Draft – the definitive version appears in JGT 10(3), 2005.

Conservative and Tiled Rasterization Using a Modified Triangle Setup

Tomas Akenine-Möller
Lund University

Timo Aila
Helsinki University of Technology
Hybrid Graphics Ltd.

Abstract

Several algorithms that use graphics hardware to accelerate processing require conservative rasterization in order to function correctly. Conservative rasterization stands for either overestimating or underestimating the size of the triangles. Overestimation is carried out by including all pixels that are at least partially overlapped by the triangle, whereas underestimation includes only the pixels that are fully inside the triangle. None or few algorithms for conservative rasterization have been described in the literature, and current hardware does not explicitly support it. Therefore, we present a simple algorithm, which requires only a small modification to the triangle setup when edge functions are used. Furthermore, the same algorithm can be used for tiled rasterization, where all pixels in a tile (e.g. 8×8 pixels) are visited before moving to the next tile.

1 Introduction

With the advent of programmable graphics hardware, lots of engineering and research work has focused on "porting" specific algorithms so that they can be run on graphics hardware. The argument for this is usually that the performance of graphics hardware grows faster than that of CPUs, and that in the long run, superior performance is achieved or can be expected. Several of these methods need to use *conservative rasterization* to report or generate correct results.

Two slightly different methods are commonly referred to as conservative rasterization. An *overestimated* footprint of a triangle includes all pixels that are at least partly overlapped by the triangle, whereas an *underestimated* footprint includes only the pixels that are completely inside the triangle. Figure 1 shows a comparison between conservative and standard rasterization. Conservative rasterization is not applicable for the actual rendering of a scene. For example, consider two triangles that share an edge. To get the expected result when rasterizing these triangles, one usually considers a pixel to be inside a triangle if the sampling point of the pixel is inside the triangle. This avoids duplicate writes to pixels, and is critical for many techniques, e.g., shadow volume rendering [3] and transparency.

Still, there are several algorithms that need conservative rasterization in order to function properly. For example, Govindaraju et al. [6] use hardware occlusion queries

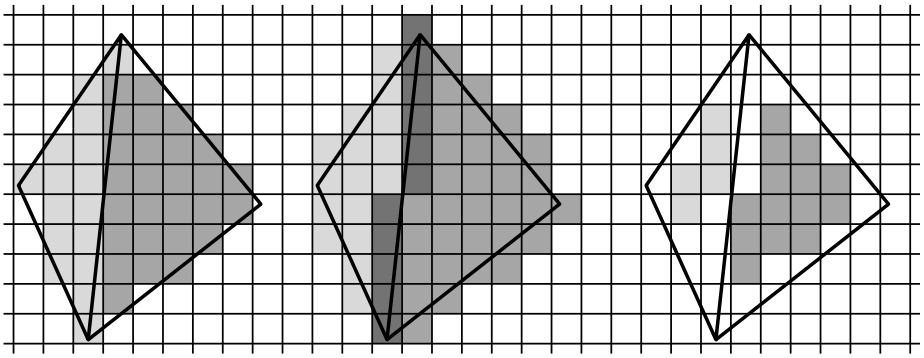

Figure 1: To the left, standard rasterization is shown for two triangles sharing an edge. There is a single sample point at the center of each pixel. In the middle, overestimating conservative rasterization is illustrated. The darkest gray indicates that both triangles have been written to those pixels. To the right, underestimating conservative rasterization is demonstrated.

to compute a potentially colliding set of objects, followed by triangle-triangle intersection testing on the CPU. Exact results can be obtained by using conservative rasterization, whereas a larger resolution can only make the problem less apparent. Koltun et al. [9] use graphics hardware for solving from-region visibility in two dimensions by utilizing a dual ray space. They need to artificially shrink all polygons to compensate the lack of conservative rasterization. Additionally, several other papers utilize conservative rasterization, e.g., Durand et al. [4]. Thus, it should be clear that there is a need for such rasterization algorithms. To our surprise, those algorithms are rarely described in any detail, and furthermore, we are not aware of any graphics hardware that exposes conservative rasterization.

