysis for data compression and combined with BTF to handle meso-structures on surfaces. Ng et al. [2][3] used wavelet transform instead of spherical harmonics to represent all-frequency shadows under environment illumination. However, these methods can be applied only to static scenes since the precomputed visibilities cannot be changed.

James et al. [26] extended PRT with precomputed deformation dynamics to support real-time interactions in limited deformation space. Mei et al. [27] tackled translation and rotation of objects. However, since their approach calculates the radiance only at each vertex and uses interpolation to calculate the radiances at the rest of the locations, dense meshes are needed in the entire scene to capture rapid changes of radiances, such as shadows cast by other ob-

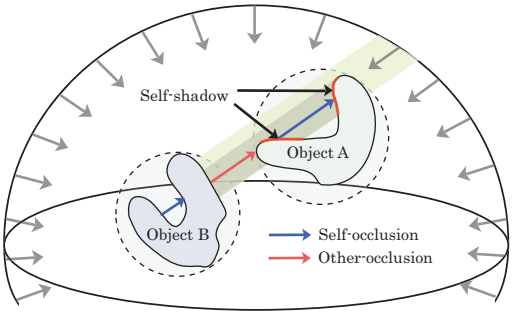


Figure 1: Each object is considered separately, for instance, object A and object B in this figure. Then, the occlusions are distinguished between self-occlusion and other-occlusion.

jects. Kautz et al. [28] proposed a method which can handle deformable objects. However, their method computes the radiance at each vertex and can produce only low-frequency shadows. Kontkanen and Laine [29] rendered dynamic scenes in real time. However, they only consider ambient occlusion, therefore, their method cannot handle environment illumination.

3 Overview

The reason why objects cannot be translated or rotated in the previous PRT methods is that the precomputed visibilities are no longer valid when the scene is changed. To solve this problem, we consider each object in the scene separately as shown in Figure 1. Here, we distinguish the occlusions as follows.

- Self-occlusion: occlusion due to its own geometry.
- Other-occlusion: occlusion due to other objects.

We call the object that causes other-occlusion (e.g. Object A for Object B in Figure 1), an *occluder*. Since the regions of shadows due to other-occlusion can change rapidly in dynamic scenes, we separately calculate the radiance due to self-occlusion and other-occlusion. We render the scene through the following three steps.

1. Render the scene with the radiance taking into account only self-occlusion to image 1 (Figure 2(a)).
2. Compute the radiance obscured by occluders only for the visible regions of the scene and store the results in image 2 (Figure 2(b)).
3. For each pixel, subtract the value in image 2 from the value in image 1 (Figure 2(c)).

When an object translates or rotates, self-occlusion does not change because we assume that the shape of each object is not deformable. Therefore, we can precompute the self-occlusion information of each object. In addition, we can respond to the rapid changes of radiances by adaptively subdividing the mesh of the object using the method proposed by Krivanek et al. [30]. Therefore, it is sufficient to compute the radiance taking into account only self-occlusion for each vertex and compute the radiances at the locations other than the vertices by interpolating the radiances at the vertices.

On the contrary, other-occlusion can change frequently in dynamic scenes. Since the region where the other-occlusion changes rapidly cannot be predicted, uniformly and densely sampled meshes are required in order to deal with the unpredictable change of the other-occlusion when we attempt to calculate the radiances only at the vertices. However, dense meshes are not efficient in terms of the computational time and the amount of the precomputed self-occlusion data. Therefore, instead of vertices, we compute the radiance obscured by occluders only in the visible regions of the scene at each pixel.

4 Deferred Shadowing on Diffuse Surfaces

In this section, we describe the details of our method on diffuse surfaces.

