Vis Comput (2010) 26: 1497–1512
DOI 10.1007/s00371-010-0507-1

ORIGINAL ARTICLE

Layered occlusion map for soft shadow generation

Kien T. Nguyen · Hanyoung Jang · JungHyun Han

Published online: 28 April 2010

Abstract This paper presents a high-quality high-performance algorithm to compute plausible soft shadows for complex dynamic scenes. Given a rectangular light source, the scene is rendered from the viewpoint placed at the center of the light source, and discretized into a layered depth map. For each scene point sampled in the depth map, the occlusion degree is computed, and stored in a layered occlusion map. When the scene is rendered from the camera's viewpoint, the occlusion degree of a scene point is computed by filtering the layered occlusion map. The proposed algorithm produces soft shadows the quality of which is quite close to that of the ground truth reference. As it runs very fast, a scene with a million polygons can be rendered in real-time. The proposed method does not require pre-processing and is easy to implement in contemporary graphic hardware.

Keywords Soft shadow algorithm · Image processing · Hardware accelerated rendering · Real-time shadowing

1 Introduction

Shadows increase the realism of rendered images, and help us understand the spatial relationships among objects in a scene. Two seminal works in shadow generation are shadow volume [7] and shadow mapping [29]. The shadow volume is an object-space technique. It describes the 3D volume occluded from a light source, and determines if a pixel to be rendered is within the volume. In contrast, shadow mapping is an image-space technique. It tests if a pixel to be rendered is visible from the light source, using a depth image from the viewpoint of the light source, stored in a texture named shadow map.

The two original works assume a point light source, and generate hard shadows where a point in the scene is either fully lit by or fully occluded from the light source. However, area or volumetric light sources in the real world generate soft shadows, where a penumbra region lies between the umbra (fully occluded) region and the fully lit region. For decades, the shadow volume and the shadow mapping techniques have been extended to generate soft shadows, and many real-time algorithms have been proposed.

Recently, many of the shadow mapping algorithms for soft shadow generation have pursued a common technique. The shadow map is viewed as a discrete representation of the scene, and each sample in the map is considered as a micro-patch. To determine the percentage or degree of occlusion at a point, the micro-patches selected as the potential occluders are back-projected from the point to the light plane, and the overlapped area is computed [4, 14].

This paper presents a high-quality high-performance algorithm along the lines of the micro-patch projection approach. Given a light source, the scene is rendered from the light source's viewpoint, and discretized into a layered depth map. (The layered depth map was initially proposed for rendering transparent objects [10].) For each scene point sampled in the layered depth map, the occlusion degree is computed through back-projection, and stored in the so-called layered occlusion map (LOM). When the scene is rendered from the camera's viewpoint, the occlusion degree of a scene point is computed by filtering the LOM.

The LOM presented in this paper is constructed using the layered depth map and micro-patch back projection algorithm. The proposed method produces soft shadows, the

K.T. Nguyen · H. Jang · J. Han (✉)
Korea University, Seoul, Korea
e-mail: jhan@korea.ac.kr

quality of which is quite close to that of the ground truth reference. It runs very fast such that a scene with a million polygons can be rendered in real-time.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 presents how to construct the layered depth map and the LOM, and Sect. 4 presents how to render the shadowed scene using the two maps. Section 5 discusses various optimization techniques used to construct the maps. Section 6 presents the implementation details, and Sect. 7 reports the experimental results. Section 8 compares the proposed method with existing work, and finally Sect. 9 concludes the paper.

2 Related work

An exhaustive survey of the numerous shadow algorithms is beyond the scope of this section; readers are referred to Woo et al. [31] for a comprehensive survey on hard shadows, and Hasenfratz et al. [16] for soft shadows. After briefly reviewing the shadow volume approach and a recently proposed algorithm based on the pre-computed shadow fields, we will focus on the shadow mapping algorithms closely related to our method.

The shadow volume approach [7] has been extended to produce soft shadows. Akenine-Möller and Assarsson [2] construct a penumbra-wedge per silhouette edge which is seen from the light source center. The penumbra-wedges are back-projected to the light source plane to accumulate the occluded area of the light source. Laine et al. [19] combine penumbra-wedges with ray tracing for planar area light source. Forest et al. [12] use penumbra-wedge to identify the visible surface points in the penumbra region. The visibility of the light source from a surface point is computed using the depth complexity between the point and a set of light samples. Recently, Forest et al. [13] propose a soft texture shadow volume to produce soft shadows for perforated triangles.

Sloan et al. [27] proposed to pre-compute radiance transfer for realtime rendering of low-frequency lighting by using a spherical harmonic basis to store the pre-computed light transport at every point in the scene. Zhou et al. [33] proposed to construct shadow fields around each single object which can be pre-computed independently of the scene configuration. This approach has been extended by [18, 20] to interactively render a dynamic scene in realtime.

Shadow mapping was introduced by Williams [29]. The scene is rendered from the light source's viewpoint, and its depths are computed and stored in the shadow map. Hard shadows are produced by transforming each screen-space pixel to the light space and then comparing its depth to the one in the shadow map. If the pixel depth is greater than the stored one, it is determined to be occluded from the light.

Following the seminal work of Williams, many algorithms have been presented to make soft shadows by extending the shadow mapping technique. Reeves et al. [21] proposed the percentage closer filtering (PCF) method to soften the hard shadow edges. Multiple sampling of the shadow map is performed around the pixel to be rendered, and the results are compared to the point's depth, as in conventional shadow mapping. The binary values are filtered to give the approximate visibility of the point. Fernando [11] extended the PCF method. Given a pixel to be rendered, the depth map is searched for its potential occluders, and the average of their depths is computed to determine the kernel size. PCF is applied to the kernel to approximate the pixel's visibility.

Heidrich et al. [17] produced soft shadows of a linear light source by sampling it at two endpoints. For each light sample, a soft shadow map is computed that consists of a conventional depth map and a visibility map. The visibility map at one light sample is computed by warping the triangulated depth points from the other light sample's depth map to its viewpoint. The soft shadows are produced by summing the two visibility maps. Agrawala et al. [1] processed a set of samples of an area light source, and generated a depth map for a sample. All points in the depth maps are warped into the viewpoint located at the light source center. Those points are added into a multi-layer buffer, called the layer attenuation map, which contains depth and visibility information. The visibility at a point is computed by counting the number of visible light samples from the point. When rendering the scene from the camera's viewpoint, a scene point is transformed to the light space with respect to the light center, and the layer attenuation map is searched to obtain the visibility of the point.

Arvo et al. [3] detected the hard shadow boundaries, in the screen space, that also store occluder information from the depth map. A modified flood-fill method is used to expand the penumbra regions from the boundaries. This method heavily depends on the number of flood-fill steps. Rong and Tan [22] improved the work of Arvo et al. by proposing a quick flood-fill method ending in a few steps. Initially, a large kernel is used to spread out the occluder information from the boundaries. Then smaller kernels are used in an iterative fashion to correct the occluder information of a pixel in the penumbra.

Eisemann and Décoret [9] rendered the scene into a multi-sliced shadow map. Each slice of the shadow map is pre-filtered. In the final rendering step, the visibility of a scene point is computed from the filtered slices using a probabilistic approach. Their algorithm is sensitive to the slice positions, i.e., where to place the slices.

Drettakis and Fiume [8] proposed a backprojection data structure to represent the visible portion of a light source from any point in the scene. Since then, many algorithms have extended the idea: the shadow map samples are converted into micro-patches, and then back-projected to the light source plane [4–6, 14]. The idea of the micro-patch has also been extended into more complicated geometries, such as micro-quads [23], micro-rects [25], and micro-tris [24].

Atty et al. [4] processed a micro-patch at a time to compute the region of the soft shadow map, affected by the micro-patch. For each pixel in the region, the micro-patch is back-projected to the light plane, and the occlusion degree is computed. The process is done for all micro-patches, and the occlusion degrees are accumulated into the soft shadow map. Finally, the soft shadow map is projected onto the scene to produce soft shadows. This algorithm distinguishes between occluders and receivers, and thus cannot produce self-shadows. Another disadvantage of the algorithm is that the shadow map (occluder map) generated by GPU is read back to the CPU that then invokes the GPU for back-projection. The communication between CPU and GPU degrades the rendering performance.

Guennebaud et al. [14] took a similar approach, but back-projection was done from a screen-space pixel. Given a pixel, the shadow map is searched for the samples working as the potential occluders. They are back-projected into the light plane. The algorithm proposed to extend the sample sizes to overcome the light leaking problem caused by the discrete nature of the shadow map. However, the solution often leads to over-shadows.

Guennebaud et al. [15] improved the shadow quality by detecting local contour edges at texel level of the shadow map. Given a pixel of the screen space, the shadow map samples working as the potential occluders are identified, as was done in the previous work [14]. Instead of back-projecting each sample in the micro-patch form, however, a local contour edge is back-projected to the light source plane. The back-projected edge is clipped by the borders of the light source, and the occluded area associated with the edge is computed and accumulated. Then, the occlusion degree of the current pixel is computed. Yang et al. [32] adopted the technique of contour edge back-projection, and proposed a so-called packet-based method, where a set of screen-space pixels is processed simultaneously. The operations of shadow map access and contour extraction are performed with respect to the set. Exploiting the penumbra coherence between the adjacent pixels leads to the performance increase.

The layered depth map was proposed by a few algorithms [5, 6, 23], to reduce the light leaking and over-shadow problems; and the goal of reducing the artifact was largely achieved. Like the work of Guennebaud et al. [14], however, these techniques process the screen-space pixels. As a result, the performance of the algorithm is sensitive to the percentage of the penumbra pixels in the screen. When the penumbra pixels take a larger part of the screen, performance drops significantly because more samples in the depth map should be processed. Guennebaud et al. [15] alleviated this limitation using a pre-computed pattern to skip some screen pixels falling in a large penumbra region. The visibility values of the skipped pixels are computed by interpolating the visibility values of the processed pixels. Yang et al. [32] proposed to use more samples of the light source to reduce the artifact caused by a single sample.

