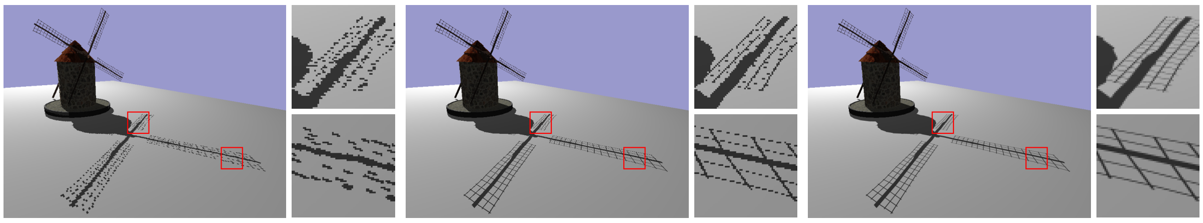
Pacific Graphics 2009
S. Lee, D. Lischinski, and Y. Yu
(Guest Editors)

Volume 28 (2009), Number 7

Fast, Sub-pixel Antialiased Shadow Maps

Minghao Pan, Rui Wang[†], Weifeng Chen, Kun Zhou, Hujun Bao

State Key Lab of CAD&CG, Zhejiang University



(a) Conventional shadow maps (b) Alias-free shadow maps, 49 FPS (c) Our method at 32× antialising, 41 FPS

Figure 1: *Shadows comparison with image resolution at 800×600.*

Abstract

Solving aliasing artifacts is an essential problem in shadow mapping approaches. Many works have been proposed, however, most of them focused on removing the texel-level aliasing that results from the limited resolution of shadow maps. Little work has been done to solve the pixel-level shadow aliasing that is produced by the rasterization on the screen plane. In this paper, we propose a fast, sub-pixel antialiased shadowing algorithm to solve the pixel aliasing problem. Our work is based on the alias-free shadow maps, which is capable of computing accurate per-pixel shadow, and only incurs little cost to extend to sub-pixel accuracy. Instead of direct supersampling the screen space, we take facets to approximate pixels in shadow testing. The shadowed area of one facet is rapidly evaluated by projecting blocker geometry onto a supersampled 2D occlusion mask with bitmasks fusion. It provides a sub-pixel occlusion sampling so as to capture fine shadow details and features. Furthermore, we introduce the silhouette mask map that limits visibility evaluation to pixels only on the silhouette, which greatly reduces the computation cost. Our algorithm runs entirely on the GPU, achieving real-time performance and is an order of magnitude faster than the brute-force supersampling method to produce comparable 32× antialiased shadows.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Color, shading, shadowing, and texture

1. Introduction

Shadows are an essential element in computer-generated scenes. Shadow maps [Wil78], one of the most popular shadow algorithms, has been widely used due to its simplicity and efficiency. The standard shadow maps algorithm takes a two-pass process that first rasterizes the scene from light source to generate a depth map and then rasterizes the screen-space to take per-pixel shadow test. However, with simplified shadow computation, these two rasterizations produce severe aliasing artifacts along shadow boundaries. Respectively, aliasing artifacts can be classified into two levels:

- **Texel-level aliasing.** Due to the limited resolution of the shadow map, the depth values stored in it are merely rough approximations of the scene. If the map's resolution is insufficient, the resulting shadow will have jagged boundaries as they are occluded by the discretized texels but not the original geometry. (Figure 1(a))

- **Pixel-level aliasing.** On the other side, rasterization on the screen plane also produces aliasing artifacts. As only a single sample is taken from each pixel for shadow test, the shadow state of the pixel would either be 0, as shadowed or 1, as lit. (Figure 1(b)). This kind of aliasing is more obvious under dynamic lights or the shadow boundary is almost parallel to the axes.

[†] Corresponding author: Rui Wang, rwang@cad.zju.edu.cn

Many algorithms have been proposed to solve the texel-level aliasing problem. Some of them follow the rasterization way but increase the texel-pixel match sophistically, such as the *warping* methods that transform the shadow map to another space for a better match between the light and camera sampling rates [SD02, WSP04, LGMM07, BLM08] or *partitioning* methods that partition the depth map into multiple parts adapted to local sampling rate [ZSXL06, CG04, LTYM06, FFB01, GW07, LSO07]. Some other methods bypass the hardware rasterization process to avoid the aliasing problem. They either use software rendering [AL04] or require new hardware features [JLBM05]. Recently, along with the advances in graphic hardware, [SEA08] presented a GPU-based method of aliasing-free shadow maps, which guarantees that the visibility is accurately computed per screen-space pixel.

On the other side, little work has been done to produce sub-pixel antialiased shadows. The brute-force approach is to supersample the screen space directly and take shadow test for each sub-pixel sample. However, for high-quality shadows, such as $64\times$ or $128\times$ sub-pixel antialiased shadows, mass of samples will lead to poor performance and cannot run in real-time.