4.1 *Radiance taking into account only self-occlusion*

We first calculate the radiance L_s taking into account only self-occlusion of each object. L_s is calculated for each vertex by using the Sloan's method [1]. L_s at the i^{th} vertex x_i can be calculated by using the following equations.

$$T(x, \boldsymbol{\omega}) = \frac{1}{\pi} v_s(x, \boldsymbol{\omega}) \max(\boldsymbol{\omega} \cdot \mathbf{n}_i, 0), \quad (1)$$

$$L_s(x_i) = \rho_d \sum_{l=0}^{t-1} \sum_{m=-l}^{m=l} L_l^m T_l^m(x_i), \quad (2)$$

where v_s is the binary visibility information taking into account self-occlusion (0 means occluded, otherwise 1), $\mathbf{n}_i$ is the normal vector of the i^{th} vertex, $\boldsymbol{\omega}$ is the direction vector to unit sphere, l, t is a positive integer, m is an integer satisfies $|m| \leq l$, ρ_d is the surface diffuse

(a) Rendering result with the radiance taking into account only self-occlusion

(b) Rendering result of the radiance obscured by occluders

(c) Final result

Figure 2: The computational process of our method. The scene composed of two objects, a floor and a plane toy. We first render (a), then render (b) afterwards. Finally, we obtain (c) by subtracting (b) from (a).

albedo, L_l^m, T_l^m are the coefficients of the environment illumination and the light transport function T corresponding to each spherical harmonic basis. Note that we exclude ρ_d from the precomputation in order to take into account texture mapping or changes of materials at runtime. After calculating the radiances of all vertices, we render the scene and obtain an image taking into account only self-occlusion as shown in Figure 2(a).

4.2 *Radiance obscured by occluders*

After the radiance L_s is calculated, we calculate the radiance L_o obscured by occluders only for the visible regions of the scene at each pixel. Figure 2(b) shows the resulting image that stores the values of L_o. Then, the final radiance L can be obtained by subtracting the radiance L_o from the radiance L_s as shown in Figure 2(c).

$$L(p) = L_s(p) - L_o(p), \tag{3}$$

where p is the point on the object which is visible at a pixel. For the sake of simplicity, we first describe the case when there is only one occluder. Next, we describe the case of multiple occluders.

4.2.1 Decomposition of environment illumination

We divide the environment illumination into several area light sources W_i using the method proposed by Kollig et al. [31] as shown in Figure 3 and calculate L_o as follows.

$$L_o(p) \approx \frac{\rho_d}{\pi} \sum_{i=1}^{q} B_i \, O(p, \boldsymbol{\omega}_i) \, \max(\boldsymbol{\omega}_i \cdot \mathbf{n}, 0), \tag{4}$$

where q is the number of distant area light sources, B_i is the total radiosity of area light

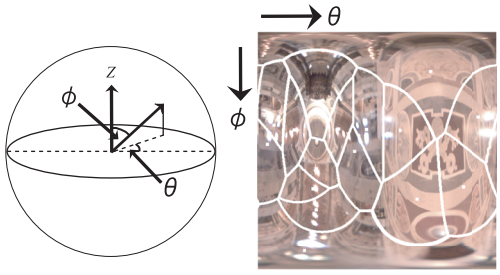


Figure 3: Decomposition of the environment illumination into several regions.

source W_i, $\boldsymbol{\omega}_i$ is the direction at the centroid of light source W_i. O is the occlusion ratio which indicates the portions of the distant area light source that are not visible from p . We represent the occlusion ratio O as a real number (0.0 to 1.0, 0.0 means completely visible and 1.0 means completely occluded) in order to obtain soft shadows with fewer light sources.

4.2.2 Computation of occlusion maps

If we compute O in Equation (4) for each visible point p, then the computational cost is high. To reduce the computational cost, we compute an *occlusion map* for each object with respect to each light source. When computing L_o, the occlusion map is projected to the visible region of the object. The occlusion map is computed as follows.

