Technical Report, Intel Labs, 2010

A Fast Scanline Micro-Rasterization Kernel and its Application to Soft Shadows

Sven Woop†

Intel Labs

Abstract

Many soft shadow algorithms approximate the visibility integral via sampling occlusion at many locations on the light source. We present a fast micro-rasterization kernel to efficiently handle a grid of such sample locations by micro-rasterizing triangular occluders onto planar lights. Our bit-parallel scanline algorithm iterates over relevant scanlines and clears occluded samples using a small number of bit operations. We show that our algorithm outperforms conventional software rasterization kernels by up to one order of magnitude for large triangles and is similarly efficient for small triangles. We achieve a speedup of 2x in a soft shadow implementation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible line/surface algorithms

1. Introduction

Many illumination problems require the computation of visibility between points in space. For instance, the common case of computing soft shadows of a small area light source is a mainly a visibility integral as cosine factors can assumed to be constant over the light. Recent work [BW09] proposed multi frustum traversal to efficiently traverse the frustum spanned by the receiver and the light and test the contained geometry against a set of sample rays. Due to their efficient frustum traversal method, the bottleneck in this method became the ray/triangle intersection.

We present an occlusion kernel can be used in such a setup to rasterize occluding triangles into small occlusion buffers. As this operation is performed per pixel, we refer to it as *micro-rasterization* similar to micro-rendering [REG*09]. Our algorithm can in general be used for computing visibility between a point and a regular sampling pattern on a surface. Besides soft shadows from planar light sources, other potential areas of application are ambient occlusion, anti-aliasing, and approximating visibility between hierarchy clusters, e.g. for hierarchical radiosity.

† sven.woop@intel.com

We present an efficient way of performing triangle setup by slightly varying the approach of homogeneous rasterization [OG97] by using Plücker tests. Our main contribution is a scanline algorithm that leverages *bit level parallelism* to efficiently compute a coverage mask of the triangle, by processing all samples of one scanline in parallel. Therefore, we occlude per scanline the range of pixels covered by the triangle through a small number of bit operations.

2. Previous Work

The purpose of a rasterizer is to determine sample points that are covered by a given triangle. This test is typically performed in 2D by projecting geometry into an image plane and testing the samples against the triangle edge equations. The edge equations are computed after projection [Pin88] or optionally before projection in 2D homogeneous coordinates [OG97]. To speed up the computation, a conservative set of candidate sample can be computed using hierarchical rasterization [Gre96,Sea08], screen tiling [Sea08], and/or using the triangle screen bounding box.

Scanline algorithms [Bou70] operate on the frame buffer scanline by scanline by managing a list of currently active edges and processing the pixels between the start and end of

the polygons. Bit operations are known in hardware implementations of scanline algorithms to compute triangle coverage masks for anti aliasing [WKP*97]. In contrast, we show how to leverage software to gain similar efficiency and demonstrate the application to soft shadow computations.

3. Setup in Homogeneous Space

For each pixel, we rasterize the triangles contained in the frustum spanned by the primary hit location C and the light source L. The light source is positioned at L_P and spanned by the vectors L_S and L_T. The point C is our projection center and the light source L is our projection plane. A regular grid of samples on the light source is given by $L(s,t) = L_p + s \cdot L_S + t \cdot L_T$ for integer locations of s and t.

As the projection center C varies per pixel, triangle setup need to be performed per pixel as well. We minimize setup costs by using a modification of homogeneous rasterization [OG97]. Homogeneous rasterization avoids the perspective divide by computing edge equations in 2D homogeneous space, which avoids costly clipping operations for triangles crossing the zero plane.

We first simplify the computation by moving the projection center from C to the zero point and the projection plane from the light L to the $z = 1$ plane. We achieve this by transforming the triangle vertices P_0, P_1, and P_2 with the inverse of the affine transformation $P_i = (L_S, L_T, L_P - C) \cdot V_i + C$ into a space with its origin at C and a coordinate system spanned by L_S, L_T, and $L_P - C$. After this geometric transform we obtain post transformed triangle vertices V_0, V_1, and V_2. Computing this transformation requires a matrix inversion once per pixel.

