Perspective Shadow Maps

Marc Stamminger and George Drettakis

REVES - INRIA Sophia-Antipolis, France, `http://www-sop.inria.fr/reves/` [*]

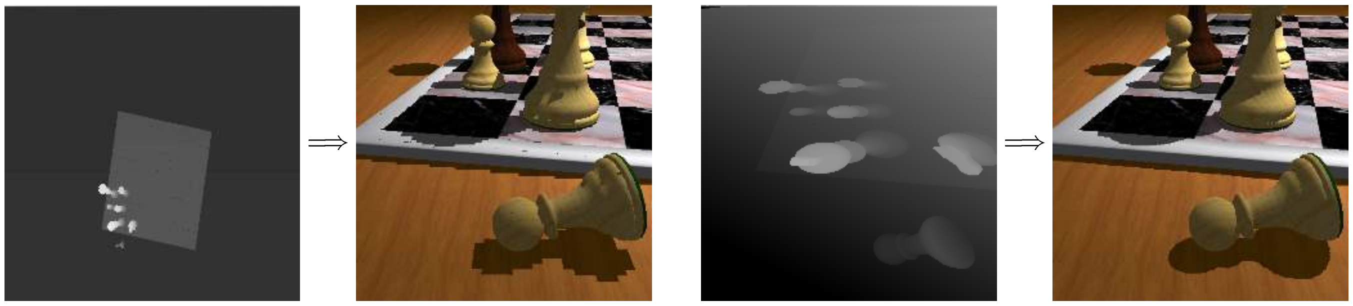


Figure 1: (Left) Uniform 512x512 shadow map and resulting image. (Right) The same with a perspective shadow map of the same size.

Abstract

Shadow maps are probably the most widely used means for the generation of shadows, despite their well known aliasing problems. In this paper we introduce *perspective shadow maps*, which are generated in normalized device coordinate space, i.e., after perspective transformation. This results in important reduction of shadow map aliasing with almost no overhead. We correctly treat light source transformations and show how to include all objects which cast shadows in the transformed space. Perspective shadow maps can directly replace standard shadow maps for interactive hardware accelerated rendering as well as in high-quality, offline renderers.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and framebuffer operations; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: Frame Buffer Algorithms, Graphics Hardware, Illumination, Level of Detail Algorithms, Rendering, Shadow Algorithms

1 Introduction

Shadows are an important element for any computer graphics image. They provide significant visual cues, aiding the user to understand spatial relationships in a scene and add realism to synthetic images. The challenge of generating shadows goes back to the early days of computer graphics, where various geometric algorithms such as those based on clipping [15], or shadow volumes [1] were introduced. These methods often suffer from robustness problems due to the geometric computations required, and may also require non-trivial data structures or involve a significant rendering overhead. In addition, such algorithms are usually a preprocess, and are thus best suited to static scenes.

The introduction of *shadow maps* [16] marked an important step in the evolution of shadow algorithms. With shadow maps, we simply render the scene from the viewpoint of the light source, creating a depth image. When rendering each pixel in the final image, the visible point is transformed into the light coordinate system, and a depth comparison is performed with the corresponding shadow map pixel to decide whether the point is hidden with respect to the light source. A single shadow map is sufficient for directional lights and spot lights; omnidirectional point lights require several shadow maps to cover all emission directions. When considering point lights in this paper, we implicitly mean spot lights, but the ideas developed here transfer to omnidirectional point lights easily.

Compared to shadow algorithms which require geometric calculations prone to robustness problems, shadow maps are much more general. Due to their simplicity, they have been incorporated into graphics hardware, first on the Infinite Reality [6] and more recently on consumer-level graphics cards such as the NVIDIA GeForce3. The possibility to have shadows for large scenes makes shadow maps particularly interesting for rendering-intensive applications like video games. Shadow maps are also widely used in offline, high-quality rendering systems because of their simplicity and flexibility. In the RenderMan standard, for example, shadow maps are the most widely used method to generate images with shadows [13].

However, as with any method based on a discrete buffer, shadow maps suffer from aliasing problems, if the shadow map resolution used is insufficient for the image being rendered. This is particularly true for scenes with a wide depth range, where nearby shadows need high resolution, whereas for distant shadows low resolution would be sufficient. Recent shadow map approaches [12, 2] adapt shadow map resolution by using multiple shadow maps of varying resolution. However, by using multiple shadow maps, several rendering passes are required, and due to the more involved data structures, the methods no longer map to hardware.