As shown in Figure 4, we consider the case of object A casts shadows on object B, with a light source in direction $\boldsymbol{\omega}_i$. We set an occluder plane H_A which passes the center of A and perpendicular to direction $\boldsymbol{\omega}_i$. Then, we project A to this occluder plane. P_A and P_B represent the boundary of A, respectively. To compute the occlusion ratio on B, we set a virtual plane H_B which passes the center of B and perpendicu-

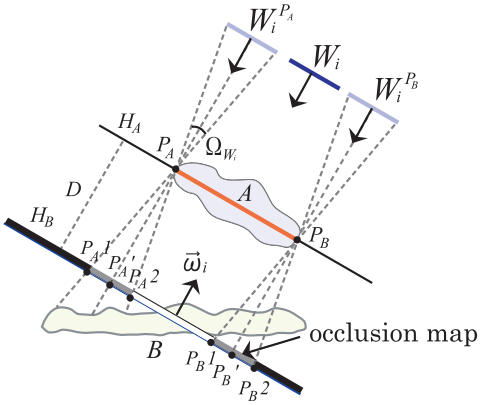
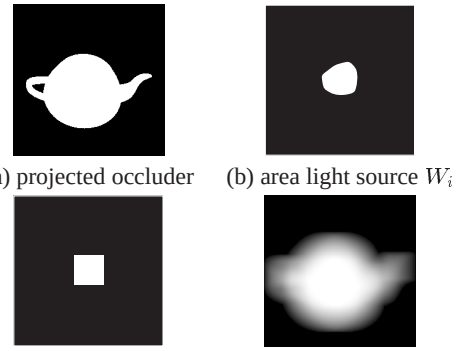


Figure 4: The computation of an occlusion map.



(a) projected occluder (b) area light source W_i

(c) simplified area light source (d) occlusion map

Figure 5: An example of an occlusion map, (a) is the projected occluder onto the virtual plane, (b) is the area light source W_i, (c) is the area light source simplified to a square, and (d) is the occlusion map obtained by convolving (a) with (c).

lar to direction $\boldsymbol{\omega}_i$. $P'_A P'_B$ is the shadow region (occlusion ratio 1.0) resulting by projecting A to H_B when we treat W_i as a parallel light source. Next, we consider the area of the light source. Since W_i is a distant light source, we can assume that the solid angles of W_i as seen from P_A and P_B are the same. The light that passes through P_A will emanate according to this solid angle, and on H_B the light will emanate in the region $P_A^1 P_A^2$. This region is the penumbra region and have the same shape as W_i. The occlusion ratio on H_B can be computed by convolving the resulting shadows (occlusion ratio 1.0) with a function with the shape of W_i.

To compute the convolution rapidly, the shape of W_i is simplified to a square. The width $s(\boldsymbol{\omega}_i)$ of the square is determined as follows. First, the area of the penumbra region $P_A^1 P_A^2$ is

$$Area(\boldsymbol{\omega}_i) = \Omega_{W_i} D^2, \tag{5}$$

where Ω_{W_i} is the solid angle of the distant area light source W_i and D is the distance between planes H_A and H_B. The width $s(\boldsymbol{\omega}_i)$ of the square can be obtained as follows.

$$s(\boldsymbol{\omega}_i) = \sqrt{Area(\boldsymbol{\omega}_i)}. \tag{6}$$

By convolving the shadow regions, resulting from projecting the occluder onto the virtual plane H_B, with a square area light source having width $s(\boldsymbol{\omega}_i)$, we obtain the occlusion map on H_B (see Figure 5). Then, we perform orthogonal projection to object B. Note that we calculate the occlusion map for each W_i and each object.

4.2.3 Case of multiple occluders

When multiple occluders exist, we cluster the occluders that are located near to each other as one group. Then, we compute an occlusion map for this group. To compute the occlusion map, we set the occluder plane to pass through the center of the group. When there are more than two groups, we compute an occlusion map for each group and then combine the results using the method proposed by Soler and Sillion [12].

4.2.4 Solution to the problem of shadow calculation

The self-occlusion and the other-occlusion might exist at the same time when viewing from a certain position in the direction $\boldsymbol{\omega}_i$. In this case, our method considers the occlusion twice, as a result the calculated radiance becomes smaller than the correct radiance.