In this paper, we propose a fast, sub-pixel antialiased shadowing algorithm to solve the pixel aliasing problem (Figure 1(c)). Our work is based on the alias-free shadow maps and extends it to achieve sub-pixel accuracy. Instead of direct supersampling the screen space, our algorithm uses facets in 3D space to approximate pixels. In shadow testing, blockers potentially shadowing facets are projected onto the screen plane and the occluded area of each pixel is evaluated by 2D occlusion masks fusion from a lookup table. Furthermore, by introducing the silhouette mask map, which limits visibility evaluation to pixels only on the silhouette, the computation cost is greatly reduced. Compared with standard alias-free shadow maps, our work incurs little cost and provides a sub-pixel occlusion sampling capable of capturing fine shadow features and details. We demonstrate that our approach is capable of running in real-time and can be an order of magnitude faster than the brute-force supersampling method to produce comparable sub-pixel antialiased shadows.

2. Related work

Ever since shadow maps was proposed, many works have been proposed to solve the problem of aliasing. However, most of them only focus on removing the texel-level aliasing.

As insufficient resolution of light view rasterization is the fundamental reason that produces texel-level aliasing, it is common to increase the texel-pixel match to solve the aliasing problem. The mostly used approaches include *warping* and *partitioning*. [SD02] introduced perspective shadow maps (PSMs) method that tries to remove perspective aliasing by using camera's perspective transform. PSMs was later refined by [MT04, WSP04, CG04] using other transforms. However, these reparameterization methods only deal with perspective aliasing. In addition, as only a global warping function is applied, complex scenes with large depth range may not be well-handled. Logarithmic shadow maps [LGMM07, BLM08] produces lower aliasing errors but requires modification to the current hardware. *Partitioning* algorithms divide the shadow map into different parts, each of which is expected to match the local sampling rate better in the camera view. [LTYM06] partitioned the shadow map according to frustum faces and [ZSXL06] did it in a similar way but along the z-axis of the frustum. Adaptive shadow maps (ASM) [FFB01] tries to solve the aliasing problem by providing additional local shadow maps in high aliasing error regions. It was later improved with a hardware implementation [LSO07]. Usually, partitioning methods require multiple rendering passes hence are relatively slow.

[SJW07] employed a history buffer to reuse information of previous frames, which potentially increased the resolution of the shadow map. From the observation that aliasing artifacts only exist along shadow boundary, shadow silhouette maps [SCH03] approximates the silhouette using piecewise linear segments. However, this method is well-limited by the shadow map resolution and may miss small features due to the fact that each texel contains at most one shadow edge point. [CD04] also utilizes this property by presenting a hybrid algorithm that performs shadow-volume computation at boundary pixels and uses a conventional shadow map elsewhere, therefore combining the strengths of both algorithms. [BWG03] also extracts silhouette edge to produce alias-free shadow boundary in their edge-and-point rendering technique.

Alias-free shadow maps [AL04] and irregular z-buffer [JLBM05] completely avoid aliasing in rasterization by projecting view-pixels to the light view and rasterizing the scene to these points instead of regular grids. However, such rasterization is not supported by current hardware and is inefficient. Along with the advances in hardware, [SEA08] presented a GPU-based alias-free shadow map method and extended it to soft shadows. Although it is guaranteed that the visibility is accurately computed per screen-space pixel, the technique still suffers from pixel-level aliasing for hard shadows.

To remove pixel-level aliasing, currently the only way is the supersampling method that rasterizes and shades much more samples than the standard resolution hence is rather slow. Multisampling [gls02] accelerates this process by shading only once for multiple samples in the same pixel. Therefore, it is not capable of the antialiased shadow computation, which requires explicit shadow test for each sample.

Instead of computing accurate per-pixel shadows, some other approaches take filtering strategies to solve the jagged boundary aliasing. Early work by [RSC87] known as percentage closer filtering filters depth maps to get blurred

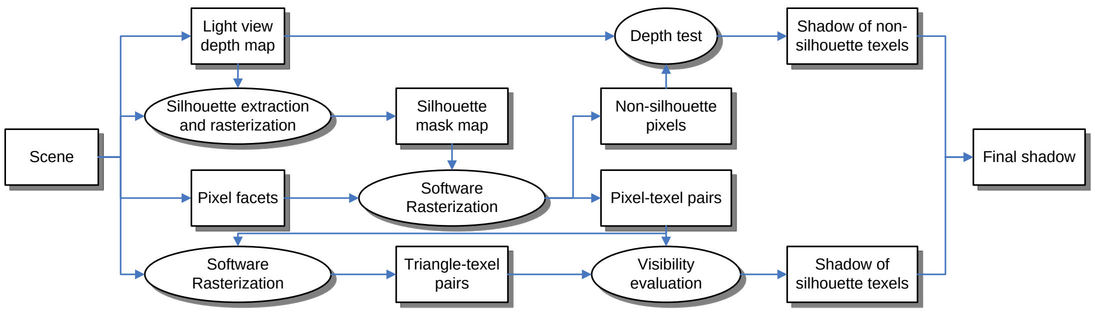


Figure 2: *Pipeline of our algorithm. Boxes denote data and ellipses denote algorithms.*