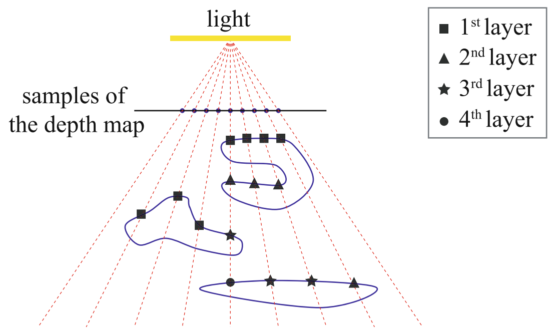

Fig. 1 Multiple layers of depths in a scene

The algorithm presented in this paper was inspired by the work of Atty et al. [4], and proposed to remedy its weaknesses. The idea of the layered depth map was adopted for this purpose. Unlike the prior work, back-projection is done in the light space, not in the screen space, and a new texture storing the occlusion degrees for the multiple layers is constructed. Final rendering of the scene is done by filtering the texture, and therefore the rendering performance is made largely constant, independent of the screen configuration.

3 Depth map and occlusion map

Like the traditional shadow mapping technique, our method is a two-pass algorithm: the first pass constructs a *depth map* and an *occlusion map* from the viewpoint of the light source, and the second pass renders the scene using the two maps from the viewpoint of the camera. This section presents the basics of the first pass, Sect. 4 presents the second pass, and Sect. 5 presents the optimization techniques of the first pass.

Both the depth map and the occlusion map are *multi-layered* or simply *layered*. The depth map discretizes the scene, and the occlusion map stores the degree of light occlusion per discretized scene point. Section 3.1 presents how to construct the depth map, and Sects. 3.2 and 3.3 present how to construct the occlusion map.

3.1 Layered depth map

We assume a rectangular light source. The viewpoint is located at the center of the light source to construct the depth map (Fig. 1). The scene is rendered in a layer-by-layer fashion using the depth peeling method [10]. At the first iteration, the scene is rendered normally with back-face culling, and the depth values of the surfaces nearest to the light source are stored in the first layer. The first layer contains the same information as the traditional shadow map.

In the second iteration, the depths less than or equal to those stored in the first layer are peeled away, and the second layer is filled. This process is repeated until the entire scene is discretized. Figure 1 shows four depth layers of a scene.

Multiple layers are packed in a texture such that a *sample* at (u, v) contains multiple depth values. A 4-channel (RGBA) texture is sufficient to represent the depth map for the scene in Fig. 1. If a scene requires more than four depth layers, we may use additional textures. Even in such a case, however, a single texture is sufficient in general, as will be discussed in Sect. 5.1.

3.2 Collecting potential occludees

The layered occlusion map (LOM) is constructed using the depth map. The two maps have the same structure, but the depth map contains a depth value per discretized scene point, whereas LOM contains its *occlusion degree*, which represents how much of light is occluded. The occlusion degree is in the range of [0,1], where 0 denotes 'fully lit' and 1 denotes 'fully occluded.'

```
for each sample s of the depth map
    Construct the micro-patches associated with s
    Get the potential occludees using the micro-patch information
    for each potential occludee
        Compute the occlusion degree by back-projecting
        the micro-patches
```

Fig. 2 Algorithm for LOM construction

Figure 2 shows the skeleton of the LOM construction algorithm. The algorithm processes a sample of the depth map at a time. The depth values are retrieved from a sample, and a rectangular *micro-patch* is placed at each depth value. For some of the scene points discretized in the depth map, the light rays from the light source may be occluded by the micro-patches. Such scene points, called *potential occludees*, are identified using the method presented in this sub-section. Then, the occlusion degree is computed for each potential occludee by back-projecting the micro-patches, as presented in Sect. 3.3.

Figure 3(a) shows the umbra and penumbra volumes caused by a micro-patch, which are collectively called the *shadow extent*. Figure 3(b) shows two micro-patches, at z_1 and z_2, obtained from a sample of the depth map. For the sake of computation, the micro-patch is assumed axis-aligned with the rectangular light source. The shadow extent of the micro-patch at z_2 is included in that at z_1. Given a sample, the shadow extent of the micro-patch farther from the light source is included in that of the closer. Therefore, the potential occludees affected by a sample are all located within the shadow extent of the sample's micro-patch that is closest to the light source.

Let us cut the shadow extent of the closest micro-patch by a plane parallel to the micro-patch, as shown in Fig. 4(a), to generate the *shadow rectangle*. When it is projected onto the depth map that is placed at the near plane of the light frustum, it is called the *kernel*. The *kernel volume* is bounded by the kernel and the shadow rectangle. It is a superset of

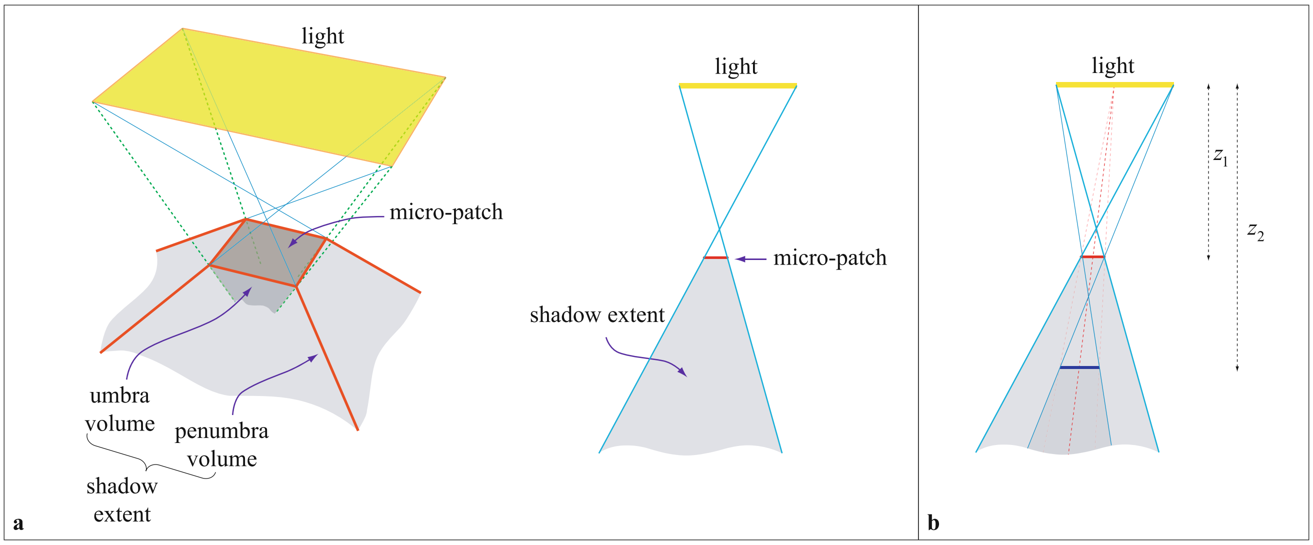

Fig. 3 Shadow extent: (**a**) a micro-patch generates a shadow extent that includes both the penumbra and umbra volumes; (**b**) given a sample, the shadow extent of the farther micro-patch is included in that of the closer, and therefore only the closest micro-patch will be considered for collecting the potential occludees

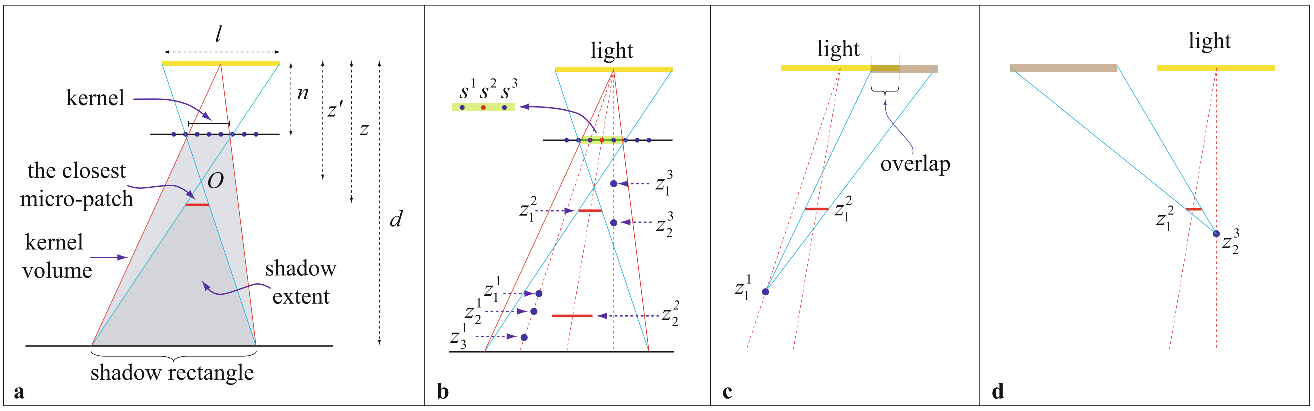

Fig. 4 Kernel and back-projection: (**a**) the kernel volume is a superset of the truncated shadow extent; (**b**) the depth values are retrieved for each sample contained in the kernel; (**c**) back-projection with overlap; (**d**) back-projection with no overlap

the truncated shadow extent, and is searched for potential occludees.

A set of samples is located within the kernel. For each sample, we can retrieve a list of depth values. Suppose that, in Fig. 4(b), s^2 is the sample under process. Its kernel, centered at s^2, includes three samples: s^1, s^2, and s^3. A list of depth values is retrieved from a sample: $\{z_1^1, z_2^1, z_3^1\}$ from s^1, $\{z_1^2, z_2^2\}$ from s^2, and $\{z_1^3, z_2^3\}$ from s^3. The scene points at these depth values are the potential occludees of s^2's micro-patches, and the occlusion degree will be computed for each potential occludee.

