Please cite as: S. Hertel, K. Hormann, and R. Westermann. A hybrid GPU rendering pipeline for alias-free hard shadows. In D. Ebert and J. Krüger, editors, *Eurographics 2009 Areas Papers*, pages 59–66, München, Germany, March/April 2009. Eurographics Association.

A Hybrid GPU Rendering Pipeline for Alias-Free Hard Shadows

Stefan Hertel
Fakultät für Informatik
Technische Universität München

Kai Hormann
Department of Informatics
Clausthal University of Technology

Rüdiger Westermann
Fakultät für Informatik
Technische Universität München

Abstract

We present a new GPU pipeline for rendering per-pixel exact shadows that are cast by point lights and parallel lights. Our approach is hybrid in that it uses kD-tree accelerated ray-tracing to determine shadow-ray intersections, and rasterization to effectively reduce both the number of shadow rays to be traversed and the number of sub-spaces to be considered along each of these rays. To achieve this we introduce conservative shadow maps, which store a conservative estimate of the first intersection with the scene for each possible shadow ray. A novel approach to build such a map is presented, which uses rasterization to compute for every shadow-map pixel the triangles intersecting this pixel. By exploiting the rasterization capacities of recent GPUs in combination with accurate ray-triangle intersection tests, we are able to efficiently compute alias-free shadows in high-resolution and spatially extended scenes where classical shadow mapping techniques have severe difficulties.

1 Introduction

Shadow maps as introduced by Williams [29] provide an efficient and elegant means to simulate hard shadows that are cast by point light sources or parallel lights. Unfortunately, classical shadow maps face the problem of aliasing, meaning that geometry features cannot be captured adequately in the underlying discrete sampling grid. As a result, shadows caused by small features can be missed and light falling between two nearby occluders can be blocked. These limitations are rooted in the point-sampling process that is used to generate a shadow map. They can only be eliminated by correctly identifying the features inside a shadow map pixel and computing exact pixel coverage for them.

With ray-tracing, on the other hand, it is conceptually simple to compute accurate hard shadows by casting at every visible surface point—the view samples—a shadow ray towards the light source. Since shadow rays are not subject to a particular sampling regime and can be tested analytically against potential occluders, they can effectively avoid the aliasing artifacts that are inherent to shadow maps. With respect to performance, however, shadow-ray tracing is still not a real competitor to shadow mapping using rasterization hardware. Especially if hierarchical space-partitions in combination with advanced rendering pipelines are employed [5], the creation of shadow maps can be performed on the GPU in a much shorter period of time than it takes to simulate shadows with ray-tracing.

In this work we present a novel GPU rendering pipeline for simulating hard shadows, which addresses the aforementioned issues by providing unlimited shadow map resolution. It is similar in spirit to alias-free shadow maps proposed by Aila and Laine [1] and Johnson et al. [19] in that it evaluates light-space visibility at projected view-samples. Thus, an irregular and view-adaptive sampling grid is employed.

In contrast to the work by Aila and Laine, however, our method uses ray-tracing for accurate visibility determination and rasterization for accelerating the ray-tracing process. GPU ray-tracing is performed using a kD-tree acceleration structure as proposed in [11]. In addition, a conservative shadow map is utilized to reduce the number of shadow rays to be cast and sub-spaces to be traversed by these rays. The shadow map is built using conservative triangle rasterization, meaning that a

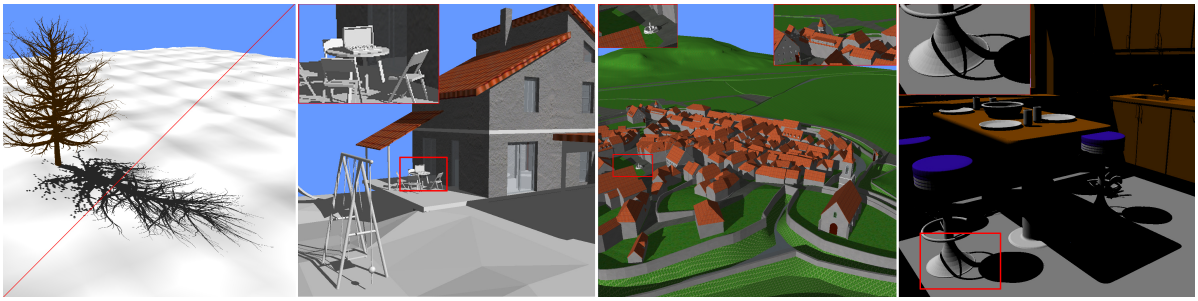

Figure 1: From left to right: a tree rendered with a 1K × 1K shadow map at 42 fps (upper left) and using alias-free shadow maps as proposed in this work at 3 fps (bottom right); the yard scene rendered with a 1K × 1K shadow map at 60 fps and with alias-free shadows at 4 fps; the village scene rendered at 7 fps with alias free shadows; the kitchen scene rendered with alias-free shadows at 9 fps. In all three examples the resolution of the conservative shadow map was 1K × 1K.

triangle is rasterized into a shadow map pixel once the triangle's light-space projection has a non-empty intersection with this pixel. Every triangle that is rasterized into the conservative shadow map leaves a unique triangle ID as well as the minimum distance of the triangle subarea covering the shadow map pixel to the light source.

