INSTANCED SHADOW MAPS

Sebastian Buntin and Marc Stamminger

Computer Graphics Group
University of Erlangen-Nuremberg
Am Weichselgarten 9
91058 Erlangen Tennenloge, Germany
{buntin,stamminger}@cs.fau.de

1. KEYWORDS

instancing, shadow mapping, shadow techniques

2. ABSTRACT

We present a shadow generation approach which allows shadow maps to be generated object-wise and to be instanced over multiple instances of an object. The algorithm is designed for parallel light conditions, thus works perfect for outdoor scenes. We generate separate shadow maps for the different scene objects and generate their shadows in an efficient postprocess that can be combined with other effects such as deferred shading.

By this, several problems of standard shadow maps can be solved. The shadow map resolution can be adapted per object and thus solve shadow map resolution problems. Further, the geometry render overhead for generating the shadow map can be reduced, because only the shadow maps for dynamic objects (and not for the whole scene) need to be updated over time. The shadow computation in a postprocessing step mitigates performance and resource problems of previous per-object shadow map algorithms.

3. INTRODUCTION

Probably shadows are the simplest, yet also most important global illumination effect. They give very important visual cues about the relative positions of scene objects, and are mandatory for realistic rendering. Due to their global nature, shadow algorithms require global knowledge of the scene, which makes them relatively expensive.

For interactive rendering, two main streams of shadow generation exist: shadow maps and shadow volumes. Shadow maps store as global representation of the scene the depth buffer of the scene's rasterization from the view point of the light source. To determine whether a particular scene point is in shadow, a simple reprojection and depth comparison is needed. With current rendering hardware, the generation of shadow maps is an additional rendering pass, the shadow map lookup is done as a cheap projective texturing operation. The fact that the shadows are determined based on a rasterized view of the scene results in aliasing problems, that can result in arbitrarily bad shadow map undersampling artifacts. A number of approaches tackle these using shadow map hierarchies [FFBG01] or projective parameterizations, e.g. [SD02].

The shadow volume of an object is the geometric description of the object's shadow region [Cro77]. It can be quickly computed for closed polyhedra with consistent orientation, however this is usually done on the CPU. The shadow test for a particular scene point then becomes an or-ed point-in-polyhedron-test for all shadow volumes, which can be done elegantly by rasterizing the shadow volumes and stencil buffer counting. Shadow volumes do not suffer from aliasing, but are generally slower than shadow maps. In particular fill rate becomes a major bottle neck for complex scenes.

In classical shadow maps, the shadow map must contain the entire scene or at least the currently visible part of it. For large scenes, very high resolution is required to obtain reasonable shadow quality. Single moving objects require regeneration of the entire shadow map. To account for this, multiple shadow maps can be used [Reg04]. The scene is split into several objects or groups, each of which gets a shadow map on its own. By this, each object can get an appropriate shadow map resolution. If some objects in the scene are dynamic, only their shadow maps need to be updated. During rendering, all these shadow maps are bound and queried, which results in significant performance problems if the number of per-object shadow maps becomes large. In this paper we propose to apply these shadow maps one by one in a postprocessing pass. First, the scene is rendered without shadows. Then, for each shadow map, we find candidate shadow pixels by rasterizing a shadow volume for the shadow caster's bounding box. Only for image pixels inside this volume the shadow map lookup is done, and pixels in shadow are marked or dimmed. The process is depicted in Figs. 1 and 2. By this, the number of shadow map queries is significantly reduced, and we avoid restrictions in the number of applicable shadow maps.

