Sample-based Visibility for Soft Shadows Using Alias-free Shadow Maps

Erik Sintorn, Elmar Eisemann, Ulf Assarsson

EUROGRAPHICS 2008 / Steve Marschner and Michael Wimmer
(Guest Editors)

Volume 27 (2008), Number 4

Sample Based Visibility for Soft Shadows using Alias-free Shadow Maps

Erik Sintorn[1] Elmar Eisemann[2] Ulf Assarsson[1]

[1]Chalmers University of Technology, Sweden [2]ARTIS-INRIA/Grenoble University, France

Abstract

This paper introduces an accurate real-time soft shadow algorithm that uses sample based visibility. Initially, we present a GPU-based alias-free hard shadow map algorithm that typically requires only a single render pass from the light, in contrast to using depth peeling and one pass per layer. For closed objects, we also suppress the need for a bias. The method is extended to soft shadow sampling for an arbitrarily shaped area-/volumetric light source using 128-1024 light samples per screen pixel. The alias-free shadow map guarantees that the visibility is accurately sampled per screen-space pixel, even for arbitrarily shaped (e.g. non-planar) surfaces or solid objects. Another contribution is a smooth coherent shading model to avoid common light leakage near shadow borders due to normal interpolation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism Shadowing

1. Introduction

Shadows are not only important for realism, but also to provide visual cues indicating the relative locations of objects. Hard shadows appear only under directional light or point light sources, which are rare in nature. Therefore, their look seems often artificial whereas soft shadows are typically more realistic and eye-pleasing.

This paper presents a method for interactive soft shadow rendering from arbitrary area- or volumetric light sources. Contrary to most real-time algorithms, visibility is correctly sampled, which is de facto standard for high quality soft shadows and our method could replace raytraced solutions.

In many cases approximate solutions fail to convince [HLHS03]. A physically plausible computation for the irradiance at a point $\mathbf{p}$ is:

$$\int_S L(\mathbf{p},y)\, v(\mathbf{p},y)\, \frac{cos(\mathbf{p}-y,n_y)\, cos(y-\mathbf{p},n\mathbf{p})}{\pi||\mathbf{p}-y||^2}\, dy \quad (1)$$

where S is the source, L is the incoming irradiance at $\mathbf{p}$, v is the binary visibility function, and n_x the normal at x. Our approach could directly sample this equation, but to keep the computations down, we use the common approximation to separate lighting and visibility. The resulting integral of visibility is still challenging to approximate. The difference of this factored solution is subtle when receivers are at a certain distance from the source.

Our first contribution is a GPU-based alias free shadow map implementation. In contrast to previous solutions, we typically require only one render-pass from the lightposition. For closed (two-manifold) shadow casters, it is also more robust and does not require any depth bias. The second and main contribution of our paper is an efficient soft shadow algorithm. Arbitrary volumetric and textured light sources can be handled. Other contributions are an efficient sample list representation(Section 3.1), a smooth coherent shading model (see Section 4.3), temporal jittering (Section 7), and an efficient penumbrae determination 4.1.

2. Previous Work

An exhaustive presentation of previous work is out of the scope of this article, for this, we refer the reader to the surveys by Hasenfratz et al. [HLHS03] and Woo et al. [WPF90].

Shadow Maps [Wil78] and Shadow Volumes [Cro77] are the two most common algorithms to create hard shadows on arbitrary receivers. While shadow volumes are fill-rate demanding, shadow maps suffer from aliasing and require careful adjustment of a scene dependent depth bias. This

becomes necessary due to the limited depth precision and a different discretization when rasterizing from the eye and the light. Numerous solutions have been proposed, including better adaption of the shadow map [MT04, WSP04, LSO07], antialiasing [AMB*07, PCdON04], or suggestions to lower the bias-concerns [HN85, WE03, Woo92]. Alias-free shadow maps exist based on either software rendering [AL04], new hardware features (still not available) [JLBM05], or depth peeling, requiring one render pass per depth layer [Arv07]. Heckbert and Herf [HH97] combine hard shadows on a plane to create shadow textures. Soft Shadow Volumes [AAM03] is inspired from the shadow volume algorithm, but uses a wedge per silhouette edge, as seen from the light source center, to add a corresponding part of penumbra. Forest et al. [FBP06] introduced a method to improve the usually overestimated shadows, with a visibility buffer per silhouette to achieve more accurate occluder fusion. Laine et al. [LAA*05] used the soft shadow volumes to produce correctly sample based soft shadows for offline rendering.

Layered attenuation maps compute an approximate shadow by combining several shadow maps [ARHM00]. Image-based raytracing traces shadow rays through a set of depth images produced from a discrete set of sample points over the light source [ARHM00] or multilayer depth images including transparencies [XTP07].

