Microquad Soft Shadow Mapping Revisited

Michael Schwarz and Marc Stamminger

University of Erlangen-Nuremberg

Abstract

Recently, real-time soft shadow research saw many important contributions from the class of soft shadow mapping algorithms, where approximate occluder geometry is reconstructed from a shadow map and backprojected onto the light source to determine its occlusion. An interesting approximation primitive is the microquad obtained from unprojected shadow map texel centers. However, its full potential has not yet been realized since correct clipping against the light and exact occlusion bitmask updates have previously been ignored motivated by performance considerations. In this paper, we first extend the microquad definition to also allow for triangles as micro-occluders. We then demonstrate how to determine the exact occlusion bitmask of a microquad with correct clipping.

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

1. Introduction

The existence of soft shadows significantly contributes to the realism in a scene, and hence rendering them at interactive or even real-time frame rates is a crucial objective. Recently, several algorithms were introduced which take a shadow map and reconstruct a subset of the occluding geometry from the shadow map texels [AHL*06,GBP06,BS06,ASK06]. By backprojecting the resulting micro-occluders onto the light source, its occlusion can be derived. Many of these so-called soft shadow mapping techniques just accumulate the areas of these backprojections, thus ignoring artifacts due to overlapping backprojections. Bitmask soft shadows (BMSS) [SS07] instead resort to point sampling and track the visibility of light points via occlusion bitmasks, and therefore are able to perform correct occluder fusion.

One interesting occluder approximation (cf. Fig. 1) and the focus of this paper is the microquad [SS07], constructed by taking the unprojected centers of four adjacent shadow map texels as vertices. Microquads feature several desirable properties like providing a good fit to the approximated geometry and implicitly avoiding light leaks. However, because a quad's backprojection onto the light area does not yield an axis-aligned rectangle in general, correct clipping and area determination or occlusion bitmask updates are complicated. As a consequence and further motivated by performance considerations, exact solutions to these chal-
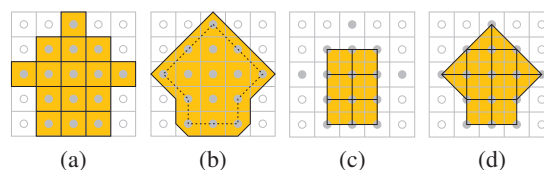
Figure 1: *Texture space footprint of occluder approximations: (a) micropatches [AHL*06, GBP06], (b) occluder contour [GBP07], (c) microquads [SS07], (d) microtris.*

lenges have been ignored so far. However, current simplifications can introduce noticeable deviations (see Fig. 3). In this paper, we focus on exact solutions and present an approach based on occlusion bitmasks. We further reduce occluder underestimation by extending the microquad definition.

2. Microtris: extending the microquad definition

A microquad is only constructed and backprojected if all four vertices are closer to the light source than the point for which light visibility is determined [SS07]. We propose to also generate and process a micro-primitive if only three of the four considered vertices pass the distance test. The resulting triangle, termed *microtri*, helps to better approximate the occluding geometry and hence reduces the tendency of the microquad approximation to underestimate the occluders' extent (cf. Fig. 1 d). Note that in case of visibility de-
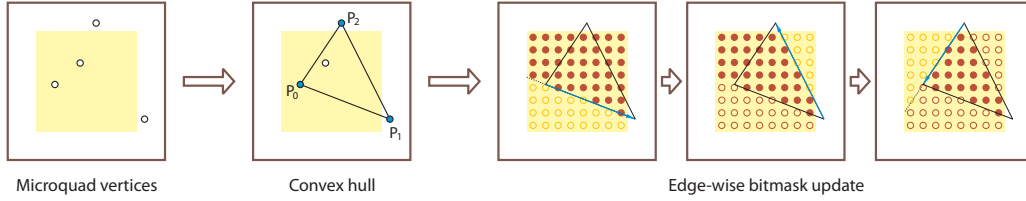
Figure 2: *Overview of the steps involved in deriving a microquad's bitmask from its backprojected vertices.*

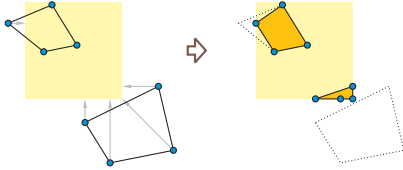termination by area accumulation, microtris are readily supported.