In this paper, we present a novel approach to adapt shadow map resolution to the current view. By using a non-uniform parameterization, a single *perspective shadow map* is generated, providing high resolution for near objects, and decreased resolution as we

[*] `{Marc.Stamminger|George.Drettakis}@sophia.inria.fr` The first author is now at the Bauhaus-Universität, Weimar, Germany.

move away from the viewpoint. An example is shown in Fig. 1. A standard shadow map (left) has insufficient resolution for nearby objects (center left). In contrast, in our perspective shadow map (center right), resolution varies such that nearby objects have sufficient shadow map resolution (right).

Our shadow map projection can still be represented by a 4×4 matrix, so that it fully maps to hardware. It can easily be used to improve any interactive and offline rendering method based on shadow maps, with only marginal overhead for the computation of the optimized projection matrix. Perspective shadow maps often reduce aliasing dramatically. In a few difficult cases, our parameterization converges to standard uniform shadow maps. Our approach is view-dependent, requiring the generation of a new shadow map at each frame or at least after major camera movements. This is necessary for scenes containing moving objects anyhow, such as those used in video games or other interactive applications. In this case, we have no significant speed penalty compared to a standard, fixed-resolution shadow map.

1.1 Shadow Map Aliasing

First, we briefly formalize the aliasing problems of shadow maps for point light sources. Every pixel in the shadow map represents a sheared pyramid of rays, passing through a shadow map pixel of size d_s x d_s on the (shadow) image plane (Fig. 2). In the following, we will call the reciprocal of d_s the *shadow map resolution*. To increase shadow map resolution we decrease d_s and vice versa. When the ray bundle through a shadow map pixel hits a surface at distance r_s under angle α, the size of the intersection is approximately $d_s r_s / \cos\alpha$. In the final image, this intersection with the surface has size:

$$d = d_s \frac{r_s}{r_i} \frac{\cos\beta}{\cos\alpha},$$

where β is the angle between the light source direction and the surface normal and r_i is the distance from the camera to the intersection.

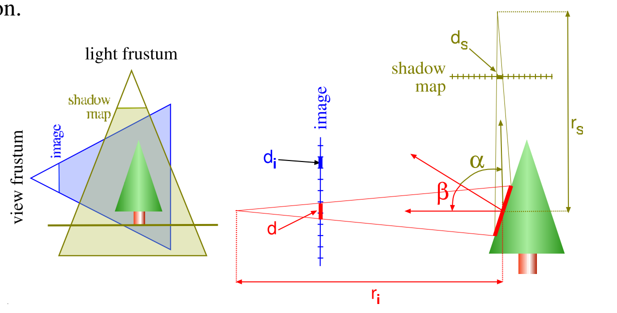


Figure 2: Shadow map projection quantities.

For directional light sources, d is independent of r_s, so d only depends on d_s/r_i. Analogously, for an orthogonal view d is proportional to $d_s r_s$.

Shadow map undersampling appears when d is larger than the image pixel size d_i. This can happen when $d_s r_s / r_i$ becomes large, which typically happens when the user zooms into a shadow boundary so that single shadow map pixels become visible. We will call this *perspective aliasing*. Due to limited memory, the shadow map resolution $1/d_s$ can only be increased up to a certain limit in practice. Perspective aliasing can be avoided by keeping the fraction r_s/r_i close to a constant. As we show in this paper, this is what is achieved when we compute the shadow map after perspective transformation.

When $\cos\beta / \cos\alpha$ becomes large, *projection aliasing* appears. This typically happens when the light rays are almost parallel to a surface, so that the shadow stretches along the surface. Projection aliasing is not treated by perspective shadow maps, since this would require a local increase in shadow map resolution. This is not possible without much more involved data structures, and it would force us to abandon hardware acceleration.

Note that there are also other sources of aliasing when using shadow maps, e.g., depth quantization, which we will not treat here.

1.2 Related Work

The literature on shadow generation in computer graphics is vast, and far beyond the scope of this paper (see [17] for an old, but nice survey). We thus concentrate our review on previous work directly related to shadows maps and our novel approach.

Reeves et al. [9] introduced the concept of percentage closer filtering. The idea is to filter the shadows obtained from shadow maps by interpolating the binary result of the depth comparison instead of the depth values. The deep shadow maps of Lokovic et al. [5] extend the idea of filtering shadow maps by storing approximate attenuation functions for each shadow map pixel, thus capturing self shadowing within hair, fur or smoke. Both approaches perform anti-aliasing in that they filter undersampling artifacts, but they do not solve the problem of undersampling as such.