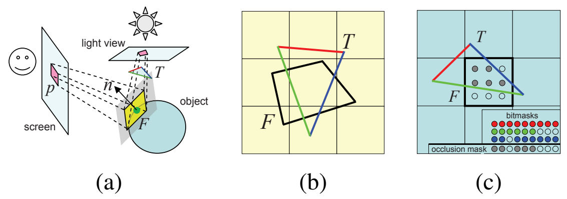


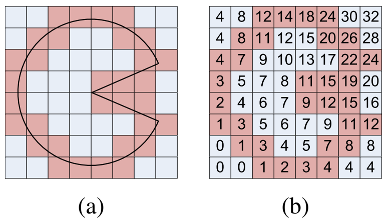
Figure 3: *(a) Facet approximation of a pixel. (b) A facet and a triangle's projections on the light view. (c) Their projections on the screen. Gray dots represent occluded samples.*

shadow boundaries. Recently, [AMS*07, AMB*08] use other filterable approximated shadow test functions for anti-aliasing shadow boundary. However, these maps are still in a discretized representation and may not be able to produce a satisfying result under low resolution. These methods focus on producing shadows that are plausible but not accurate.

3. Algorithm

3.1. Overview

In alias-free shadow maps or irregular z-buffer methods, pixels are regarded as 3D points to perform shadow test. Consequently, the shadow state of the pixel would either be 0 or 1, which can miss fine shadow details or features. To achieve sub-pixel accuracy, it is necessary to record the part of occluded area within a pixel, which is not easy as the underlying geometry covered by a pixel can be very complex. In our method, we use a 3D element, the *facet*, to approximate each pixel. A facet is a small piece of quadrangle on the tangent plane determined by a pixel's position and normal (Figure 3(a)). Under this approximation, for each blocker that potentially occludes a facet (Figure 3(b)), the occluded area is computed by projecting the blocker to the tangent plane and then to the screen plane. On the screen plane, occlusion masks that consists of multiple samples are used to record the sub-pixel shadows. These multiple samples are uniformly distributed in pixel's grid and represented by a bitmask such that each set bit indicates one sample's shadow state. The area a blocker triangle occludes inside each pixel is the intersection of several half-planes (section 3.4). For a discrete set of half-planes, the value of bits in the occlusion mask can be precomputed and stored in a lookup table. Accordingly, the occlusion mask of a pixel with respect to a blocker triangle is obtained by combining the bitmasks of the triangle's edges via bit operations (Figure 3(c)).



4	8	12	14	18	24	30	32
4	8	11	12	15	20	26	28
4	7	9	10	13	17	22	24
3	5	7	8	11	15	19	20
2	4	6	7	9	12	15	16
1	3	5	6	7	9	11	12
0	1	3	4	5	7	8	8
0	0	1	2	3	4	4	4

Figure 4: *(a) The silhouette mask map marks the silhouette texels (in red) which intersects at least one silhouette edge. (b) The summed area table built from the map.*

From the fact that aliasing only exists along shadow boundaries, we do not need to compute sub-pixel antialiased shadows for pixels not on the shadow boundaries. This leads to a compact representation, the *silhouette mask map*, which marks *silhouette texels*, texels that potentially has shadow boundary in it (Figure 4(a)). Pixels not intersecting any silhouette texels are either completely shadowed or lit and can be easily determined via a conventional depth test. This greatly reduces the computational cost.

Figure 2 shows the main building blocks of our algorithm. The main framework follows the shadow map that render the scene from light source and camera view respectively but involves more stages and multiple passes to extract silhouette, rasterize triangles and facets, and evaluate visibility.

We start by rendering a conventional depth map from the light source and creating the silhouette mask map (section 3.2). Next, the scene is rendered from the camera view and a facet is created regarding each pixel's position and normal. Facets are projected to light view and classified into non-silhouette and silhouette ones. If a facet is non-silhouette, shadow of the corresponding pixel can be conservatively computed by the depth test of standard shadow mapping. Otherwise, the pixel is identified as a *silhouette pixel* and its facet's projection on the light view will be rasterized using a software rasterization algorithm and stored with intersected texels as pixel-texel pairs (PT-pair) for further processing. To find triangles that potentially occlude these facets, all triangles are rasterized from the light view and then stored with intersected texels as triangle-texel pairs (TT-pair)(section 3.3). The TT-pairs and the PT-pairs are combined to calculate the occlusion ratio of pixels. For each triangle in a TT-pair, it will be projected to the screen plane and tested with pixels in the corresponding PT-pair. The occlusion mask lookup table makes the shadow test fast and with sub-pixel antialiasing (section 3.4). Finally, shadow of non-silhouette and silhouette pixels are combined to get the final shadowed image.