It is simple to compute the kernel. In Fig. 4(a), O is the apex of a triangular side of the truncated pyramid for the shadow extent. The distance z' from the light plane to O is computed as follows:

$$\frac{1}{z'} = \frac{1}{z} + \frac{w}{nrl}, \tag{1}$$

where z is the distance from the light plane to the closest micro-patch, w and n are the width and distance of the light frustum's near plane, respectively, r is the resolution of the depth map, and l is the width of the light source. Then, the kernel width w_k is computed as follows:

$$w_k = \frac{ln}{w}\left(\frac{1}{z'} - \frac{1}{d}\right) = \frac{ln}{w}\left(\frac{1}{z} - \frac{1}{d}\right) + \frac{1}{r}, \tag{2}$$

where d is the distance from the light plane to the shadow rectangle. The kernel width w_k is computed separately for each of the u- and v-directions of the depth map.

Recall that the shadow extent is cut by a plane to generate the shadow rectangle, and the distance d from the light source to the shadow rectangle determines the kernel width w_k. Therefore, the position of the cutting plane has to be chosen carefully. This issue will be discussed in Sect. 5.4.

3.3 Computing occlusion degrees

The pseudo code for LOM construction is shown in Fig. 5, which is a detailed version of the algorithm presented in Fig. 2. In the previous sub-section, we identified the potential occludees using the kernel, as illustrated in Fig. 4(b). It corresponds to the first five lines of the pseudo code in Fig. 5. This sub-section discusses the innermost for loop of the pseudo code, i.e., for each potential occludee, we compute the occlusion degree caused by all micro-patches of the sample under process.

The micro-patches are *back-projected* from a potential occludee to the light plane to compute the occlusion degree. Figure 4(c) shows the result of back-projecting the micro-patch of z_1^2 from the potential occludee at z_1^1. The light source and the projected image overlap, and therefore the rays from the light source are determined to be partially occluded. The occlusion degree at z_1^1 is determined using the overlap area. Computing the overlap area is quite simple, and readers are referred to [14].

The micro-patch of z_2^2 need not be back-projected to compute the occlusion degree at z_1^1, because z_2^2 is farther from the light source than z_1^1, and therefore the micro-patch of z_2^2 cannot cast shadows to z_1^1. Similarly, only the micro-patch of z_1^2 is back-projected to compute the occlusion degree of z_2^1. In contrast, both micro-patches of z_1^2 and z_2^2 are back-projected for z_3^1.

After back-projection, we may have more than one micro-patch that are overlapped with the light source. Then, following the method proposed by Bavoil et al. [6], we choose the one that is closest to the potential occludee under process, and take its occlusion degree.

Now, consider $\{z_1^3, z_2^3\}$ retrieved from s^3. Neither z_1^2 nor z_2^2 is closer to the light source than z_1^3, and therefore no micro-patch is back-projected. The occlusion degree at z_1^3

```
1: for each sample of the depth map
2:     Retrieve the depth values $z_i$s
3:     Compute the kernel size
4:     for each sample in the kernel
5:         Retrieve the depth values $z_j$s
6:         for each potential occludee at a depth value $z_j$
7:             Compute the occlusion degree by back-projecting
               the micro-patches at $z_i$s
```

Fig. 5 Pseudo code for LOM construction

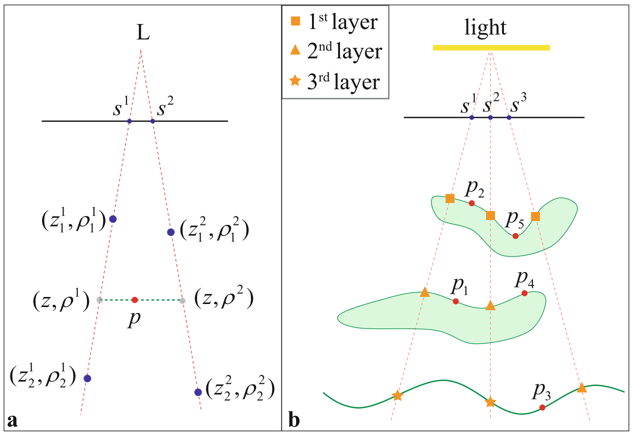

Fig. 6 LOM filtering: For a fragment p to be rendered, the depth map and LOM are referenced and a depth-based tri-linear interpolation is performed. (**a**) 2D illustration of the tri-linear interpolation; (**b**) p_1 is processed by the basic tri-linear interpolation, p_2 and p_3 are processed with the aid of virtual bounds, p_4 requires special handling of the occlusion degrees to be interpolated, and p_5 shows an artifact that rarely appears for a depth map of a reasonable size

is set to zero. With respect to z_2^3, only the micro-patch of z_1^2 is back-projected. However, there is no overlap between the light source and the projected image, as shown in Fig. 4(d), and the occlusion degree is set to zero.

Both of z_1^3 and z_2^3 are located outside the shadow extent, and cannot be occluded by all micro-patches of the sample s^2. They are collected as potential occludees because the kernel volume is a superset of the shadow extent, as shown in Fig. 4(c). However, the occlusion degrees are correctly computed and set to zero, through simple computation.

The same process is done for the depth values of the current sample. In Fig. 4(b), the occlusion degree brought by the micro-patch of z_1^2 is computed for z_2^2. However, computing the occlusion degree is not needed for the closest depth z_1^2.

Recall that the depth map and the LOM have the same structure. For example, if z_1^1, z_2^1, and z_3^1 are stored in the R, G, and B channels of the depth map texture, respectively, the occlusion degrees for them are also stored in the R, G, and B channels of the LOM texture.

4 LOM filtering

The filtering method we have adopted is depth-based and follows a tri-linear interpolation. When the scene is rendered from the viewpoint of the camera, each fragment is transformed into the light space, and the four neighbor samples in the depth map are located. In the 2D illustration of Fig. 6(a), s^1 and s^2 are located for the fragment p. Its depth value z is used to identify the *bounding layers* at each sample. For example, the first and second layers of s^1 bound p because $z_1^1 < z < z_2^1$. When the bounding layers are identified, the LOM is searched for the occlusion degrees at the same layers. (Recall that both the depth map and LOM have the same structure.)

In Fig. 6(a), each scene point is described by a pair of its depth and occlusion degree ρ. For s^1, the distances, $(z - z_1^1)$ and $(z_2^1 - z)$, are used to interpolate ρ_1^1 and ρ_2^1 and determine ρ^1. Similarly, ρ^2 is computed for s^2. Finally, ρ^1 and ρ^2 are interpolated to determine the occlusion degree of p. In Fig. 6(b), the occlusion degree of p_1 is computed using the tri-linear interpolation.

However, the bounding layers are not always identified. Consider the fragment p_2 in Fig. 6(b). With respect to s^2, the first layer is the lower bound, and there is no upper bound. For such a case, a virtual bound, called the 0th layer, is placed at the light source, and its occlusion degree is set to 0 (fully lit).

On the other hand, consider p_3 in Fig. 6(b). With respect to s^3, the second layer is the upper bound, and there is no lower bound. For such a case, a virtual bound is placed at the far distance of the light frustum, and its occlusion degree is set to 1 (fully occluded). The occlusion degrees at p_2 and p_3 are computed through the tri-linear interpolation with the aid of the virtual bounds.

Consider p_4 in Fig. 6(b) that is obviously 'fully occluded.' When the tri-linear interpolation is applied, however, p_4 will be determined to be 'partially occluded' since the upper bounds along s^2 and s^3 are fully lit, even though the lower bounds are fully occluded. We check if the four upper bounds are of the same layer, to handle such a case. If so, our heuristic assumes that a surface patch connecting the four points occludes light rays to the current fragment. This is true for p_4, where all the upper bounds belong to the first layer. When all of the upper bounds belong to a layer, their occlusion degrees are ignored, and instead 1s (denoting 'fully occluded') are used for tri-linear interpolation.

We have so far discussed the complete process of LOM filtering. Unfortunately, the filtering method reveals a problem. Consider p_5 in Fig. 6(b), which is obviously fully lit. Its upper bounds belong to the first layer, and their contributions to the tri-linear interpolation are taken as 1s. Then, the resulting degree of occlusion must be far from 'fully

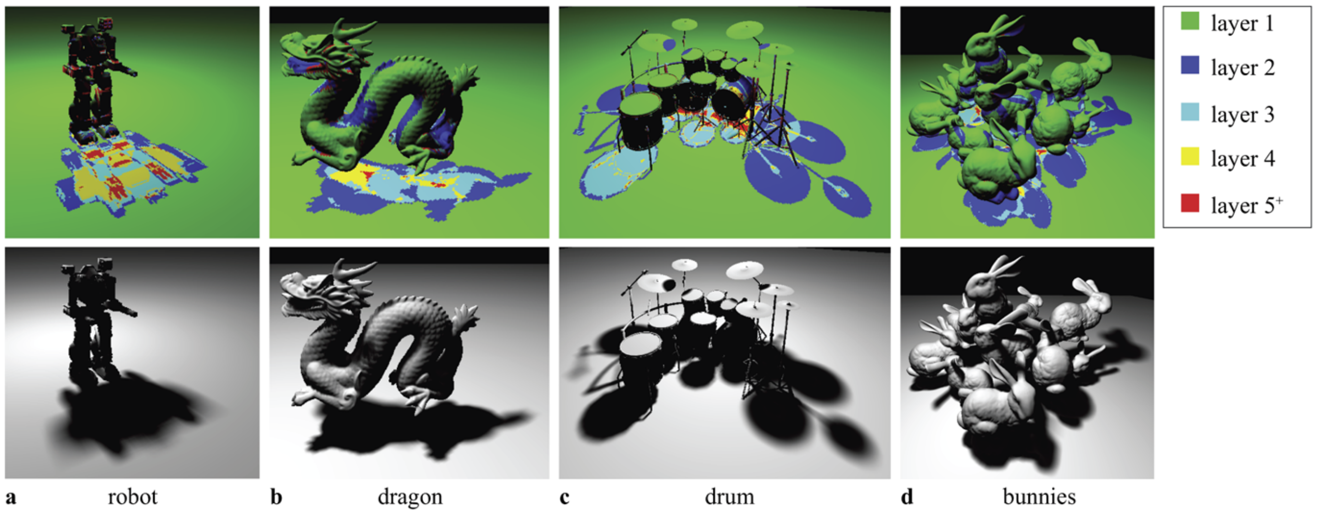

Fig. 7 Layer visualization: The *first row* illustrates the layers in different colors, and the shadowed scenes are rendered at the *second row*

lit.' This error occurs, since p_5 is deeper than its upper bound points, i.e., p_5 locates in a dent of the surface patch connecting its upper bounds. Such a dent usually appears when we use a coarse resolution for the depth map. As will be discussed in Sect. 7.2, the 512×512-resolution proves to be a good choice satisfying both the high frame rate and visual quality. Fortunately, the artifact shown at p_5 of Fig. 6(b) is hard to find in such a fine resolution, since the surface patch tends to be planar and rarely has a dent.