To solve this problem, as shown in Equation (7), visibility taking into account self-occlusion v_s at p is multiplied to the occlusion ratio O to eliminate the influence of the other-occlusion when the self-occlusion exists.

$$L_o(p) \approx \frac{\rho_d}{\pi} \sum_{i=1}^{q} B_i \, v_s(p, \boldsymbol{\omega}_i) \, O(p, \boldsymbol{\omega}_i) \, \max(\boldsymbol{\omega}_i \cdot \mathbf{n}, 0). \tag{7}$$

Using the light transport function T, Equation (7) is transformed to

$$L_o(p) \approx \rho_d \sum_{i=1}^{q} B_i \, T(p, \boldsymbol{\omega}_i) \, O(p, \boldsymbol{\omega}_i). \tag{8}$$

Obviously, we need to evaluate T at the visible point p. The detail algorithm for computing T at p is described in Section 6.2.

5 Deferred Shadowing on Glossy Surfaces

In this section, we describe the details of our method on glossy surfaces.

5.1 Radiance taking into account only self-occlusion

The radiance on glossy surfaces is also calculated for each vertex by using spherical harmonics similar to the Sloan's method [1]. However, since we only consider direct illumination, we propose a method that can reduce the amount of the precomputed data compared to the Sloan's method [1]. The radiance L'_s at the i^{th} vertex x_i of glossy surfaces viewing from direction $\boldsymbol{\omega}_o$ taking into account only self-occlusion can be calculated by using the following equation.

$$L'_s(x_i, \boldsymbol{\omega}_o) = \int_\Omega L_{\text{ENV}}(\boldsymbol{\omega}) f_r(x_i, \boldsymbol{\omega}, Ref(\boldsymbol{\omega}_o)) \\ v_s(x_i, \boldsymbol{\omega})\, d\boldsymbol{\omega}, \quad (9)$$

where f_r is the Phong BRDF, Ref is the reflection vector. We calculate the coefficients corresponding to each spherical harmonic basis of L_{ENV} and v_s. In Sloan's method [1], v_s is represented as a matrix. However, since we only consider direct illumination, instead of v_s, we represent the coefficients of L_{ENV} as a matrix. In this case, the coefficients of v_s are represented simply as a vector. As a result, we can reduce the amount of the precomputed data since we do not have to store a matrix for each vertex. The coefficients of the spherical harmonic basis of L_{ENV} and v_s are then computed as follows.

$$L_{ll'}^{mm'} = \int_\Omega L_{\text{ENV}}(\boldsymbol{\omega})\, y_l^m(\boldsymbol{\omega})\, y_{l'}^{m'}(\boldsymbol{\omega})\, d\boldsymbol{\omega}. \quad (10)$$

$$v_l^m(x_i) = \int_\Omega v_s(x_i, \boldsymbol{\omega})\, y_l^m(\boldsymbol{\omega})\, d\boldsymbol{\omega}. \quad (11)$$

The glossy light transport function G is calculated by using the convolution of spherical harmonics.

$$G(x_i) = f_r(x_i, \boldsymbol{\omega}, (0,0,1)) \otimes (L_{ll'}^{mm'}\, v_{l'}^{m'}(x_i)). \quad (12)$$

Equation (9) is transformed by using Equation (12) as follows.

$$L'_s(x_i, \boldsymbol{\omega}_o) = \sum_{l=0}^{t-1} \sum_{m=-l}^{m=l} G(x_i)\, y_l^m(Ref(\boldsymbol{\omega}_o)). \quad (13)$$

G does not change as long as the positional relation between an object and the environment illumination does not change.

5.2 Radiance obscured by occluders

Similar to the case of diffuse surfaces, the radiance L'_o obscured by occluders on glossy surfaces can be calculated as follows.

$$L'_o(p, \boldsymbol{\omega}_o) \approx \sum_{i=1}^{q} B_i\, f_r(p, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)\, O(p, \boldsymbol{\omega}_i). \quad (14)$$

To solve the problem as described in Section 4.2.4, visibility v_s at p is multiplied.

$$L'_o(p, \boldsymbol{\omega}_o) \approx \sum_{i=1}^{q} B_i\, f_r(p, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)\, v_s(p, \boldsymbol{\omega}_i)\, O(p, \boldsymbol{\omega}_i). \quad (15)$$

6 Efficient Calculation Using GPU

In this section, we explain the implementation for an efficient computation of our method using GPU. Currently, we implemented our method on the OpenGL platform. However, it is also possible to implement our method on the DirectX platform.

