Soft Shadows from Extended Light Sources with Penumbra Deep Shadow Maps

Jean-François St-Amour
LIGUM, Dép. I.R.O.
Université de Montréal

Eric Paquette
LESIA, Software & IT Engineering Dept.
École de technologie supérieure, Montréal, Canada

Pierre Poulin
LIGUM, Dép. I.R.O.
Université de Montréal

Abstract

This paper presents a new method of precomputing high-quality soft shadows that can be cast on a static scene as well as on dynamic objects added to that scene. The method efficiently merges the visibility computed from many shadow maps into a penumbra deep shadow map (PDSM) structure. The resulting structure effectively captures the changes of attenuation in each PDSM pixel, and therefore constitutes an accurate representation of light attenuation. By taking advantage of the visibility coherence, the method is able to store a compact representation of the visibility for every location within the field of view of the PDSM. Modern programmable graphics hardware technology is used by the method to cast real-time complex soft shadows.

Key words: Rendering, illumination, graphics hardware, visibility, shadow, compression.

1 Introduction

Shadows provide important cues to understand spatial relationships between lights and objects. Soft shadows are even more desirable as they add a great deal of realism to synthetic images. However, because computing soft shadows can be very computationally intensive, generating them has traditionally been an off-line (non-interactive) process.

Recent advances in computer graphics hardware make it possible to generate soft shadows in real time, at the expense of trading quality for speed. While these methods produce good results, they lack the very high quality of off-line methods.

Therefore, this paper proposes a method to generate soft shadows, based on the deep shadow map (DSM) structure [13], to bridge the gap between off-line and real-time methods. The major contributions of our method can be summarized as: (1) high-quality precomputed soft shadows, which can be rendered using modern graphics hardware, and thus can be integrated to any graphics engine ; (2) seamless integration of dynamic objects to the scene, *after* precomputation of shadows. Furthermore, unlike many other real-time soft shadow methods, our method gracefully handles small to large extended light sources, inner and outer penumbrae, and any object that can be rendered with the z-buffer algorithm.

With the PDSM, the attenuation function can be queried for any location in the field of view of the light, which allows the correct shadowing of dynamic objects after the high-quality shadow precomputation. Since the precomputed PDSM does not include shadow information for the dynamic objects, the shadow of these objects must be computed with a standard shadow map or one of the methods discussed in Section 2.1. This approach is justified since studies [19] have shown that lower quality shadows for moving objects are acceptable.

2 Previous Work

While shadow determination is closely related to visibility determination [7], a problem so ubiquitous in computer graphics, it has its particularities, which lead to the development of numerous algorithms and structures dedicated to resolving the shadowing problem. Most methods are covered in surveys and books [3, 10, 22] and this section concentrates on the methods most relevant to the PDSM technique.

2.1 Real-Time Methods

Algorithms appropriate for real-time computation of shadows can be roughly divided in two classes: (1) methods based on shadow maps [20] that create depth images to store the closest visible point as seen from the light source; and (2) methods based on shadow volumes [8] that create invisible 3D shadow polygons to answer the point-in-shadow-volume request.

Shadow map algorithms have always been very popular: they have early on been integrated in graphics hardware pipelines [16]; they have been transformed to better fit perspective views of a 3D scene [14, 18, 21]; they have been adaptively refined for efficiency and quality [9, 17]. Some extensions of the shadow map algorithm also apply to the determination of soft shadows.

Percentage closer filtering [15] projects the 3D point to be shadowed in a standard shadow map, but instead of determining the visibility against a single pixel, it tests with respect to a finite kernel of pixels, thus blurring the shadow borders. Although it was primarily applied to antialias shadows, it can be used to approximate soft shadows. The size of the created penumbra is incorrect since it does not take into account occluder-occludee relative distance. Creating large penumbrae also results in a large increase in computation time because of the increased size of the kernel.

Smoothies [6] extend silhouette edges into a set of joined rectangles (smoothies) parallel to the shadow map. A clever function using the projection of the 3D point on the smoothie, the distance to its smoothie edges, and the depth to the 3D smoothie provides a reasonable approximation of penumbra. In the work of Wyman and Hansen [23], a similar method replaces the smoothies by 3D slanted shadow rectangles and cones, rendered into a penumbra map. Both of these methods achieve real-time soft shadows, but the umbra region does not shrink as it should when the light source increases in size. They base the computation of the umbra region on the shadow map computation and add the penumbra region outside of the shadow map umbra, thus capturing only half of the effects of extended light sources.