A single depth map is insufficient to generally represent an occluder even for a non-pinhole camera (seeing *around* objects) like Mo and Wyman's [MW07], but can lead to plausible soft shadows as in [AHL*06]. Here, each depth sample is downprojected on a plane, where its corresponding occlusion is accumulated. Guennebaud et al. [GBP06] take a very similar approach. Instead of the downprojection, occluding pixels are searched in the depth map, which allows self-occlusion. In [GBP07], they improve search and quality by tracing occluder contours. Bala et al. [BWG03] also combine sparse sampling and discontinuity events to achieve interactive performance. Schwarz et al. [SS07] apply bit arithmetic to combine samples correctly, but need depth peeling for accurate shadows (see Figure 1). Kautz et al. [KLA04]

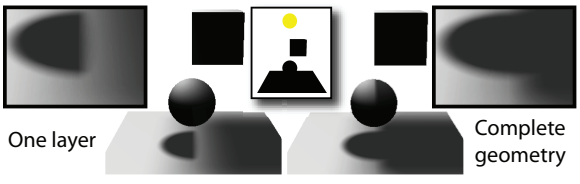

Figure 1: *Left: One depth layer & accurate shading. This case needs four and has an aligned face. Right: Our solution*

have a CPU determination of illumination using bit arithmetic. Each triangle of an LOD model is tested against each vertex. A fast approximation is presented in [ED06], where several scene slices are combined using a filtering process.

Our algorithm is most closely related to the work by Laine et al. [LAA*05] and Eisemann and Décoret [ED07]. Both compute accurately sampled visibility. Soft Shadow Volumes are combined with ray tracing in [LAA*05]. The latter is GPU-based, targets real-time purposes, and uses an overestimation of the penumbra, *the influence region* from triangles to project the occlusion onto a planar/height field receiver. We refer to this paper for compelling arguments why classic approximations often give insufficient result.

3. Alias Free Shadow Maps for Hard Shadows

Overview - Initially, the scene is rendered from the eye into a *view texture* to find points (*view-samples*) for which shadow computation is needed. These are foremost the visible points from the eye. Samples on back-facing geometry w.r.t. the light source can also be excluded since they self occlude (see Figure 2). The view-samples are then projected and stored in the shadow map (SM).

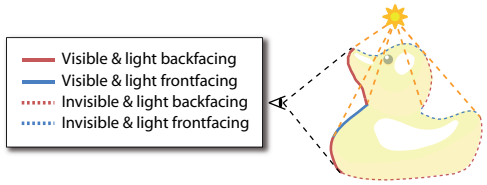

Figure 2: *Only visible and light front-facing view-samples (non-dashed blue) are stored in the shadow map. The world space coordinate and shading status of each sample is maintained in a local list at the shadow map pixel. For closed objects, only light back-facing triangles (red dashed and non-dashed) are used as shadow casters.*

Several of the view-samples can fall in the same pixel (see Figure 3 - left), which is the source of aliasing in standard shadow mapping. To avoid collisions, we store them in list's for each SM pixel (Section 3.1).

Next, we test whether each view-sample is lit or in shadow. For this, we render the scene from the light source as follows. For each triangle t, a fragment shader will be executed on the SM pixels that are partially intersected by t's projection. We achieve this by using conservative rasterization (Section 3.2). For each covered pixel, a fragment shader traverses the list of view-samples stored at this location and determines whether t hides the view-sample from the point light. All triangles are treated separately. After each pass, the shadow information (lit/unlit) is combined with the result for the previously tested geometry (Section 3.3).

Once all triangles are processed, the shadows are applied by rendering a screen covering quad in the observer's view, and each view-sample recovers its corresponding shadow value.

3.1. Constructing and storing the shadow map lists

Filling the shadow map lists with the view-samples uses CUDA's scattering capabilities. The process is memory compact and faster than stencil routing [MB07].

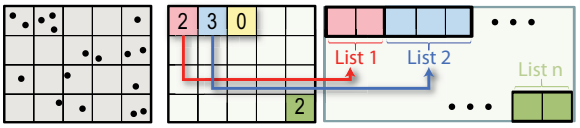

Figure 3: *Memory layout of the SM Lists: view-samples projected into the SM (left). An offset and length per pixel allow to access the sequentially stored lists (right).*

Our goal is to store in each pixel of the SM a *list length* s and a *list offset*. The list offset points into a global array, I_A (see Figure 3), from where all consecutive s elements correspond to the entries of the *local list*. With this information, one can iterate over the list elements of each SM pixel by reading successive elements in I_A. Further, each view-sample has an *offset* into I_A. It is stored in the view-texture and allows to lookup a value, associated to a view-sample. In the context of our shadow computation, each entry of I_A holds the view-sample's status information (lit/shadowed) as well as its world position.