The conservative shadow map can be used to determine view-samples that might have been incorrectly classified as "in-shade" or "illuminated" by standard shadow mapping, and, therefore, allows reducing the number of shadow rays to be traced. Furthermore, the depth values stored in the conservative shadow map provide conservative estimates of the first possible intersections along the shadow-rays. This information can be used in turn in the ray-tracing procedure to skip large parts of the domain.

Our paper makes the following specific contributions:

- A conservative rasterization method for triangles that avoids construction and rendering of additional geometry.
- A method to efficiently determine view-samples that might be incorrectly classified by standard shadow mapping.
- The use of a conservative shadow map as an acceleration structure for shadow-ray tracing.

Compared to the GPU implementation of alias-free shadow maps proposed by Sintorn et al. [26], our method differs in a number of aspects. Firstly, the construction of a dynamic list structure on the GPU, which is used to assign projected view-samples to shadow map entries, can entirely be avoided. Secondly, since we use a conservative shadow map to aggressively reduce the number of shadow rays to be traced as well as the number of triangles these rays have to be tested with, we end up with considerably fewer intersection tests on the GPU. Especially for complex scenes having high depth complexity, the assumed logarithmic complexity of ray-tracing in the number of polygons can deploy its full potential and results in faster rendering times.

The remainder of this paper is organized as follows: In the next chapter we review previous work that is related to ours. Then, we introduce conservative shadow maps, discuss their construction, and show how to use them for view-sample classification. Next, we briefly describe the GPU ray-tracer we use in the current work. We conclude the paper with a detailed performance analysis of the different stages of our approach, and we show a number of examples.

2 Related Work

Interactive shadow rendering techniques have been at the core of computer graphics research for many years, and today there exists an extensive amount of literature on this subject which we cannot attempt

to review in detail here. Woo et al. [31], Haines et al. [14] and Hasenfartz et al. [15] discuss a number of previous and current algorithms and provide many useful references on this subject.

Interactive rendering algorithms for hard shadows can roughly be grouped into three different categories: *geometry-based techniques*, *image-based techniques* and *ray-tracing*. Into the first category falls research on shadow volumes [8] and its variants [7, 18, 20, 22], an object-space approach that is based on computing a boundary representation of the region occluded from a light source. The shadow volume technique renders accurate per-pixel shadows, but it requires the creation of the shadow geometry as well as some rasterization overhead to render this geometry, and its efficiency thus strongly depends on the geometric complexity of the shadowing object.

In the second category, research is mainly pursued on the improvement of shadow mapping [29], which is currently the most used shadowing technique in real-time scenarios. Shadow mapping rasterizes the scene into a depth map as seen from the light source and then uses this map to test whether the light is visible from a view sample or occluded. Since shadow map construction only involves rendering the scene from the light source, it is reasonably fast and does not depend on the model's geometric complexity.

Shadow mapping, on the other hand, suffers from discretization artifacts due to limited shadow map resolution causing loss of shadow details and temporal coherence. To avoid this, some efforts have been undertaken to improve the virtual shadow map's resolution according to the viewer's perspective, for instance by using warped projection schemes [27, 30] and adaptively refined shadow maps [10]. Alias-free shadow maps [1], on the other hand, provide a solution to the aliasing problem by avoiding the regular sampling grid underlying classical shadow mapping. It uses projected view-samples as irregularly distributed sampling points and can thus simulate pixel-exact hard shadows. Conceptually similar approaches utilizing more advanced GPU features to speed up the irregular sampling process were proposed in [19, 4, 26].

Another concern in shadow mapping is shadow map filtering to provide screen-space anti-aliasing for both hard and soft shadows. Percentage closer filtering [25] samples the result of the light-view depth test at multiple positions around the projected view-samples, and then averages these samples to compute a fractional view-sample occlusion. Variance shadow maps [9, 21] calculate an upper bound for the probability of a receiver fragment to be shadowed by samples around some point in a shadow map. A Fourier series expansion of the visibility step function at shadow boundaries is used in convolution shadow maps [2] to blur the visibility along such boundaries. Recently, exponential shadow maps [3] have been proposed, which replace the hard shadow ramp by an approximate exponential decay.

