Stylized Haloed Outlines on the GPU

Jörn Loviscach*
Hochschule Bremen

1 Introduction

Methods to emphasize outlines form a major branch of non-photorealistic rendering. Recently, McGuire and Hughes [2004] have demonstrated how to render stylized outline strokes entirely with graphics hardware, extending an approach proposed by Card and Mitchell [2002]. To this end, not only the original 3D mesh but also additional geometry is sent to the graphics card. The additional geometry contains degenerated quadrangles along all edges of the original mesh. A special vertex shader tests for each edge if it forms a border between a front- and a back-facing polygon. If so, the zero-area quadrangle is extruded into a visible fin.

We present a related approach with the following improvements, see Fig. 1: The outline strokes are visible both inside and outside the silhouette and possess soft halos; smoothly curved outlines are generated even from low-resolution meshes; complex outline styles such as sketch-like strokes are generated; the amount of data per vertex is reduced.

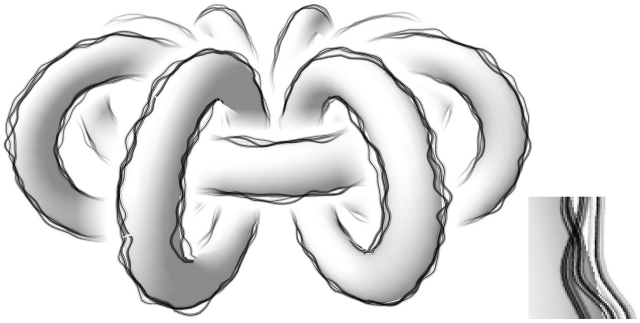

Figure 1: A stylized rendering with sketch-like strokes and soft halos. The inset shows a detail of the strokes.

2 Extraction and Rendering of Silhouettes

In a preprocessing step, the edges of the original mesh are collected and used to build an edge mesh. This is rendered several times per frame with different settings. Edges that form a part of the silhouette are extruded inside and/or outside through a vertex shader, yielding quadrangles oriented toward the viewer. These serve as depth mask or as canvas for the stylized stroke.

We use a sequence of render operations on the main buffer and one additional off-screen buffer. The off-screen buffer is mainly used to collect depth and intensity information about the outer, soft part of the halo. To blend soft halos together, we employ the `Max` operation instead of linear alpha blending. This effectively suppresses the artifacts of the silhouette extraction such as zigzag structures [Isenberg et al. 2002], which would otherwise become visible behind semi-transparent polygons.

The original objects are rendered into the main buffer first normally and then shrunk along the vertex normals through a vertex shader.

*e-mail: jlovisca@informatik.hs-bremen.de

The latter form is used as depth mask, which allows to draw strokes that extend to both the inside and the outside of the silhouette. A pixel shader paints the quadrangles employing 1D texture lookups to control the profile of a stroke. In Fig. 1 the pixel shader adds three different strokes in a single pass, applying a different offset to each stroke. The offset is read from a texture using image-space position as *uv* coordinates.

In addition, the pixel shader introduces a cubic deformation to the 2D screen projection of the silhouette line so that the strokes appear smooth, no longer polygonal, see Fig. 2. The deformed curve runs perpendicular to the screen projection of the vertex normals and thus is tangent continuous at the vertices.

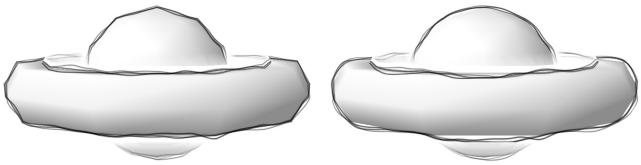

Figure 2: Using cubic deformation, even coarsely tessellated meshes (left) lead to smooth outlines (right).

Both to save memory bandwidth and to free per-vertex attributes for other uses, we strove to store the data in a compact format. The face normals of the two adjacent faces are used to decide if an edge is on the silhouette. A little algebra allows to store the two face normals via four floating point values with no need for trigonometric functions. The face normals are perpendicular to the edge, so that its direction is already determined by them. Thus, the vector from one vertex of an edge to the other can be stored with just one additional floating point value. In total, for every edge of the original mesh a quadrangle is stored that uses additional vertex attributes amounting to 52 floating point values, where McGuire and Hughes [2004] use 76 floats with less complex stylization.

3 Conclusion

We have presented a method to render complex outlines with soft, semi-transparent halos entirely on the GPU. The prototype has been implemented in C# and HLSL. It can render 3D objects containing tens of thousands of edges in real time on current graphics cards.

References

CARD, D., AND MITCHELL, J. L. 2002. Non-photorealistic rendering with pixel and vertex shaders. In *ShaderX: Vertex and Pixel Shaders Tips and Tricks*, W. Engel, Ed. Wordware, 319–333.

ISENBERG, T., HALPER, N., AND STROTHOTTE, T. 2002. Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. *Computer Graphics Forum 21*, 3, 249–258.

MCGUIRE, M., AND HUGHES, J. F. 2004. Hardware-determined feature edges. In *Proc. NPAR 2004*, 135–147.