Extensions of shadow volume algorithms, such as the penumbra wedges [2, 4], produce quality soft shadows in real time, but they suffer from high fill-rate requirements. As an object-space method, its cost also increases rapidly with scene complexity.

2.2 Multiple Shadow Maps

To compute the shadowing from an extended light source at a given 3D point, the fraction of the light reaching this point is needed. Brotman and Badler [5] keep many shadow maps generated from point light sources randomly positioned on the extended light source. Unfortunately, memory consumption quickly becomes prohibitively large as the number of shadow maps increases to improve the quality of the shadows. Yet, much of the information is inherently coherent. Agrawala *et al.* [1] group the shadow maps in a single *layered attenuation map*. They warp every shadow map to the layered attenuation map, which has two main disadvantages: (1) as the light source becomes larger, splatting is required to prevent holes from appearing; (2) it provides no information where there are no receiving surfaces in the original scene. To cast soft shadows on dynamic objects added to the scene the method of Agrawala *et al.* thus requires the recomputation of the layered attenuation map for every frame of an animation.

2.3 Deep Shadow Maps

With the DSM, Lokovic and Veach [13] introduce a visibility structure to capture partial occlusion due to the cumulative occlusion by tiny or semi-transparent objects. A function storing the increase in attenuation is constructed and stored for each pixel of a shadow map. Kim and Neumann [11] efficiently construct this structure using graphics hardware.

The standard DSM from a point light source contains for each pixel the attenuation as a function of depth. Each DSM pixel is constructed from many shadow map pixels with jittered positions (or simply sub-pixels). For example, a single DSM pixel can typically correspond to 256 shadow map sub-pixels.

3 Penumbra Deep Shadow Maps

This section presents the adaptation of the DSM structure to capture the attenuation from an extended light source, in an approach similar to that of Agrawala *et al.* [1]. The PDSM method stores the attenuation function for every 3D location in the PDSM field of view. New objects can then be added in the scene with high-quality soft shadows cast on them without recomputing the PDSM. The PDSM also has advantages compared to approaches such as lightmaps since it does not require any surface parameterization of the objects. As can be seen in Figure 8 (g), it can even be cast on objects that do not have any surface, such as fog.

3.1 Construction

The fraction of the light source visible is needed when rendering a 3D point. The PDSM encodes this information as the attenuation of light for every 3D location in its field of view. Each pixel in the PDSM corresponds to a 3D ray from the PDSM center of projection through the center of the pixel. A PDSM pixel captures the attenuation of light along this ray.

Algorithm 1: PDSM construction.

```
1 Generate $k$ random sample points on the light source.
  foreach sample point do
2     Compute a shadow map (SM).
      // Merge the SM information to the PDSM.
      foreach PDSM pixel do
3         Compute the associated 3D PDSM ray.
4         Project this PDSM ray in the SM.
          foreach SM pixel traversed by the ray do
5             if visibility changed then
6                 Insert an event into the PDSM.
```

The pseudo-code for the construction of the PDSM is presented in Algorithm 1. The attenuation factor is computed by first randomly distributing a set of *sample point* light sources over the extended light source (step 1). When distributing the k sample points on the light, stratified sampling is used to ensure lower variance; other distributions could be used, as well as different importance functions. A shadow map (a standard shadow depth map) is then computed from each of these sample points (step 2). Since there are k shadow maps, each shadow map contributes for $1/k$ of the attenuation function.

To compute the attenuation contributed by one shadow map, each 3D PDSM ray is projected into the shadow map (steps 3 and 4) to compute the sections of the ray that are in shadow (step 5). These sections correspond to *entry* and *exit* events which are then inserted with the events of the other shadow maps in the PDSM pixel of the current ray (step 6). Steps 5 and 6 are illustrated in Figures 1 and 2. Step 2 is computed using graphics hardware, while steps 3 to 6 are computed in software.