Some authors have implemented conservative rasterization by moving the edges of the triangle either inwards or outwards. Unfortunately, the technique exaggerates the size of an overestimated triangle, especially at sharp corners. This approach also comes with the cost of moving the edges and computing new vertices—a process that can be prone to numerical problems.

In this paper, we present the details of a conservative rasterization algorithm based on edge functions [5, 13]. It can be used in both hardware and software. An advantage of this algorithm is that it requires only a small modification to the triangle setup of a rasterizer, while the rest of the pipeline is left unmodified. Furthermore, we show that the same algorithm can be used for tiled rasterization, which is used to improve memory coherence [10, 11], to do simple forms of culling [2, 12], and for different types of analysis to accelerate rendering [1]. The algorithm allows enabling conservative rasterization separately for each edge.

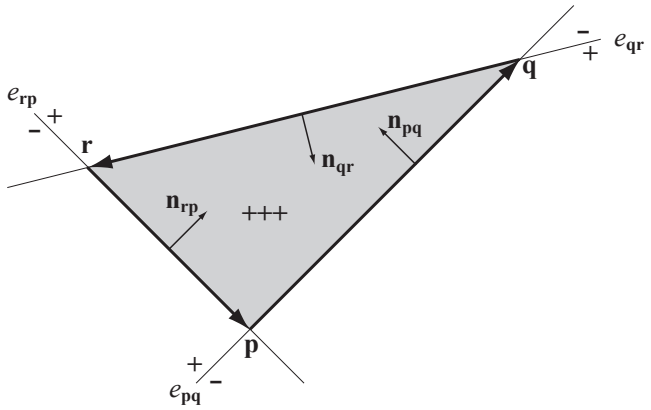

Figure 2: An edge function, e, is a line equation defined by the two vertices of a directed edge. The equation divides the plane into positive and negative half-planes. A separate edge function is defined for each edge of a triangle, and a point is inside the triangle if all three edge functions are positive at the point.

2 Rasterization using Edge Functions

The majority of modern rasterizers use edge functions [13] for rasterizing a triangle. The theory of edge functions is briefly reviewed in this section.

Assume that a triangle $\Delta\mathbf{pqr}$ shall be rasterized, and that the points $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{r}$ are two-dimensional points in screen space. Each edge of the triangle defines an edge function $e(\mathbf{s})$, which is simply a line equation in implicit form. The edge function for the edge $\mathbf{pq}$ is

$$e_{\mathbf{pq}}(\mathbf{s}) = (-(q_y - p_y), q_x - p_x) \cdot (\mathbf{s} - \mathbf{p}) = \mathbf{n} \cdot (\mathbf{s} - \mathbf{p}) = \mathbf{n} \cdot \mathbf{s} + c, \tag{1}$$

where $\mathbf{n} = (n_x, n_y)$ is the normal vector of the edge and $c = -\mathbf{n} \cdot \mathbf{p}$. The *triangle setup* of a rasterizer initializes at least the constants $\mathbf{n}$ and c for each edge, and usually also a set of constants for interpolation of other parameters. A point $\mathbf{s}$ is inside the triangle if $e(\mathbf{s}) \geq 0$ for all edge functions.[1] This assumes that the vertices are in counter-clockwise order, and that the origin is located in the lower left corner. See Figure 2 for a triangle with its corresponding edge functions.

The edge function for a point $\mathbf{s} + \mathbf{t} = (s_x + t_x, s_y + t_y)$ can be written as

$$e(\mathbf{s} + \mathbf{t}) = \mathbf{n} \cdot (\mathbf{s} + \mathbf{t}) + c = \mathbf{n} \cdot \mathbf{s} + e(\mathbf{t}). \tag{2}$$

In conservative rasterization, the triangle setup initializes each edge function with an offset $\mathbf{t}$ that depends on the orientation of the edge. The subsequent rasterization then uses Equation 2 instead of Equation 1.

[1] This is not entirely true since a pixel center $\mathbf{s}$ can lie exactly on an edge shared by two triangles. McCool et al. [10] present a simple solution to this problem based on the signs of n_x and n_y.