6.1 Radiance taking into account only self-occlusion

To compute the radiance taking into account only self-occlusion on diffuse and glossy surfaces efficiently, we propose an implementation which makes use of the parallel processing capability of modern GPU. After we compute the radiances taking into account only self-occlusion at all the vertices, for accuracy, we render the scene into a 16-bit floating point p-Buffer.

6.1.1 Diffuse surfaces

The radiance L_s taking into account only self-occlusion can be obtained by calculating the inner product in Equation (2). Since this calculation is independent for each vertex, we can efficiently compute L_s by using the programmable pixel shader which can perform a rendering operation for each pixel on a screen or a texture.

We first prepare a texture (16-bit floating point p-Buffer) for using pixel shader. This texture is used to store the radiances and we call this texture, *radiance texture*. Each pixel of the radiance texture is allocated to each vertex of the object. Then, we compute L_s as follows.

1. Send L_l^m (Equation (2)) to pixel shader as shader constant values.

2. Send T_l^m (Equation (2)) of a vertex by drawing a point primitive with a width of one pixel to the corresponding pixel. Here, T_l^m are set as the attribute values of the point.
3. Compute the radiance L_s using pixel shader's vectors inner product capability between L_l^m and T_l^m for each vertex in the pixel shader.

After performing the above-mentioned computation, the radiance texture stores the radiances of all vertices (without taking into account the surface diffuse albedo) of each object. When the scene is rendered, the value of each pixel in the radiance texture is multiplied with the surface diffuse albedo at the corresponding vertex and the results are used in the vertex shader. As long as the relation between each object and the environment illumination does not change, we can reuse the same radiance texture for rendering. Only when the objects are rotated, the coefficients of the spherical harmonic are transformed using the method used in Sloan et al. [1].

6.1.2 Glossy surfaces

For glossy surfaces, the glossy light transport function G in Equation (12) is first calculated for each vertex and the result is stored in *transport textures* which are 16-bit floating point p-Buffers. To compute G, we have to perform a multiplication between a matrix $L_{ll'}^{mm'}$ (Equation (10)) and a vector v_l^m (Equation (11)). Since a matrix-vector multiplication can be performed by computing a vector-vector inner product multiple times, we can employ the method described in Section 6.1.1. The resulting vectors from the above computation is convolved with the BRDF kernel by using the method as shown in Sloan et al. [1]. G does not change as long as the positional relation between each object and the environment illumination and the BRDF at the vertex do not change. Therefore, the transport textures that store G can be reused.

Radiance L_s' (Equation (13)) is evaluated by using G whenever the viewpoint moves. We calculate L_s' based on the diffuse case. The details for computing L_s' are as follows.

1. Send the viewpoint to pixel shader as shader constant values.
2. Send the position and the normal vector of each vertex to the corresponding pixel.
3. Calculate the coefficients corresponding to each spherical harmonic basis for the reflection vector at each pixel.
4. Obtain the radiance L_s' through the inner product computation between G (obtained from the transport textures) and the coefficients of the reflection vector.

As a result, the radiance texture that stores the radiances for all vertices can be obtained. The value of each pixel in the radiance texture is used in the vertex shader when the scene is rendered.

6.2 Radiance obscured by occluders

To compute the obscured radiance, first we have to determine the occlusion maps, the visible regions, and finally compute the obscured radiance using the former two information.

6.2.1 Creation of occlusion maps

An occlusion map is computed by projecting occluders and performing convolution. For fast computation, we perform the convolution using GPU. In our method, the convolution is separable for each dimension because we simplify the shape of area light sources to a square which can be regarded as box function and so the computation cost can be reduced.

When rendering a scene, we can think of computing occlusion maps only for the visible objects in the scene. However, this approach requires to check the visibility of each object in each frame and thus complicates the rendering process. Instead, for each object in the scene, we compute an occlusion map with respect to each area light source and store the results in GPU. Only when the objects in the scene are moved, we recompute the occlusion maps.