The depth peeling process presented in Sect. 3.1 samples the scene in multiple depth layers. Two nearby samples of a surface may be stored in different layers, and their occlusion degrees may be stored in different layers of LOM. As a result, a bi-linear filtering in the same layer often gives incorrect results. This is why the tri-linear interpolation is adopted in our method.

5 Optimization techniques

5.1 Number of depth layers

As discussed in Sect. 3.1, a single texture is sufficient for the depth map of a scene with up to four layers, but additional textures would be needed if a scene has more than four layers. This sub-section shows that, in general, a single texture is sufficient even for the latter case.

Figure 7 visualizes the layers for four scenes, together with their shadowed images. Layers 1, 2, 3, and 4 are colored in green, blue, cyan, and yellow, respectively. The fifth and deeper layers (denoted by 5^+) are colored in red. (Colored in black are the back-faces.) Observe that, as we go down the layers, the areas taken by the layers are significantly reduced.

Note that the points on the first layer are either fully lit ($\rho = 0$) or partially occluded. In contrast, the points on the second or deeper layers are either partially occluded or fully occluded ($\rho = 1$). Table 1 shows the percentage and average occlusion degree of the partially occluded points in the second and deeper layers. In the robot scene of Fig. 7(a), for example, 0.3% of the fifth and deeper layers' points is partially occluded (this implies that 99.7% is fully occluded), and the occlusion degree of the points is on average 0.92. In the scenes of Table 1, most of the points in the fourth and deeper layers are fully occluded, and the occlusion degrees are high even for the partially occluded.

Consider a point p on the fifth layer of the robot scene. Suppose that p will be a discretized point in the depth map if the depth map is extended beyond the fourth layer. However, a 4-channel texture is used, and therefore p is not discretized. Suppose that p happens to be a fragment to be rendered in the screen. Then, its occlusion degree ρ is obtained through the tri-linear filtering algorithm, and is 1 for almost all cases.[1]

If the point p on the fifth layer is discretized, it will be either partially occluded or fully occluded. Suppose that it is fully occluded. Then, its occlusion degree ρ_o computed through the back-projection process will be 1. There is then little difference between ρ_o and ρ.

Conversely, suppose that p is partially occluded. Then, ρ_o is not 1, and ρ can be taken as an *over-estimation* of ρ_o. However, such an over-estimation is not prominent due to

[1]The upper bound lies at the fourth layer, and its occlusion degree is 1 with the probability of 98.1%, according to Table 1. In contrast, the lower bound is virtual, and its occlusion degree is taken as 1. When these two values are interpolated, the result is 1 for almost all cases.

Table 1 Percentage and average occlusion degree of the 'partially occluded' points in the second and deeper layers: The statistics are obtained by discretizing each of the four scenes in Fig. 7 from ten distinct light positions, and counting the non-NULL depth values in each layer. The depth map resolution is 512 × 512

Layer	Robot		Dragon		Drum		Bunnies	
	Percentage (%)	Occlusion degree	Percentage (%)	Occlusion degree	Percentage (%)	Occlusion degree	Percentage (%)	Occlusion degree
2	61.1	0.84	59.3	0.87	65.2	0.87	66.0	0.84
3	13.3	0.86	12.2	0.84	6.8	0.83	5.5	0.88
4	1.9	0.90	3.9	0.86	1.1	0.86	1.4	0.94
5^+	0.3	0.92	4.9	0.96	0.1	0.93	3.4	0.97

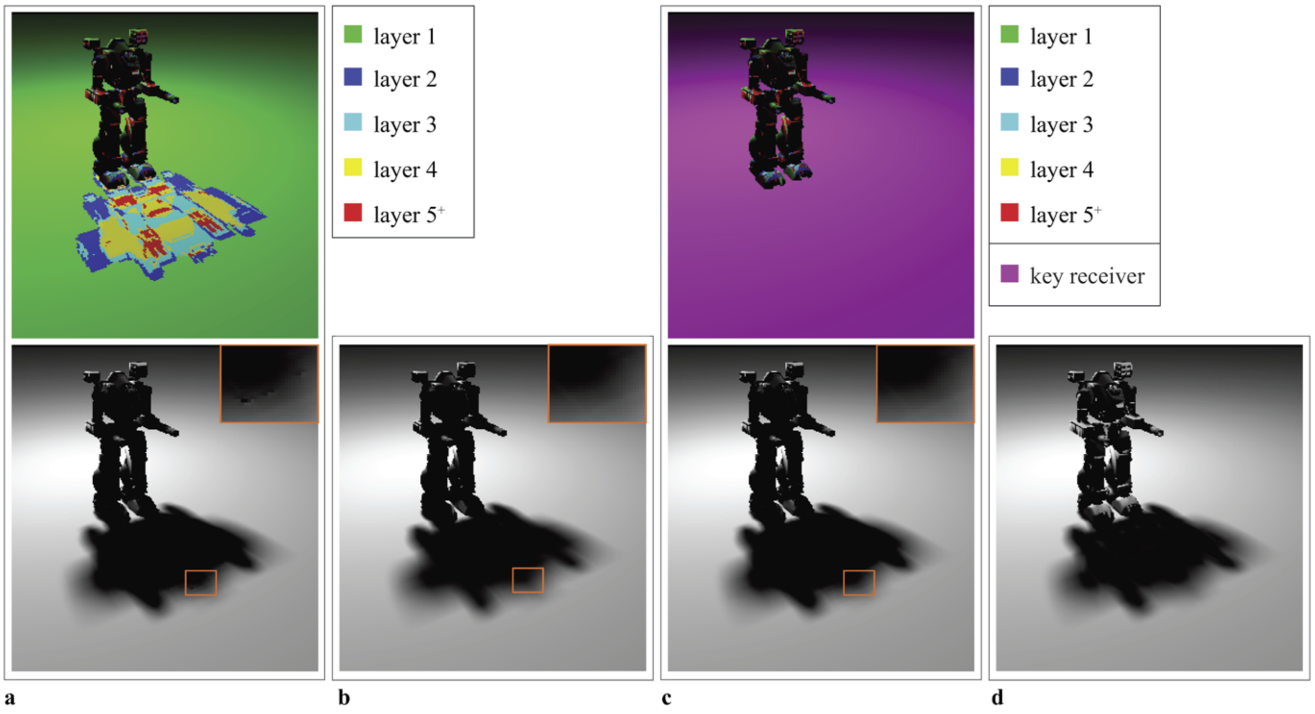

Fig. 8 The robot scene: (**a**) rendered with four layers; (**b**) reference image (1024 light samples); (**c**) rendered by specifying a key receiver; (**d**) rendered by separating the receiver from the occluder [4]

the following: (1) the fifth and deeper layers take a small fraction, as visualized in Fig. 7(a), (2) the partially occluded points such as p take 0.3% even in the layers, and (3) the amount of over-estimation is small because the correct degree ρ_o is also heavily occluded, on average 0.92. Such a small over-estimation in an infinitesimal area is hardly noticeable. We can make similar discussions for the other scenes in Fig. 7 and Table 1. Our experiments in a variety of scenes prove that a single texture is sufficient to discretize a scene with little noticeable artifact. If we add more textures, more computing time and memory are needed, but little gain is obtained.

5.2 Receiver specification

In virtual environments such as video games, some *shadow receivers* such as floors and walls often play key roles in helping us identify the spatial information of the scene because they usually take a large part in the scene and the shadows are cast on them for a relatively long time. Suppose that the number of depth layers is limited to four, and some pixels of such key receivers are over-estimated when rendered. Figure 8(a) shows an example. Compare it to the reference image in Fig. 8(b), which is rendered with 1024 point light sources sampled from the area light. Even though the artifact

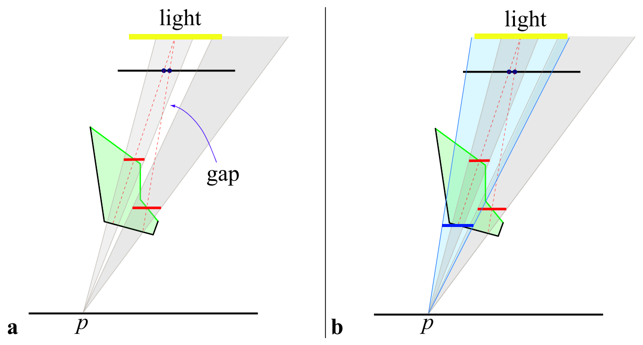

Fig. 9 Light leaking problem alleviated using back-faces: (**a**) light leaking; (**b**) use of the back-faces to reduce the light leakage

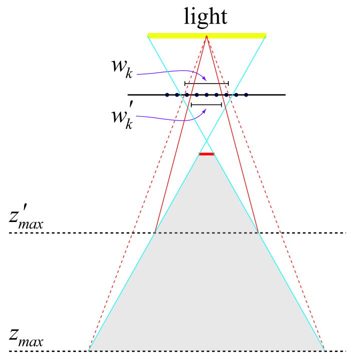

Fig. 10 Kernel size reduction: Using $z_{\max}$, the depth value of the deepest scene point, we compute w_k. Within the kernel defined by w_k, the new deepest value $z'_{\max}$ is retrieved, and then the reduced kernel size w'_k is computed. The hierarchical depth map aids this process

is hard to perceive, it would be desirable for it to disappear because it is generated on the key receivers.