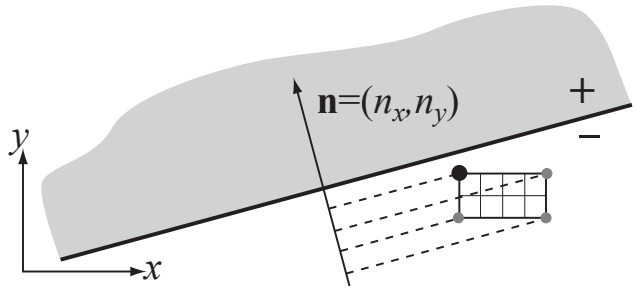

Figure 3: In this example of overestimating rasterization, the tile consists of 4×2 pixels. To determine at which corner of the tile the edge function should be evaluated, the four corners are projected onto the normal vector of the edge. The corner that corresponds to the largest dot product between the normal and the corner is selected. In this case it is the corner with the large black circle. Note that the gray half-plane is potentially inside the triangle.

3 Modification of Triangle Setup

In this section we describe how the triangle setup can be modified to enable conservative rasterization. We will present the general case of rasterizing with tiles that are rectangles of $w \times h$ pixels. Rasterizing with pixels is just the special case of $w = h = 1$.

In tiled rasterization, a tile is excluded if it is fully outside either the axis-aligned bounding rectangle of the triangle or at least one of the three edge functions. As the rectangle test is trivial, we concentrate on the latter. McCool et al. [10] perform this step by testing all four corners of the tile against each edge. In this paper, we develop a less expensive method. Haines and Wallace [7] observe that the intersection of a box and a plane can be detected by considering the two corners of the box that define a line segment that most closely aligns with the normal vector of the plane. Hoff [8] further suggests that only one of these corners is needed for determining if the entire box is either in the positive or negative half-space. Therefore, a single corner of a tile is sufficient for testing if the entire tile is in the negative half-plane of an edge function. In general, the corner needs to be selected separately for each edge.

For overestimating rasterization, the correct corner is the one whose dot product with the normal vector of the edge is the largest, as illustrated in Figure 3. The corner can be efficiently selected by using the orientation of the normal vector. Revisit the example in Figure 3: since $n_y > 0$, one of the upper two corners must be used. Furthermore, $n_x < 0$, and thus it can be concluded that the upper left corner must be used.

In order to correctly initialize Equation 2 for tiled rasterization, the offset $\mathbf{t}$ should be set to either $(0,0)$, $(w,0)$, $(0,h)$ or (w,h) depending on the value of $\mathbf{n}$:

$$t_x = \begin{cases} w, & n_x \geq 0 \\ 0, & n_x < 0 \end{cases}, \qquad t_y = \begin{cases} h, & n_y \geq 0 \\ 0, & n_y < 0 \end{cases} \tag{3}$$

The additional setup code is executed once per edge, and is very inexpensive in

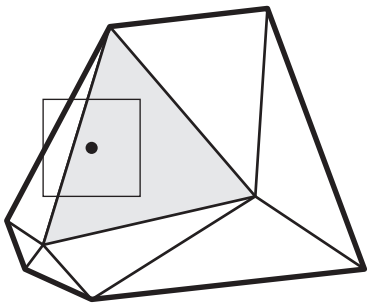

Figure 4: This figure shows why conservative rasterization cannot be used for underestimating the footprint of a mesh. The square illustrates a pixel, and the black dot is the sampling point. Assume that underestimating conservative rasterization is applied only to the silhouette edges of a mesh (bold lines), and traditional rasterization to the interior edges (thinner lines). In this example, the pixel is partially outside the footprint of the mesh, and should not be processed. Unfortunately, the pixel is included when the light gray triangle is rasterized. Preventing this would require more involved methods and global knowledge of the mesh.

terms of computations: two sign comparisons for selecting the corner (Equation 3) and two additions and two multiplications (or shifts when w and h are powers of two) for initializing the edge function ($e(\mathbf{t})$ in Equation 2).

There are many different ways of traversing the pixels/tiles of a triangle, and our technique can be used with any of those as long as edge functions are involved, and thus we omit a discussion of any particular traversal method.