6.2.2 Determination of visible regions

To determine the visible regions, we first render the scene once and store the depth information. Then, we render each object independently taking into account the stored depth information. After the visible regions are determined, the computed occlusion map for each light source is then orthogonally projected to these regions for each object using GPU.

6.2.3 Computation of obscured radiance

The obscured radiances of the visible regions are stored in 16-bit floating point p-Buffers.

In the diffuse case, the obscured radiance for each light source (Equation (8)) can be obtained by multiplying together the value of the projected occlusion map, the radiosity of each light source, and the light transport function T. As described in Section 4.2.4, we need to evaluate T in the direction of each light source ω_i for each visible point p in order to overcome the overlapping problem of the self-occlusion and the other-occlusion. We first expand T in the direction of all light sources in advance for each vertex of the object, then T of each vertex is set as the vertex attribute and T of the visible points on the objects other than the vertices are interpolated using GPU when the scene is rendered.

In the glossy case, to compute v_s, we use the same method used in computing T in the diffuse case. f_r is evaluated at each visible point p. To add the obscured radiances due to all light sources, we use the floating point blending [32] since accuracy is insufficient when we use the usual blending.

7 Results and Discussion

Our results were computed on a machine with a 3.2 GHz Pentium 4 processor and a nVIDIA GeForce 6800 GT graphics hardware. For the environment illumination, we use the Paul Debevec's light probe images [33]. The size of the environment illumination is 128×128. We use 5^{th}-order (36-term) spherical harmonics for diffuse surfaces and 3^{rd}-order (16-term) spherical harmonics for glossy surfaces. The sizes of the images are 512×512 pixels.

7.1 Shadows quality

We performed several experiments to examine the quality of the generated shadows. We varied the number of area light sources to 16, 32, and 64 and the sizes of the occlusion maps to 32×32, 64×64, and 128×128.

Figure 6 shows two of the resulting images and the reference image rendered using ray tracing method. High-frequency shadows can be seen in Figures 6 (a) and (b). Most of the time, we found that our method generates high quality shadows when we used occlusion maps with sizes 128×128.

For the number of area light sources, generally, 32 area light sources are sufficient to produce high quality shadows. In summary, we found that shadow quality depends on the resolutions of occlusion maps rather than the number of area light sources.

7.2 Rendering performance

We examined the rendering performance (fps) using our method while changing the rendering parameters. We performed the experiments using two scenes, a teapot scene consists of 2421 vertices and a Buddha statue scene consists of 44856 vertices. For each scene, we performed the experiments twice by setting the object as a diffuse surface and a glossy surface.

Results of our experiments are plotted in the graphs shown in Figure 7. For real-time application, we found that 32 area light sources and occlusion maps with sizes 64×64 are appropriate in terms of the balance between the rendering speed and the shadows quality. When it is sufficient to render at interactive rate, occlusion maps with sizes 128×128 can be used for producing a better quality images.

7.3 Rendering dynamic scenes

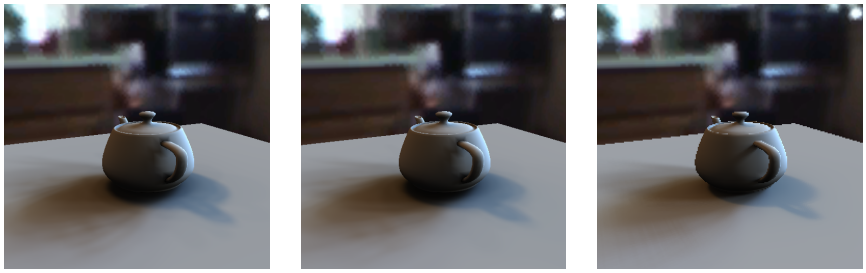
Based on the facts described in the previous sections, we rendered animations of two scenes with 32 area light sources and occlusion maps with sizes 128×128. Figure 8 shows the results. The statistics of the two scenes are shown in Table 1.

Table 1: Statistics of the rendering examples.