3.2. Constructing the silhouette mask map

To find silhouette texels, all silhouette edges must be identified in the scene. An edge is a silhouette if one of its two adjacent faces is a front face with respect to the light source and the other is a back face. Edges with only one adjacent face are also considered silhouette in the case of meshes with boundary. The silhouette edges are then rasterized to the silhouette mask map.

It is a common case that silhouette edges passing through a texel are all occluded by other triangles closer to the light source, resulting in an unnecessary silhouette texel. To remove such texels, the depth of the rasterized silhouette edges are compared with the depth map in each texel. If the depth value stored in the depth map is smaller, then all silhouette edges in this texel are occluded by triangles closer to the light source and the texel can be safely marked as non-silhouette.

3.3. Creating pixel-texel pairs and triangle-texel pairs

Removing non-silhouette pixels. To identify non-silhouette facets, all facets are projected to the light view. If there are no silhouette texels in the bounding rectangle of a projected facet, it is not on the silhouette. To speed up texel traversal, a *summered area table(SAT)* is built in a pre-pass from the silhouette mask map (Figure 4(b)). Only 4 references from the SAT are required to count the number of silhouette texels in a rectangle.

Software rasterization. Silhouette facets are rasterized to create the PT-pairs. As several projected facets may intersect with the same texel, we have to rasterize them in software to avoid collision. A three pass rasterization algorithm is employed here:

In the first pass, N_s, the number of silhouette texels in the bounding rectangle of a projected facet is counted using the *SAT*. The array of N_s is scanned to get the list size needed for a global list to store all the PT-pairs. Then texels in the bounding rectangle of each projected facet is traversed and a PT-pair is created for each silhouette texel and written to the global list. The write offset of each projected facet in the global array is exactly the corresponding scanned sum at the index of that facet.

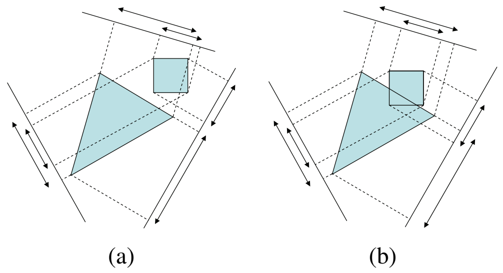


Figure 5: *A convex primitive and a texel in its bounding rectangle only intersect if their projections on any axis perpendicular to one of the primitive's edges overlap.*

In the second pass, all PT-pairs are processed in parallel to test if the projected facet actually intersects that texel. The *separating axis theorem* states that two convex polygons do not intersect only if their projection onto an axis perpendicular to one of their edges do not overlap. In our case, since the texel is in the bounding box of the projected facet, the projections onto axes perpendicular to the texel's edges are guaranteed to overlap. Thus, only the projected facet's edges are to be considered (Figure 5). Furthermore, projection of the facet to these axes can be precomputed once for each facet and used for testing all the texels in its bounding rectangle.

In the last pass, the global list is compacted to remove PT-pairs that fail the intersection test. The list is then sorted according to the texel index and scanned to get the offset to each texel's local list.

The triangles are rasterized analogously except that TT-pairs are created for texels with a non-empty local PT-pair list rather than silhouette texels.

3.4. Visibility evaluation

With TT-pairs and PT-pairs created, we parallelize the computation in TT-pairs. For the texel in a TT-pair, pixels in the corresponding local PT-pair list are traversed and shadow tests are carried out between the identified triangles and pixels. Our shadow test takes a two-step projection. The triangle is projected to the pixel's tangent plane (the plane of pixel's facet) while the occluded area is determined, then the occluded area is projected to the screen plane to recover the view samples it covers.

The occluded area may have different shapes based on the relative position of the tangent plane, the triangle and the light source. Nevertheless, this area is always bounded by the projected edges of the triangle, plus the intersection line of the triangle and the tangent plane if they intersect. Therefore, we represent a triangle using three half-planes defined by its edges. If the triangle and the facet intersects, an extra half-plane defined by the intersection line is added. The projection of these half-planes onto the tangent plane are also half-planes, forming the occluded area. (Figure 6).

(a) and (b) panel labels are inside the figure.