In the paper [OG97] the 2D homogeneous edge equations are computed by taking the rows of the inverse of the matrix (V_0, V_1, V_2). We vary this approach, by using the Plücker ray/triangle test [Eri97] and compute:

$$U = V_1 \times V_2 \quad (1)$$
$$V = V_2 \times V_0 \quad (2)$$
$$W = V_0 \times V_1 \quad (3)$$

These vectors U,V, and W are the rows of the adjoint matrix of (V_0, V_1, V_2), thus we are not computing the full inverse which saves as some setup computations. We also compute the determinant $D = det(V_0, V_1, V_2) = U \cdot V_0$ and discard triangles that are back facing $D \leq 0$.

We can now directly test if a homogeneous point $(x, y, 1)$ on the projection plane is inside the triangle by testing if all of the following u,v, or w are ≥ 0:

$$u = U \cdot (x, y, 1) \quad (4)$$
$$v = V \cdot (x, y, 1) \quad (5)$$
$$w = W \cdot (x, y, 1) \quad (6)$$

To perform near/far clipping we compute additional clip edges. In the Plücker test, the distance of the hit computes as $d = D/(u + v + w)$ and can be tested for a valid intersection between the near and far clipping plane through $near \leq d$ and $d \leq far$. We introduce two new edge equations $N = (0,0,D) - near \cdot (U + V + W)$ and $F = far \cdot (U + V + W) - (0,0,D)$ and compute:

$$n = N \cdot (x, y, 1) \quad (7)$$
$$f = F \cdot (x, y, 1) \quad (8)$$

This makes the hit test more consistent as the triangle is now hit if all of the u, v, w, n, and f are ≥ 0. Note that for a near clipping plane at 0, the clip edge can be discarded as the edge got already tested through backface culling.

4. Bit-parallel Scanline Rasterization

We have derived a set of 5 edge equations that could be used in a traditional rasterizer to test if the samples are covered by the triangle. However, as we are not interersted in depth, barycentric coordinates, etc. but only if we hit or not we can perform scanline rasterization and do some additional bit-level optimization that speeds up the computation significantly.

Instead of operating on single samples, scanline rasterization operates on scanlines by computing the interval of covered samples per scanline (see Figure 1). By doing a few more setup operations, we compute for each edge equation the open interval where it is larger or equal than zero the following way:

$$0 \leq U_x \cdot x + U_y \cdot y + U_z \quad (9)$$
$$x \overset{U_x>0}{\geq} -\frac{U_y}{U_x} \cdot y - \frac{U_z}{U_x} \quad \texttt{or} \quad x \overset{U_x<0}{<} -\frac{U_y}{U_x} \cdot y - \frac{U_z}{U_x} \quad (10)$$

The location where the edge equation sign flips is either a lower bound (for $U_x > 0$) or upper bound (for $U_x < 0$) of the triangle. Using some min/max operations, we intersect these lower and upper bounds of the different edge equations to get the covered interval of the triangle with the current rasterline. Unfortunately, being a lower or upper bound depends on the sign of the x-component of an edge equation, making

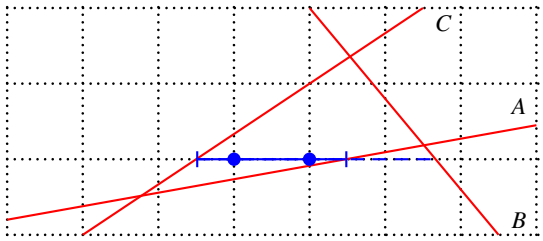


Figure 1: *Scanline rasterization computes per scanline the interval of pixels covered by the triangle. In this figure edges A and B are upper bounds and edge C is a lower bound.*

different versions of the inner loop necessary to achieve best performance.

We choose a 32x32 bit occlusion buffer for the discussions in the rest of the paper. Our algorithm leverage bit level parallelism to efficiently occlude the samples covered by the triangle. We represent a scanline as a 32bit integer which is initially set to all 1s and will at the end contain 0s for occluded samples. We clear bits covered by the triangle by computing a triangle coverage mask that is 1 outside the triangle and 0 inside. For instance, for the scanline interval $[2.5,4.5]$, we first compute a bitvector with the bits 3 to 32 not set $...00000111 = not(-1 << 3)$ and one with bits 0 to 4 not set $...11100000 = (-2) << 4$. Computing the logical or of both bitvectors yields the desired triangle coverage mask for the scanline.