Construction - To obtain this structure, we intialize the local lists' lengths in the SM to zero. Then, each view-sample, $\mathbf{p}_{cam}$, is projected into the SM, to a position $\mathbf{p}_{SM}$. The current local list size s at $\mathbf{p}_{SM}$ is read and incremented instantaneously using the atomicInc() instruction in CUDA. The read value is stored at $\mathbf{p}_{cam}$'s position in the view texture. This process associates a unique index i to each view-sample that can be used as an offset into the local list it projects in. At the same time, on the SM side, this counter ultimately indicates the number of elements in the pixel's local list.

To concatenate the local lists in the global array I_A, each one needs a particular offset. For this, a running sum scan [HSO07] is evaluated on the list length map. This process interprets the 2D texture as a 1D array (line by line) and stores in each pixel the sum of all proceeding values. These are exactly the offsets that represent the start of each list in the global array I_A.

To directly access the data in I_A for a view-sample, we find its offset by adding the list offset of the list at $\mathbf{p}_{PM}$ and the view-samples index i. This final offset will not change until the end of the frame and we store it in the view-texture.

Storage - In practice, instead of having one array I_A with heterogenous data (world position and lit/shadowed state) we store them in separate arrays using the same offsets we derived for I_A. This allows us to pack the shadow data densely in single bits. This is beneficial because even though the world position remains the same throughout one frame, the shadow state will be updated for each processed triangle and it becomes possible to use the card's bitwise blending operation for this. Nowadays, eight render targets and 32-bit channels allow us to update 1024 view-sample shadow state values per list and render pass. If more view-samples are in a list, blocks of 1024 samples are treated, and multi-pass rendering is used until all view-samples have been processed. However, in practice, even 128 samples prove enough in most cases.

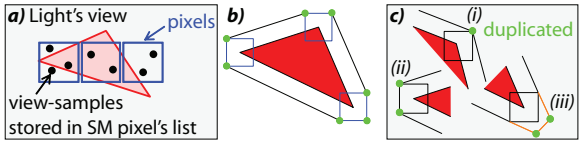

Figure 4: *a) Without conservative rasterization, only pixels whose center is inside the triangle are rasterized. Since sample points can lie anywhere in the pixel, conservative rasterization is required. b) Conservative rasterization is done in the geometry shader, by computing the convex hull of the triangle's vertex extension to a pixel-sized quad [HAMO05]. c) Three cases can occur at corners of the convex hull. Performance is improved significantly if the area is approximated with only two vertices at each corner. For case iii) edges are extended by the width of a pixel.*

3.2. Conservative rasterization

Having lists of view-samples in each pixel of the SM, we next rasterize the scene's triangles one by one into the SM. For each pixel intersected by a triangle t, the local list is traversed and each view-sample is tested for occlusion by t with respect to the point light source. The rasterization needs to be conservative (Figure 4). Otherwise, if the center of the list pixel is not covered, no fragment would be produced and thus the computation will not be applied to the list's elements (which in turn, could lie in the partially covered region). We use a version of Hasselgren and Akenine-Möllers's algorithm [HAMO05] executed on the geometry shader, but we always output a fixed number of 6 vertices (4 triangles) instead of up to 9 vertices (7 triangles) since that turned out to be significantly faster. Our tradeoff is that case (iii), shown in Figure 4 c), can be overly conservative by one pixel, which has no effect on correctness in our situation.

3.3. Evaluating visibility

A given view-sample is tested against a triangle t by computing distances to the planes given by t's supporting plane and the pyramid defined by t and the light source. This has the advantage that we can compute the plane equations once in the geometry shader and then perform rapid tests in the fragment shader.

Avoiding bias - The algorithm has a nice property: For two-manifold (closed) shadow casters, either the light back-facing surfaces or the light front-facing surfaces suffice to determine the shadows. This fact can be utilized to avoid incorrect self shadowing and the corresponding problem of surface acne, without using the classic scene dependent

depth bias [Wil78], triangle ID's [HN85], or depth values inside closed objects [Woo92,WE03]. Light back-facing view-samples do not need to be transferred into the SM lists, since they always will be in shadow by the shading. Thus, selecting only the light back-facing triangles as shadow casters eliminates self shadowing. A triangle will never be both a shadow caster and receiver. This also avoids numerical imprecision problems from floating point transformations between coordinate systems [AL04, Arv07]. Theoretically, problems might remain where front- and back-faces touch, but in practice we did not observe such artifacts. No example in the paper uses any depth bias (e.g. see left part of Figure 14 that renders in 66 fps). To avoid even this potential problem, an epsilon value could be introduced to offset the triangle's supporting plane slightly. There are two possible offsets (toward the light and away). Due to our culling strategy, self shadowing does not occur and both are valid. One favors light-, the other shadow leaks. The latter are less disturbing than light leaks (a bright spot in the dark is more obvious than a slightly extended shadow). Moving the supporting plane towards the light is thus the better choice.