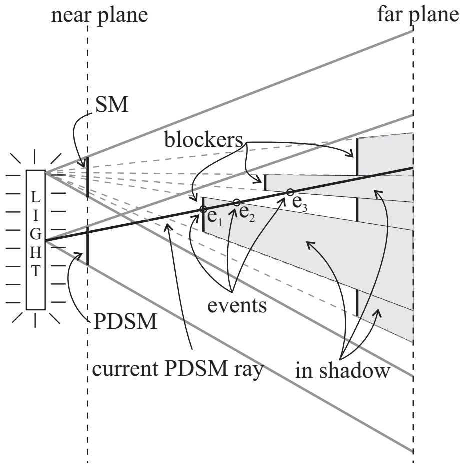


Figure 1: A PDSM ray corresponding to a single PDSM pixel. The PDSM ray is used to gather attenuation information as computed from the shadow map (SM).

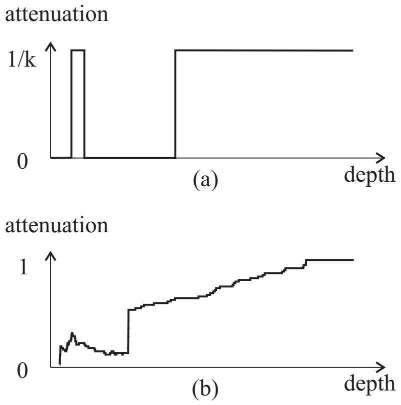


Figure 2: (a) Attenuation along a PDSM ray for one shadow map. (b) The final attenuation function along the PDSM ray as the sum of the contributions of all the shadow maps.

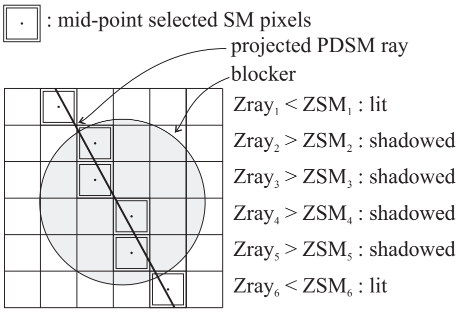

: mid-point selected SM pixels
projected PDSM ray
blocker
$Zray_1 < ZSM_1$: lit
$Zray_2 > ZSM_2$: shadowed
$Zray_3 > ZSM_3$: shadowed
$Zray_4 > ZSM_4$: shadowed
$Zray_5 > ZSM_5$: shadowed
$Zray_6 < ZSM_6$: lit

Figure 3: The 3D PDSM ray is projected in the shadow map (SM) and the pixels used to determine the visibility of the ray are identified by software scan-converting the ray using the mid-point algorithm.

Determination of the shadowed portions of a projected PDSM ray is obtained by software scan-converting the ray into the shadow map (Figure 3). The algorithm then keeps track of which parts of the PDSM ray fail the standard shadow map test. As mentioned previously, this information is kept in entry (attenuation $+1/k$) and exit (attenuation $-1/k$) events of each shadowed interval. Once a pixel is identified as containing a visibility event, the depth along the PDSM ray where this event occurred must be determined. Both the shadow map and the ray scan-conversion provide information about this. Figure 4 illustrates (a) when to use the shadow map depth information, (b) when to use the PDSM ray scan-conversion depth information, and (c) how to compute the depth of exit events. Care must be taken during this step, because while the shadow map is usually the most accurate source of depth information, it can also provide some erroneous data and as such must always be validated with the scan-conversion information. PDSM rays that project in a single pixel are handled as a special case and only add an entry event.

When adding the shadow map information in the PDSM, depths in the shadow map must be transformed to depths in the PDSM for insertion. This computation can be optimized if the light source is planar and if the PDSM and the shadow maps are parallel to each other and parallel to the light source plane [1]. In this context, a depth in the shadow map is exactly the same depth in the PDSM, thus requiring no transformation.