The key receiver is enforced to be discretized in constructing the depth map for this purpose. For example, the floor in Fig. 8 is processed separately to generate a single-layer depth map. Then, the remaining objects in the scene are processed normally in the depth-peeling mode to construct a three-layer depth map. Finally, the two maps are packed in a four-layer depth map. Figure 8(c) shows the rendering result using the combined depth map.

It is important to understand that unconditionally processing such a key receiver in our method has little to do with the traditional method of partitioning the scene into occluders and receivers. A good example showing the difference is self-shadow or cast shadow among occluders; this cannot be generated when the scene is partitioned into occluders and receivers. The image in Fig. 8(d) is generated by the algorithm of [4], where the robot is taken as the occluder, and the floor is taken as the receiver. Note that, in Fig. 8(d), self-shadow is not generated on the occluder.

5.3 Back-faces and optimal layer configuration

The gaps between the micro-patches may often result in *light leaking*. For example, the point p in Fig. 9(a) receives some light rays even though it is fully occluded in reality. If the back-faces are used for computing the occlusion degree, the light leaking problem can be greatly alleviated, as illustrated in Fig. 9(b).

In the current implementation, the back-faces are stored in a separate depth map, and are used only for computing the occlusion degrees for the front-faces. (The front-face depth map and the LOM have the same structure.) The occlusion degrees need not be computed for the back-face points because they are always fully occluded.

For a deeper point in the scene, the light leaking problem is scarcely observed because multiple layers above it usually close the gap. In our experiments, we observe that light leaking mostly occurs on the second (front-face) layer. For a closed object, the first back-face layer lies between the first and second front-face layers, and therefore greatly contributes to reduce the light leaking problem.

According to our experiments, the best configuration of the layers is set to the following: two or three front-face layers, one layer for the key receiver, and one back-face layer. The first two are packed in the front-face depth map. In this configuration, the light leaking problem has not been observed.

5.4 Kernel size reduction

As discussed in Sect. 3.2 and Fig. 4, the shadow extent is cut by a plane to generate the shadow rectangle. In (2), the distance d from the light source to the shadow rectangle determines the kernel width w_k. The cutting plane's position should be correctly computed such that all of the potential occludees affected by the currently processed sample are located above it. Otherwise, some of the potential occludees may be missing.

If the far plane of the light frustum is used as the cutting plane, all of the scene points are considered and therefore no potential occludee can be missing. However, it is an extremely inefficient method. The farther the cutting plane is, the larger the kernel size is. The kernel size should be made as small as possible to save the occlusion degree computation.

For the purpose of kernel size reduction, a hierarchical depth map is constructed from the depth map. Level 0 of the hierarchy is constructed by taking the largest (deepest) value per sample. Then, 2×2 neighbors of level i are recursively combined into a texel of level $i + 1$ which stores the largest of the neighbors. The top level of the hierarchy stores the depth value of the deepest scene point. Let us call it $z_{\max}$.

Figure 10 illustrates the kernel size reduction process. The initial kernel size w_k is computed using (2) and $z_{\max}$. Then, the level l to visit in the hierarchy is defined by

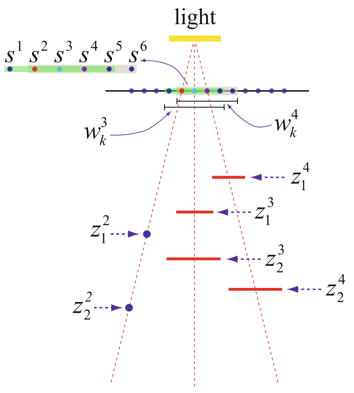

Fig. 11 Sample grouping to reduce multiple fetches of a depth value: The kernels of s^3 and s^4 share sample s^2. When s^3 and s^4 are grouped and processed at a time, the micro-patches at z_1^3 and z_1^4 are back-projected from z_1^2 simultaneously. Grouping 2×2 samples leads to around 25% performance gain due to the decreased number of texture fetches

$\lceil \log_2(w_k) \rceil$, and the largest depth $z'_{\max}$ is retrieved from the four neighboring texels at level l. The cutting plane is moved to $z'_{\max}$, and the new kernel size w'_k is computed. The reduction step can be repeated but, in practice, a single step is sufficient.

The hierarchy we have implemented is similar to that of [14]. Some other methods such as [23, 25] provide more precise results at the cost of more expensive computation.

5.5 Reduced texture fetches

A set of potential occludees affected by a sample is processed at a time in constructing LOM. Note that adjacent samples usually share some potential occludees. Figure 11 shows an example. The kernels of s^3 and s^4 are $\{s^1, s^2, s^3, s^4, s^5\}$ and $\{s^2, s^3, s^4, s^5, s^6\}$, respectively. They overlap. For instance, they share s^2. Then, the potential occludee at z_1^2 will be fetched twice if we process s^3 and s^4 separately. This is inefficient. The texture fetch cost can be reduced by grouping s^3 and s^4.

Let us construct a texture whose size is a quarter of the original depth map by grouping the 2×2 samples. The new texture stores the smallest depth value of the four samples, e.g., z_1^4 in Fig. 11 when s^3 and s^4 are grouped. (The smallest is chosen because the micro-patch closest to the light source produces the shadow extent encompassing all potential occludees.)

We then need small changes in the algorithm of Fig. 5. The samples in the 'new texture' are processed one at a time. However, the depth values associated with the original four samples are fetched from the depth map, and their micro-patches are computed. The kernel size is computed using (2), where z and r are with respect to the 'new texture,' i.e., z is the smallest depth among the four samples, and r is the reduced resolution. Finally, the micro-patches from the four samples are back-projected for determining the occlusion degree. In Fig. 11, for example, the micro-patches at z_1^3 and z_1^4 are back-projected from z_1^2 and accumulated to determine the occlusion degree of z_1^2.

We could choose to group 3×3 or larger samples, but grouping 2×2 samples performs best. Grouping 2×2 samples leads to about 25% performance gain.

```
Vertex shader
    Retrieve the depth values $z_i$s
    Compute the kernel size

Geometry shader
    Triangulate the kernel

Pixel shader
    Retrieve the depth values $z_j$s
    for each potential occludee at a depth value $z_j$
        Compute the occlusion degree by back-projecting
        the micro-patches at $z_i$s
```

Fig. 12 Shaders for LOM construction

6 Implementation

Figure 12 shows how the LOM construction algorithm of Fig. 5 is performed by the vertex, geometry, and pixel shaders of the Shader Model 4.0. The vertex shader is invoked for each sample of the depth map, and performs two tasks: (1) it retrieves the depth values z_is for the current sample and passes them to the pixel shader, (2) it computes the kernel, i.e., w_ks along u- and v-directions, using the algorithms presented in Sects. 3.2 and 5.4.

The sample position and w_ks are passed to the geometry shader, which then converts the kernel quad centered at the sample position into two triangles in a triangle strip. The rasterizer produces the fragments of the triangles, that correspond to the samples in the kernel. Each fragment goes through the pixel shader, where the depth values z_js for the fragment are read out from the depth map, the micro-patches are computed from the depth values z_is passed from the vertex shader, and then the micro-patches are back-projected from each z_j to compute the occlusion degree.

We use a 16-bit RGBA floating-point texture for the depth map, and a 32-bit RGBA floating-point texture for the LOM. The pixel shader uses the 32-bit floating-point hardware blending capability to accumulate the occlusion degrees for a discretized scene point. This does not suffer from the 16-bit blending accuracy, which Atty et al. [4] discussed thoroughly.

7 Experimental results

The experiments were performed on an Intel Core2 CPU 1.86 GHz with an NVIDIA GeForce 8800 GTS graphics card. The GPU code is in DirectX 10 and HLSL. All images were rendered in 1024 × 768 resolution.

7.1 Scene complexity

Figure 13(a) shows the result of rendering the dragon model of 870K polygons using the proposed algorithm running on 512 × 512-resolution depth map and LOM. Two front-face layers and a back-face layer are used for the dragon, and a layer is assigned for the ground specified as the key receiver. The proposed algorithm runs at 47 fps/footnote. The frame rates reported in this section were measured for the entire rendering process including the depth map and LOM construction, soft shadow filtering, lighting and shading. Figure 13(b) shows the reference image rendered with 1024 point light sources sampled from the area light.

Figure 14 shows a series of scenes with increasing complexity, and Fig. 15 shows their frame rates. (The texture resolution is 512 × 512.) This proves the proposed method scales well with the scene complexity. The scene in Fig. 14(d) has more than 1M polygons, and is rendered at 32 fps.

7.2 Texture resolution

In Fig. 16, the scene is rendered by changing the texture resolutions, i.e., the resolutions of the depth map and LOM. The first row shows that the soft shadows become smoother as the texture resolution increases. The second row shows the difference images, which visualize the pixel-wise differences between the images of the first row and the reference image. The pixel-difference is visualized as a gray color, where a lighter pixel implies a larger difference. It is obvious that the difference or error decreases as the texture resolution increases.