While these bit operation principally work for all intervals when long shifts are available, modern instruction sets handle the shift amount modulo 32. We fix this issue by forcing the lower bound to be ≥ 0 and the upper bound to be ≤ 31. For similar reasons we need to ignore empty intervals by oring the bitvector with the value $-(lower > upper)$ which is all 1s when the interval is empty and all 0s otherwise. As an example, our innermost loop looks for $U_x > 0$, $V_x > 0$, $N_x > 0$ (lower bounds) and $W_x < 0$, $F_x < 0$ (upper bounds) like:

```
for (int y=y0; y<=y1; y++) {
 lower = ceili(max(max(0,u),max(v,n)));
 upper = floori(min(31,min(w,f)));
 V[y] &= -int(lower > upper) |
          ~(-1«lower) | (-2«upper);
 u+=U.y; v+=V.y; w+=W.y; n+=N.y; f+=F.y;
}
```

During setup, we also compute a conservative range $[y0,y1]$ of scanlines that need to get tested. To do so we perform the perspective division and compute the bounding of y locations of the triangle points. Triangles that are partially behind the zero plane complicate this bounding computation and are rare, thus we conservatively iterate over all scanlines of the micro raster in this case.

Unfortunately, there are some special cases left that need to get adressed. If one of the edges is parallel to the x axis the equation 10 is undefined as we divide by zero. We address this issue in our implementation by setting the x component of the edge equation to a small fraction of the y component in this case. This makes the edge slightly misaligned and works even if the y component is very small. If the x and y components are both 0 the triangle lies on the $z = 0$ plane and can be discarded. Handling these special cases puts more burden on the setup stage but does not require any special handling in the innermost loop. It further might result in small light leaks.

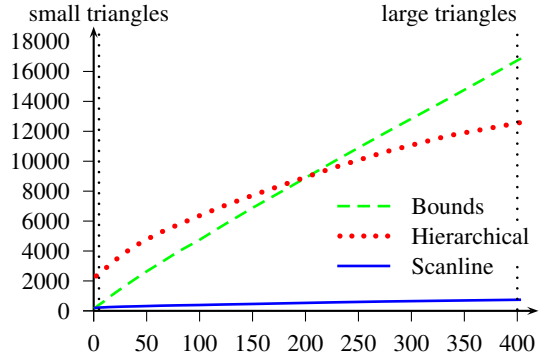


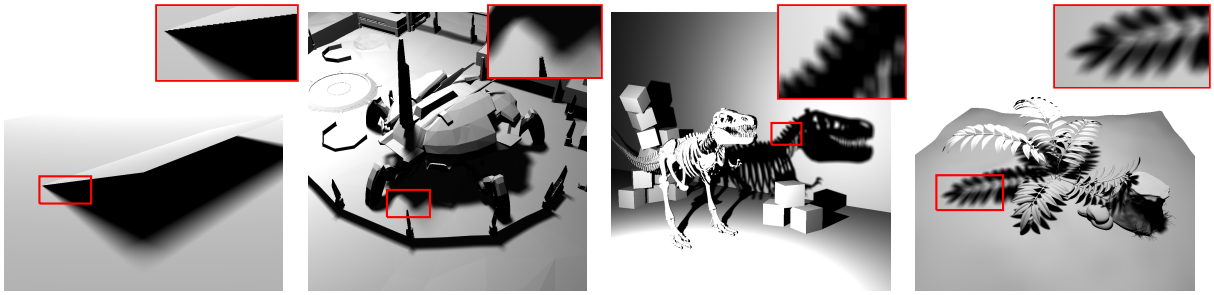
Figure 2: *Cycles required to render triangles of different number of occluded samples using three rasterization kernels into a 32x32 occlusion buffer.*

5. Results

We evaluate the algorithmic complexity of our kernel by comparing execution times using an Intel® Core™ 2 Duo CPU clocked at 2.26 GHz. We have *not* vectorized any kernel to SSE instructions and are using single threaded code compiled with ICC. Our near clipping plane is at 0 effectively removing the near edge equation. We use a micro-occlusion buffer of size 32x32 pixels and to simplify bounding box computations we do not render triangles that cross the zero plane in this comparison. We compare the following rasterization kernels with each other:

- **Bounds**: Evaluates the edge equations for all samples inside the bounding box of the projected triangle.
- **Hierarchical**: Hierarchical Rasterization Algorithm [Gre96] using homogeneous edge equations. A two level hierarchy of 8x8 blocks of size 4x4 performed best.
- **Scanline**: The scanline algorithm described in this paper. We have 16 different versions of the innermost loop as described in Section 4.