Tadamura et al. [12] introduce multiple shadow maps with varying resolution as a solution to the aliasing problem for large outdoor scenes. This solves the problem of perspective aliasing, but it does not map to hardware and is thus not appropriate for interactive applications.

An interesting improvement to traditional shadow maps is the adaptive shadow map method [2]. This approach replaces the “flat” shadow map with an adaptive, hierarchical representation, which is updated continuously during a walkthrough. For each frame, a first rendering pass is required to determine which parts of the hierarchical shadow map need refinement. This is achieved using OpenGL texture mip-mapping capabilities. Perspective and projection aliasing are detected. The most critical parts of the shadow map are then rendered at higher resolution, read back, and inserted into the hierarchical shadow map structure. Oversampled parts of the hierarchy are truncated. The method is a software solution, since it requires the traversal and refinement of a hierarchical data structure instead of a shadow map, and thus cannot entirely map to hardware. Rapid view changes make it necessary to update a large number of nodes in the hierarchy, so either the frame rate drops or aliasing appears until the hierarchical structure has been updated. The generation of a new node essentially requires an entire rendering pass, albeit with a small part of the model, but with the full cost of a frame-buffer readback to update the data structure. Scenes with moving objects would require the generation of the entire hierarchical structure at each frame, which would require many rendering passes.

Shadow maps are closely related to textures, so we assume the reader is familiar with the texturing and the projective mapping process. Most standard graphics textbooks treat these issues [3]. A particularly nice and thorough description can be found in [4].

2 Overview

Perspective shadow maps are computed in post-perspective space of the current camera, or, using standard graphics terminology, the space of normalized device coordinates [3]. In the traditional rendering pipeline, perspective is obtained by transforming the world to a perspectively distorted space, where objects close to the camera are enlarged and distant objects are shrunk (see Fig. 3). This mapping is projective [4] and can thus be represented by a 4×4 matrix with homogeneous coordinates. It projects the current viewing frustum to the unit cube; the final image is generated by a parallel projection of this cube along z.

The idea of perspective shadow maps is to first map the scene to post-perspective space and then generate a standard shadow map in this space by rendering a view from the transformed light source to the unit cube. We can work in post-perspective space almost as in world space, with the exception of objects behind the viewer, which will be handled in Sect. 4. Because the shadow map "sees" the scene *after* perspective projection, perspective aliasing (see Sect. 1.1) is significantly decreased, if not completely avoided.

To explain this principle a special case is depicted in Fig. 3. The scene is illuminated by a vertical directional light source. In post-perspective space the shadow map is an orthogonal view from the top onto the unit cube and the final image is an orthogonal view from the front. Consequently, all shadow map pixels projected onto the ground in this scene have the same size in the image. Thus, in the sense of Sect. 1.1, perspective aliasing is completely avoided for this case.

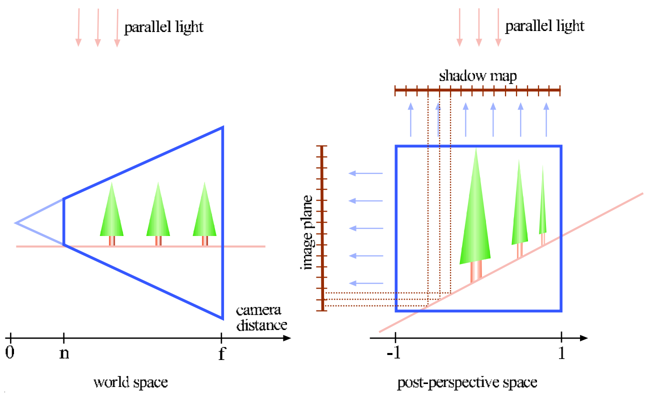


Figure 3: The scene is illuminated by a vertical directional light source (left). By applying the eye-view perspective transformation and then generating the shadow map (right), the shadow map pixels projected on the ground are evenly distributed in the image.

3 Post-Perspective Light Sources

To apply our new method, we first transform the scene to post-perspective space, using the camera matrix. We then transform the light source by the same matrix, and generate the shadow map. The different cases which arise are described next.

3.1 Directional Light Sources

Directional light sources can be considered as point lights at infinity. The perspective mapping can move these sources to a finite position; the set of possible cases is shown in Fig. 3 and 4. A directional light source parallel to the image plane remains at infinity. All other directional light sources become point lights on the *infinity* plane at $z = (f+n)/(f-n)$, where n and f are the near and far distance of the viewing frustum (see Fig. 3).