Our approach is particularly tailored for large outdoor scenes with a large number of objects (typically vegetation) generated by instancing. In this case, shadow maps can be

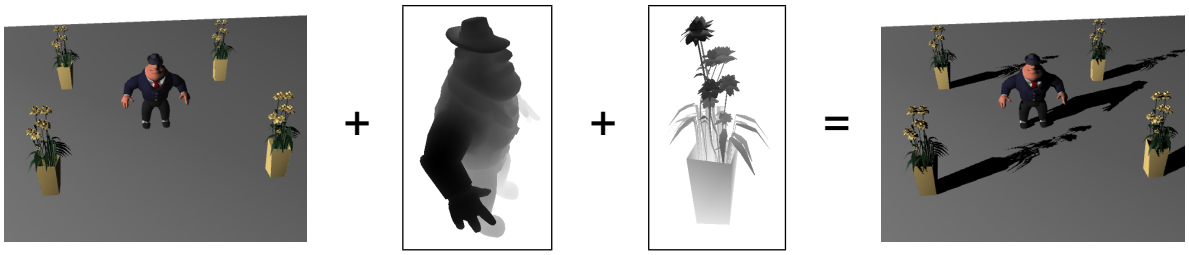

Figure 1: *We use multiple shadow maps for the different objects in the scene. For instanced objects, shadow maps are reused.*

reused for all instances. We present a simple approach to instance shadow maps also for rotated instances under parallel lighting conditions.

In this paper we present an approach which is tailored for large (e.g. outdoor) scenes with vegetation generated by instancing. For such scenes, shadow maps usually suffer from aliasing problems due to the size of the scene, whereas shadow volumes are not practical for vegetation. Our approach is based on shadow maps, but also exploits ideas from shadow volumes. The idea is simple: instead of a single shadow map for the entire scene, we generate the shadows for each shadow casting object seperately. After the scene has been rendered without shadows, shadows are generated as a post-process on the rendered image. For each shadow caster, we find candidate shadow pixels by rasterizing a shadow volume for the shadow caster's bounding box. Only for image pixels inside this volume a shadow map lookup is done, and pixels in shadow are marked or dimmed. The process is depicted in Fig. 1. For instanced objects, shadow maps can be reused for all instances. We present a simple, yet efficient and practical approach to instance shadow maps also for rotated instances.

Instanced shadow mapping results in an overhead compared to a single scene-wide shadow map. However, we gain valuable flexibility which allows for significant optimizations. Shadow map resolution is automatically focused to shadow casters, so the typical undersampling artifacts of single objects in large scenes can be solved. Shadow map updates can be restricted to dynamic objects, reducing the shadow map generation costs. The shadow maps can be reused for instanced objects in the scene, thus allowing the instanced shadow maps. Finally, we can generate transparent shadows for objects with uniform transparancy.

In this paper, we describe the algorithmic details of such instanced shadow maps, in particular the per-object shadow generation post process, and applications of instanced shadow maps. We further elaborate on instanced shadow maps for instanced objects, where the instances vary in rotations around one axis, as it is typical e.g. for objects in outdoor scenes.

Ideally, the approach is combined with deferred shading, where the shading computation is only done when all shadow information is available. Our approach is very flexible and can be easily combined with other approaches. So it is not a problem to generate shadows partially by instanced shadow maps and by shadow volumes. Furthermore, most advanced shadow map techniques, e.g. penumbra maps for the generation of soft shadows [WH03], can be combined with instanced shadow maps.

4. PREVIOUS WORK

A rather old, yet still valuable overview on shadowing techniques has been written by Woo et al. [WPF90]. We will not discuss simplistic approaches such as planar shadows or non-interactive ones such as ray traced shadows. Instead we focus on shadow maps and shadow volumes, which are the most often used algorithms in interactive graphics. We will also only discuss algorithms for hard shadows. We are aware that in terms of realism soft shadows would be a much better choice. However, despite a lot of research in this area, soft shadow methods are still very expensive and suffer from robustness problems.

Shadow Maps have been presented in the early days of computer graphics [Wil78]. Early, aliasing problems of shadow maps have been described. Reeves et al. [RSC87] proposed filtering on the shadow map lookup results in order to avoid aliasing (percentage closest filtering). Much later, several approaches have been presented to support omnidirectional lights and to adapt shadow map resolution according to the current view [BAS02, FFBG01, SD02, WSP04, MT04]. Shadow maps are in general the fastest method for shadow generation, however despite significant improvements aliasing remains a major problem and is often visible, also in commercial games.