A similar trick is actually often used for standard shadow maps to ameliorate the problem of surface acne. For closed shadow casters, only the back facing triangles with respect to the light source are rendered into the shadow map. However, since standard shadow maps are not alias-free, artifacts will still appear near all silhouettes, and careful adjustment of a bias and resolution is required to suppress the problems.

3.4. Short Discussion of Alias Free Shadow Maps

Our method shows many similarities with the work by Johnson et al. [JLBM05]. The main differences are that Johnson et al. proposed hardware extensions to treat more general cases than just hard shadows. Their algorithm makes heavy use of dynamically allocated linked lists. Besides the adapted memory structures, a supplementary processing unit and waiting queues were proposed to synchronize work. We represent lists differently, making the access faster in our context (linked list iterators need to follow pointers). They are implementable on current hardware and can be easily parallelized (only the cheap atomicIncr() operation needs synchronization). No dynamic reallocation is required. Each sample's final position is given by an offset in a global array, which is of constant length (tightly bound by the view-texture resolution). We compute shadows only where needed. Conservative rasterization improves upon previous work. We introduced a way to avoid depth fighting and decrease the number of caster triangles. In a 512^2 viewport, our method is usually only ≈ 3 times slower than SM (8192^2).

4. Soft Shadows

Only two steps of the previous SM algorithm need to be modified for soft shadows by arbitrary volumetric sources. First, the conservative rasterization needs to cover the entire umbra and penumbra region of each shadow casting triangle, the so-called *influence region*. Second, the fragment shader testing for occlusion needs to consider multiple light samples. This is solved using a 3D-texture as explained below in Section 4.2. This pipeline is similar to [ED07]. The major differences are that we treat arbitrary receivers with our lists, compute tight influence regions, give an efficient way to apply conservative rasterization to them, allow volumetric light sources, and we introduce a temporal anti-aliasing.

4.1. Computing triangle's influence region

A point $\mathbf{p}$ can only be shaded by a triangle t, if it lies in the union of t's projections from all light source points onto a plane passing through $\mathbf{p}$. The same holds, for a plane further away (as seen from the light's center). Therefore, conservative choices for an arbitrary point of the scene include the far plane (see Figure 5), a floor plane (if the scene exhibits one), or a plane at a distance of the furthest view-sample. As can be seen in Figure 5 (right), when the distance between the light center and far plane grows, the ratio between the size of the influence region and the base of the light frustum is asymptotically bound by the ratio of their respective angles (green and blue). Thus, the size of the influence region is typically more dependant on the triangle's distance to the light than the choice of the far-plane.

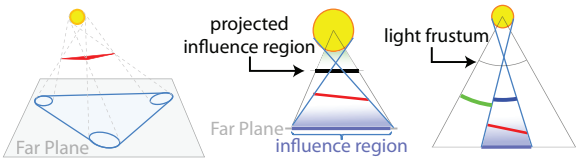

Figure 5: *Left: a triangle's influence region at a far plane. Middle: affected shadow map pixels when rasterizing the influence region. Right: the influence region size as seen from the light, is asymptotically bound by the ratio between the blue and green angle.*

In the case of a planar source, it is relatively simple to calculate the influence region from a shadow casting triangle. Let us first concentrate on a spherical light source. Here, the influence region on a plane is more complex. It is the convex hull of the source's projection through the vertices which form ellipses in the plane.

Previous work approximated influence regions coarsely with one bounding quad [ED07] and was restricted to planar sources and receivers. We provide a solution that allows arbitrarily tight bounding regions for spherical sources and apply it to arbitrary receivers (see Figure 5-left). For each triangle vertex v, we compute a regular bounding polygon of a desired degree (e.g. a hexagon for 6 points) for the light's silhouette as seen from v. This silhouette is a circle whose radius and position can be easily inferred and thus the construction of a bounding polygon $\mathcal{P}_v$ is direct. We then project $\mathcal{P}_v$ through v onto the far plane. This is done by computing the intersection of the lines through v and each vertex

of $\mathcal{P}_v$ with the desired far plane. These intersection points form bounding polygons of the ellipses and their convex hull bounds the influence region. The convex hull is computed with Graham's scan [Gra72]. The method is $O(n)$ if all points are in clockwise angular order. This is easy to assure for the points of each ellipse, but these point groups do not share a common center. Fortunately, merging them is possible in $O(n)$ time. This makes the full convex hull computation very rapid. In practice, we chose hexagons and CUDA was found to be up to twice as fast as geometry shaders.