Figure 3: *Clipping a quad against the rectangular light area by clamping is fast but affects its footprint and can thus even introduce erroneous occluders.*

3. Exact microquad occlusion processing

Referring to Fig. 4, several different configurations can arise when backprojecting a quad □**ABCD**. While the orange area might seem reasonable in cases (a) – (c) (and area accumulation correctly yields them by splitting the quad in two triangles and adding their signed areas), the foldover case (d) questions the suitability of using a quad's interior as its occlusion footprint. Reasoning that a microquad should have the same footprint as the union of all of its microtris, we hence suggest taking the convex hull of a microquad's backprojected vertices as occluder approximation.
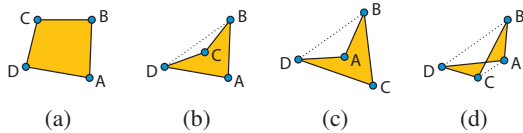

Figure 4: *Various cases of backprojected quads.*

Our general approach to exactly determine a microquad's bitmask, outlined in Fig. 2, therefore starts with deriving the CCW-oriented convex hull. In order to determine the bits corresponding to occluded sample points, we employ a convex hull bitmask initialized to all bits set. For each oriented edge of the convex hull, we determine the bitmask for the corresponding half-space and perform a bitwise AND with the convex hull bitmask. Finally, we update the occlusion bitmask to incorporate the microquad's occlusion footprint by ORing it with the convex hull bitmask.

Note that correct clipping against the light area is automatically performed by the edge-wise convex hull bitmask

updates. Moreover, microtris are naturally supported while previous techniques [SS07] have fundamental problems with them since they approximate quads by axis-aligned rectangles for bitmask updates.

3.1. Convex hull determination

In a first step, we determine the convex hull of the microquad or microtri such that its vertices are enumerated in CCW order. Note that in case of a microquad, the (non-degenerated) convex hull may be either a quadrilateral or a triangle.
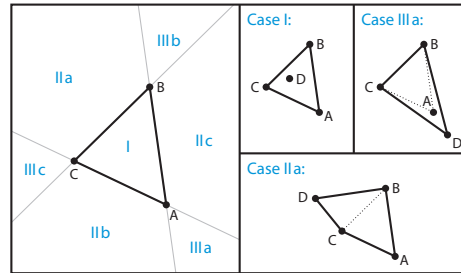

Figure 5: *Cases for vertex* **D** *in convex hull determination.*

In our shader-friendly approach, we start with the triangle △**ABC** made up of the microquad's first three vertices or the microtri, respectively, determine its orientation and swap **B** and **C** if it is clockwise. In case of a microtri, we are already done. Otherwise, we compute the barycentric coordinates of vertex **D** with respect to the triangle, to identify which of the seven cases depicted in Fig. 5 holds. If necessary, we reorder the vertices to ensure a CCW order.

3.2. Convex hull bitmask construction via half-space bitmask lookup texture

To determine the convex hull bitmask, we precompute the bitmasks for a number of half-spaces and store them in a lookup texture, similar to Eisemann and Décoret [ED07]. The big advantage of this approach is that it puts no restrictions on the placement of sample points, and hence naturally allows random distribution patterns.

More precisely, we parameterize the half-spaces by the Hough transform [DH72] of the corresponding lines, i.e. their angles θ and signed distances r to the origin. We

$d_y > 0$:

$\Delta l = 0$

$\Delta r = d_x/d_y$

$l = 0$

$r = a_x - a_y \Delta r$

$d_y < 0$:

$\Delta l = d_x/d_y$

$\Delta r = 0$

$l = a_x - a_y \Delta l$

$r = n$

$d_y = 0$:

$\Delta l = 0$

$l = 0$

$d_x > 0$:

$\Delta r = 2n$

$r = n - 2n\left(\lfloor a_y \rfloor + \frac{3}{4}\right)$

$d_x \leq 0$:

$\Delta r = -2n$
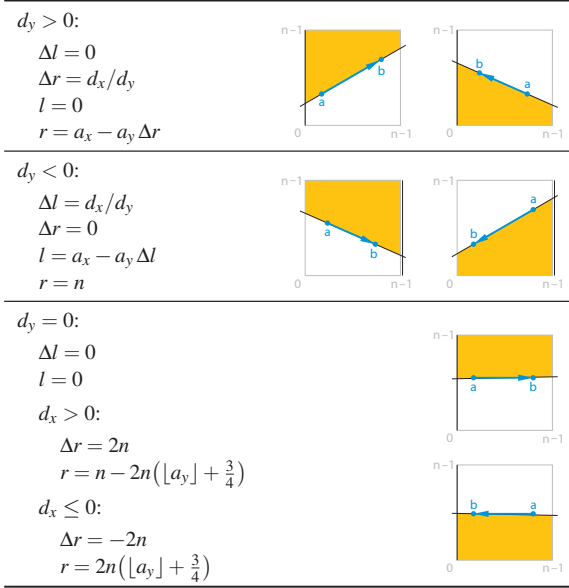
$r = 2n\left(\lfloor a_y \rfloor + \frac{3}{4}\right)$

Figure 6: *Setup of exact bitmask computation parameters for a directed edge* $\mathbf{a} \to \mathbf{b}$ *where* $\mathbf{d} = \mathbf{b} - \mathbf{a}$.

regularly sample (θ, r) from the relevant domain $[-\pi, \pi] \times [-\sqrt{2}, \sqrt{2}]$, determine the corresponding bitmask values and store them in a lookup texture. At run-time, for each edge of the convex hull, we compute its Hough parameters, query the lookup texture (with circular wrapping for θ and clamping for r) and update the convex hull bitmask.