Shadow volumes [Cro77] don't suffer from aliasing problems. For all shadow casters, the polyhedral shadow region (shadow volume) is computed. Shadows are then generated by rasterizing the shadow volumes and counting entry/exit events for every pixel in the stencil buffer [Hei91]. Those pixels with more entry than exit events lie in a shadow volume and are thus in shadow. Robustness problems have been solved by Everitt et al. [EK03]. Generally, shadow generation by shadow volumes suffers from fill rate problems. Lloyd et al. attack these by selective rendering of the potentially large shadow volumes [LWGM04]. Furthermore, the shadow volume computation is still mostly done on the CPU and thus relatively expensive. First approaches to shift this to the GPU have been presented [BS03], and latest hardware with geometry shaders [Bly06] will support this better.

Figure 2: *Shadow generation for Al. Left: image without shadows. Center: bounding shadow volume rasterized hits red pixels. For all red pixels a shadow map lookup is performed. Right: red pixels that are hidden by the shadow map are darkened.*

There have also been approaches to combine shadow maps and shadow volumes. McCool [McC00] generates shadow volumes from shadow maps. Chen et al. [CD04] render shadows using shadow maps with bilinear filtering. For all pixels with ambiguous shadow map result, shadow volumes are rerendered. In ideal cases, the quality of shadow volumes can be achieved at largely reduced fill rate.

Our algorithm shows similarities to another hybrid approach by Arvo et al. [AA03]. In their work, for every light source all pixels inside the light frustum are determined by rasterizing the light frustum and restricting the lighting and shadow computation to the hit pixels. This approach is particularly helpful for many spot lights with restricted area of influence. Instanced shadow maps work best for scenes with a few light sources and multiple scattered objects. We do not rasterize the light frusta, but a "bounding shadow volume" for every object, in order to be able to handle separate shadow maps and restricting computations to relevant pixels.

The idea of object-wise shadow maps has been presented in [Reg04]. However, their way to apply these shadows is rather different. All shadow maps are bound at the same time and a fragment program tests all bound shadow maps for occlusion. In contrast, we bind the shadow maps one by one and restrict the shadow map lookup to a tight candidate set of potential shadow receiving pixels. By this, we have no limitations in the number of possible shadow maps and reduce the number of shadow map lookups at the expense of increased fillrate. Last but not least, our approach opens the door for instancing shadow maps.

5. INSTANCED SHADOW MAPS

5.1. Light Source/Shadow Caster Pairs

The scene has to be partitioned into shadow casters. Typically, such shadow casters are players or scene objects which are modeled separately and then composited to the complete scene by a scene graph. Objects that only receive shadows, such as walls or the ground can be omitted here.

For scenes with one light source, each of the shadow casters gets a shadow map on its own. Multiple instances of single objects can easily share shadow maps, as long as they only differ by translation and scaling. Shadow map resolution can vary for different objects, depending on object characteristics, object importance, distance to the viewer etc. For the case of multiple light sources, generally a shadow map is needed for each light source/shadow caster pair. For each light source/shadow caster pair, a matrix describing the relative transformation is stored. Optimizations are possible if the light sources have a limited range of influence (e.g. spot lights, or lights with limited reach), so that light source/shadow caster pairs that cannot influence each other can be omitted. We will not further elaborate on this, but want to emphasize the gain in flexibility.

As a result, for each light source/shadow caster pair, we know a shadow map, and a matrix describing the relative transformation. Of course, each transformation is usually selected such that the shadow caster is fully visible in the shadow map without wasting pixels at the boundary. This can be achieved well by projecting the bounding box of the shadow caster into the light view and searching the smallest enclosing window on the image plane that encloses the bounding box.