A directional light source shining from the front becomes a point light in post-perspective space (Fig. 4, left), whereas a directional light source from behind is mapped to a "inverted" point light source (center). In this context, "inverted" means that all light rays for these light sources do not emanate, but converge to a single point. Such inverted point light sources can easily be handled as point lights with their shadow map depth test reversed, such that hit points *furthest* from the point source survive. The extreme cases are directional lights parallel (facing or opposite to) the view direction, which are mapped to a point light just in front of the viewer (right).

3.2 Point Light Sources

The different settings for point light sources are shown in Fig. 5. Point light sources in front of the viewer remain point lights, (left).

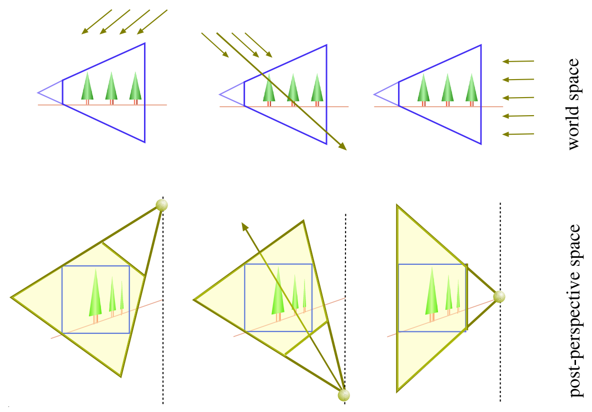


Figure 4: Directional lights in world space (top row) become point lights in post perspective space (bottom row) on the infinity plane. Lights from behind become inverted, i.e., the order of hits along a ray is inverted.

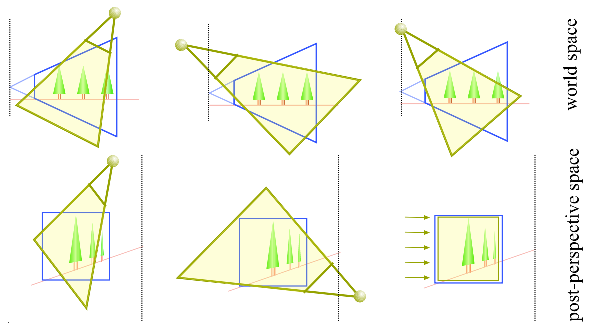


Figure 5: Mapping of point lights in world space (top row) to post-perspective space (bottom row). A point light in front of the user remains a normal point light (left), a point light behind the viewer becomes inverted (center). Boundary cases are mapped to directional lights (right).

Point lights behind the viewer are mapped beyond the infinity plane and become inverted (center). Point lights on the plane through the view point which is perpendicular to the view direction (camera plane), however, become directional (right).

3.3 Discussion

In post-perspective space, the final image is an orthogonal view onto the unit cube. Following our observations in Sect. 1.1, this means that perspective aliasing due to distance to the eye, r_i, is avoided. However, if the light source is mapped to a point light in post-perspective space, aliasing due to the distance to the shadow map image plane, r_s, can appear.

The ideal case occurs when, after the perspective mapping, the light source is directional. Perspective aliasing due to r_s is thus also avoided. This happens for:

- a directional light parallel to the image plane (see Fig. 3),
- a point light in the camera plane. The typical example is the miner's head-lamp, that is a point light just above the camera. It is known that this setting is also ideal for standard shadow maps [7]. With our parameterization, the offset can be arbitrarily large, as long as the point remains in the camera plane.

On the other hand, the less optimal cases appear when we obtain a point light with a large depth range in post-perspective space. The extreme example is a directional light parallel to the viewing direction. In post-perspective space, this becomes a point light on the infinity plane on the opposite side of the viewer (see Fig. 4 right). In this worst case the two perspective projections with opposite viewing directions cancel out mutually and we obtain a standard uniform shadow map.

In general, for directional light sources the benefit is maximal for a light direction perpendicular to the view direction. Since these are also parallel lights in post-perspective space, perspective aliasing is completely avoided in this case. Consider the smaller of the two angles formed between the light direction and the viewing direction. The benefit of our approach decreases as this angle becomes smaller, since we move further away from the ideal, perpendicular case. If the angle is zero our parameterization corresponds to a uniform shadow map.