Conservative rasterization of the influence region is done by adapting the rules of Figure 4. We only output one vertex in case (i) and for case (ii) if the angle between the edges is $< 90^\circ$ their intersection point is used to output only one vertex. To ensure conservative coverage of the influence regions for an arbitrary volumetric light source, we first approximate it by an ellipsoid. A *virtual* scaling of the entire scene allows us to reestablish a spherical light source for which the previous algorithm can be used.

4.2. Sampling Visibility of Volumetric Sources

For hard shadows, we used a single bit to represent the visibility of the point light source. To compute soft shadows, we associate a *bitmask* to each view-sample, where each bit corresponds to the visibility of one light sample. This means that 8 MRTs allow us to output 128 light samples for 8 view-samples in a single pass. Multipass rendering is used until all view samples are computed.

To sample visibility in the influence region, we use a light sample dependent lookup table LU. LU takes a 3D-plane as input and returns a bit pattern corresponding to the light-samples behind the plane (Figure 6 top-left). Combining the bitmasks of the planes defined by the view-sample and the edges of the triangle t via an AND operation establishes the correct visibility with respect to t (Figure 6 top-right). This can be seen as an extension of the method by Eisemann and Décoret [ED07] to handle non-planar light sources and shadow receivers. To accumulate the results, we use, as before for the hard shadows, the blending capacities of graphics hardware (namely the bitwise OR operation).

We can represent LU with a 3D texture by describing the plane with two angles and a distance.

To minimize discretization errors, we choose LU's size relative to the number of samples, typically 128^3 per 3D-texture. It leads to good shadow quality with moderate memory cost. If more samples are needed, we can precompute additional k LU-textures for $k \times 128$ different sample locations. As pointed out in [ED07], for symmetric sources, sample textures can be reused by rotating the normal around the symmetry axes. This can be done on a per pixel basis to achieve jittering at almost no cost. We further propose temporal jittering where samples change in each time frame. The solution converges rapidly to a high quality image via accumulation, if the scene is static and the camera does not move (see Figure 7). As pointed out in [SJW07], lower quality is acceptable for moving cameras and for light designers it is useful to have a fast feedback concerning shadows.

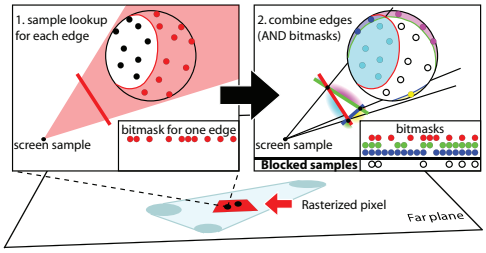

Figure 6: *Visibility is tested for each view-sample in each rasterized fragment (bottom). A texture lookup returns a bitmask indicating the light samples behind the plane defined by an edge and the view-sample (left). The occluded light samples are identified by AND:ing these values (right). The light samples can be arbitrarily located.*

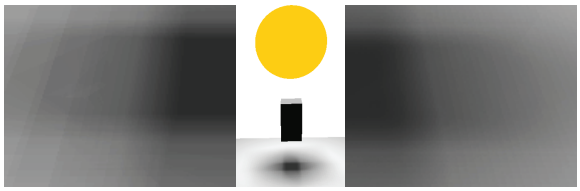

Figure 7: *Left: No jittering. Right: jittering and frame accumulation, after a second. Both: 128 light samples.*

4.3. Optimizations

Parametrization - To keep view-sample lists short, existing techniques to maximize the SM resolution (e.g. [MT04]) can be used to optimize the sample repartition. A good solution, we employed, is to fit the frustum to a minimal 2D-axis aligned bounding box of the projected view-samples, and we readapt the light frustum between each pass of our algorithm. The parametrization lead to speed-ups of around 30% in scenes like one in Figure 9. The same step also computes the maximum view-sample distance to the light source. This allows us to fit an optimal far plane. This keeps influence regions smaller, but its influence on performance is weaker.

Restrict computations - With the stencil buffer we can block all pixels with empty lists. Further, penumbrae can only be cast from silhouette edges. Here, *silhouette* means that there is a point on the light for which the edge is a silhouette. Their detection is done in the geometry shader or using CUDA by computing the relative position of the light w.r.t. the planes defined by the adjacent triangles [LAA*05]. The edges' influence region is then rendered into the stencil of the SM. This blocks unnecessary computations in the umbra/lit region, where a hard shadow from any sample point is enough. The same reasoning applies for triangles.