Figure 17 illustrates the frame rates obtained by changing the texture resolutions. Together with Fig. 16, this shows the tradeoff between the frame rate and the visual quality. As the bottleneck of the proposed method lies in LOM construction, it is important to determine the LOM's resolution. Our experiments show that a 512 × 512-resolution provides satisfactory results in both frame rate and visual quality. As can be found in Fig. 16, the soft shadows in the resolutions of 512 × 512 and 1024 × 1024 do not make a big difference. More discussion on the relation between the texture resolution and frame rate will be given in Sect. 7.3.

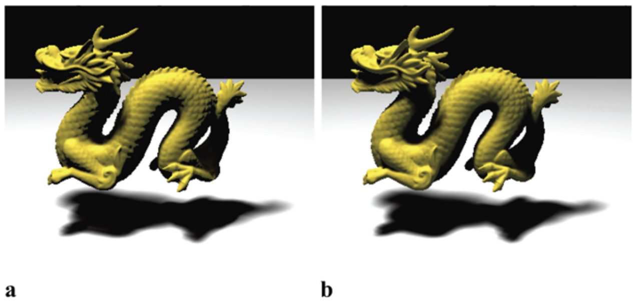

Fig. 13 Dragon of 870K polygons: (**a**) rendering result of the proposed algorithm using 512 × 512-resolution maps (47 fps); (**b**) reference image (1024 light samples)

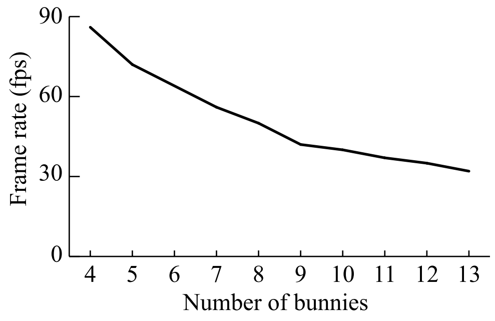

Fig. 15 The frame rates with the varying number of bunnies in the scenes of Fig. 14

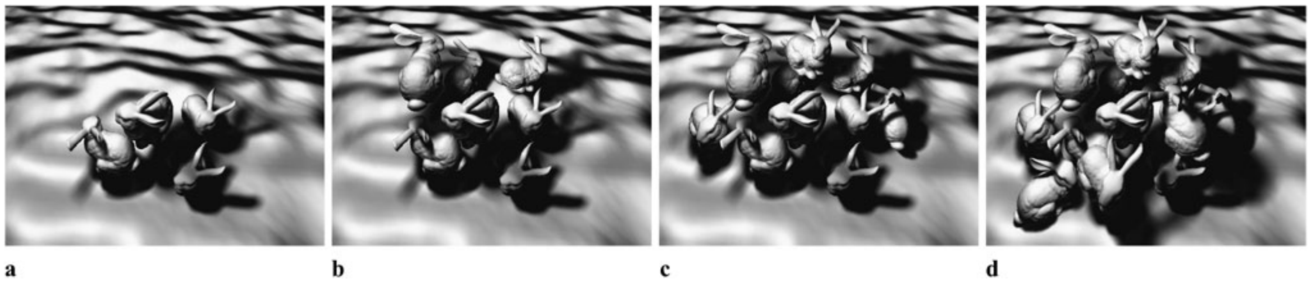

Fig. 14 The bunny has 69K (69,451) polygons, and the terrain has 160K polygons: (**a**) 4 bunnies & 438K polygons in total (86 fps); (**b**) 7 bunnies & 646K polygons in total (56 fps); (**c**) 10 bunnies & 854K polygons in total (40 fps); (**d**) 13 bunnies & 1M polygons in total (32 fps)

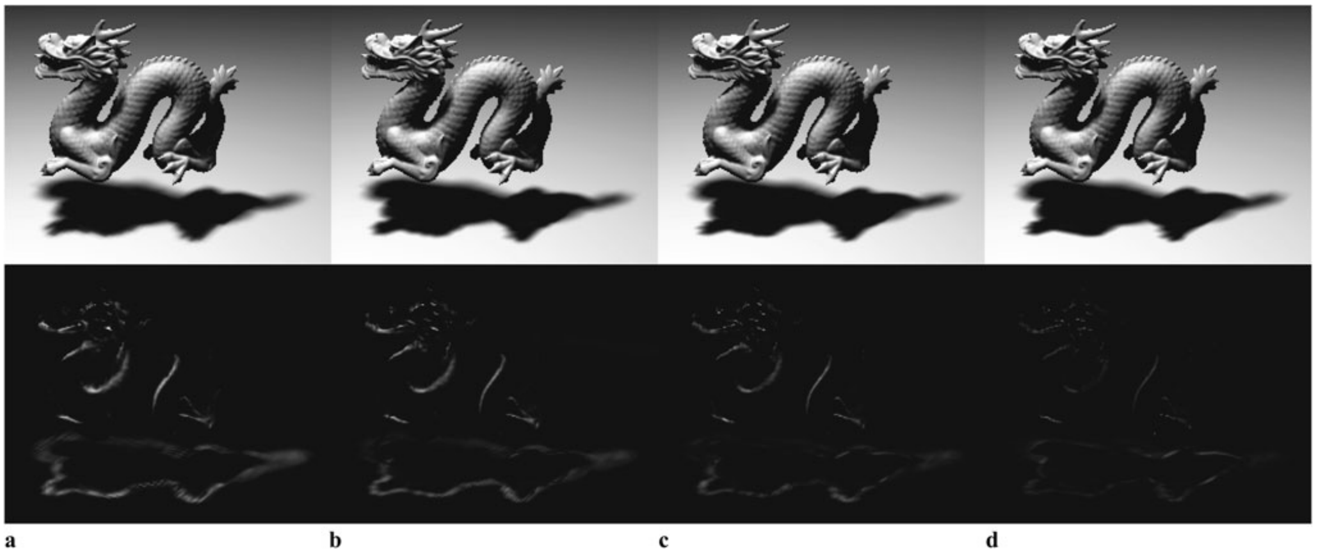

Fig. 16 (*First row*) Soft shadows with the varying resolution of the depth map: The soft shadows become smoother as the depth map resolution increases. (*Second row*) Difference between the images of the *first row* and the reference image: The pixel-difference is visualized as a gray color, and the difference decreases as the depth map resolution increases: (**a**) 128 × 128; (**b**) 256 × 256; (**c**) 512 × 512; (**d**) 1024 × 1024

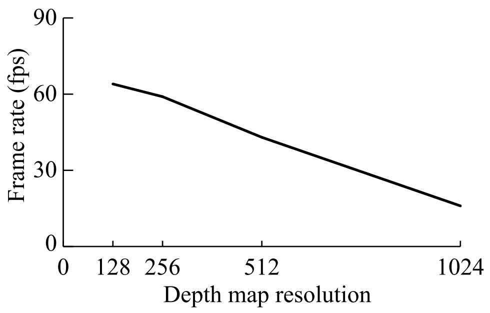

Fig. 17 The frame rates with the varying depth map resolution in the scene of Fig. 16 (Dragon, 870K polygons)

Table 2 Performance comparison of three cases: `I`. (texture resolution, light size) = (256 × 256, 0.5), `II`. (texture resolution, light size) = (512 × 512, 0.5), and `III`. (texture resolution, light size) = (512 × 512, 1). The row of 'Depth map construction' includes building the hierarchy presented in Sect. 5.4, and the rendering time includes LOM filtering as well as lighting and image texturing

	`I`	`II`	`III`
Depth map construction	1.01	1.67	1.67
LOM construction	1.48	5.32	11.24
Rendering	0.92	0.95	0.95
Total (ms)	3.41	7.94	13.86
fps	293.26	125.94	72.15

7.3 Light source size

Figure 18 shows how soft shadows change when the light source size changes. Even though our algorithm handles a rectangular light source, for the sake of simplicity, we assume a square light source with a side length of a. The bounding box of the bunny model is computed, and the longest edge of length b is selected. In Fig. 18(a), the ratio of a and b is made 0.05, i.e., $a/b = 0.05$. It simulates a point light source, and an almost hard shadow is generated. As a/b increases, the shadow becomes softer, i.e., the penumbra area becomes larger.

If the size of the light source increases, the kernel size accordingly increases in our method, as can be noticed in (2). The larger the kernel, the larger is the set of the potential occludees. Therefore, more time is taken to construct the LOM. Note that, however, the vertex and geometry shaders presented in Fig. 12 do the same work regardless of the light source size. Only the pixel shader's work increases.

Table 2 compares the performances of three cases with different texture resolutions and light sizes, for the bunny scene in Fig. 18. Compare Cases `II` and `III`. When the light size is magnified four times, the LOM construction time increases about twice, not four times, because only the pixel shader's work increases. (Obviously, the depth map construction time and rendering time do not change between `II` and `III` because the texture resolution is fixed.) Figure 19 shows the frame rates as a function of the light source size in the test scene of Fig. 18 with the depth map resolution of 512 × 512.