For point lights, the analysis is harder, since the point light source also applies a perspective projection. Informally, the advantage of our parameterization is largest when the point light is far away from the viewing frustum, so that it is similar to a parallel light, and if the visible illuminated region exhibits a large depth range. For the case of a miner's lamp, which is known to be ideal for uniform shadow maps [7], our parameterization again converges to the uniform setting.

A common problem in shadow maps is the bias necessary to avoid self-occlusion or surface acne [7]. This problem is increased for perspective shadow maps, because objects are scaled non-uniformly. We use a constant offset in depth in shadow map space, which may require user adjustment depending on the scene.

4 Including all Objects Casting Shadows

Up to now, we ignored an important issue: our shadow map is optimized for the current viewing frustum. But the shadow map must contain all objects within that frustum *plus* all potential occluders outside the frustum that can cast a shadow onto any visible object.

4.1 World Space

More formally, we define the set of points that must appear in the shadow map as follows: Let $\mathcal{S}$ be an envelope of the scene objects, typically its bounding box. $\mathcal{V}$ is the viewing frustum and $\mathcal{L}$ the light source frustum. The light source is at position l (for directional lights $\mathcal{L} = R^3$ and l is at infinity). First, we compute the convex hull $\mathcal{M}$ of $\mathcal{V}$ and l, so $\mathcal{M}$ contains all rays from points in $\mathcal{V}$ to l. From $\mathcal{M}$ we then remove all points outside the scene's bounding box and the light frustum: $\mathcal{H} = \mathcal{M} \cap \mathcal{S} \cap \mathcal{L}$ (shown in yellow in Fig. 6 (right)).

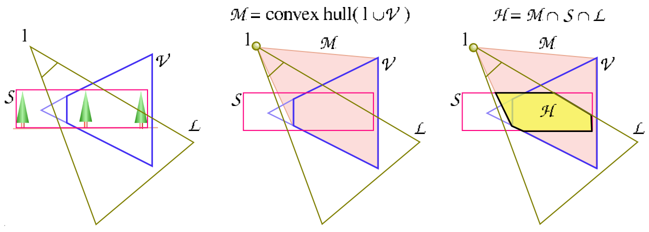

$\mathcal{M}$ = convex hull(l ∪ $\mathcal{V}$)
$\mathcal{H} = \mathcal{M} \cap \mathcal{S} \cap \mathcal{L}$
l
$\mathcal{V}$
$\mathcal{S}$
$\mathcal{L}$
$\mathcal{M}$
$\mathcal{H}$

Figure 6: The current view frustum $\mathcal{V}$, the light frustum $\mathcal{L}$ with light position l, and the scene bounding box $\mathcal{S}$ (left). $\mathcal{M}$ is obtained by extending $\mathcal{V}$ towards the light (center). Then $\mathcal{M}$ is intersected with $\mathcal{S}$ and $\mathcal{L}$ to obtain the set of points that need to be visible in the shadow map (right).

In our implementation we use the computational geometry library CGAL (`www.cgal.org`) to perform these geometric computations. As a result, the window of the shadow map is chosen so that all points in $\mathcal{H}$ are included. However, the shadow map will not see $\mathcal{H}$, but its transformation to post-perspective space, so we have to consider how $\mathcal{H}$ changes by the perspective mapping.

4.2 Post-perspective Space

Under projective transformations lines remain lines, but the order of points along a line can change. For perspective projections, this happens when a line intersects the camera plane, where the intersection point is mapped to infinity. As a result, we can quickly transform the convex set $\mathcal{H}$ to post-perspective space by transformation of its vertices, as long as $\mathcal{H}$ is completely in front of the viewer. An example where this is not the case is shown in Fig. 7. Points behind the viewer had to be included, because they can cast shadows into the viewing frustum. However, in post-perspective space, these points are projected to beyond the infinity plane. In this case, possible occluders can be on both sides of the infinity plane.

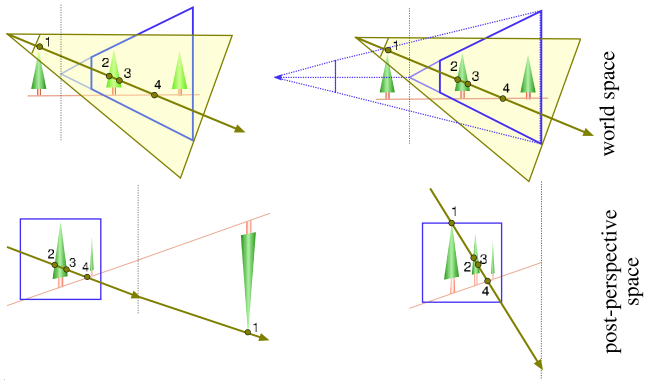


Figure 7: The scene is lit by a directional light source coming from behind (top left). After perspective projection, an object behind the viewer is inverted and appears on the other side of the infinity plane (lower left). To treat this we move the center of projection back (upper right), so that we are behind the furthermost point which can cast a shadow into the view frustum. After this, a standard shadow map in post-perspective space is again sufficient (lower right).