5.2. Shadow Map Updates and Reuse

For a static light source/shadow caster pair, shadow maps can be reused from frame to frame. This includes pairs which are moved in the scene but are fixed with respect to each other, e.g. a lamp with a light source and a casing as shadow caster. If a rigid object is only translated (and not rotated) under parallel light (typical for outdoor scenes), the shadow map can also be reused; only the matrix must be updated. As an example, the shadow map of a racing car under sun light only needs to be updated if the car rotates (significantly).

Analogously, a shadow map can be reused for multiple instances of an object under parallel light, as long as the objects are not rotated relative to each other. This means that we use the same shadow map multiple times, but under varying transformations. Later, we will present an approach that can also handle multiple instances under simple rotations.

5.3. Shadow Rendering

Instanced shadow maps are applied *after* the complete scene has been rasterized. We need the resulting z-Buffer of the camera view as input texture. The method computes the image pixels which are shadowed by every light source/shadow caster pair.

Each light source/shadow caster pair will only affect a possibly small portion of the screen. We thus need a quick estimate of the pixels that are affected by a particular pair. This estimate is obtained by an approach in the style of shadow volumes: First, we generate a shadow volume for the bounding box of the shadow caster. In the following we will call this the "bounding shadow volume". We then determine all pixels inside this volume by rasterizing the bounding shadow volume.

Finding these pixels inside the bounding shadow volume can be done like in traditional shadow volume rendering by rasterizing front and back faces and counting entry/exit events in the stencil buffer. For our purpose, we can simplify this test, and avoid the costly detour via the stencil buffer. Because our bounding volumes of the shadow casters are convex, also their bounding shadow volumes are, so only one single entry and exit event is possible. Our entry test is made by the z-test of rasterization: we render all front-facing facets of the bounding shadow volume with enabled z-test. All generated fragments belong to a pixel with an entry event. In the upper image of Fig. 3 these pixels are marked in red. What leaves is to cull all pixels that also have an exit event. We thus test for every fragment whether it is "behind" the volume (lower image) by evaluating the plane equation for all backfacing volume facets, which can be done in a fragment program. All such pixels (shown in yellow) can be immediately killed.

For all remaining pixels, we have to do the shadow map lookup. For this, the pixel is transformed to shadow map space using the current transformation matrix. This step requires to read the depth value of the pixels, which should be provided to the fragment shader as a simple texture. Then, a simple shadow map lookup tells us whether the pixel in question is shadowed. If not, we kill the fragment. All surviving pixels are shadowed by the current shadow caster with respect to the current light source. We can simply dim these pixels using alpha-blending or can set a corresponding bit in the stencil buffer, to obtain per-light source shadow masks.

The fact that instanced shadow maps are applied as a post-process makes further extensions possible. As an example, we can assign transparencies to the shadow casters and accumulate transparent shadows. Instanced shadow maps can also be well combined with deferred shading techniques. On the downside, the need to bind the depth buffer as texture foils hardware anti-aliasing.

5.4. Alternatives

If implemented as described above, the algorithm suffers from the same problem as traditional shadow volumes do: if the viewer is inside a volume, entry points will not be rasterized, because they are behind the viewer. Thus, the test will fail incorrectly, leading to missing shadows. A solution is to use a z-fail-method instead of z-pass, as described in [EK03]. If we render the backfacing facets of the bounding shadow volume with reversed depth test, fragments are only generated if they are behind the current scene. Thus we get all pixels that do not have an exit event, and we have to test the remaining pixels against all front facing bounding shadow volume facets. This approach can only work, if the far plane for the current viewer is set to infinity, so that the bounding shadow volumes are not incorrectly clipped at the far plane.