In the third category, methods have been developed to simulate shadows using classical ray-tracing [28]. Ray-tracing provides the most intuitive method for simulating hard shadows in that it simply spawns at each view-sample an additional shadow ray towards the light source. Ray-tracing, on the other hand, is in general too slow to be used in interactive environments. However, over the last few years considerable effort has been put into the implementation of ray tracing on programmable graphics hardware. Inspired by the early work of Purcell et al. [24] and Carr et al. [6], in a number of succeeding implementations it was shown that the capabilities of recent GPU stream architectures can effectively be used for ray tracing. Foley and Sugerman [11] and Popov et al. [23] independently examined stack operations—a feature not well supported by the GPU—and they reported a significant performance gain by using a stackless traversal algorithm for kD-trees. Most recently, Günther et al. [13] introduced so-called ropes to avoid re-starting the tree traversal from the root node in case no hit was found in a sub-space.

3 Hybrid GPU Shadow Rendering

The idea behind our method is to combine a fast but low-resolution shadow map with a pixel-wise exact shadow test. In addition, the shadow map is used to reduce the number of shadow rays to be tested and to effectively restrict the ray intervals that have to be considered.

The method can be split into two stages: Firstly, a shadow map is constructed, which, as will be explained later, stores additional information about the triangles rasterized into this map. Secondly, the

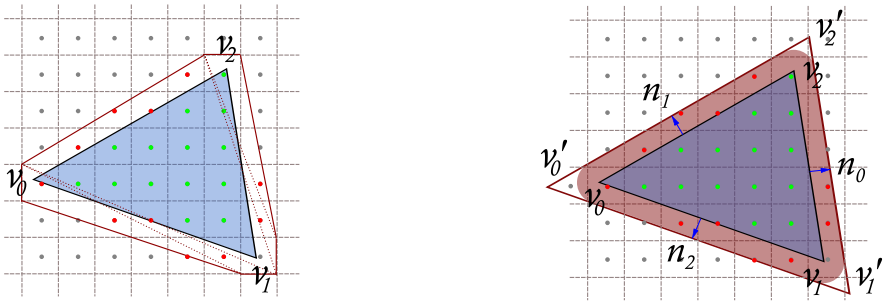

Figure 2: Left: The expansion of a triangle (blue) to a polygon (red) as suggested by [16], with one possible triangulation (dotted). Right: The expansion of a triangle by moving the edges along their normal (blue). The light red shape shows the area a distance smaller than l to the triangle (l is the length of a pixel's diagonal.)

scene is rendered as seen from the camera, and every visible view-sample is tested against the shadow map for being in one of the following three regions:

- lit
- shadowed
- uncertain

If a view-sample is categorized as "uncertain", it cannot directly be classified as being in light or in shadow with only the information stored in the shadow map. Then, an exact visibility test has to be performed.

4 Conservative Shadow Maps

To perform the classification of view-samples into "lit", "shadowed", and "uncertain", a particular kind of shadow map is employed—a so-called *conservative shadow map*. Such a map stores for every pixel the minimum light-space depth of all triangle subareas overlapping this pixel after the light-space projection. Therefore, we utilize the graphics API's functionality, i.e., using the slope scaled depth-bias, to shift the triangles towards the light such that the depth value at covered pixel centres are less than the aforementioned minimum depth. By using the resulting depth information, a simple comparison between the light-space depth of a view sample and the depth stored in the shadow map can be used to classify lit view samples. If the stored depth is greater than the sample's depth or if no depth is stored, then the sample is illuminated.

4.1 Construction

To generate a conservative shadow map we use *conservative rasterization*, which works by rasterizing a triangle into a shadow-map pixel once the triangle's light-space projection has a non-empty intersection with this pixel. This method enlarges the triangles' projections in view-space in such a way as to cover the centres of all shadow-map pixels that are touched by the projection. This is in contrast to classical shadow mapping, where fragments are only generated for those triangles that cover a shadow-map pixel centre. Hasselgren et al. [16] proposed a conservative triangle rasterization method that expands projected triangles to polygons with up to nine vertices. The positions of the new vertices are computed from the positions of the corners of those pixels containing the original triangle vertices (see Figure 2).

On Shader Model 4.0 capable graphics hardware the construction of the enlarged polygon can be performed using the geometry shader. However, as it was already pointed out by Sintorn et al. [26], geometry amplification in the shader can lead to severe performance limitations. Due to this reason

we developed an alternative approach that only resizes the initial triangle in the geometry shader by moving the edges about the length of a pixel's diagonal l in the direction of the edge normals but—in contrast to the suggestion in [16]—avoids intersecting the enlarged triangle with the shadow-map pixels that are overlapped by the triangle. Instead, a fragment shader is used to discard all fragments with a distance to the triangle that is larger than the length of the pixel diagonal l (see Figure 2).