3.1 Limitations of Underestimating Conservative Rasterization

The presented technique can also be used for determining the pixels or tiles that are fully inside a triangle. This underestimated footprint of a triangle is obtained by simply selecting the "diagonally opposite" corner of a tile when initializing the edge function. However, for a mesh one typically wants to apply underestimation only to the silhouette pixels of the footprint of the mesh. This calls for more involved algorithms due to the required global knowledge, as shown in Figure 4.

4 Discussion

The presented algorithm is so simple, and requires so few modifications to an existing rasterizer that we hope the algorithm will find its way to a hardware implementation. The only modification needed into an OpenGL-alike API is the support for selecting one of the two conservative rasterization modes, and ideally a possibility for enabling the selected mode separately for each edge.

With overestimating rasterization, one needs to be careful not to sample depth and other attributes outside the triangle. One solution for this is to silently clamp the

barycentric coordinates to be inside the triangle before proceeding with pixel shading.

It is possible that current graphics chips use the presented technique internally for z_{min}/z_{max}-culling, but no document seems to verify that hypothesis. After we developed and implemented our algorithm in software, it came to our knowledge that a similar algorithm has been briefly mentioned in the US Patent no. $6,480,205$ "Method and apparatus for occlusion culling in graphics systems" by Greene and Hanrahan.[2] Their presentation lacks the details presented here, and furthermore, it is not at all well-known in the computer graphics community.

Acknowledgements Thanks to Jacob Ström, Lauri Savioja and Timo Haanpää for proofreading.

References

[1] Timo Aila, Ville Miettinen, and Petri Nordlund, "Delay Streams for Graphics Hardware," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 792–800, 2003.

[2] Tomas Akenine-Möller and Jacob Ström, "Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 801–808, 2003.

[3] Franklin C. Crow, "Shadow Algorithms for Computer Graphics," *Computer Graphics (SIGGRAPH '77 Proceedings)*, vol. 11, no. 3, pp. 242–248, July 1977.

[4] Frédo Durand, George Drettakis, Joëlle Thollot and Claude Puech., "Conservative Visibility Preprocessing using Extended Projections," *Proceedings of ACM SIGGRAPH 2000*, pp. 239–248, 2000.

[5] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks Jr., John G. Eyles and John Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes," *Computer Graphics (SIGGRAPH '85 Proceedings)*, vol. 19, no. 3, pp. 111–120, 1985.

[6] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha, "CULLIDE: Interactive Collision Detection between Complex Models in Large Environments using Graphics Hardware," *Graphics Hardware 2003*, pp. 25–32, July 2003.

[7] Eric Haines and John Wallace,"Shaft Culling for Efficient Ray-Traced Radiosity" in P. Brunet and F.W. Jansen, eds., *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, Springer-Verlag, pp. 122–138, 1994.

[8] Kenneth E. Hoff III, "A Faster Overlap Test for a Plane and a Bounding Box, 1996. http://www.cs.unc.edu/ hoff/research/vfculler/boxplane.html

[2] They did not patent conservative rasterization but rather an algorithm for occlusion culling.

[9] Vladlen Koltun, Daniel Cohen-Or, and Yiorgos Chrysanthou, "Hardware-Accelerated From-Region Visibility Using a Dual Ray Space," *12th Eurographics Workshop on Rendering*, pp. 204–214, 2001.

[10] Michael D. McCool, Chris Wales, and Kevin Moule, "Incremental and Hierarchical Hilbert Order Edge Equation Polygon Rasterization," *Graphics Hardware 2001*, pp. 65–72, 2001.

[11] Joel McCormack and Robert McNamara, "Tiled Polygon Traversal using Half-plane Edge Functions," *Graphics Hardware 2000*, pp. 15–21, 2000.

[12] Steve Morein, "ATI Radeon HyperZ Technology," *Workshop on Graphics Hardware, Hot3D Proceedings*, August 2000.

[13] Juan Pineda, "A Parallel Algorithm for Polygon Rasterization," *Computer Graphics (SIGGRAPH '88 Proceedings)*, vol. 22, no. 4, pp. 17–20, August 1988.