Because of the finite depth resolution of the shadow map, many events from different shadow maps can occur at the same depth or be very close to each other. Events

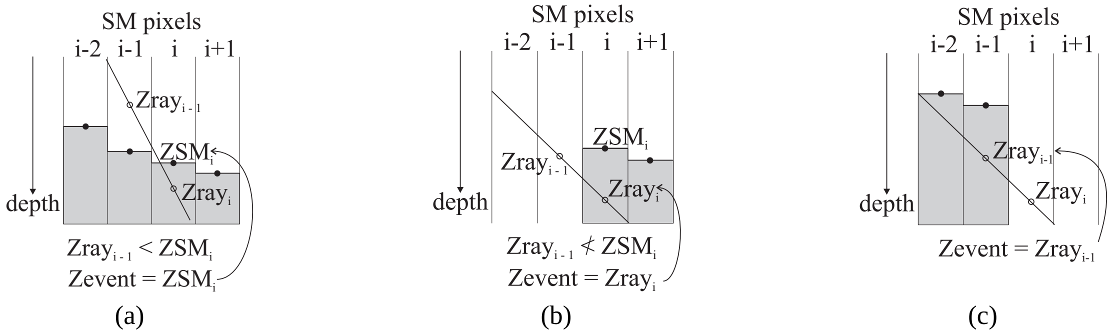


Figure 4: Shadow map (SM) pixels extend as vertical regions i-2, i-1, i, *etc. Pixel* i *is under examination. Points represent the depth information at the center of shadow map pixels (ZSM). The PDSM ray traverses the shadow maps along increasing depth. The selection between the PDSM ray depth (Zray) and the shadow map depth (ZSM) is based on the comparison between these two depths: (a) shadow map depth is appropriate, (b) shadow map depth is wrong thus Zray must be used. Finally, (c) illustrates that the* exit event *uses the depth of the previous pixel.*

occurring in a depth range close to the resolution of the shadow map are simply merged together during step 6 of the construction algorithm.

The constructed PDSM is composed of a series of events with depth and attenuation values for each PDSM pixel. One such attenuation function is illustrated in Figure 2 (b). It should be noted that if no dynamic objects are to be added to the scene, only a subset of all the events are needed. As Agrawala *et al.* [1], the PDSM method could keep only the attenuation values where there are surfaces, thus greatly reducing the number of events and required storage.

3.2 Compression

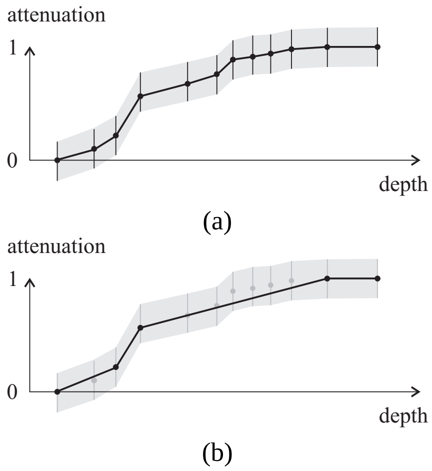


Figure 5: The depth events (a) of the PDSM are compressed in a piecewise linear approximation (b) that respects the error bounds.

Because the number of events in the attenuation functions can be very high, it needs to be compressed, especially if the PDSM is to be stored on graphics hardware. This is accomplished by using the compression technique described in the original DSM paper [13]. The basic idea is that at each step, the algorithm draws the longest possible segment, while staying within the confines of the pre-defined error bounds. Figure 5 illustrates the technique. As will be seen in the next section, compressing the number of events is important to reduce both memory consumption and rendering time.

4 Rendering

Rendering shadows with the PDSM is described in Algorithm 2. This algorithm can be used in software rendering engines, but can also be implemented on programmable graphics hardware (details of our hardware implementation follow in Section 4.1). The 3D point to be shaded is projected in the PDSM (step 2), the attenuation is computed from the function stored in the PDSM (steps 3 and 4), and the shading is multiplied by this attenuation factor (step 5).

Algorithm 2: Rendering.

```
1 foreach 3D point to shade do
      // $3D\ point(x,y,z)_{world} \rightarrow (x,y,z)_{PDSM}$
2     Project in the PDSM.
      // $(x,y)_{PDSM} \rightarrow f(\,)$
3     Retrieve the appropriate attenuation function.
      // $f((z)_{PDSM}) \rightarrow attenuation$
4     Retrieve the attenuation value.
      // $attenuation \rightarrow pixel\ color$
5     Modulate the shading by this attenuation.
```

The preprocessing phase computes depth images from the light source and constructs a PDSM that contains attenuation functions for every location in its field of view. The attenuation functions are stored in each PDSM pixel as a list of events (depth and attenuation values). As can be seen in Figure 5, an attenuation function is a piecewise linear function, interpolating the attenuation between adjacent events.