The distance is the minimum of the perpendicular distances to the triangle edges and the three corners. It can be calculated by using the barycentric coordinates of a generated fragment with respect to the triangle that this fragment originates from. For a planar triangle T with vertices v_0, v_1, v_2 we let n_i be the outward pointing unit normal vectors of the edges $e_i = v_{i+1} - v_i$ (with indices considered modulo 3). The enlarged triangle T' is now constructed by intersecting the lines that are parallel and at (positive) distance l to the edges of T (where l is the length of the pixel diagonal). Simple geometric reasoning shows that the vertices v_i' of T' can be computed as

$$v_i' = v_i + l\left(\frac{e_{i-1}}{e_{i-1}\cdot n_i} + \frac{e_i}{e_i \cdot n_{i-1}}\right),$$

where ' $\cdot$ ' denotes the standard dot product.

Since the edges are moved by the length of the pixel diagonal, we have an over-conservative rasterization. Rendering the enlarged triangle T' will cause the generation of more fragments than needed for a conservative rasterization. Especially at sharp angles the vertices will be moved far away from the original position and the enlarged triangle will cover a lot of pixels.

For all fragments of T' we let the shader calculate the distance $d(w, T)$ of the pixel centre w to the triangle T and discard it whenever it is larger than the length l of the pixel diagonal. In order to calculate $d(w, T)$ efficiently, we resort to some tricks that are based on the concept of *barycentric coordinates*. The barycentric coordinates $\lambda_i(u)$ of a point u with respect to the triangle $[v_0, v_1, v_2]$ in the plane are defined as the ratios of triangle areas,

$$\lambda_i(u) = \frac{\text{area}[u, v_{i+1}, v_{i+2}]}{\text{area}[v_0, v_1, v_2]}.$$

These three values sum up to 1 and allow us to write u as an *affine combination* of the vertices v_i,

$$u = \lambda_0(u)v_0 + \lambda_1(u)v_1 + \lambda_2(u)v_2,$$

hence their name. An important property of barycentric coordinates is that they are linear in u. Thus, if we compute the barycentric coordinates of the vertices v_i' in the geometry shader and pass them on as vertex attributes, we can let the rasterizer do the linear interpolation so as to provide the barycentric coordinates $\lambda_i = \lambda_i(w)$ of the pixel centre to the fragment shader.

Due to the properties of barycentric coordinates we have

$$\begin{aligned} w - v_i &= \lambda_{i-1}v_{i-1} + \lambda_i v_i + \lambda_{i+1}v_{i+1} - v_i \\ &= \lambda_{i-1}v_{i-1} + (1 - \lambda_{i-1} - \lambda_{i+1})v_i + \lambda_{i+1}v_{i+1} - v_i \\ &= \lambda_{i-1}(v_{i-1} - v_i) + \lambda_i(v_{i+1} - v_i) \\ &= -\lambda_{i-1}e_{i-1} + \lambda_{i+1}e_i, \end{aligned}$$

so that the squared distance between w and some vertex v_i of T is

$$d(w, v_i)^2 = \|w - v_i\|^2 = \lambda_{i-1}^2\|e_{i-1}\|^2 + \lambda_{i+1}^2\|e_i\|^2 - \lambda_{i-1}\lambda_{i+1}(e_{i-1}\cdot e_i). \tag{1}$$

Note that the three squared lengths $\|e_i\|^2$ and the three dot products $e_{i-1}\cdot e_i$ are constant for each triangle and can thus be pre-computed and stored as face attributes. Moreover, since (1) involves only scalar values, all three squared vertex distances can be computed efficiently in parallel in the fragment shader. Barycentric coordinates can also be used to compute the (signed) distances of w to the edges e_i of T, because

$$d(w, e_i) = \lambda_{i-1}d(v_{i-1}, e_i),$$

and as before, the (signed) distances $d(v_{i-1}, e_i) = n_i \cdot e_{i+1}$ can be pre-computed and stored as face attributes. Note that by this definition of signed distances, the points contained in T are exactly those for which all edge distances are *negative*.

Finally, the distance $d(w, T)$ of the pixel centre to the triangle is the minimum of the three vertex distances $d(w, v_i)$ and the three absolute edge distances $|d(w, e_i)|$. Hence, if all squared vertex distances are larger than l^2 and all edge distances exceed l, we conclude that the pixel w does not overlap T and discard it. On the other hand, if all edge distances are smaller than $-l$, then we can be sure that the whole pixel is covered by T (note that in this case the vertex distances do not need to be considered as they are always larger than the absolute edge distances).