Figure 8: *With standard vertex normals, light leaks in back-faces (red line). We bend normals for coherent shading.*

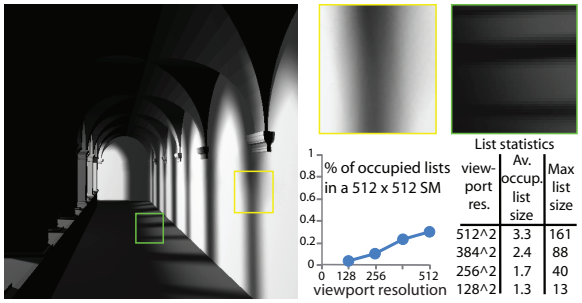

Figure 9: *Sponza (73k tris), this view: 20k shadow casting tris, 256 light samples. Inlay: SM list length information.*

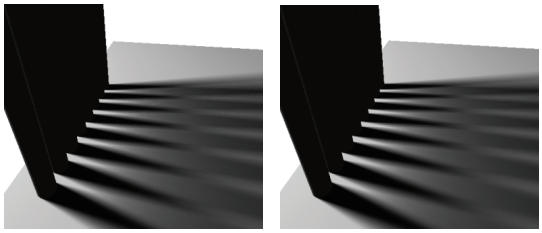

Figure 10: *Shadow quality: 128 samples/512 samples*

If models are watertight, triangles can be eliminated based on conservative back-face culling [ED07], also we can cut off the influence region based on the triangle's supporting plane [LLA06], this integrates well in our convex hull computation. Both measures made the soft shadow pass win up to 50%. Another possibility would be to apply a perceptual measure, e.g. based on the frequency of standard lighting and texturing to choose only a view-sample subset. A reconstruction step yields the final image. A first step in the context of shadows was presented in [GBP07]. We did not rely on this for our timings to show the actual cost.

Shading - The standard Gouraud/Phong shading models use per vertex normals for a smooth interpolant. This might lead to *light bleeding* into back-facing triangles (see figure 8). It introduces artifacts at light silhouette edges because accurate shadows lead to self-shadowing and create a strong discontinuity. This is particularly disturbing for hard shadows. To avoid this behavior, we bend vertex normals with respect to the adjacent faces. The geometry shader input (triangles with adjacency) does not provide all triangles incident to a vertex. Nevertheless, passing face normals in a texture and adjacent face indices as texture coordinates allows a rapid GPU evaluation. In case the number of neighbors is small, normals can be sent directly through texture coordinates. Interpolating the minimum illumination based on adjacent face normals delivers a smooth result that ensures blackness of all light back-facing triangles. Formally, we compute at each vertex: $l_{bend} = min_{F \in adj(v)} dot(l, n_F)$ where n_F is the normal of face F, l the light vector, and $adj(v)$ the adjacent faces to v. Per fragment, we can securely and continuously blend back to Phong illumination l_f, using: $\alpha := min(1, max(c\, l_{bend}, 0))$, $\alpha\, l_f + (1-\alpha)\, l_{bend}$. $c := 3$ works well. Soft shadows are smoother and thus bent normals with respect to the source's center are often sufficient. The definition could be extended to the minimum with respect to the entire source or a bounding volume (e.g. a cube).

5. Results

All measurements were performed using 512^2 resolution and a Geforce 8800GTS-512. Five test scenes where used. A *fairy* of 734 triangles, *columns* of 840 triangles, a *hairball* of 44ktris, a *torus-knot* of varying tessellation degrees, the *Sponza Atrium* of 73k triangles, and very high polygon scenes like a ring in grass of 379ktris and others shown in Figure 15. Figure 10 stresses the sampling quality for a large penumbra. The visual difference between 512 and 1024 samples was negligible in this case. Figure 12 shows a linear dependence when varying the light radius. For the case of increasing samples (Figure 13) the rendering cost roughly only doubles from 128 to 1024. Our method is about an order of magnitude faster than using a corresponding amount of shadow maps, which would be an alternative for producing sample based soft shadows on arbitrary receivers, but they exhibit aliasing artifacts. It works robustly even for complicated scenes like the hairball (Figure 14).

Figure 11 uses 128 light samples and varies the tessellation degree. In this case, the algorithm behaves mostly linear in the number of triangles. Nevertheless our optimizations (section 4) to transfer only visible view-samples and eliminate empty lists (figure 9 shows typical statistics), as well as the restriction to penumbra regions lead to important gains. The rightmost image in Figure 15 shows a scene from [LAA*05]. The computation time using Laine et al.'s algorithm reported 75 seconds for a 512^2 image of this scene. In contrast, we achieve a frame-rate of almost constant 0.65 Hz independently of the viewpoint, which corresponds to a speed-up of 48.75. Figure 9 shows the Sponza Atrium, running in 2.6 fps with 256 light samples, compared to 9 seconds for Overbeck et al. [ORM07]. We thus provide a 23.4 times speedup. Increasing the number of samples in this scene did not lead to noticeable quality improvements, but in general, Overbeck et al.'s solution is exact, not sampled like ours.