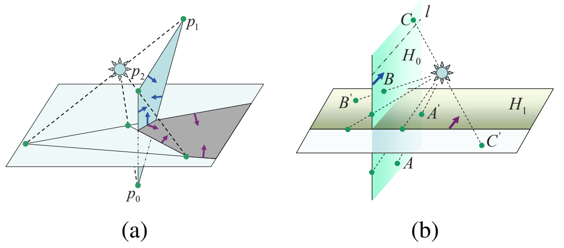


Figure 6: *(a) A blocker is represented as a set of half-planes (blue arrows). Projections of these half-planes on the tangent plane (purple arrows) bound the occluded area. (b) Projection of a half-plane H_0 is still a half-plane H_1 whose boundary is determined by projecting two points on the boundary of H_0. To determine which side of the boundary is H_1, an arbitrary point in H_0 is selected. If the point lies between the projection center and the light source (B in figure) or is on the other side of the tangent plane (A in figure), then its projection is in H_1, otherwise (above line l as in figure, e.g. C), it is not in H_1.*

Next, the occluded area is projected to the camera view similarly to obtain a group of half-planes on the screen plane. Similar to [ED07], we precompute a lookup table which takes a half-plane as input and returns a bit pattern corresponding to the covered view samples. We bitwise AND the patterns of all half-planes and then bitwise OR the result to the pixel's *occlusion mask* to be accumulated with occlusion from other triangles.

Validity mask. As the facet determined by the pixel center is used to approximate the geometry within the entire pixel, sometimes shadow will be over- or under-estimated. This happens when the facet intersects nearby triangles and causes light leaking or incorrect self-shadow (Figure 7(a)). To compensate for such incorrect estimation, another bitmask, named *validity mask*, which has the same bit depth as *occlusion mask*, is linked with each pixel. Each bit of *validity mask* corresponds to the validity of one view sample and denotes whether the sample should be counted in occlusion ratio calculation.

For each triangle intersecting a facet, the intersection line is calculated and projected to the view plane. Unlike half-planes, the line does not have explicit information indicating which of its two sides should be marked as invalid. However, with the fact that pixels are only rasterized if a triangle covers its center, it is guaranteed that a view sample at the pixel center should always be valid. Therefore, if the returned bitmask from the lookup table marks the pixel center as invalid, the bitmask is negated using bitwise NOT before being bitwise ANDed to the pixel's *validity mask*. This method requires a sample to be placed at the center of a pixel in all sample configurations.

The validity mask calculation can be easily integrated into the visibility evaluation function. The intersection line of tri-

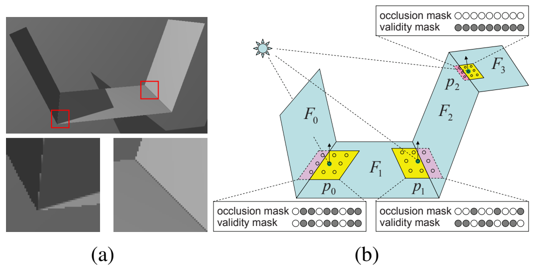


Figure 7: *(a). Light leaking and incorrect self-shadow problem. (b) We introduce validity mask to eliminate invalid samples.*

angle and facet is already computed during occluded area calculation, therefore the only additional operation required is bitwise NOT and AND of the *validity mask*. Occlusion ratio calculation function should also be modified. Instead of counting the number of 1 bits in *occlusion mask* and dividing it with the total number of view samples in one pixel, *occlusion mask* is first bitwise ANDed with the *validity mask* and occlusion ratio is computed by dividing the number of 1 bits in the ANDed result with the number of 1 bits in the *validity mask*.

By using the validity mask, sample rate in some pixels are reduced as some view samples do not contribute to the occlusion ratio. It is worth mentioning that at convex locations (e.g. p_2 in Figure 7), we do not mask the samples in the pink area since the intersecting face (F_2 in figure) is farther from the light source than the facet. This makes our method retain full sample rate for convex objects.

4. Implementation

We built an implementation of our algorithm on the GPU using CUDA and shaders.

First, a conventional depth map is rendered from the light view. Next, silhouette edges are identified using CUDA and then rasterized to obtain the silhouette mask map. For each silhouette edge, we conservatively draw a line with width of 2 and two points with size of 3 at the edge's endpoints. In the fragment shader, an intersection test is used to check whether the edge intersects the fragment. It also compares the fragment's depth with the value stored in the depth map to eliminate hidden silhouette. A SAT is then built from the silhouette mask map [MSO07].

Next, the scene from the camera view is rasterized to get the position and normal of each pixel for constructing the facets. To prevent normal interpolation which causes incorrect facet orientation, face normals are used instead of vertex normals.