When rendering, shaded images are computed from the view of the camera. Computing the pixel colors of an image requires the shading of the 3D points corresponding to the pixels of the image (step 1). A 3D point to shade is projected in the PDSM image to determine the PDSM pixel that contains the relevant attenuation function (step 2 and 3). To evaluate the piecewise linear attenuation function, the algorithm searches for the two events between which the 3D point to shade is located. The list of events is sequentially visited and the attenuation values are linearly interpolated to obtain the attenuation value at the depth of the 3D point to shade (step 4). Compressed functions (Section 3.2) are used in the examples presented in this paper since it reduces the length of the lists of events and thus reduces both memory consumption and rendering time.

As in many other image-based approaches, instead of evaluating the attenuation value from a single PDSM pixel, the PDSM pixels around the projected 3D point can be considered and the attenuation values filtered. Filters of various widths can be applied by evaluating the attenuation functions and weighting the results. The examples presented in this paper use bilinear filtering since it provides a nice balance between quality and cost.

4.1 Hardware Rendering

To take advantage of the processing power of programmable graphics hardware, the whole PDSM rendering algorithm can be implemented in hardware using a fragment shader. Because the PDSM structure is usually very sparse, encoding it efficiently in graphics hardware texture memory is important. Encoding the PDSM in a 3D data structure potentially wastes huge amounts of memory, which is why we encode it into two 2D textures: an *index texture* and a *data texture*. The data texture contains all the PDSM functions encoded sequentially (Figure 6 (b) and (c)) and the index texture simply provides information on where a specific PDSM function is located in the data texture, and how many texels it occupies (Figure 6 (a)).

Every step of Algorithm 2 is mappable to graphics hardware. Using regular projective texturing for the index texture gives the address and length of the PDSM function to evaluate (steps 2 and 3). This evaluation is done at the fragment level, where the fragment program fetches the function data, finds the needed events, and interpolates the attenuation values (step 4). Because PDSM compression greatly reduces both average and maximum function lengths, this evaluation can be done very efficiently, allowing the rendering of high-quality soft shadows in real time. When dynamic branching is available at the fragment shader level, this method of encoding also allows for arbitrary length attenuation functions and early-exit when searching through the attenuation function.

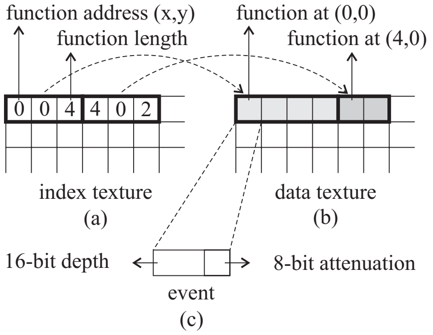


Figure 6: Texture encoding on the graphics hardware.

5 Results

The images in Figure 8 show common setups to illustrate soft shadows. The method generates smooth transitions from hard to soft shadows as can be seen in Figure 8 (a) and (b). The method also handles the smooth merging of soft shadows, as well as penumbra sizes according to the relative occluder-occludee distances (Figure 8 (c)). Various soft/hard shadows cast by a single complex object (Figure 8 (d)), or by a complex scene (Figure 8 (e) and (f)), are automatically treated, improving the realism of shadows. The volumetric shadow information of the PDSM allows to handle shadows within smoke and fog (Figure 8 (g)).

Figure 7 shows the linearity of various construction parameters. For k shadow maps of resolution $m \times m$ encoded in a PDSM of resolution $n \times n$, the complexity of the construction is $O(k\ m\ n^2)$. This means that the resolution of the PDSM should be kept relatively low, but that the accuracy of the shadow maps can be increased by a factor of four with an increase in computation time of only a factor of two. Finally, since the construction time is linear in the number of shadow maps, this gives a good trade-off between accuracy and precomputation time.