Concerning resources, a two channel `UINT32` texture suffices for 8×8 sample points, whereas for 16×16 points a four channel `UINT32` texture array with two array slices is required, and 32×32 points even necessitate eight array slices and hence eight texture fetches per edge of the convex hull. To keep the shader register count to a minimum, we refrain from keeping a complete representation of the convex hull bitmask for cases with more than 128 sample points, i.e. where multiple texture fetches are performed per edge. Instead, we only employ a `uint4` variable as working set and both construct the convex hull bitmask and update the occluder bitmask 128-sample-point-wise by appropriate instruction interleaving.

3.3. Exact convex hull bitmask computation

Since the gap between offered computational power and available memory bandwidth is widening, keeping a shader's arithmetic intensity high can help performance, especially in the long run. Consequently, it seems desirable to avoid texture lookups and determine a convex hull's bitmask merely by computations. However, it is way too expensive to loop over sample positions and check for each whether it is contained in the convex hull. On the other hand, for regular sample point patterns of size $n \times n$, bitmask updates are already feasible at interactive frame rates.
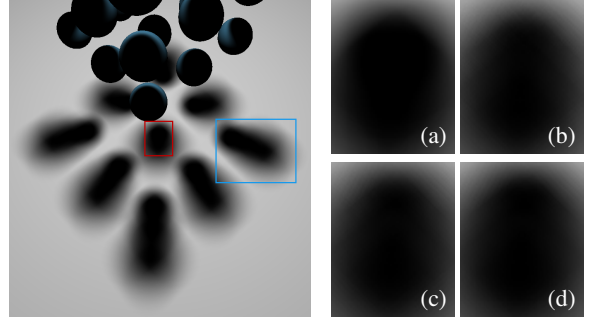
Figure 7: *Left: reference soft shadows cast by 27 spheres. Right: close-up of the red region: (a) reference, (b) microquads/BMSS, (c) microquads/lookup texture, (d) microtris/lookup texture (each time: 32×32 points, regular sample pattern). Compared to (b), the L^1 errors regarding (a) are 25% and 48% lower for (c) and (d), respectively.*

In particular, if the sample points are located at integer positions (i, j), $i, j = 0 \dots n - 1$, it is possible to efficiently determine a whole row of bits $(\cdot, k)$. To this end, we keep two values l and r, with $\lceil l \rceil$ to $\lceil r \rceil$ (excluded) specifying the range of bits to be set within the row. After each row, l and r are incremented by Δl and Δr, respectively. These parameters are derived as listed in Fig. 6 after the convex hull vertices have been transformed into sample space. Note that horizontal edges are readily supported by taking the parameters for a slightly tilted and shifted edge that yields the same bitmask, thus exploiting the regular sample point placement.
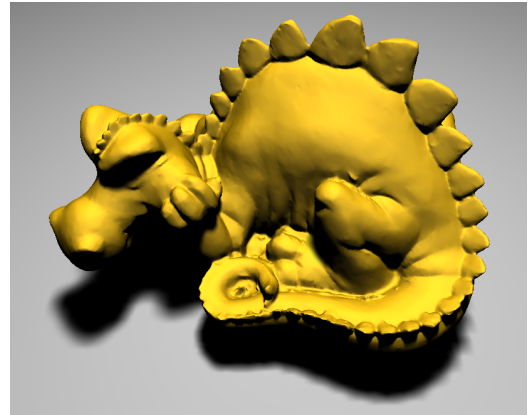

Figure 8: *Phlegmatic dragon example (here: lookup texture, with microtris, 32×32 randomly distributed points).*