Non-silhouette facets are identified and removed in the way described in section 3.3. The rest facets are then rasterized using the software rasterization algorithm implemented

Scene	Figure	Triangles	Silhouette texels	Silhouette pixels	Visibility evaluations	Sample rate	Create silhouette mask map	Create PT-pairs and TT-pairs	Shadow calculation	FPS
Windmill	Fig. 1(c)	40712	9365	36113	1295363	32	2.3ms	6.5ms	13.2ms	41
Bunny	Fig. 8(a)	69473	3254	18523	609901	32	2.1ms	6.3ms	6.8ms	61
Pterosaur	Fig. 10(a)	10000	1828	12991	329740	32	1.6ms	4.7ms	4.3ms	80
Cage	Fig. 11(d)	59780	37646	53576	2915995	128	4.2ms	10.0ms	18.9ms	28
Bicycle	Fig. 12(a)	30922	4277	40954	2996154	32	2.1ms	6.6ms	15.3ms	47
Horse	Fig. 12(b)	77500	4317	26202	2138740	32	2.3ms	6.5ms	17.3ms	32
Plant	Fig. 12(c)	20926	15263	59692	1446287	32	2.1ms	6.2ms	10.8ms	50

Table 1: *Statistics of the test scenes.*

in CUDA to create the PT-pair lists. The algorithm primitives used, including *scan*, *sort* and *compact*, are implemented according to [SHZO07]. Triangles are also rasterized similarly to create the TT-pairs for texels with a non-empty local PT-pair list.

All TT-pairs are processed in parallel. For each TT-pair, we traverse all pixels in the PT-pair list of that texel. Visibility is evaluated for each pixel against the triangle as described in section 3.4. *Occlusion mask* and *visibility mask* of that pixel is updated using the atomic instructions in CUDA.

PT-lists of different texels may have distinct lengths. When TT-pairs are processed in parallel for visibility evaluation, each of them can traverse a different number of pixels, which may result in bad multi-thread load balance. To solve this problem, TT-pairs are sorted according to the length of PT-pair list of that texel beforehand so that nearby pairs (particularly pairs for the same warp of CUDA threads) traverse a similar number of pixels.

Finally, occlusion ratio of each pixel is calculated based on the *occlusion mask* and *visibility mask*. These values are applied to the scene image with standard shading.

5. Results and discussion

We tested our algorithm with several scenes. The tests are run on a PC with 3.0G Hz CPU, 2.0G host memory and an NVIDIA GTX280 video card with 1.0G video memory.

Table 1 lists the statistics and performance of the test scenes. All the scenes are rendered with an image size of 800×600. Depth maps and shadow silhouette mask maps have 512×512 resolution.

The fourth and fifth columns of the table list the number of silhouette texels and silhouette pixels in each scene respectively. The sixth column is the number of visibility evaluations, i.e. the total number of occlusion tests between triangles and pixels. The next three columns list time spent every frame for each stage of the algorithm, corresponding to section 3.2, 3.3 and 3.4 respectively.

5.1. Discussion

Quality. Our algorithm evaluates shadow on the facet of each pixel instead of the complex underlying geometry, therefore may introduce error along the shadow boundary.

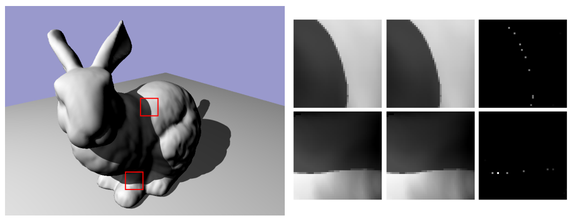

(a) (b)

Figure 8: *(a) Shadows on the bunny model at 32× sample rate. (b) Zoomed in views (left column), reference image from a ray-tracer (middle column), 10× error image (right column).*

Figure 8 compares our result with a ray-traced ground truth image both at 32× sample rate. The difference between the two images is visually small (Figure 8(b)). For shadow cast on a plane, our facet approximation is accurate and produces the same result as the reference image. However, if a pixel covers the geometry with varying depths, errors will be produced. This is a limitation of our algorithm and is due to the fact that we only rasterize the scene at standard resolution therefore the geometric variation within a pixel is missed. Nevertheless, this error is limited to the sub-pixel level and does not cause noticeable artifacts.

The validity mask we introduced may reduce sample rate in concave locations as stated in section 3.4. Figure 9 shows shadow cast on a bumpy receiver. The receiver is generated from a height field and is densely tesselated that the projection area of most triangles are less than one pixel and even triangles nearest to the camera cover no more than 4×4 pixels. The scene is rendered at 32× sample rate and the average number of valid samples per pixel on the shadow boundary is 18.5. However, in the extreme case that the scene is a highly tesselated concave object, our method will fall back to point sampling.