	Venus	Trumpet
#Vertices	5776	8557
Precomputation time (sec)	85	109
Precomputation data (KB)	813	1212
Average frame rate (fps)	30	25

8 Conclusion and Future Work

In this paper, we have presented an efficient method that is able to render dynamic scenes under environment illumination in real time. Our method distinguishes the shadows between self-shadow and shadows cast by other objects, and



(a) 32 lights, 128×128 (b) 64 lights, 128×128 (c) reference image

Figure 6: Changing the numbers of area light sources and the sizes of the occlusion maps. (a), (b) are the resulting images with $32, 64$ area light sources and occlusion maps with sizes 128×128, (c) is the reference image rendered using ray tracing method.

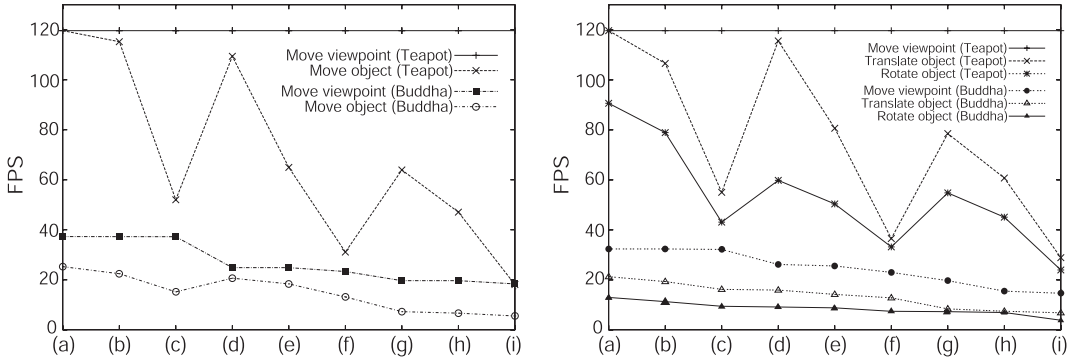


Figure 7: Statistics of the performance on diffuse surfaces (*left*) and glossy surfaces (*right*). The number of area light sources and the sizes of occlusion maps are (a) $16, 32 \times 32$, (b) $16, 64 \times 64$, (c) $16, 128 \times 128$, (d) $32, 32 \times 32$, (e) $32, 64 \times 64$, (f) $32, 128 \times 128$, (g) $64, 32 \times 32$, (h) $64, 64 \times 64$, (i) $64, 128 \times 128$.

computes both of them efficiently using GPU. Through experiments, we showed that the proposed method can render high quality soft shadows fast.

As future work, we are interested in developing a fast method for rendering dynamic scenes consist of deformable objects under environment illumination.

Acknowledgment

We thank Prof. Chandrajit Bajaj (University of Texas), Yoshihiro Kanamori, and Ryoichi Mizuno for many useful suggestions.

References

[1] P. P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proc. SIGGRAPH 2002*, pages 527–536, 2002.

[2] R. Ng, R. Ramamoorthi, and P. Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Transactions on Graphics*, 22(3):376–381, 2003.

[3] R. Ng, R. Ramamoorthi, and P. Hanrahan. Triple product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics*, 23(3):477–487, 2004.

[4] nVIDIA. Deferred shading. 2004.

[5] T. Nishita, I. Okamura, and E. Nakamae. Shading models for point and linear sources. *ACM Transactions on Graphics*, 4(2):124–146, 1985.

[6] T. Nishita and E. Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In *Proc. SIGGRAPH 85*, pages 23–30, 1985.

[7] T. Nishita and E. Nakamae. Continuous tone representation of three- dimensional objects illuminated by sky light. In *Proc. SIGGRAPH 86*, pages 125–132, 1986.

[8] L. Williams. Casting curved shadows on curved surfaces. In *Proc. SIGGRAPH 78*, pages 270–274, 1978.

[9] F. C. Crow. Shadow algorithms for computer graphics. In *Proc. SIGGRAPH 77*, pages 242–248, 1977.

[10] P. S. Heckbert and M. Herf. Simulating soft shadows with graphics hardware. January 1997. Technical report CMU-CS-97-104, Carnegie Mellon University, January 1997.

[11] W. Heidrich, S. Brabec, and H. P. Seidel. Soft shadow maps for linear lights. In *Proc. 11th Eurographics Workshop on Rendering*, pages 269–280, 2000.