4.2 Classification

By comparing a view-sample's light-space depth to the depth value stored in the conservative shadow map, many samples can be classified as "lit." In addition, shadow-map pixels can be marked as "fully covered" based on the distances of the pixel's center to the edges of the triangle that was rasterized into it. In the previous section we described how to efficiently perform this computation using barycentric coordinates. We can now refine the classification further by using this information. If a view-sample is projected into a fully covered pixel and the stored depth is greater (or smaller) than its own depth, then the sample can be classified as "lit" (or "shadowed"). If the pixel was partially covered, an exact test between the shadow ray originating from the view sample and the triangle touching the pixel is necessary.

We accommodate this by storing at every shadow map pixel the unique ID of the triangle being captured at this pixel. Once a view-sample has been projected into the conservative shadow map, it first reads the stored ID via nearest neighbour texture lookup and then reads the three vertices corresponding to this ID. By projecting the view sample into the plane spanned by the triangle, it can be efficiently determined whether the view sample is covered by this triangle. If this is not the case, the sample is classified as "uncertain" and some further tests have to be carried out.

The triangle ID can also be used to avoid another well known problem of shadow maps—self shadowing. By simply comparing the ID that was read from the shadow map with the ID of the triangle the view sample belongs to, self shadowing can be detected and pixels with false shadowing can be eliminated. A depth bias as it is typically used to avoid this problem is not necessary.

5 GPU Ray-Tracing

For every view sample that is labelled "uncertain", a shadow ray is traced on the GPU. Since the ray-tracer is used to trace shadow rays, it only has to search for any intersection, and it can terminate the ray traversal once an intersection has been found. The exact position of this intersection does not need to be computed. GPU ray-tracing is performed using a kD-tree acceleration structure and a special traversal routine that avoids the need for a stack as in classical recursive tree traversal. For a detailed description of this approach let us refer to [11] and [17].

The ray-tracing approach takes advantage of the information that is stored in the conservative shadow map to speed up ray traversal. Specifically, the depth values stored in this map can effectively be used to restrict the ray intervals that have to be considered. At first glance, this kind of acceleration seems to be redundant because the kD-tree should already provide an effective means to skip empty space. However, by letting the shadow rays start at the stored light-space depth and by traversing them towards the view sample where they were spawned from, an intersection point will most likely be found after a few steps along the ray, i.e., by only traversing a few sub-spaces, because in many cases an intersection point is close to the stored depth estimate. Furthermore, depending on the built kD-tree, generated sub-spaces do not always provide very tight bounds to the enclosed geometry. In this case, many sub-spaces will be tested without finding any intersection. By using the depth information stored in the conservative shadow map, the number of sub-spaces to be tested can be reduced to some extent.

To initiate shadow-rays tracing on the GPU, a full-screen quad is rendered and a pixel shader is employed to discard those view-samples which are back-facing the light source or can securely be

	r_{polygon}	r_{discard}	speedup
village (247432 tris)	21	13	40%
kitchen (103351 tris)	19	13	21%
tree on snow (78646 tris)	8	7	21%

Table 1: Rendering time (ms) for conservative rasterization by expanding the triangles to polygons (r_{polygon}) as proposed by Hasselgren et al. [16] and our approach by discarding fragments with larger distances to the triangle than a pixel's diagonal (r_{discard}).

	RT_{class}	RT_{insec}	RT_{opt}	# rays	# insection rays
4 buildings (11154 tris)	158	114	95	32079	55294
village (247432 tris)	422	288	217	236350	905290
tree on snow (78646 tris)	3115	2670	2585	342105	623890
kitchen (103351 tris)	956	641	610	198576	967655

Table 2: Rendering times (ms) for hard shadows in different scenes using GPU ray tracing (RT_{class}), GPU ray tracing of only the "uncertain" shadow rays (RT_{insec}), and GPU ray tracing of only the "uncertain" shadow rays with reduced ray lengths (RT_{opt}). The last two columns give the number of shadow rays cast without and with the proposed classification scheme. Timings include the construction of the conservative shadow map.

classified as "lit" or "shadowed" as described before. Otherwise, a shadow ray is spawned and traversed through the kD-tree acceleration structure until the first intersection is found or the ray leaves the domain. In these cases the sample is classified as "shadowed" and "lit", respectively.

6 Results

To validate the efficiency and accuracy or the proposed method, we have tested the proposed GPU technique for rendering alias-free hard shadows in a number of different scenarios consisting of several thousands up to hundreds of thousands of triangles. Resulting images are given in Figure 1 and the colour plate at the end of the manuscript.