Scalability. Supersampling requires every sample to undergo a shadow test, thus the computation time grows almost linearly to the sample rate. On the contrary, our algorithm evaluates visibility only once for each pixel therefore has a significantly better scalability. For example, to accommodate twice the number of samples, our algorithm only need to double the bit depth of the masks, resulting in twice the size of the precomputed lookup table which is very small,

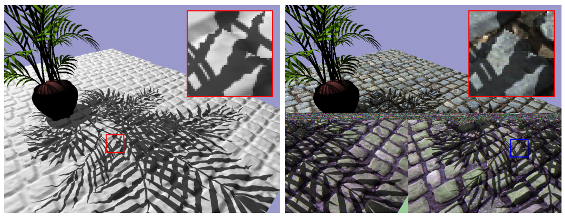
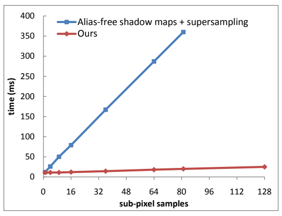

(a) No texture (b) With Texture

Figure 9: *Shadow cast on a bumpy receiver.*



(a) (b)

Figure 10: *Performance comparison between our method and alias-free shadow map + supersampling on the pterosaur scene at different sample rates. Performance of supersampling at 128× is not listed because it requires a framebuffer larger than the limit of hardware.*

and twice the number of bitwise operations which are rather fast. For comparison we implemented the GPU-based alias-free shadow maps according to [SEA08] and added supersampling functionality. Figure 10 compares its performance with ours under different sample rates.

Figure 11 compares shadows of the cage scene under different sample rates. Shadows of the cage lines are missed even at 8× samples. The 32× image captures almost all these features. Due to the good scalability of our algorithm, an 128× sample rate is able to be rendered in real-time and delivers smoother shadows.

Since the sub-pixel shadow calculation only affects the silhouette, our algorithm grows linearly to the number of triangles on the silhouette rather than the total number of triangles in the scene. In some extreme cases, such as scenes with a large number of silhouettes, the silhouette mask map may not improve any performance but rather require some extra time to build it. However, this overhead is quite small compared to the total time.

Custom sample configuration and filters. Currently, a grid configuration for sample positions is used in all the scenes. Nevertheless, as the only assumption our method makes is placing one sample at the center of a pixel, it is compatible with other patterns like rotated grid. For a new sample pattern, only the lookup table needs to be recomputed to reflect the change of sample positions and the rendering pipeline is kept unchanged. However, patterns with dynamic positions like Poisson disc or jittered grid are not supported as the lookup table could not be computed on-the-fly.

Besides box filter, which averages all samples in each pixel, it is also interesting to use other filters like Gaussian. This requires modifying the function that calculates occlusion ratio so that instead of counting the number of 1 bits, a weighted sum is computed. Although this may deliver a better quality, it will be much slower especially when the support of the filter is large, in which case masks of nearby pixels must be accessed. Therefore, we are leaving it as future work.

6. Conclusion

In this paper, we have presented a versatile solution that enables sub-pixel shadow antialiasing. Our algorithm outperforms supersampling by an order of magnitude while keeping similar shadow quality. The algorithm can be completely implemented in latest graphics hardware, yielding real-time performance. We have demonstrated that our approach is robust and efficient for a variety of complex scenes and is capable of capturing fine shadow features and details that can not be sampled by alias-free shadow maps. An interesting future work would be extending this framework to shading antialiasing.

Acknowledgements

This project is supported by the 973 program of China (No. 2009CB320803).

References

[AL04] Aila T., Laine S.: Alias-free shadow maps. In *Proceedings of the Eurographics Symposium on Rendering* (2004), pp. 161–166.

[AMB*08] Annen T., Mertens T., Bekaert P., Seidel H.-P., Kautz J.: Exponential shadow maps. In *Proceedings of Graphics Interface* (May 2008).

[AMS*07] Annen T., Mertens T., Seidel H.-P., Flerackers E., Kautz J.: Convolution shadow maps. In *Rendering Techniques 2007: Eurographics Symposium on Rendering* (June 2007).

[BLM08] Brandon Lloyd Naga Govindaraju C. Q. S. M., Manocha D.: Logarithmic perspective shadow maps. *ACM Transactions on Graphics 27*, 4 (Oct. 2008).

[BWG03] Bala K., Walter B., Greenberg D. P.: Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph. 22*, 3 (2003), 631–640.

[CD04] Chan E., Durand F.: An efficient hybrid shadow rendering algorithm. In *Proceedings of the Eurographics Symposium on Rendering* (2004), Eurographics Association, pp. 185–195.

[CG04] Chong H., Gortler S. J.: A lixel for every pixel. In *Proceedings of the Eurographics Symposium on Rendering* (2004).