One solution would be to generate two shadow maps in this case. A first one as described previously, and a second one that looks "beyond" the infinity plane. We avoid this awkward solution by a virtual camera shift. We virtually move the camera view point backwards, such that $\mathcal{H}$ lies entirely inside the transformed camera frustum; the near plane distance remains unchanged and the far plane distance is increased to contain the entire previous frustum. Note that this camera point displacement is only for the shadow map generation, not for rendering the image.

By this, we modify the post-perspective space, resulting in decreased perspective foreshortening. If we move the camera to infinity, we obtain an orthogonal view with a post-perspective space that is equivalent to the original world space; the resulting perspective shadow map then corresponds to a standard uniform shadow map.

Consequently, we can say that by moving back the camera, our parameterization degrades towards that of a standard shadow map. By this, perspective aliasing is reintroduced, however, in practice this is only relevant for extreme cases, so we found it preferable to the double shadow map solution. It is interesting to note that for the cases that are ideal for perspective shadow maps, no such shift is necessary, because $\mathcal{H}$ will always be completely in front of the camera. The shift distance will be significant only if the light source is far behind the viewer. But in this case, our parameterization converges to the uniform shadow map anyway.

5 Point Rendering

Point rendering (e.g., [8, 10, 11, 14]) has proven to be a very effective means for rendering complex geometry. Points are particularly well suited for natural objects. However, for the generation of uniform shadow maps point rendering loses most of its benefits. A very large, dense point set needs to be generated in order to render a high-resolution uniform shadow map. Frequent regeneration of the shadow map for dynamic scenes becomes highly inefficient.

On the other hand, our projective shadow map parameterization fits nicely with the point rendering approaches presented in [14, 11], in the sense that the point sets generated by these methods can also be used for the rendering of the shadow map. The reason is that these approaches generate random point clouds, with point densities adapted to the distance to the viewer. In post-perspective space, these point clouds have uniform point density. Since perspective shadow maps are rendered in this space, the shadow map can assume that point densities are uniform. It is thus easy to select the splat size when rendering the shadow map such that holes are avoided.

6 Implementation and Results

We implemented perspective shadow maps within an interactive rendering application using OpenGL hardware assisted shadow maps. The following results, and all sequences on the accompanying video, were obtained under Linux on a Compaq AP550 with two 1 GHz Pentium III processors and an NVIDIA GeForce3 graphics accelerator. We used the shadow map extensions `GL_SGIX_depth_texture` and `GL_SGIX_shadow`. The shadow maps are rendered into p-buffers (`GLX_SGIX_pbuffer`) and are applied using register combiners (`GL_NV_register_combiners`).

In order to obtain optimal results from perspective shadow maps, we require that the near plane of the view be as far as possible. We achieve this by reading back the depth buffer after each frame, searching the minimal depth value, and adapting the near plane accordingly. Due to the fast bus and RAM of our computer, the speed penalty for reading back the depth buffer is moderate, e.g., 10ms for reading back depth with 16 bits precision at 640 by 480 pixel resolution. The increase in quality is well worth this overhead.

Fig. 1 shows a chess board scene, lit by a spot light source. The left image shows a uniform shadow map, the center left image is the resulting rendering with well visible shadow map aliasing. The far right image was obtained with a perspective shadow map of the same size, which is shown on the center right.

Fig. 8 shows Notre Dame in Paris, augmented with a crowd and trees. With point based rendering for the trees and the crowd, the views shown are still obtained at about 15 frames/sec.

In the street scene in Fig. 9 the cars and two airplanes are moving. The figure shows three snapshots from an interactive session; all images were obtained at more than ten frames per second with perspective shadow maps of size 1024×1024.

Our final test scene is a complete ecosystem, consisting of hundreds of different plants and a few cows. The view shown in Fig. 10 contains more than 20 million triangles. We show the image without shadows (left), with shadows from a 1024×1024 uniform shadow map (center) and with shadows from a projective shadow map of the same size (right).