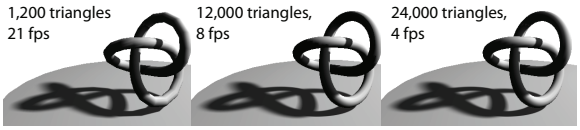

Figure 11: *Varying number of triangles.*

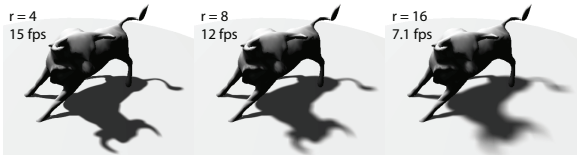

Figure 12: *Varying light radius r. 256 samples. 6K tris.*

6. Conclusion and Future work

We presented an accurate solution to the general soft shadow sampling problem. Our soft-shadow algorithm outperforms Soft Shadow Volumes for ray tracing by 1-2 orders of magnitude, and shadow rays with 2-3 orders and has similar image quality. Our alias free shadow maps, have comparable performance to [LSO07], but the implementation effort of our solution is much lower and has virtually *infinite* resolution.

Approximate transparency is possible by multiplying the transparency values of shadow casting triangles and weighting by the percentage of coverage. For correct transparency, each sample would need to store one RGBA-value per light sample, which requires more bits than currently available.

We considered using silhouette edges to integrate visibility for each view-sample [LAA*05], but then counters (instead of bits) are needed for each light sample, reducing their number. Potential problems are inaccurate depth complexity and the limited counters, but transparency can be handled gracefully. Concurrently, such a method was published [FBP08].

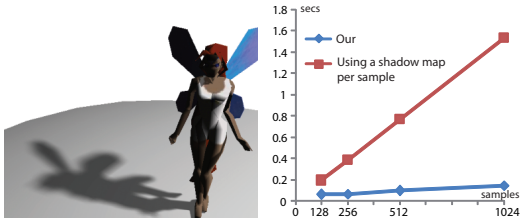

Figure 13: *Performance comparison vs. shadow maps.*

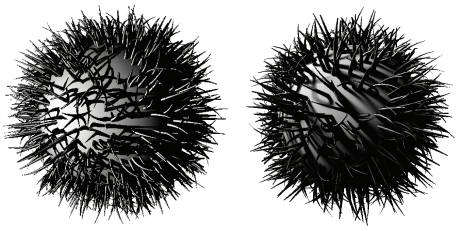

Figure 14: *Hard and soft shadows (44ktris).*

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Trans. Graph. 22*, 3 (2003).

[AHL*06] ATTY L., HOLZSCHUCH N., LAPIERRE M., HASENFRATZ J.-M., HANSEN C., SILLION F.: Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum 25*, 4 (2006).

[AL04] AILA T., LAINE S.: Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering* (2004), Eurographics Association.

[AMB*07] ANNEN T., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Convolution shadow maps. In *Proceedings of Eurographics Symposium on Rendering* (2007), vol. 18, Eurographics.

[ARHM00] AGRAWALA M., RAMAMOORTHI R., HEIRICH A., MOLL L.: Efficient image-based methods for rendering soft shadows. In *Proceedings of SIGGRAPH* (2000), Annual Conference Series, ACM SIGGRAPH.

[Arv07] ARVO J.: Alias-free shadow maps using graphics hardware. *journal of graphics tools 12*, 1 (2007).

[BWG03] BALA K., WALTER B., GREENBERG D. P.: Combining edges and points for interactive high-quality rendering. *ACM Transactions on Graphics 22*, 3 (2003).

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *Computer Graphics (SIGGRAPH 1977) 11*, 3 (1977).

[ED06] EISEMANN E., DÉCORET X.: Plausible image based soft shadows using occlusion textures. In *Proceedings of SIBGRAPI* (2006), IEEE Computer Society.

[ED07] EISEMANN E., DÉCORET X.: Visibility sampling on gpu and applications. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3 (2007).

[FBP06] FOREST V., BARTHE L., PAULIN M.: Realistic soft shadows by penumbra-wedges blending. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/Eurographics symposium on Graphics hardware* (2006), ACM.

[FBP08] FOREST V., BARTHE L., PAULIN M.: Accurate Shadows by Depth Complexity Sampling. *Computer Graphics Forum (Proceedings of Eurographics)* (2008).

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Real-time soft shadow mapping by backprojection. In *Eurographics Symposium on Rendering, Nicosia, Cyprus, 26/06/06-28/06/06* (2006), Eurographics.

[GBP07] GUENNEBAUD G., BARTHE L., PAULIN M.: High-Quality Adaptive Soft Shadow Mapping. *Computer Graphics Forum, Eurographics 2007 proceedings* (2007).

