# On Depth-Testing Wide Outlines

László Szécsi, Balázs Hajagos and Tamás Umenhoffer

Budapest University of Technology and Economics, Hungary

**Abstract**

*We investigate the visibility problem of crease outlines in stylistic and engineering rendering. We discuss why the problem is ill-defined, and offer two different, consistent formulations that lead to artistic and technical drawing styles. We propose real-time, flicker-free GPU algorithms for both problems.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

## 1. Introduction and previous work

Outlines are used in both artistic and technical depictions to emphasize discontinuities. Drawn outlines include *silhouettes* at image-space boundaries of object surfaces (separating front- and back-facing parts in object space), and *creases* at discontinuities of the surface normal. Outlines often must be rendered as wide, textured, semi-transparent strips without seems, folds or flickering.

Outline detection can be performed in **image space**, which is simple but inconsistent as it loses subpixel detail [IFH*03], or in **object space**. On manifold triangle meshes, silhouettes can be found by processing all triangles [HZ00], e.g. in a geometry shader. Creases form *halfedge* loops delimiting *smoothing groups* of triangles. These can be identified as a part of the triangle adjacency computation.

Compositing object-space outlines with surfaces in a 3D scene is a major challenge. It can be seen as a hidden line removal problem that can be solved geometrically in **object space** [App67], which is expensive, even if accelerated by an image-space lookup [MAH00], or using image space **depth testing** [IHS02], where filtering must be used to alleviate instabilities. One technique we propose is based on the latter approach. The outline, rendered as a wide strip, will be considered visible in all its width where its centerline is not hidden. This is similar to an artist painting strokes on paper, lifting the brush roughly where outlines would go behind objects. We will refer to this approach as the *Wide Outlines with Approximate Rendering Technique*, or *WO/ART*.

In technical drawings, enhancing feature edges must not modify shape contours or interfere with exact occlusion. Drawing the polygon wireframe [BNGL08] only where the object itself is visible—thus, practically, onto its surface—is such a technique. We extend this from polygons to non-planar, potentially self-occluding smoothing groups of arbitrary topology to get on-surface crease edge rendering. We will refer to this approach as *Wide Outlines with Precise Rendering of Occlusions*, or *WO/PRO*.

## 2. The wide outline visibility problem

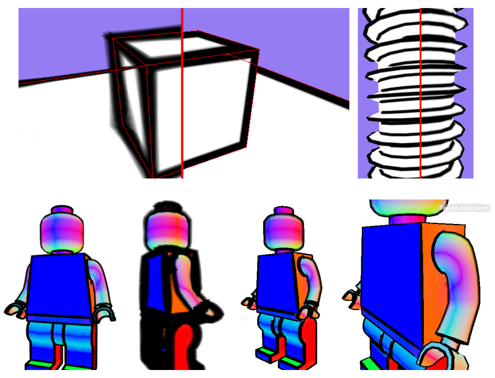

**Figure 1:** *Outline comparision (up) and two renders both with WO/ART and WO/PRO (down).*

A wide outline strip is a 3D surface itself. It has to be aligned with the object outlines when projected onto the image plane, but it does not have a well-defined depth, save for its centerline. There is no universal method to orient the strips so that depth compositing provides an acceptable image. One option is to place all strips on the image plane, draw

DOI: 10.2312/conf/EG2013/posters/015-016

them in an arbitrary order, but hide them where the centerline is not visible, as in the WO/ART approach. This hides inconsistencies behind artifacts that we accept as features of brush art (Figure 1). Another solution, that of WO/PRO, is to abandon the part of the strip extending over the surface, and project the internal half onto it as a decal. This makes outline geometry a subset of the original surface geometry, making simple depth testing consistent. The projection of the outline strip onto the surface, however, is far from trivial, as self-occlusions present another ill-defined problem. When is a surface point—coinciding with an outline strip in image space—part of the outline, and when is it on a surface occluding it? Intuitively, the strip should keep its topology in object space, but its projected boundary in object space is as topologically complex and prone to numerical inaccuracies as silhouettes themselves. Resolving these complexities would be possible using object space processing, but doing so in a temporally coherent way is difficult. Thus, we propose a technique that computes the on-surface, but screen-geometry-factor-scaled distance of a surface point to the outline edge, and classifies it as part of the outline only if the distance is below the desired outline width. This models a rubber band attached to the creases, which attempts to stretch along the surface as far as to provide uniform screen space width, but maintains its topology. To compute on-surface distance, we first use a shortest path algorithm on the edge graph to find the nearest creases for all mesh vertices, then, for any surface point on a triangle, we can evaluate distance to all creases stored in the three vertices.