Figure 2 shows the number of cycles required for rasterizing triangles of different size. Triangle size is measured in number of occluded samples. Our kernel outperforms all other rasterization kernels for all triangle sizes by up to one order of magnitude. Our kernel has a setup cost of 193 cycles and required 17.2 cycles per iterated scanline. For large triangles covering 400 samples we need 743 cycles to rasterize one triangle into the 32x32 pixel occlusion buffer, which is a throughput of 1.5 samples per cycle. For this workload the hierarchical rasterization algorithm performs second with 12600 cycles, but we are still 17x faster. Our speedup can be explained by our high fill rate due to clearing many samples per scanline at once. The bounding box kernel shows to be very efficient for small triangles that cover only one sample. However, with 142 cycles compared to 193 cycles



(a) Quad (4 triangles) (b) Refinery (300k tris) (c) T-Rex (69k triangles) (d) Fern (212k triangles)

Figure 3: *Scenes rendered with soft shadows at 1024x1024 pixel resolution and using a 32x32 pixel micro occlusion buffer.*

we are only about 35% slower. We suffer in this case by our higher setup cost through computing equation 10.

6. Application to Soft Shadows

To evaluate the micro-rasterization kernel with a realistic workload, we implemented a simple soft shadow algorithm: For every pixel, a frustum is traced from the primary hit location to the light source by culling BVH nodes against the frustum planes. All triangles inside the frustum are rasterized into a 32x32 occlusion buffer using different raster kernels.

For axis aligned geometry the regularity of the sampling pattern can reduce the quality of the shadow to a step function with 32 steps. We worked around this issue by misaligning the sampling pattern (we rotate it slightly) and by shifting the pattern per pixel. The first fix reduces the likelihood of issues to occur and the second transforms artefacts into into less objectionable noise. However, the noise will only occur at locations where artefacts would occur otherwise. As we setup a projection per pixel anyway, these fixes are efficient to perform.

	Quad	Refinery	T-Rex	Fern
Bounds	4.0s	18.5s	9.5s	9.1s
Hierarchical	2.9s	15.1s	10.3s	16.7
Scanline	1.9s	6.7s	4.9s	5.7s

The above table shows the time required to traverse the light source frusta and rasterizing the triangles. We do not include the time required for computing primary visibility, shading, or building datastructures. See Figure 5 for pictures of the scenes. Our kernel speeds up the computation by about a factor of two compared to the fastest of the other kernels. The speedup is not higher as we also need to rasterize many small triangles and suffer from our not optimal traversal kernel that makes up 50% of the computation. An efficient frustum traversal technique [BW09] would help to fix this issue.

7. Conclusions

We have demonstrated the strengths of software rendering by designing an appropiate raster kernel for the particular visibility problem arising in soft shadow computations. Our fast scanline rasterization kernel uses bit level parallelism to speed up the innermost loop by operating on complete scanlines in parallel. We demonstrated that our kernel can speed up the rendering of soft shadows by a factor of 2.

References

[Bou70] BOUKNIGHT W. J.: A procedure for generation of three-dimensional half-toned computer graphics presentations. *Commun. ACM 13*, 9 (1970), 527–536.

[BW09] BENTHIN C., WALD I.: Efficient ray traced soft shadows using multi-frusta tracing. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), ACM, pp. 135–144.

[Eri97] ERICKSON J.: Pluecker Coordinates. *Ray Tracing News* (1997). http://www.acm.org/tog/resources/RTNews/html/-rtnv10n3.html#art11.

[Gre96] GREENE N.: Hierarchical polygon tiling with coverage masks. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 65–74.

[OG97] OLANO M., GREER T.: Triangle scan conversion using 2d homogeneous coordinates. In *HWWS '97: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (New York, NY, USA, 1997), ACM, pp. 89–95.

[Pin88] PINEDA J.: A parallel algorithm for polygon rasterization. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), ACM, pp. 17–20.

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers* (New York, NY, USA, 2009), ACM, pp. 1–8.

[Sea08] SEILER L., ET AL.: Larrabee: a many-core x86 architecture for visual computing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (New York, NY, USA, 2008), ACM, pp. 1–15.

[WKP*97] WINNER S., KELLEY M., PEASE B., RIVARD B., YEN A.: Hardware accelerated rendering of antialiasing using a modified a-buffer algorithm. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 307–316.