Figure 7 (d) shows how PDSM size can increase rapidly when sampling the light more finely and how the compression of the PDSM functions provides satisfying results with a fairly stable memory usage. Compression

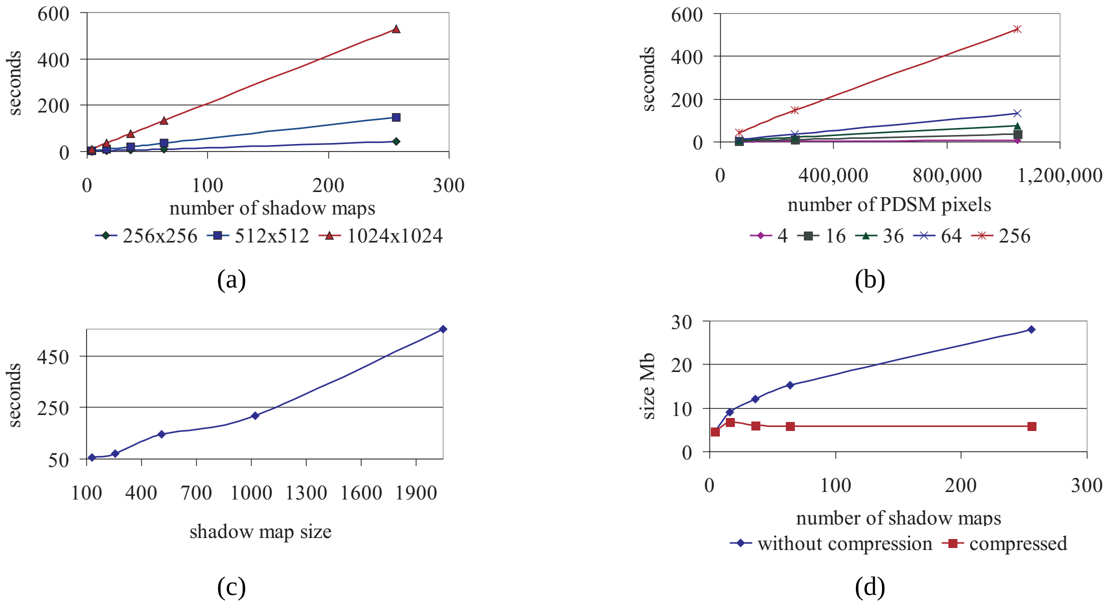


Figure 7: Various statistics computed for the Stanford Dragon scene. The construction time is directly proportional to (a) the number of shadow maps and to (b) the number of pixels of the PDSM. PDSM of (b) were constructed from 4, 16, 36, 64, and 256 shadow maps. It is also quite proportional to the size of the shadow maps (c) (resolutions of 128×128 *up to* 2048×2048*). Finally, (d) presents the compression of the PDSM attenuation functions for a* 512×512 *PDSM with different numbers of shadow maps.*

results in an important reduction of the number of events. As a result of the compression on the scene with the most complex soft shadows (the park scene), the average number of events per pixel goes down from 63 to 5 and the maximal number of events per pixel goes from 482 to 40.

The table below gives statistics about both PDSM construction and rendering for the scenes in Figure 8.

	Cylinder	Quads	Dragon	Park
Res. PDSM/SM	512^2	512^2	512^2	1024^2
Num. Samples	64	144	64	64
Construction (sec.)	30	67	33	161
Rendering (fps)	1150	840	80	40

All the results presented in this section were computed on an AMD Athlon 64 3500+ with 2GB of memory and a GeForce 6800 GT.

As previously mentioned, because the PDSM can be evaluated at any location in 3D space, shadows are cast not only on the static parts of the scene that were used to precompute the shadows, but also on any dynamic object that might have been added to the scene afterward. This is illustrated in the accompanying video [12].

When adding dynamic objects, information about how they should cast shadows is not present in the precomputed PDSM. While nothing prevents the recomputation of the PDSM with the new positions of these dynamic objects, or even to incrementally add the contributions of two PDSM (one static and one dynamic), the time required to compute the new PDSM is significant compared to rendering time (several seconds, depending on PDSM resolution and the number of shadow maps, see Figure 7). In our current implementation, we simply used shadow mapping with percentage closer filtering [15].