Let us now consider the relation between the texture resolution and frame rate. In Table 2, the texture resolutions

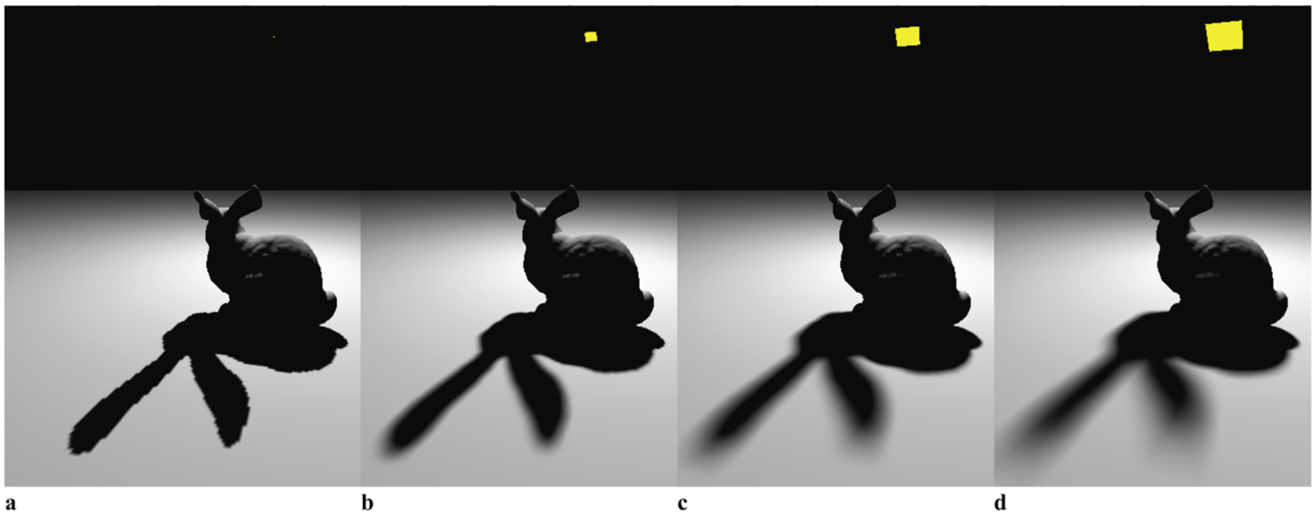

Fig. 18 The light source is *square*, and its side length is a. The bounding box is computed for the bunny, and the longest edge is of length b: (**a**) $a/b = 0.05$ (simulation of a point light source); (**b**) $a/b = 0.25$; (**c**) $a/b = 0.5$; (**d**) $a/b = 0.75$

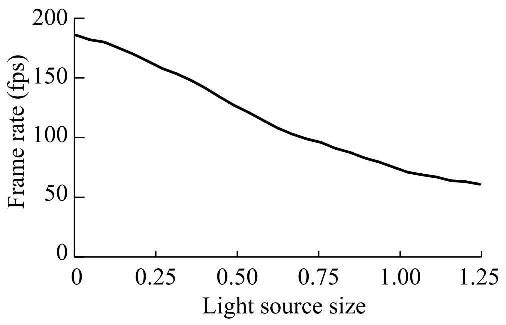

Fig. 19 The frame rates with varying light source size in the scene of Fig. 18

of Cases I and II are 256 × 256 and 512 × 512, respectively. As expected, the LOM construction time increases about four times when the texture resolution is changed from 256 × 256 to 512 × 512. In contrast, the depth map construction time does not. It is because the depth map construction time depends not only on the texture resolution but also on the scene complexity, which is identical for Cases I and II. Table 2 shows that, when the texture resolution is magnified four times, the overall performance decreases by about half. This observation is compatible with the graph in Fig. 17.

The relationship between the light source size and the texture resolution is worth noting. As shown in Fig. 16, the difference or error is usually found in the penumbra area. Note that a scene with a larger light source has a larger penumbra area, where the error caused by the small texture resolution is harder to observe. Therefore, a smaller resolution can be chosen for a larger light.

8 Discussion

Our method was inspired by the work of Atty et al. [4], especially by the techniques of accumulating the occlusion degrees and using two depth layers (front- and back-face layers). However, our work is distinct in many aspects. First, unlike the work of Atty et al. where the occluder and the receiver are clearly separated and consequently no self-shadow is generated, we do not distinguish between them, and self-shadow is obtained. Such a difference was illustrated in Fig. 8. As presented in Sect. 5.2, a specific surface can be designated as the key receiver, but its purpose is to discretize and correctly shadow the surface. Both the key receiver and the other objects can be self-shadowed in our method.

Secondly, our method does not send data back to the CPU and all computation is performed within GPU, whereas Atty et al. [4] send the depth map to CPU and many drawcalls are invoked, degrading the performance.

Thirdly, our method pursues a nearly-optimal layer configuration: two or three front-face layers, one back-face layer, and one layer for the key receiver, as discussed in Sect. 5.3. Figure 20(a) shows the light leaking artifact of the work of Atty et al., where the front-face and back-face depth maps are constructed for the tree designated as the occluder, and the floor is taken as the receiver. Compare this to our result shown in Fig. 20(e), which adopts the nearly-optimal layer configuration and therefore rarely reveals light leaking. Our result is quite close to the reference image shown in Fig. 20(f).

In general, the algorithms based on micro-patch back-projection suffer from the light leaking artifact caused by

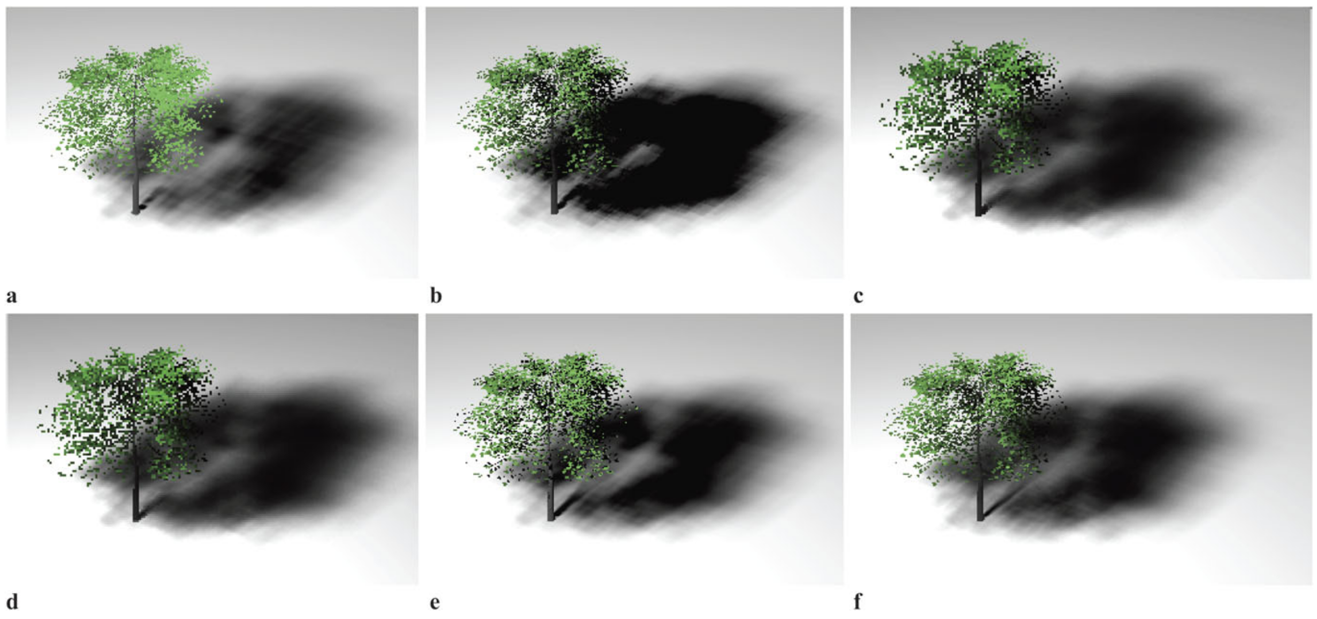

Fig. 20 Comparison of soft shadows: (**a**) light leak in Atty et al. [4] at 12 fps; (**b**) over-shadow in Guennebaud et al. [14] at 27 fps; (**c**) improved shadow quality and degraded fps in Guennebaud et al. [15] at 16 fps; (**d**) compatible shadow quality and improved fps in Yang et al. [32] (a single light sample) at 35 fps; (**e**) compatible shadow quality and higher fps in our method at 46 fps; (**f**) reference image (1024 light samples)

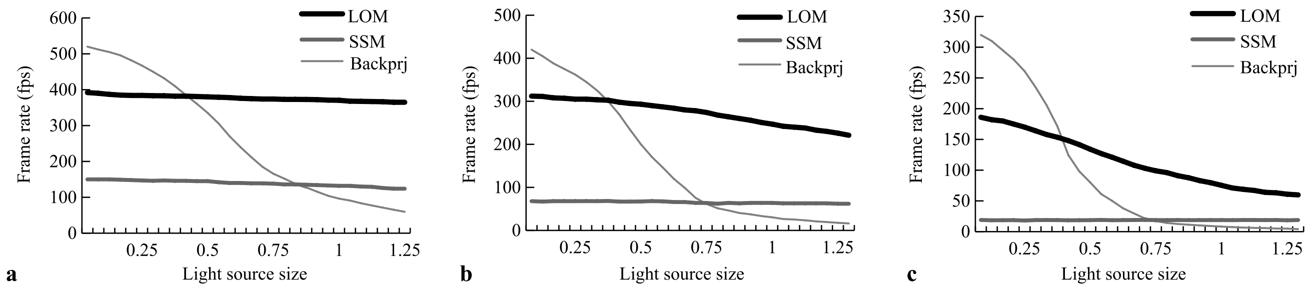

Fig. 21 Performance comparison of our method (LOM), Atty's soft shadow map method (SSM), and Guennebaud's back-projection method (Backprj) with the scene in Fig. 18 and texture resolution of: (**a**) 128 × 128; (**b**) 256 × 256; (**c**) 512 × 512

the discrete nature of the depth map, as discussed in Sect. 2. Our method alleviates this by pursuing a nearly-optimal layer configuration. In contrast, the work of Guennebaud et al. [14] adopted the so-called gap-filling method. Unfortunately, this solution leads to the over-shadow artifact, as illustrated in Fig. 20(b). Guennebaud et al. [15] proposed to detect the shadow contours for improving the shadow quality, and Fig. 20(c) shows the result. However, it comes at the cost of degraded fps.