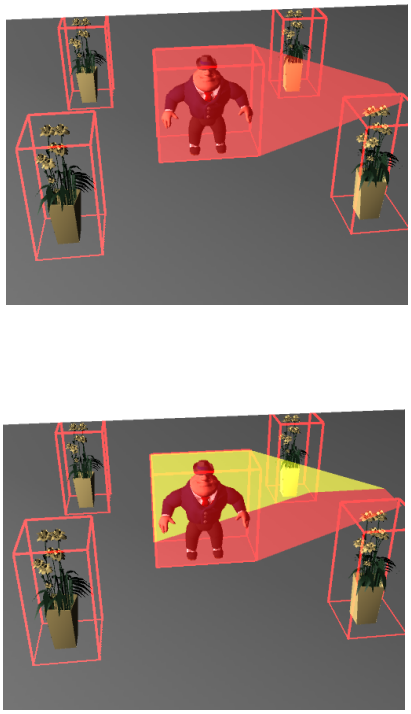

Figure 3: *Determining image pixels inside a bounding shadow volume. Top: pixels with an entry event (red), bottom: pixels with entry and exit event (yellow).*

Another alternative is to skip the exit point (z-pass) or entry point test (z-fail) in the fragment program. In Fig. 3 this would mean that we perform the shadow map lookup for all red pixels in the center left image, before the yellow pixels in the center right image have been culled away. Since fragment shader length can easily become a delimiting factor, this might result in an overall improvement. However, our shadow map lookup is a dependent texture lookup, because the shadow map position depends on the image depth of the pixel, which in turn has to be read from a texture. Thus the time while this texture lookup is done can well be used for the additional test.

5.5. Instancing

Under parallel light, we can reuse a single shadow map of an object for various instances of the object, as long as the object is only translated and scaled. This has been used in Fig. 1, where only one single shadow map of the flower vase is used to generate the shadows of all four instances.

However, in practical examples, instancing is often tried to be hidden, so that the instances are rotated and scaled relative to each other. If the instance rotation angles are restricted to a small set of angles, we can also precompute a small set of shadow maps for all these rotations and select the appropriate one for each instance. Alternatively, we restrict rotation to rotations around a single axis, and generate shadow maps at e.g. 30^o steps. For instances, we allow arbitrary rotation angles, and interpolate the shadow from the two closest shadow maps. The simplest solution is to determine the shadow for both shadow maps and interpolate the result. However, this makes the interpolation well visible.

For instances of vegetation, we apply a trick that generates plausible shadows. We rotate the two closest shadow maps to the desired position, and interpolate the two obtained depth values. Then, the depth comparison is done with this interpolated depth value. As a result, we achieve hard shadows for all angles. The approach works well, as long as the object is roughly cylinder shaped. Errors mainly happen on the object due to wrong self-occlusion, however on trees and bushes such errors are simply masked by the complex structure. Popping becomes visible, if the light source or the object are animated. This precomputation of a set of rotated shadow map also works if single objects are dynamically rotated around a single axis, e.g. a racing car as long as it does not roll over. The approach is not applicable in the case of point light sources. Here sampling a single rotation angle is not sufficient, we would have to add another dimension for view distance, where the curse of dimensionality results in an explosion of required shadow maps.

6. IMPLEMENTATION AND RESULTS

We implemented instanced shadow maps under Windows using OpenGL and GLSL 1.10. Light source/shadow caster pairs are provided by the modeler. At the beginning of each frame, all necessary shadow maps are generated or updated and the matrices determined. A frame buffer object is used for the initial, unshadowed rendering pass. The depth buffer is bound as a texture to make it available to the following render passes.

Fig. 4 shows the fillrate spent by instanced shadow maps. For a single shadow map (left), part of the screen is overdrawn once (red and green pixels with positive shadow test result for the red pixels). For a more complex scene (center), the bounding shadow volumes overlap, resulting in moderately increased overdraw. The fillrate required for a shadow is independent of the object's geometric complexity, but directly related to its size, i.e. the small shadows for distant trees are very cheap.

Fig. 5 shows rendering of an example scene with 105 instances of a tree with 36.000 triangles each, i.e. a total of 3.8 million triangles. The trees are arbitrarily rotated around the vertical axis. 12 shadow maps at 30 degrees steps with resolution 512^2 are used for the tree instances, occupying 12 Megabytes of video memory. Without shadows, the scene renders on an nVIDIA GeForce 8800 at 63 frames per second with resolution 1024x768. With shadows, the frame rate drops to 43 fps (without shadow map regeneration), 30 fps with shadow map regeneration. Standard shadow mapping had a frame rate of 31 fps but with strong shadow artefacts (using a 2048^2 shadow map).