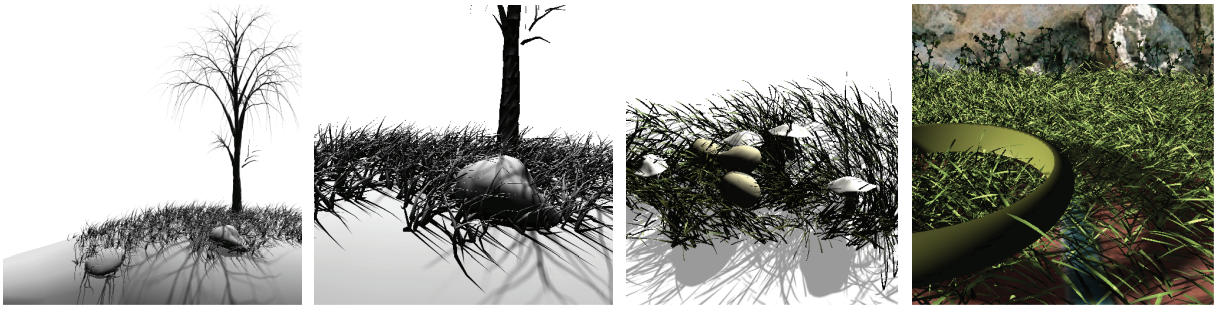

Figure 15: *Left to right: 90k triangles (7 fps), zoomed (4.5 fps), 70k triangles (4.8 fps), ring scene: 379k triangles (0.65 fps).*

[Gra72] GRAHAM R. L.: An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters 1* (1972).

[HAMO05] HASSELGREN J., AKENINE-MÖLLER T., OHLSSON L.: Conservative rasterization on the gpu. In *GPU Gems 2* (2005), Adison-Wesley.

[HH97] HECKBERT P. S., HERF M.: *Simulating Soft Shadows with Graphics Hardware*. Tech. Rep. CMU-CS-97-104, Carnegie Mellon University, 1997.

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22*, 4 (2003). State-of-the-Art Reviews.

[HN85] HOURCADE J.-C., NICOLAS A.: Algorithms for antialiased cast shadows. *Computer & Graphics* (1985).

[HSO07] HARRIS M., SENGUPTA S., OWENS J. D.: Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*. Addison-Wesley, 2007, ch. 39.

[JLBM05] JOHNSON G. S., LEE J., BURNS C. A., MARK W. R.: The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph. 24*, 4 (2005).

[KLA04] KAUTZ J., LEHTINEN J., AILA T.: Hemispherical rasterization for self-shadowing of dynamic objects. In *Proceedings of Eurographics Symposium on Rendering* (2004), Eurographics Association.

[LAA*05] LAINE S., AILA T., ASSARSSON U., LEHTINEN J., AKENINE-MÖLLER T.: Soft shadow volumes for ray tracing. *ACM Transactions on Graphics 24*, 3 (2005).

[LLA06] LEHTINEN J., LAINE S., AILA T.: An improved physically-based soft shadow volume algorithm. *Computer Graphics Forum 25*, 3 (2006).

[LSO07] LEFOHN A. E., SENGUPTA S., OWENS J. D.: Resolution matched shadow maps. *ACM Transactions on 26*, 4 (2007).

[MB07] MYERS K., BAVOIL L.: Stencil routed a-buffer. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches* (2007), ACM.

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the 2nd EG Symposium on Rendering* (2004), Springer Computer Science, Eurographics Association.

[MW07] MO Q., WYMAN C.: Interactive backprojected soft shadows with an occlusion camera shadow map. In *SIGGRAPH Posters* (2007), ACM.

[ORM07] OVERBECK R., RAMAMOORTHI R., MARK W. R.: A Real-time Beam Tracer with Application to Exact Soft Shadows . In *Rendering Techniques* (2007), Eurographics Association.

[PCdON04] PAGOT C. A., COMBA J. L. D., DE OLIVEIRA NETO M. M.: Multiple-depth shadow maps. In *Proceedings of SIBGRAPI* (2004), IEEE Computer Society.

[SJW07] SCHERZER D., JESCHKE S., WIMMER M.: Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)* (2007), Eurographics Association.

[SS07] SCHWARZ M., STAMMINGER M.: Bitmask soft shadows. *Computer Graphics Forum 26*, 3 (2007).

[WE03] WEISKOPF D., ERTL T.: Shadow Mapping Based on Dual Depth Layers. In *Proceedings of Eurographics '03 Short Papers* (2003).

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH 1978) 12*, 3 (1978).

[Woo92] WOO A.: The shadow depth map revisited. In *Graphics Gems III*. Academic Press Professional, Inc., 1992.

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

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proceedings of the 2nd EG Symposium on Rendering* (2004), Springer Computer Science, Eurographics Association.

[XTP07] XIE F., TABELLION E., PEARCE A.: Soft Shadows by Ray Tracing Multilayer Transparent Shadow Maps . In *Rendering Techniques* (2007), Eurographics Association.