7 Conclusions

We have introduced *perspective shadow maps*, a novel parameterization for shadow maps. Our approach permits the generation of shadow maps with greatly improved quality, compared to standard uniform shadow maps. By choosing an appropriate projective mapping, shadow map resolution is concentrated where appropriate. We have also shown how our method can be used for point rendering, thus allowing interactive display of very complex scenes with high-quality shadows. Perspective shadow maps can be used in interactive applications and fully exploit shadow map capabilities of recent graphics hardware, but they are also applicable to high-quality software renderers.

8 Acknowledgments

The first author has been supported by a Marie-Curie postdoctoral Fellowship while doing this work. L. Kettner implemented the frustum intersection code in CGAL (`www.cgal.org`). Thanks to F. Durand for his very helpful comments. Most of the models in the result section were downloaded from `www.3dcafe.com` and `www.help3d.com`.

References

[1] F. C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (Proc. of SIGGRAPH 77)*, 11(2):242–248, 1977.

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

[3] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. Computer graphics, principles and practice, second edition. 1990.

[4] P. Heckbert. Survey of Texture Mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[5] T. Lokovic and E. Veach. Deep shadow maps. *Proc. of SIGGRAPH 2000*, pages 385–392, 2000.

[6] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal. Infinite-reality: A real-time graphics system. *Proc. of SIGGRAPH 97*, pages 293–302, 1997.

[7] nvidia. webpage. `http://developer.nvidia.com/view.asp?IO=cedec_shadowmap`.

[8] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. *Proc. of SIGGRAPH 2000*, pages 335–342, 2000.

[9] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (Proc. of SIGGRAPH 87)*, 21(4):283–291, 1987.

[10] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. *Proc. of SIGGRAPH 2000*, pages 343–352, 2000.

[11] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In S. Gortler and K. Myszkowski, editors, *Rendering Techniques 2001 (12th Eurographics Workshop on Rendering)*, pages 151–162. Springer Verlag, 2001.

[12] K. Tadamura, X. Qin, G. Jiao, and E. Nakamae. Rendering optimal solar shadows with plural sunlight depth buffers. *The Visual Computer*, 17(2):76–90, 2001.

[13] S. Upstill. *The RenderMan Companion*. Addison-Wesley, 1990.

[14] M. Wand, M. Fischer, I. Peter, F. Meyer auf der Heide, and W. Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. *Proc. of SIGGRAPH 2001*, pages 361–370, 2001.

[15] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *Computer Graphics (Proc. of SIGGRAPH 77)*, 11(2):214–222, 1977.

[16] L. Williams. Casting curved shadows on curved surfaces. *Computer Graphics (Proc. of SIGGRAPH 78)*, 12(3):270–274, 1978.

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

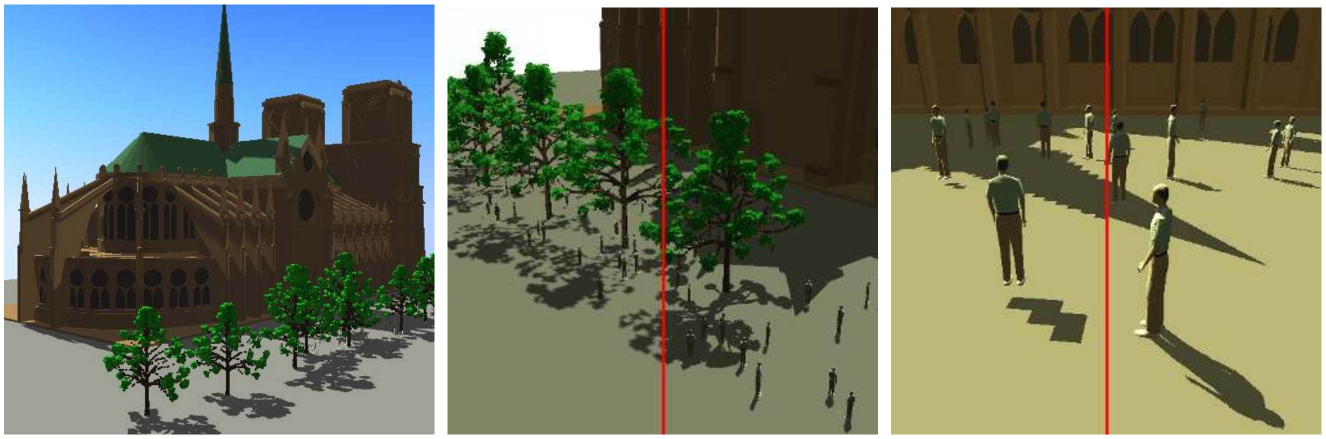


Figure 8: Notre Dame rendered with shadow maps of size 1024×1024. The left image and the right halves of the center and right image were rendered with perspective shadow maps at about 15 frames per second. The left sides of the center and right image have been generated with uniform shadow maps of the same size. The rendering of the trees and the crowd is accelerated by point rendering.