Yang et al. [32] followed the approach of Guennebaud et al. [15], but grouped a number of nearby pixels in the screen space and processed the group at a time for the purpose of improving performance. Figure 20(d) shows the result. Our method in Fig. 20(e) shows a higher fps, and a reason is that the penumbra coherence exploited in [32] is broken for the scene of tree with foliage.

Bavoil et al. [6] followed the framework of Guennebaud et al. [14], and adopted multiple depth maps instead of gap filling to avoid the over-shadow artifact. Their method can produce better results, but performance is low especially when the penumbra pixels take a larger part in the screen, as discussed in Sect. 2. Like Bavoil et al. [6], our method also used the layered depth map, but back-projection is done in the light space, not in the screen space, to generate LOM, and the LOM is filtered at the rendering time. As a result, the rendering performance is made relatively constant, independently of the screen configuration.

Figure 21 compares the performances of our method, the work of Atty et al. [4], and the work of Guennebaud et al. [14]. The work of Guennebaud et al. performs better than the others when the light source size is kept small. It is because their work discards fully lit or fully occluded pixels, and processes only the penumbra pixels. When the light size is small, the penumbra area is small, and therefore fewer pixels are processed. When the light size increases, however, more pixels have to be processed, and therefore performance degrades. Figures 20 and 21 show that, in general, our method excels the others.

9 Conclusions and future work

In this paper, we presented a high-performance and high-quality algorithm to produce soft shadows. The algorithm is image-based, requires no pre-computation, maximally utilizes the GPU, demands no CPU read-back, runs quite fast, and generates self-shadows.

Although not physically correct, the resulting shadows are close to those of the ground truth reference image. The proposed method can be combined with some of the advanced shadow map techniques [28, 30] which enhance the object discretization quality in the shadow map.

Using a fixed number of layers in the depth and occlusion maps, we have obtained high-quality soft shadows, i.e., the resulting artifact is hardly prominent. Nonetheless, there exists a possibility of further enhancing the proposed method by using, for example, the shadow map list [26] in order to complement the information for missing discretized points.

Acknowledgements We would like to thank Gaël Guennebaud and Kiwon Um for valuable discussions and suggestions. The Stanford Bunny and Dragon models were digitized and kindly provided by the Stanford University Computer Graphics Laboratory. The Waterlily, Robot, Drum set and Tree models appearing in this paper and in the accompanying video are downloaded from 3dxtras.com. This research was supported by MKE, Korea under ITRC NIPA-2010-(C1090-1021-0008).

References

1. Agrawala, M., Ramamoorthi, R., Heirich, A., Moll, L.: Efficient image-based methods for rendering soft shadows. In: Proc. of SIGGRAPH'00, pp. 375–384 (2000)
2. Akenine-Möller, T., Assarsson, U.: Approximate soft shadows on arbitrary surfaces using penumbra wedges. In: Rendering Techniques, pp. 297–305 (2002)
3. Arvo, J., Hirvikorpi, M., Tyystjärvi, J.: Approximate soft shadows using image-space flood-fill algorithm. Comput. Graph. Forum **23**(3), 271–280 (2004). (Proc. of Eurographics'04)
4. Atty, L., Holzschuch, N., Lapierre, M., Hasenfratz, J.M., Hansen, C., Sillion, F.: Soft shadow maps: Efficient sampling of light source visibility. Comput. Graph. Forum **25**(4) (2006)
5. Bavoil, L., Silva, C.T.: Real-time soft shadows with cone culling. In: Technical Sketches and Applications at SIGGRAPH'06 (2006)
6. Bavoil, L., Callahan, S.P., Silva, C.T.: Robust soft shadow mapping with backprojection and depth peeling. J. Graph. Tools **13**(1), 19–30 (2008)
7. Crow, F.C.: Shadow algorithms for computer graphics. Comput. Graph. **11**(2), 242–248 (1977). (Proc. of SIGGRAPH'77)
8. Drettakis, G., Fiume, E.: A fast shadow algorithm for area light sources using backprojection. Comput. Graph. Forum **28**, 223–230 (1994)
9. Eisemann, E., Décoret, X.: Occlusion textures for plausible soft shadows. Comput. Graph. Forum **27**(1), 13–23 (2008)
10. Everitt, C.: Interactive order-independent transparency. Tech. rep., NVIDIA Corporation (2001)
11. Fernando, R.: Percentage-closer soft shadows. In: ACM SIGGRAPH 2005 Sketches and Applications, p. 35 (2005)
12. Forest, V., Barthe, L., Paulin, M.: Accurate shadows by depth complexity sampling. Comput. Graph. Forum **27**(2), 663–674 (2008). (Eurographics 2008 Proceedings)
13. Forest, V., Barthe, L., Guennebaud, G., Paulin, M.: Soft textured shadow volume. Comput. Graph. Forum **28**(4), 1111–1121 (2009). (Eurographics Symposium on Rendering 2009)
14. Guennebaud, G., Barthe, L., Paulin, M.: Real-time soft shadow mapping by backprojection. In: Proc. of Eurographics Symposium on Rendering, pp. 227–234 (2006)
15. Guennebaud, G., Barthe, L., Paulin, M.: High quality adaptive soft shadow mapping. Comput. Graph. Forum **26**(3) (2007). (Proc. of Eurographics'07)
16. Hasenfratz, J.M., Lapierre, M., Holzschuch, N., Sillion, F.: A survey of real-time soft shadows algorithms. Comput. Graph. Forum **22**(4), 753–774 (2003)
17. Heidrich, W., Brabec, S., Seidel, H.P.: Soft shadow maps for linear lights. In: Proc. of the 11th Eurographics Workshop on Rendering, pp. 269–280 (2000)
18. Iwasaki, K., Dobashi, Y., Yoshimoto, F., Nishita, T.: Precomputed radiance transfer for dynamic scenes taking into account light interreflection. In: Proc. of the Eurographics Symposium on Rendering, pp. 35–44 (2007)
19. Laine, S., Aila, T., Assarsson, U., Lehtinen, J., Akenine-Möller, T.: Soft shadow volumes for ray tracing. ACM Trans. Graph. **24**(3), 1156–1165 (2005)
20. Pan, M., Wang, R., Liu, X., Peng, Q., Bao, H.: Precomputed radiance transfer field for rendering interreflections in dynamic scenes. Comput. Graph. Forum **26**(3), 485–493 (2007)
21. Reeves, W.T., Salesin, D.H., Cook, R.L.: Rendering antialiased shadows with depth maps. Comput. Graph. **21**(4), 283–291 (1987). (Proc. of SIGGRAPH'87)
22. Rong, G., Tan, T.S.: Jump Flooding: An Efficient and Effective Communication Pattern for Use on GPUs, pp. 185–192. Charles River Media (2006). Chap. 3
23. Schwarz, M., Stamminger, M.: Bitmask soft shadows. Comput. Graph. Forum **26**(3) (2007). (Proc. of Eurographics'07)
24. Schwarz, M., Stamminger, M.: Microquad soft shadow mapping revisited. In: Eurographics'06 Short Paper (2008)
25. Schwarz, M., Stamminger, M.: Quality scalability of soft shadow mapping. In: Proc. of Graphics Interface'08 (2008)
26. Sintorn, E., Eisemann, E., Assarsson, U.: Sample-based visibility for soft shadows using alias-free shadow maps. Comput. Graph. Forum **27**(4), 1285–1292 (2008). (Proc. of the Eurographics Symposium on Rendering 2008)
27. Sloan, P.P.J., Kautz, J., Snyder, J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In: Proc. of SIGGRAPH'02, pp. 527–536 (2002)
28. Stamminger, M., Drettakis, G.: Perspective shadow maps. ACM Trans. Graph. **21**(3), 557–562 (2002). (Proc. of SIGGRAPH 2002)
29. Williams, L.: Casting curved shadows on curved surfaces. Comput. Graph. **12**(3), 270–274 (1978). (Proc. of SIGGRAPH'78)
30. Wimmer, M., Scherzer, D., Purgathofer, W.: Light space perspective shadow maps. In: Rendering Techniques 2004 (Proc. Eurographics Symposium on Rendering), pp. 143–151 (2004)

31. Woo, A., Poulin, P., Fournier, A.: A survey of shadow algorithms. IEEE Comput. Graph. Appl. **10**(6), 13–32 (1990)
32. Yang, B., Feng, J., Guennebaud, G., Liu, X.: Packet-based hierarchal soft shadow mapping. Comput. Graph. Forum **28**(4), 1121–1130 (2009). (Proceedings of Eurographics Symposium on Rendering 2009)
33. Zhou, K., Hu, Y., Lin, S., Guo, B., Shum, H.Y.: Precomputed shadow fields for dynamic scenes. ACM Trans. Graph. **24**(3), 1196–1201 (2005)

Kien T. Nguyen is a Ph.D. student in the Department of Computer Science and Engineering at Korea University. He received his B.S. degree in Applied Mathematics from Hanoi University of Science in 1999 and M.S. degree in Computer Science from Le-Quy-Don Technical University in 2003. His research interests include global illumination, shadows and image based rendering.

Hanyoung Jang is a Ph.D. student at Korea University. He received both of M.S. and B.S. degrees in the College of Information and Communications at Korea University. Since 2005, he has been working in the fields of collision detection and accessibility analysis for robotics. Currently, his primary research interest lies in real-time rendering of complex scenes.

JungHyun Han is a professor in the Department of Computer Science and Engineering at Korea University, where he directs the Interactive 3D Media Laboratory and Game Research Center supported by Korea Ministry of Culture, Sports, and Tourism. Prior to joining Korea University, he worked at the School of Information and Communications Engineering of Sungkyunkwan University, in Korea, and at the Manufacturing Systems Integration Division of the US Department of Commerce National Institute of Standards and Technology (NIST). He received a B.S. degree in Computer Engineering at Seoul National University, an M.S. degree in Computer Science at the University of Cincinnati, and a Ph.D. degree in Computer Science at USC. His research interests include real-time simulation and animation for games.