6 Conclusion

This paper presents an efficient construction algorithm of DSM for extended light sources represented by a distribution of sample point light sources. Each PDSM ray is software scan-converted in each shadow map to accumulate the distribution of total incoming light along the ray. The resulting PDSM structure captures quality (hard and soft) shadows for light sources ranging from point light to large extended sources. When compressing this attenuation function, modern graphics hardware can be exploited for the real-time display of static shadows. The PDSM structure is well adapted to real-time interactive applications where complex shadows can be cast over animated characters. A complementary real-time adapted shadow algorithm can be applied on these animated characters to ensure their shadows are properly cast within the scene.

There are many interesting directions to improve the PDSM method. The scan-conversion of PDSM rays into each shadow map should exploit more of the inherent coherence of this structure, such as scan-converting “slices” of the PDSM view pyramid rather than individual rays. The compression of the attenuation function is also performed on a single PDSM ray, where adjacency information and more perception-based compression would be preferable. Exploiting recent adaptive shadow map extensions [9, 17, 18] and developing a better sampling of the extended light sources according to the shadow regions should also improve the quality of the constructed PDSM.

Acknowledgements

Luc Leblanc integrated our technique in his modeling system, and he was instrumental in providing a general, stable, and flexible environment, as well as producing the images and animations. Philippe Beaudoin and Andrew Woo commented on an early draft. We are grateful for the continuous financial support from NSERC.

References

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

[2] T. Akenine-Möller and U. Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Eurographics Workshop on Rendering*, pages 297–306, June 2002.

[3] T. Akenine-Möller and E. Haines. *Real-time Rendering*. AK Peters, 2nd edition, 2002.

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

[5] L. Brotman and N. Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics & Applications*, 4(10):71–81, October 1984.

[6] E. Chan and F. Durand. Rendering fake soft shadows with smoothies. In *Eurographics Symposium on Rendering*, pages 208–218, June 2003.

[7] D. Cohen-Or, Y.L. Chrysanthou, C.T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Trans. on Visualization and Computer Graphics*, 9(3):412–431, jul-sep 2003.

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

[9] R. Fernando, S. Fernandez, K. Bala, and D.P. Greenberg. Adaptive shadow maps. In *SIGGRAPH 2001*, pages 387–390, August 2001.

[10] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of real-time soft shadows algorithms. In *Computer Graphics Forum*, volume 22, 2003.

[11] T.-Y. Kim and U. Neumann. Opacity shadow maps. In *Eurographics Workshop on Rendering*, pages 177–182, June 2001.

[12] LIGUM. www.iro.umontreal.ca/labs/infographie /papers/St-Amour-2005-PDSM, March 2005.

[13] T. Lokovic and E. Veach. Deep shadow maps. In *SIGGRAPH 2000*, pages 385–392, July 2000.

[14] T. Martin and T.-S. Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Eurographics Symposium on Rendering*, pages 153–160, 2004.

[15] W.T. Reeves, D.H. Salesin, and R.L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH 87*, volume 21, pages 283–291, July 1987.

[16] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P.E. Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH 92*, volume 26, pages 249–252, July 1992.

[17] P. Sen, M. Cammarano, and P. Hanrahan. Shadow silhouette maps. *ACM Trans. on Graphics*, 22(3):521–526, July 2003.

[18] M. Stamminger and G. Drettakis. Perspective shadow maps. *ACM Trans. on Graphics*, 21(3):557–562, July 2002.