All of our tests were run on an Intel Core 2 Quad PC, equipped with 2 GB RAM and an Nvidia Gefore 8800 GTS graphics card. The proposed shadow algorithm was implemented as part of a viewer for spatially extended city models, which uses occlusion culling as described in [5] for both the generation of view samples and the conservative shadow map. The kD-tree structure is constructed in a preprocess step on the CPU using the SAH heuristics [12]. All tests were rendered into a 1280×1024 view port using a conservative shadow map of size 1K $\times$ 1K.

Our first test demonstrates the efficiency of conservative rasterization as proposed in this paper compared to the approach by Hasselgren et al. [16]. For different scenes, Table 1 shows timings in milliseconds for constructing the shadow map on the GPU. As it can be seen, our method clearly outperforms previous approaches independently of the scene complexity. We attribute this to the fact that our method requires a significantly lower geometry throughput on the GPU, and instead exploits the computing and memory access capacities in the fragment units.

Table 2 gives representative timings for the rendering of alias-free hard shadows using classical GPU ray-tracing and the improved method proposed in this work. The first column shows the time (ms) it took to render accurate shadows in various scenes using GPU ray-tracing. Timings in the second column refer to the rendering of shadows using GPU ray-tracing, but only for those view samples that were classified as "uncertain". The third column shows the time it took to render the scenes if the ray intervals were restricted with respect to the depth values stored in the conservative shadow map. The last two columns give the number of shadow rays that were cast without and with the proposed classification scheme.

As can be seen, by means of the proposed acceleration schemes our technique can simulate hard

shadows in a significantly shorter period of time than classical GPU ray tracing, while at the same time providing the same accuracy (see the comparison in Figure 5). Compared to classical shadow mapping, on the other hand, the quality can be improved to some extent (see Figure 3). In particular the tree scene demonstrates the strength of our method. In this scene, the kD-tree acceleration structure cannot deploy its full potential due to the relatively small empty sub-spaces and the fine granularity of the branches. This prohibits early-ray termination for most of the shadow rays, meaning that many of them have to traverse the entire space interval towards the light source. Our method, on the other hand, can effectively classify many view samples as "lit" and "shadowed" and can, therefore, greatly reduce the number of shadow rays. Moreover, as the statistics in Table 2 shows, many shadow rays can effectively benefit from interval reduction as described.

7 Conclusion and Future Work

In this work we have described a technique for GPU rendering of alias-free hard shadows. By exploiting the strength of rasterization-based shadow mapping and GPU ray-tracing in combination with a novel shadow map type, we have demonstrated a considerable improvement in rendering performance over classical GPU ray-tracing. At the same time, typical discretization artifacts as they are paramount to shadow mapping can entirely be avoided. As our timings indicate, the proposed technique enables interactive rendering of pixel-accurate hard-shadows in complex scenes. Since we use a view-independent scene representation based on a kD-tree, the method is independent of both the viewing position and the light position.

In the future we want to investigate the extension of the current approach towards the use for shadow filtering. One possibility might be to directly use the barycentric coordinates to estimate sub-pixel coverage of projected triangles. Since we have already utilized barycentric coordinates to determine whether a pixel is fully covered by a triangle, the next step is to investigate whether these coordinates can already provide an estimate of the per-pixel coverage. Another possibility is to use cones of rays or simply multiple rays to estimate the coverage.

References

[1] T. Aila and S. Laine. Alias-free shadow maps. In *Rendering Techniques 2004*, Eurographics Symposium on Rendering, pages 161–166, Norrköping, Sweden, June 2004. Eurographics Association.

[2] T. Annen, T. Mertens, P. Bekaert, H.-P. Seidel, and J. Kautz. Convolution shadow maps. In *Rendering Techniques 2007*, Eurographics Symposium on Rendering, pages 51–60, Grenoble, France, June 2007. Eurographics Association.

[3] T. Annen, T. Mertens, H.-P. Seidel, E. Flerackers, and J. Kautz. Exponential shadow maps. In *Proceedings of Graphics Interface 2008*, pages 155–161, Windsor, ON, May 2008. Canadian Information Processing Society.

[4] J. Arvo. Alias-free shadow maps using graphics hardware. *Journal of Graphics Tools*, 12(1):47–59, 2007.

[5] J. Bittner, M. Wimmer, H. Piringer, and W. Purgathofer. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum*, 23(3):615–624, Sept. 2004. Proceedings of Eurographics 2004.

[6] N. A. Carr, J. D. Hall, and J. C. Hart. The ray engine. In *Proceedings of Graphics Hardware 2002*, pages 37–46, Saarbrücken, Germany, Sept. 2002. Eurographics Association.

[7] E. Chan and F. Durand. An efficient hybrid shadow rendering algorithm. In *Rendering Techniques 2004*, Eurographics Symposium on Rendering, pages 185–195, Norrköping, Sweden, June 2004. Eurographics Association.

