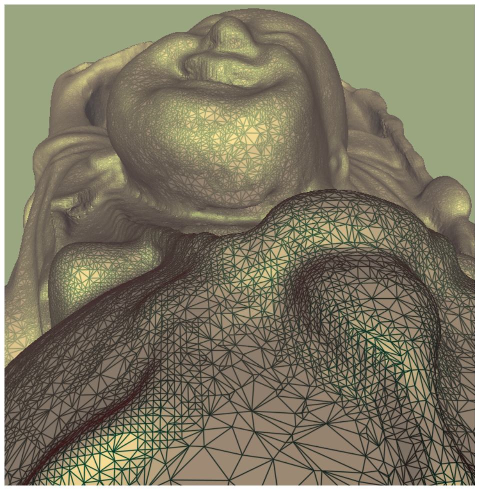
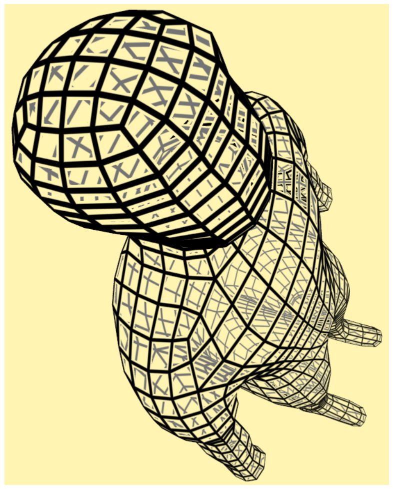
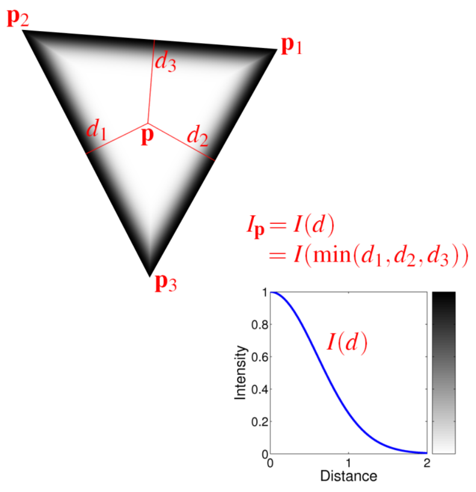
Single-pass Wireframe Rendering

Andreas Bærentzen[1] Steen L. Nielsen[1,2] Mikkel Gjøl[3] Bent D. Larsen[1,3] Niels Jørgen Christensen[1]

[1] Informatics and Mathematical Modelling, Technical University of Denmark, [2] Flux Studios, [3] Dalux







The standard procedure for wireframe drawing with hidden line removal on a graphics card has not changed for a long time: First the filled polygons are drawn, laying down the depth buffer. Next, the polygon edges are drawn as lines with a small depth offset to ensure that polygons do not occlude their own edges.

The depth offset is required because the procedure for rasterizing lines is not exactly the same as the one for rasterizing polygons. Consequently, when rasterizing a polygon edge as a line, a given fragment may have a depth value that is different from when the corresponding polygon itself is rasterized. This leads to stippling artefacts. However, adding an offset is not an ideal solution since this offset can result in disocclusions of lines that should be hidden. Moreover, there is usually stippling in any case near steep slopes in the mesh where a very large offset is sometimes required. The only real fix is a slope dependent offset but that tends to make disocclusion problems much worse. A few authors have proposed improved techniques, but either these are not intended for modern graphics hardware [Wang et al.] or they incur a performance hit [Herrel et al.].

Our solution does not use the line primitive at all. Instead polygon edges are drawn directly as a part of polygon rasterization. For each fragment, we compute the shortest distance to the edges of the polygon and map that distance to an intensity value, I, as shown in the figure (right). This mapping is not just a step function but a smooth function, $I = \exp_2(-2d^2)$ which amounts to antialiasing by *prefiltering*.

This method works for convex polygons (triangles or quads most likely), and it suffers from none of the artefacts associated with the offset based methods, but it does have one drawback. If a polygon does not have a neighbouring polygon (e.g. near a hole or a silhouette) the line is drawn from one side only. This means that silhouette lines are thinner and not antialiased. However, in practice the quality is still far better than using the offset based method, and the performance is almost invariably better. On a Geforce 7800 GTX the Happy Buddha mesh was rendered at 25 fps using our method and only 5 fps using the offset based method. Thus, our method seems to be particularly well suited for the rendering of dense triangle meshes such as the increasingly common laser scanned models. Furthermore, many variations of the method are possible. In fact, the two images in the figure show the method using attenuation of line intensity (left) and thickness (center). In the center image, alpha testing was used to remove the interior of the quads.

Implementation

Observe that the 2D distance from a point to a polygon edge is an affine function. Such functions are reproduced by linear interpolation. For this reason, we can compute the distance at each vertex and simply interpolate linearly. Thus, for each vertex of, say, a triangle we must send the other two vertices as attributes in order to compute the line distance. The graphics card will do the interpolation, but there is one problem: Perspective correct interpolation is currently hardwired into the interpolation used by modern graphics cards. It is possible, however, to negate this interpolation by premultiplying the distance (computed in a vertex program) with the w value. The interpolated distance is then postmultiplied in the fragment program by the interpolated $1/w$.

It is worth mentioning that Direct3D 10 offers an even simpler solution: It is possible to switch off perspective correction in D3D 10. Moreover, D3D 10 supports *geometry shaders* which are executed after vertex shaders but before rasterization. In such shaders, all vertices of a triangle can be accessed making it possible to compute the distance at each vertex without sending additional attributes per vertex. A similar shader type (called *primitive shaders*) is planned for OpenGL 3.0.

References

Herrell, R., Baldwin, J., and Wilcox, C. 1995. High-quality polygon edging. *IEEE Comput. Graph. Appl. 15*, 4, 68–74.

Wang, W., Chen, Y., and Wu, E. 1999. A new method for polygon edging on shaded surfaces. *J. Graph. Tools 4*, 1, 1–10.

[1] (jab|njc)@imm.dtu.dk
[2] steen@flux-studios.com
[3] (mig|bdl)@dalux.dk