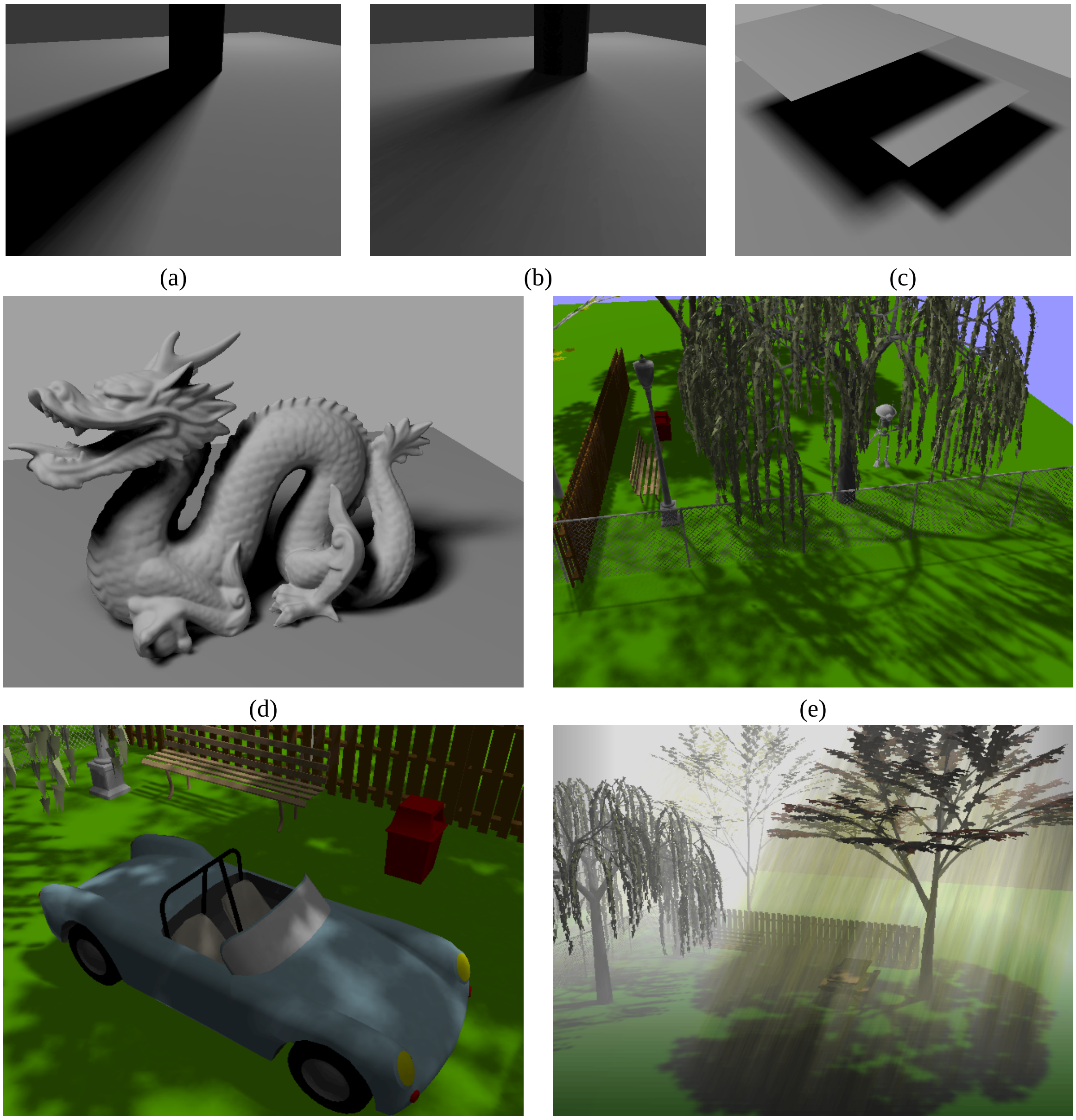
[19] L. Wanger. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *Symposium on Interactive 3D Graphics*, volume 25, pages 39–42, March 1992.

[20] L. Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH 78*, volume 12, pages 270–274, August 1978.

[21] M. Wimmer, D. Scherzer, and W. Purgathofer. Light space perspective shadow maps. In *Eurographics Symposium on Rendering*, 2004.

[22] A. Woo, P. Poulin, and A. Fournier. A survey of shadow algorithms. *IEEE Computer Graphics & Applications*, 10(6):13–32, November 1990.

[23] C. Wyman and C. Hansen. Penumbra maps: Approximate soft shadows in real-time. In *Eurographics Symposium on Rendering*, pages 202–207, June 2003.



(a) (b) (c)

(d) (e)

(f) (g)

Figure 8: Examples of soft shadows computed with the PDSM. From top to bottom, left to right: (a) A narrower penumbra behind a cylinder extends as (b) the light source is enlarged. (c) Different sizes of penumbrae according to the relative distances between blockers/receivers. (d) Complex self-shadowing effects on the Stanford Dragon. (e) Park scene with complex geometry and shadows. (f) Complex shadows cast in real time on a car inserted in the precomputed shadows; the shadows cast by the car are computed with percentage closer filtering [15]. (g) Fog lighted with the PDSM, showing how the attenuation can be evaluated everywhere in the scene. No special treatment is needed, the PDSM shader is simply added to the regular fog shader.