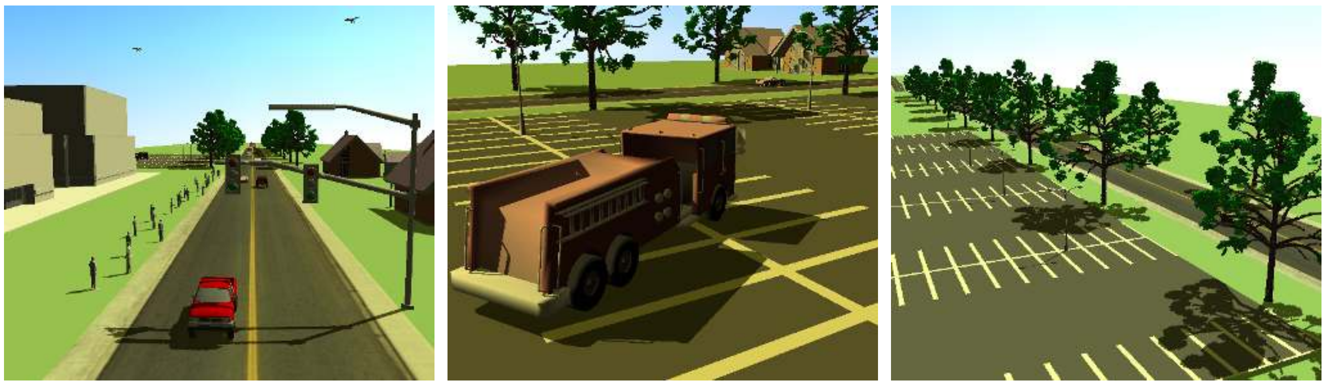
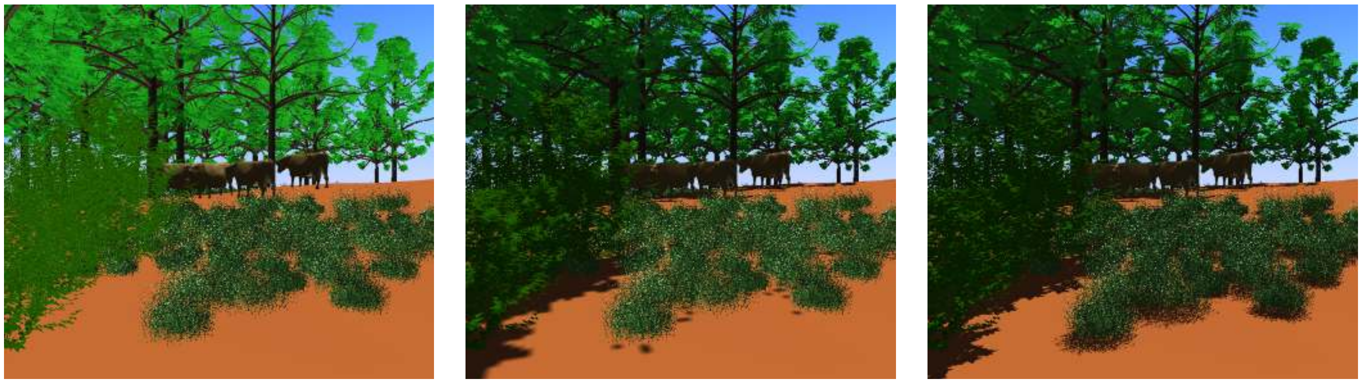


Figure 9: A street scene with moving cars and planes. The images were rendered at more than 10 frames per second with perspective shadow maps of size 1024×1024. Note the different shadow details at varying distances.



no shadows uniform shadow map perspective shadow map

Figure 10: This view into an eco system contains more than 20 million triangles, rendered by a mix of points and polygons in one second (left). By adding shadows from a uniform shadow map the quality improves significantly (center), but only the perspective shadow map captures fine nearby shadows (right). The total rendering time for the right image is 1.2 seconds.