SVGPU: Real Time 3D Rendering to Vector Graphics Formats

Apollo I. Ellis[1], Warren Hunt[2], and John C. Hart[1]

[1] *University of Illinois Urbana Champaign*
[2] *Oculus VR*

1 Introduction & Previous Work

As the output of a 3-D scene renderer, a vector image (a planar map of 2-D polygons) has several advantages over the common raster image of pixels. Vector images are resolution independent, providing an intermediate representation that can be efficiently rasterized as needed for various applications, from watches to videowalls, for variable resolution head mounted displays or for crisp printing. A high-performance vector renderer would also reduce the bandwidth needed for cloud gaming, where games are played on a server but displayed on a remote network device. In this work we focus on using modern game-oriented graphics techniques to accelerat a real-time triangle mesh vector rendering system that converts a 3-D meshed scene into a planar map of 2-D triangles.

The hidden line problem was well studied decades ago [Sutherland et al. 1974]. Modern approaches mostly utilize Appel's algorithm [Appel 1967] which extracts continuous silhouette components of a meshed model to display, computing a delicate quantitative invisibility to detect and remove hidden silhouette segments.

Robert's algorithm is an even older but more robust approach that simply compares all pairs of scene polygons, clipping and culling their occluded portions [Roberts 1963]. While theoretically slower, this approach maps better to modern GPU hardware, and its quadratic complexity can be moderated through binning.

A similar GPU approach has been explored, based on an all pairs comparison of edges [Auzinger et al. 2013]. Our approach is triangle based, using the silhouette edges only for clipping, and our experiments show it performs better.

2 SVGPU

We use standard game-engine techniques to transform, clip and bin 3-D world-space meshes into 2-D screen-space bins. For each bin, our SVGPU approach efficiently extracts silhouette edges, clips triangles to those edges, and determines which of the resulting clipped triangle are visible.

Silhouette Extraction. The silhouette (formally the visual contour) lies between front and back facing polygons of a watertight mesh. SVGPU detects these edges with a GPU spatial hash table [Lefebvre and Hoppe 2006]. Each of the three edges of every triangle form a hash key from the sorted x, y, z coordinates of each edge's two vertices. We enter each triangle into the hash table at the three locations corresponding to its three edge keys. The silhouette is extracted from pairs of hash table entries whose screen-space normal z coordinates differ in sign.

Clipping. For each bin, SVGPU clips every triangle to every silhouette edge. We first perform a variety of geometric tests to cull unnecessary clip tests from consideration. The result of this stage is a table of triangles and the silhouette edges to which it should be clipped. We then clip each triangle to its list of silhouette edges, using a GPU work queue and double buffering since clipping a triangle can yield additional triangles.

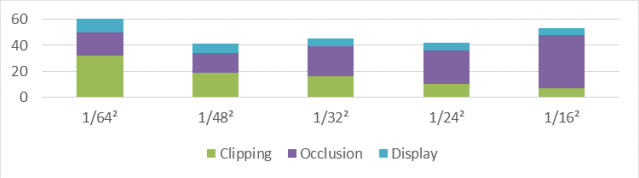
Occlusion. At this stage, every triangle is either completely visible or completely occluded (by one or more triangles). We apply the plane separation theorem to all pairs of triangles in the bin to cull any occluded triangles.

3 Results & Conclusion

SVGPU is about three times faster than previous GPU vector rendering results on Bunny [Auzinger et al. 2013].

Model	in → out	bins (non-∅)	max | ave. $\frac{\triangle}{\text{bin}}$		t
Bunny	69K → 280K	2,304 (33%)	698	139	41
Dragon	202K → 950K	2,304 (19%)	1,740	555	222
Armadillo	212K → 1.2M	1,152 (44%)	2,076	356	256
Buddha	293K → 1M	2,304 (43%)	712	200	140

The Dragon and Armadillo models are about $3\times$ the size of the bunny, but require roughly $6\times$ as much time to render, indicating some impact of the quadratic all-pairs triangle occlusion step.



The above bargraph, for Bunny, is useful for tuning bin size (as a fraction of screen size). Larger bins require less clipping but quadratically more occlusion tests. Silhouette extraction runs required less than one ms of total time.

We have shown that the GPU can implement a vector rendering system suitable for generation of line art or for rasterization with improved edge antialiasing and visible surface processing. By binning geometry into small screen tiles, about $1/48^2$ of the screen size, we achieve an optimal domain decomposition that distributes a parallel clipping workload evenly while limiting the impact of an all-pairs quadratic triangle occlusion test. The result yields about a $3\times$ improvement over the state of the art.

References

APPEL, A. 1967. The notion of quantitative invisibility and the machine rendering of solids. *Proc. 22nd ACM Natl. Conf.*, 387–393.

AUZINGER, T., WIMMER, M., AND JESCHKE, S. 2013. Analytic visibility on the gpu. *Computer Graphics Forum (Proc. Eurographics) 32*, 2 (May), 409–418.

LEFEBVRE, S., AND HOPPE, H. 2006. Perfect spatial hashing. *Proc. SIGGRAPH, ACM TOG 25*, 3, 579–588.

ROBERTS, L. 1963. Machine perception of three-dimensional solids. Tech. Rep. TR 315, Lincoln Laboratory, MIT.

SUTHERLAND, I. E., SPROULL, R. F., AND SCHUMACKER, R. A. 1974. A characterization of ten hidden-surface algorithms. *ACM Comput. Surv. 6*, 1, 1–55.