7. CONCLUSION

In this paper, we presented a simple, yet efficient approach to apply object-wise shadow maps. Instead of querying all per-object shadow maps for every fragment, we apply the shadow maps in a postprocess one by one, and restrict the shadow map queries to the pixels that can be affected by the shadow map. Compared to other per-object shadow map techniques, our approach can efficiently handle a large number of shadow maps. It is not necessary to bind a large number of textures at once or to pack multiple shadow maps into an atlas. Shadow maps can be reused for multiple instances of an object. Our approach is particularly suited for large outdoor scenes with many static, instanced objects (such as vegetation) and some animated objects.

The main idea is to have separate shadow maps for all scene objects. Compared to the normal single-shadow map approaches, this gives us flexibility to provide sufficient shadow map resolution to all objects, to partially reuse shadow maps also in animated scenes, and to reuse shadow maps for object instances.

Instanced shadow maps require memory overhead compared to standard shadow mapping. In terms of fill rate, they behave roughly like shadow volumes. With well-chosen shadow casters and shadow map resolutions, shadow quality close to shadow volumes can be achieved. Compared to shadow volumes, we avoid stencil buffer operations and do not need to compute the shadow volume of a general object, which is still costly and badly supported by current GPUs. Instead, we have to generate a shadow volume of a box, which can be done with a simple vertex shader, and render per-object shadow maps, which only requires standard rasterization.

Moreover, instanced shadow maps can be easily combined with other techniques, such as precomputed shadow textures, shadow volumes, and many advanced shadow map techniques.

8. BIOGRAPHY

Sebastian Buntin is a PhD student at the Computer Graphics Group of the University of Erlangen-Nuremberg, located in Bavaria, Germany. His main research focus is currently the visualization and simulation of complex plant ecosystems using modern graphics hardware.

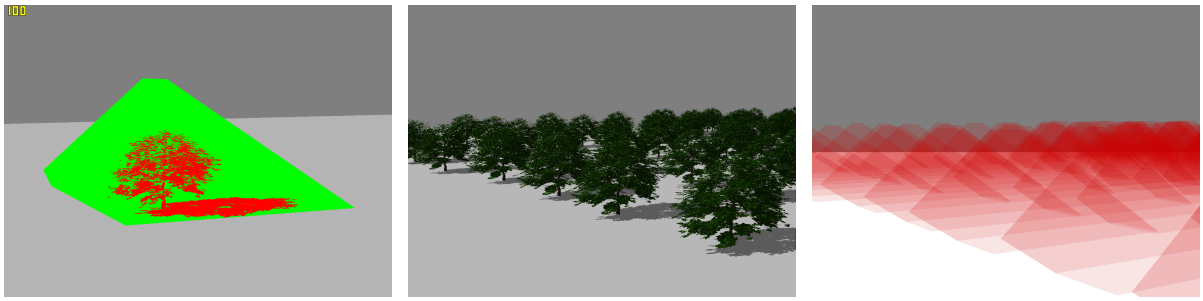

Figure 4: *Left: colored pixels are candidate pixels for the tree's shadow. Shadowed pixels are drawn in red, unshadowed ones in green. Center/right: complex scenes with visualization of generated overdraw.*