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

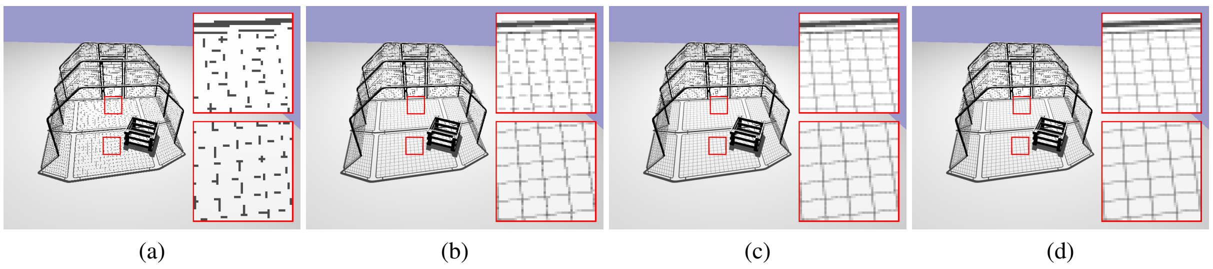


Figure 11: *Shadow of the cage scene at different sample rates. (a) 1×, (b) 8×, (c) 32×, (d) 128×.*

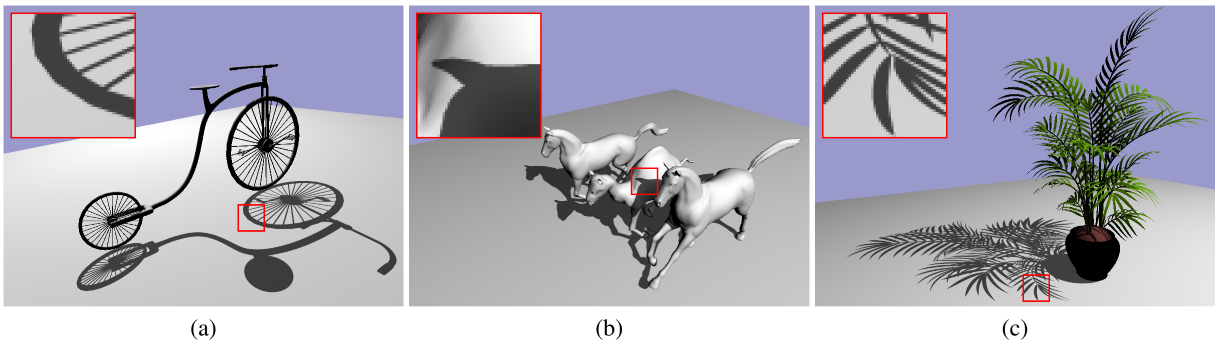


Figure 12: *Shadows on the (a) bicycle, (b) horse and (c) plant.*

[FFB01] Fernando R., Fernandez S., Bala K.: Adaptive shadow maps. In *Proceedings of SIGGRAPH '01* (2001), ACM SIGGRAPH, pp. 387–390.

[gls02] Gl_arb_multisample, 2002. `http://www.opengl.org/registry/specs/ARB/multisample.txt`.

[GW07] Giegl M., Wimmer M.: Queried virtual shadow maps. In *Proceedings of ACM SIGGRAPH 2007 Symposium on Interactive 3D Graphics and Games* (Apr. 2007), pp. 65–72.

[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), 1462–1482.

[LGMM07] Lloyd D. B., Govindaraju N. K., Molnar S. E., Manocha D.: Practical logarithmic rasterization for low-error shadow maps. In *GH '07: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2007), pp. 17–24.

[LSO07] Lefohn A. E., Sengupta S., Owens J. D.: Resolution-matched shadow maps. *ACM Transactions on Graphics 26*, 4 (Oct. 2007), 20:1–20:17.

[LTYM06] Lloyd B., Tuft D., Yoon S., Manocha D.: Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006* (2006), Eurographics Association, pp. 215–226.

[MSO07] Mark H., Shubhabrata S., Owens J. D.: Parallel prefix sum (scan) with cuda. *GPU Gems 3* (2007).

[MT04] Martin T., Tan T.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the 2nd EG Symposium on Rendering* (2004).

[RSC87] Reeves W. T., Salesin D., Cook R. L.: Rendering antialiased shadows with depth maps. *Computer Graphics Proceedings of SIGGRAPH '87* (1987), 283–291.

[SCH03] Sen P., Cammarano M., Hanrahan P.: Shadow silhouette maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)* (2003).

[SD02] Stamminger M., Drettakis G.: Perspective shadow maps. In *Proceedings of ACM SIGGRAPH* (July 2002), ACM SIGGRAPH, pp. 557 – 562.

[SEA08] Sintorn E., Eisemann E., Assarsson U.: Sample based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2008) 27*, 4 (2008), 1285–1292.

[SHZO07] Sengupta S., Harris M., Zhang Y., Owens J. D.: Scan primitives for gpu computing. In *Graphics Hardware 2007* (Aug. 2007), ACM, pp. 97–106.

[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)* (June 2007), pp. 45–50.

[Wil78] Williams L.: Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings) 12*, 3 (Aug. 1978), 270–274.

[WSP04] Wimmer M., Scherzer D., Purgathofer W.: Light space perspective shadow maps. In *Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering)* (June 2004).

[ZSXL06] Zhang F., Sun H., Xu L., Lun L. K.: Parallel-split shadow maps for large-scale virtual environments. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications* (2006).