[8] F. C. Crow. Shadow algorithms for computer graphics. *ACM SIGGRAPH Computer Graphics*, 11(2):242–248, Aug. 1977. Proceedings of SIGGRAPH 1977.

[9] W. Donnelly and A. Lauritzen. Variance shadow maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pages 161–165, Redwood City, CA, Mar. 2006. ACM.

[10] R. Fernando, S. Fernandez, K. Bala, and D. P. Greenberg. Adaptive shadow maps. In *Proceedings of SIGGRAPH 2001*, pages 387–390, Los Angeles, CA, Aug. 2001. ACM.

[11] T. Foley and J. Sugerman. Kd-tree acceleration structures for a GPU raytracer. In *Proceedings of Graphics Hardware 2005*, pages 15–22, Los Angeles, CA, July 2005. ACM.

[12] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[13] J. Günther, S. Popov, H.-P. Seidel, and P. Slusallek. Realtime ray tracing on GPU with BVH-based packet traversal. In *Proceedings of the 2007 Symposium on Interactive Ray Tracing*, pages 113–118, Ulm, Germany, Sept. 2007. IEEE.

[14] E. Haines and T. Akenine-Möller. Real-time shadows. Lecture at Game Developers Conference, Mar. 2001.

[15] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, Dec. 2003.

[16] J. Hasselgren, T. Akenine-Möller, and L. Ohlsson. Conservative rasterization on the GPU. In M. Pharr, editor, *GPU Gems 2*, chapter 42, pages 677–690. Addison-Wesley, 2005.

[17] D. R. Horn, J. Sugerman, M. Houston, and P. Hanrahan. Interactive k-d tree GPU raytracing. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, pages 167–174, Seattle, WA, Apr./May 2007. ACM.

[18] S. Hornus, J. Hoberock, S. Lefebvre, and J. C. Hart. *ZP+*: correct *Z-pass* stencil shadows. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pages 195–202, Washington, D.C., Apr. 2005. ACM.

[19] G. S. Johnson, W. R. Mark, and C. A. Burns. The irregular Z-buffer and its application to shadow mapping. Technical Report TR-04-09, Department of Computer Sciences, The University of Texas at Austin, Apr. 2004.

[20] S. Laine. Split-plane shadow volumes. In *Proceedings of Graphics Hardware 2005*, pages 23–32, Los Angeles, CA, July 2005. ACM.

[21] A. Lauritzen and M. McCool. Layered variance shadow maps. In *Proceedings of Graphics Interface 2008*, pages 139–146, Windsor, ON, May 2008. Canadian Information Processing Society.

[22] B. Lloyd, J. Wendt, N. Govindaraju, and D. Manocha. CC shadow volumes. In *Rendering Techniques 2004*, Eurographics Symposium on Rendering, pages 197–205, Norrköping, Sweden, June 2004. Eurographics Association.

[23] S. Popov, J. Günther, H.-P. Seidel, and P. Slusallek. Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum*, 26(3):415–424, Sept. 2007. Proceedings of Eurographics 2007.

[24] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002. Proceedings of SIGGRAPH 2002.

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

[26] E. Sintorn, E. Eisemann, and U. Assarsson. Sample based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum*, 27(4):1285–1292, June 2008. Proceedings of the Eurographics Symposium on Rendering 2008.

[27] M. Stamminger and G. Drettakis. Perspective shadow maps. *ACM Transactions on Graphics*, 21(3):557–562, July 2002. Proceedings of SIGGRAPH 2002.

[28] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[29] L. Williams. Casting curved shadows on curved surfaces. In *Seminal Graphics: Pioneering Efforts That Shaped the Field*, pages 51–55. ACM, New York, NY, 1998.

[30] M. Wimmer, D. Scherzer, and W. Purgathofer. Light space perspective shadow maps. In *Rendering Techniques 2004*, Eurographics Symposium on Rendering, pages 143–151, Norrköping, Sweden, June 2004. Eurographics Association.