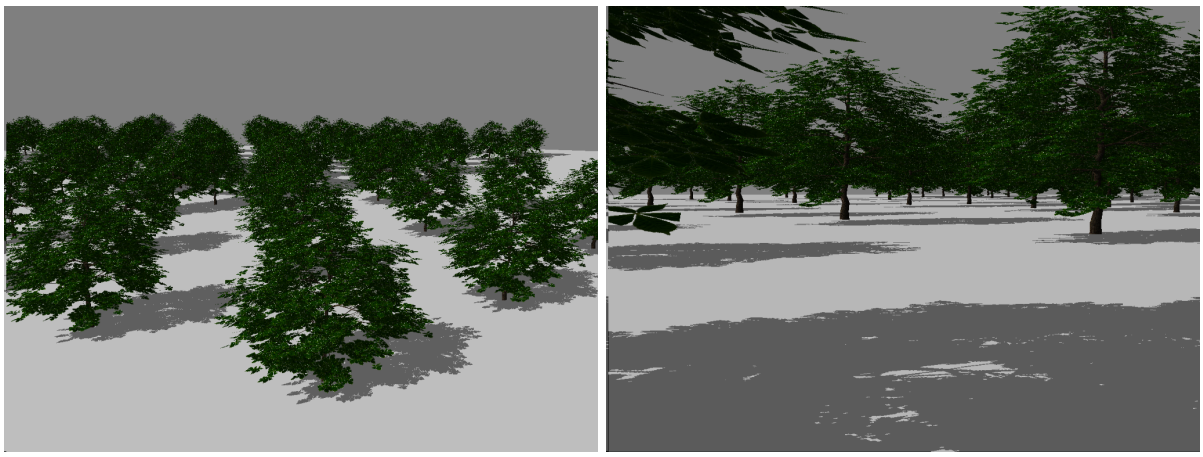

Figure 5: *105 rotated instances with 3.8 million triangles in total, rendered at 43 frames per second.*

References

[AA03] ARVO J., AILA T.: Optimized shadow mapping using the stencil buffer. *Journal of Graphics Tools 8*, 3 (2003), 23–32.

[BAS02] BRABEC S., ANNEN T., SEIDEL H.-P.: Practical shadow mapping. *Journal of Graphics Tools 7*, 4 (2002), 9–18.

[Bly06] BLYTHE D.: The direct3d 10 system. *ACM Transactions on Graphics 25*, 3 (July 2006), 724–734.

[BS03] BRABEC S., SEIDEL H.-P.: Shadow volumes on programmable graphics hardware. *Computer Graphics Forum 22*, 3 (Sept. 2003), 433–440.

[CD04] CHAN E., DURAND F.: An efficient hybrid shadow rendering algorithm. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering* (June 2004), pp. 185–196.

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 77)* (July 1977), vol. 11, pp. 242–248.

[EK03] EVERITT C., KILGARD M.: Optimized stencil shadow volumes. *Game Developer Conference* (2003). http://developer.nvidia.com/docs/IO/8230/GDC2003_ShadowVolumes.pdf.

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *Proceedings of ACM SIGGRAPH 2001* (Aug. 2001), Computer Graphics Proceedings, Annual Conference Series, pp. 387–390.

[Hei91] HEIDMANN T.: Real shadows, real time. *The IRIS Universe, Silicon Graphics 18* (1991), 23–31.

[LWGM04] LLOYD D. B., WENDT J., GOVINDARAJU N. K., MANOCHA D.: Cc shadow volumes. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering* (June 2004), pp. 197–206.

[McC00] MCCOOL M. D.: Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics 19*, 1 (Jan. 2000), 1–26.

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering* (June 2004), pp. 153–160.

[Reg04] REGE A.: Shadow considerations, 2004. download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_Shadows.pdf.

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (July 1987), vol. 21, pp. 283–291.

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective

shadow maps. *ACM Transactions on Graphics 21*, 3 (July 2002), 557–562.

[WH03] WYMAN C., HANSEN C.: Penumbra maps: Approximate soft shadows in real-time. In *Eurographics Symposium on Rendering: 14th Eurographics Workshop on Rendering* (June 2003), pp. 202–207.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)* (Aug. 1978), vol. 12, pp. 270–274.

[WPF90] WOO A., POULIN P., FOURNIER A.: A survey of shadow algorithms. *IEEE Computer Graphics & Applications 10*, 6 (Nov. 1990), 13–32.

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering* (June 2004), pp. 143–152.