Fast, Sub-pixel Antialiased Shadow Maps

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

State Key Lab of CAD&CG, Zhejiang University

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

(c) 2009 The Author(s)
Journal compilation (c) 2009 The Eurographics Association and Blackwell Publishing Ltd.
Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA
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 sophisticatedly, 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 $64 \times 128$ 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 its 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. [LYTM06] 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.

[SBW07] 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
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)).

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

---

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.

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.
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 anti-aliasing (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 summed 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, \( N_i \), the number of silhouette texels in the bounding rectangle of a projected facet is counted using the SAT. The array of \( N_i \) 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.

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).
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 triangles 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.
in CUDA to create the PT-pair lists. The algorithm primitives used, including scan, sort and compact, are implemented according to [SHZ07]. 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.

![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).](image)

Table 1: Statistics of the test scenes.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Figure</th>
<th>Triangles</th>
<th>Silhouette texels</th>
<th>Silhouette pixels</th>
<th>Visibility evaluations</th>
<th>Sample rate</th>
<th>Create silhouette mask map</th>
<th>Create PT-pairs and TT-pairs</th>
<th>Shadow calculation</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windmill</td>
<td>Fig. 12(c)</td>
<td>40712</td>
<td>9365</td>
<td>38113</td>
<td>289583</td>
<td>32</td>
<td>2.3ms</td>
<td>6.5ms</td>
<td>13.2ms</td>
<td>41</td>
</tr>
<tr>
<td>Bunny</td>
<td>Fig. 8(a)</td>
<td>69473</td>
<td>3254</td>
<td>18533</td>
<td>609901</td>
<td>32</td>
<td>2.1ms</td>
<td>6.3ms</td>
<td>13.4ms</td>
<td>61</td>
</tr>
<tr>
<td>Pterosaur</td>
<td>Fig. 30(a)</td>
<td>10000</td>
<td>1828</td>
<td>12991</td>
<td>329740</td>
<td>32</td>
<td>1.6ms</td>
<td>4.7ms</td>
<td>4.3ms</td>
<td>80</td>
</tr>
<tr>
<td>Cage</td>
<td>Fig. 31(d)</td>
<td>59780</td>
<td>37646</td>
<td>53576</td>
<td>2915995</td>
<td>128</td>
<td>4.2ms</td>
<td>10.0ms</td>
<td>18.9ms</td>
<td>28</td>
</tr>
<tr>
<td>Bicycle</td>
<td>Fig. 12(a)</td>
<td>30922</td>
<td>4277</td>
<td>40954</td>
<td>2996154</td>
<td>32</td>
<td>2.1ms</td>
<td>6.6ms</td>
<td>15.3ms</td>
<td>47</td>
</tr>
<tr>
<td>Horse</td>
<td>Fig. 12(b)</td>
<td>77500</td>
<td>4317</td>
<td>26202</td>
<td>2138740</td>
<td>32</td>
<td>2.3ms</td>
<td>6.5ms</td>
<td>17.3ms</td>
<td>32</td>
</tr>
<tr>
<td>Plant</td>
<td>Fig. 12(c)</td>
<td>20926</td>
<td>15263</td>
<td>59692</td>
<td>1295363</td>
<td>32</td>
<td>2.3ms</td>
<td>6.8ms</td>
<td>10.8ms</td>
<td>50</td>
</tr>
</tbody>
</table>

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 tessellated 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 tessellated 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,
Figure 9: Shadow cast on a bumpy receiver.

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 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

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.