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

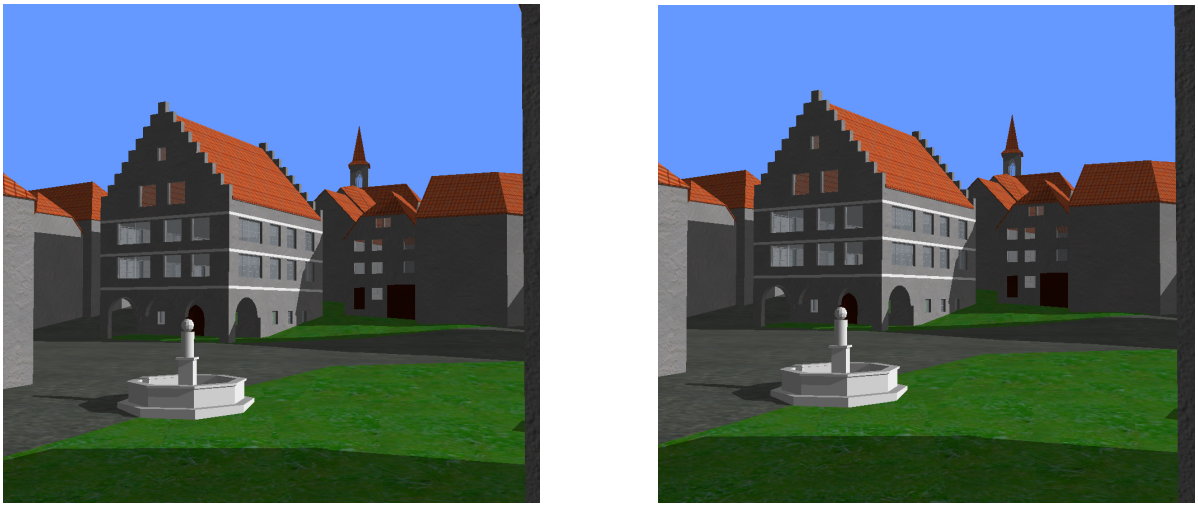

Figure 3: Comparison between a ray traced shadow rendered at 2 fps (left) and our method using a 1K × 1K conservative shadow map rendered at 7 fps (right).

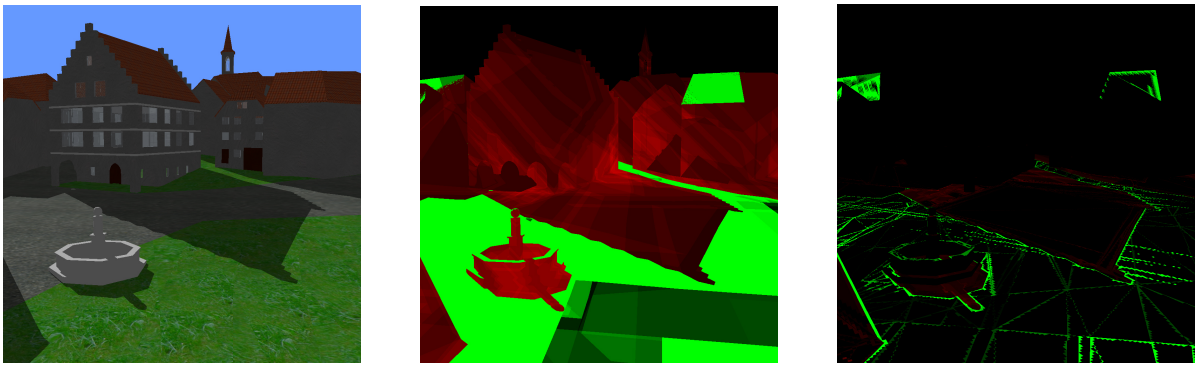

Figure 4: Visualization of the efficiency of our acceleration. Colour coding of the number of triangle intersection tests (the brighter the colour the more tests were made, red colour shows a view sample in shadow, green samples are lit). Left: The rendered image; centre: standard GPU ray tracing; right: our method.

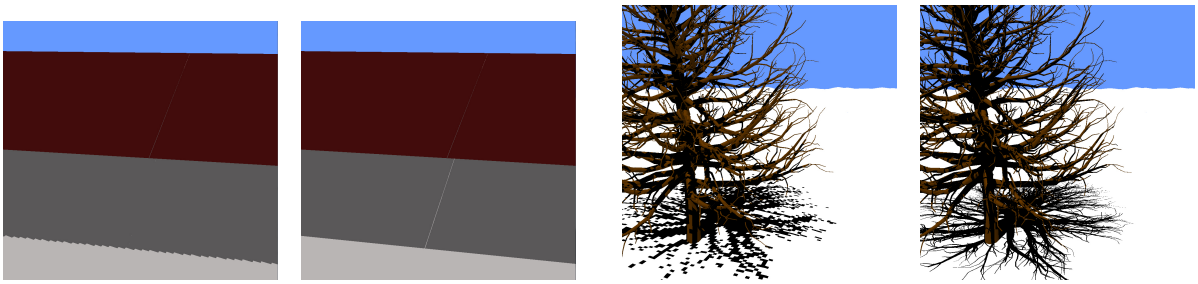

Figure 5: From left to right: For the left image a 1K × 1K shadow map was used. Note the gap between the faces in the left two pictures is missed completely by the map. The same picture rendered with a 1K × 1K conservative shadow map. The gap is clearly visible in the shadow. The tree rendered with a 1K × 1K shadow map at 45 fps. The same scene rendered with a conservative shadow map at 3 fps.