[12] C. Soler and F. X. Sillion. Fast calculation of soft shadow textures using convolution. In *Proc. SIGGRAPH 98*, pages 321–332, 1998.

Figure 8: Rendering results, moving a diffuse Venus statue under the kitchen illumination (*top*), moving a glossy trumpet under the St. Peter's Basilica illumination (*bottom*).

[13] M. Agrawala, R. Ramamoorthi, A. Heirich, and L. Moll. Efficient image-based methods for rendering soft shadows. In *Proc. SIGGRAPH 2000*, pages 375–384, 2000.

[14] J. Arvo, M. Hirvikorpi, and J. Tyystyjarvi. Approximate soft shadows using an image-space flood-fill algorithm. In *Proc. Eurographics 2004*, pages 271–280, 2004.

[15] E. Chan and F. Durand. Rendering fake soft shadows with smoothies. In *Proc. 14th Eurographics Workshop on Rendering*, pages 208–218, 2003.

[16] C. Wyman and C. Hansen. Penumbra maps: approximate soft shadows in real-time. In *Proc. 14th Eurographics Workshop on Rendering*, pages 202–207, 2003.

[17] T. Akenine-Moller and U. Assarsson. Shading and shadows: Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Proc. 13th Eurographics Workshop on Rendering*, pages 297–306, 2002.

[18] U. Assarsson and T. Akenine-Moller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics*, 22(3):511–520, 2003.

[19] A. Keller. Instant radiosity. In *Proc. SIGGRAPH 97*, pages 49–56, 1997.

[20] Y. Dobashi, K. Kaneda, H. Yamashita, and T. Nishita. A quick rendering method for outdoor scenes using sky light luminance functions expressed with basis functions. *The Journal of the Institute of Image Electronics Engineers of Japan*, 24(3):196–205, 1995.

[21] R. Ramamoorthi and P. Hanrahan. An efficient representation for irradiance environment maps. In *Proc. SIGGRAPH 2001*, pages 497–500, 2001.

[22] J. Kautz, P. P. Sloan, and J. Snyder. Shading and shadows: Fast, arbitrary BRDF shading for low-frequency lighting using spherical harmonics. In *Proc. 13th Eurographics Workshop on Rendering*, pages 291–296, 2002.

[23] J. Lehtinen and J. Kautz. Matrix radiance transfer. In *Proc. Symposium on Interactive 3D Graphics 2003*, pages 59–64, 2003.

[24] P. P. Sloan, J. Hall, J. Hart, and J. Snyder. Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics*, 22(3):382–391, 2003.

[25] P. P. Sloan, X. Liu, H. Y. Shum, and J. Snyder. Bi-scale radiance transfer. *ACM Transactions on Graphics*, 22(3):370–375, 2003.

[26] D. L. James and K. Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics*, 22(3):879–887, 2003.

[27] C. Mei, J. Shi, and F. Wu. Rendering with spherical radiance transport maps. In *Proc. Eurographics 2004*, pages 281–290, 2004.

[28] J. Kautz, J. Lehtinen, and T. Aila. Hemispherical rasterization for self-shadowing of dynamic objects. In *Proc. Eurographics Symposium on Rendering 2004*, pages 179–184, 2004.

[29] J. Kontkanen and S. Laine. Ambient occlusion fields. In *Proc. Symposium on Interactive 3D Graphics 2005*, pages 41–48, 2005.

[30] J. Krivanek, S. Pattanaik, and J. Zara. Adaptive mesh subdivision for precomputed radiance transfer. In *Proc. 20th Spring Conference on Computer Graphics*, pages 106–111, 2004.

[31] T. Kollig and A. Keller. Efficient illumination by high dynamic range images. In *Proc. 14th Eurographics Workshop on Rendering*, pages 45–50, 2003.

[32] nVIDIA. High dynamic range rendering on GeForce 6 series GPU. 2004. GPU Jackpot 2004.

[33] P. Debevec. Light probe image gallery. http://www.debevec.org/Probes/index.html.