4. Results and conclusion

By means of the fused soft shadow of three spheres, Fig. 7 demonstrates that non-approximate processing of microquads yields better results than the approximations from our BMSS paper. Moreover, additionally accounting for mi-
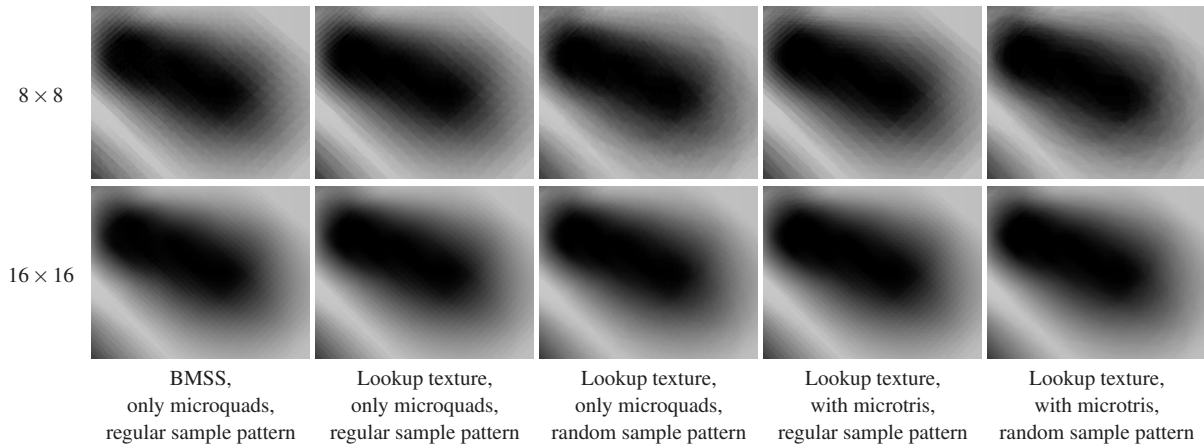
| | BMSS,
only microquads,
regular sample pattern | Lookup texture,
only microquads,
regular sample pattern | Lookup texture,
only microquads,
random sample pattern | Lookup texture,
with microtris,
regular sample pattern | Lookup texture,
with microtris,
random sample pattern |

Figure 9: *Close-up of the blue region in Fig. 7.*

Micropatches			
Area accumulation		59.3	
BMSS	16×16, jittered	34.6	
Microquads without/with microtris			
Area accumulation		37.2	29.6
BMSS	8×8, regular	30.2	
BMSS	16×16, regular	24.3	
BMSS	16×16, jittered	23.3	
BMSS	32×32, regular	11.5	
Lookup texture	8×8	16.9	13.6
Lookup texture	16×16	13.2	10.7
Lookup texture	32×32	4.2	3.4
Exact computation	8×8, regular	11.7	9.2

Table 1: *Frame rates in Hz for the example scene from Fig. 8 on a GeForce 8800 GTX (viewport: 1024×768).*

crotris reduces occluder underestimation and hence the deviations from the reference.

For a different set of spheres, Fig. 9 additionally contrasts different bit field sizes and sample patterns. The random placement of sample points (we employed stratification in our examples) made possible by resorting to a lookup texture successfully trades noticeable discretization artifacts for noise. For the 16×16 case, the results achieved with the random pattern are roughly on par with the jittered placement strategy (not shown) from [SS07].

Performance data for the dragon scene from Fig. 8 is listed in Table 1. We note that support for microtris adds an overhead of about 20%. The convex hull and lookup texture based microquad processing roughly halves the frame rate compared to the approximate approach from our BMSS paper. Finally, determining half-space bitmasks merely by arithmetic instructions appears to be not yet competitive on current hardware.

In conclusion, we realized some of the microquads' idle potential by studying some previously ignored aspects. We introduced microtris and identified the convex hull as a more appropriate occlusion footprint. Building on top of that, we showed how to determine the exact bitmask with correct clipping, and presented two possible implementations.

Acknowledgements

References

[AHL*06] ATTY L., HOLZSCHUCH N., LAPIERRE M., HASEN-FRATZ J.-M., HANSEN C., SILLION F. X.: Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum 25*, 4 (2006), 725–741.

[ASK06] ASZÓDI B., SZIRMAY-KALOS L.: Real-time soft shadows with shadow accumulation. In *Eurographics 2006 Short Papers* (2006), pp. 53–56.

[BS06] BAVOIL L., SILVA C. T.: Real-time soft shadows with cone culling. In *ACM SIGGRAPH 2006 Sketches and Applications* (2006).

[DH72] DUDA R. O., HART P. E.: Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM 15*, 1 (1972), 11–15.

[ED07] EISEMANN E., DÉCORET X.: Visibility sampling on GPU and applications. *Computer Graphics Forum 26*, 3 (2007), 535–544.

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Real-time soft shadow mapping by backprojection. In *Proceedings of Eurographics Symposium on Rendering 2006* (2006), pp. 227–234.

[GBP07] GUENNEBAUD G., BARTHE L., PAULIN M.: High-quality adaptive soft shadow mapping. *Computer Graphics Forum 26*, 3 (2007), 525–533.

[SS07] SCHWARZ M., STAMMINGER M.: Bitmask soft shadows. *Computer Graphics Forum 26*, 3 (2007), 515–524.