## 3. GPU implementation

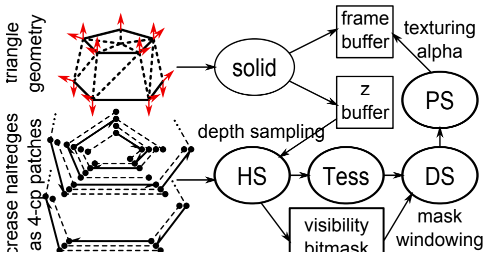

**Figure 2:** *WO/ART operation.*

**WO/ART** can be implemented (Figure 2) as a two-pass method. The first pass renders solid surface geometry, also producing a depth buffer with a small depth slope bias. The second pass renders crease halfedges—augmented with adjacent vertices along the crease halfedge loop—as four-control-point patches. The *constant hull shader* computes and outputs crease normals at vertices, and samples visibility using *comparison filtering* along the three-segment line strip, producing an output *visibility bitmask*. The number of samples depends on the screen-space size of the halfedge, but we limited it to 64 to fit the bitmask into two integers. The tessellator is set up to convert the patch to a quad strip with a linear tessellation factor along the $v$ axis roughly corresponding to the sample density. The *domain shader* positions the strip vertices offsetting them along the interpolated crease normals, at a distance proportional to their visibility. Vertex visibility is computed by averaging relevant bits of the visibility mask. Completely hidden parts will have zero width, the strip will taper off where visibility vanes, and numerical inconsistencies are hidden by averaging multiple visibility samples. The pixel shader applies texturing with alpha-blending.

For the **WO/PRO** method, all vertices on creases store the crease normals. In the first pass, we use a compute shader to propagate these to non-crease vertices, so that all of them store the nearest crease normals and the distance to the respective crease edges. In the second pass, mesh geometry is rendered using a geometry shader aggregating all crease data for a triangle, and a pixel shader that evaluates crease proximity.

## 4. Results and limitations

We have proposed two solutions to the outline visibility problem, and described GPU implementations. These run at 200-300 FPS on a notebook GPU. Some artifacts with joining silhouettes to creases remain to be resolved, just like the parameterization, texturing and stylization of the outlines. Exploiting self-similarity [BCGF10] is promising for WO/ART, but the WO/PRO case is more challenging.

This work has been supported by OTKA PD-104710.

## References

[App67] Appel A.: The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 1967 22nd national conference* (1967), ACM, pp. 387–393. 1

[BCGF10] Bénard P., Cole F., Golovinskiy A., Finkelstein A.: Self-similar texture for coherent line stylization. In *Proceedings of the 8th Symposium on Non-Photorealistic Animation and Rendering* (2010), ACM, pp. 91–97. 2

[BNGL08] Bærentzen J., Nielsen S., Gjøl M., Larsen B.: Two methods for antialiased wireframe drawing with hidden line removal. In *Proceedings of the 24th Spring Conference on Computer Graphics* (2008), ACM, pp. 171–177. 1

[HZ00] Hertzmann A., Zorin D.: Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 517–526. 1

[IFH*03] Isenberg T., Freudenberg B., Halper N., Schlechtweg S., Strothotte T.: A developer's guide to silhouette algorithms for polygonal models. *Computer Graphics and Applications, IEEE 23*, 4 (2003), 28–37. 1

[IHS02] Isenberg T., Halper N., Strothotte T.: Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. In *Computer Graphics Forum* (2002), vol. 21, Wiley Online Library, pp. 249–258. 1

[MAH00] Markosian L., Adviser-Hughes J.: *Art-based modeling and rendering*. Brown Univ., 2000. 1