Curved PN Triangles

Rick van Veen (s1883933) Laura Baakman (s1869140)

December 9, 2015

Point normal triangles, or PN triangle for short, were introduced by [1] to improve the visual quality of existing triangle-based art in realtime entertainment, such as computer games. An important aspect of this approach is, that it uses the existing informations that is available per triangle primitive. Therefore no changes to authoring tools and hardware designs are needed while providing a smoother, although not everywhere tangent continuous, silhouette and more organic shapes (see figure 1). We provide a brief review of the construction of the PN triangle, advantages and disadvantages, and where this fits in the hardware pipeline.

The PN triangle is specified by its normal and geometric component. The geometry of a curved PN triangle is defined by a cubic bézier patch which is shown in (1). Consequently the flat triangle needs to be converted to a control net or control polyhedron of coefficients. Three sets of coefficients b_{ijk} are grouped together: Vertex coefficients, Tangent coefficient and the center coefficient.

$$b(u,v) = \sum_{i+j+k=3} b_{ijk} \frac{3!}{i!j!k!} u^i v^j w^k \tag{1}$$

To obtain the control coefficients which describe the cubic control net, the first step is to spread out the intermediate control points using the formula shown in (2). The vertex coefficients are the intermediate coefficients that lay on the original vertices. The tangent coefficients are derived by projecting the intermediate control points, that lie on the edges of the original flat triangle, onto the plane that is defined by the vector that is orthogonal to the closest by vertex normal. The only vertex left is the center coefficient, which lies in the center of the triangle. This position is obtained by computing the average of the final tangent coefficients and raising it by the the difference between the tangent and vertex coefficients divided by two (see original paper by Vlachos et al.).

$$b_{ijk} = (iP_1 + jP_2 + kP_3)/3 \tag{2}$$

By using cubic bézier patches for the geometry, has as consequence that quadratically varying normals are needed to capture the inflection points (see (3)). These normals can be computed by reflecting the averaged end-normals on/by/bleh the plane perpendicular to the edge.

$$n(u,v) = \sum_{i+j+k=2} n_{ijk} u^i v^j w^k \tag{3}$$

Sharp edges... blunt edged who cares...

As stated in the first paragraph PN triangles provide better visual quality of existing triangle based art, but this is not always the case as observed by Vlachos et al.. If two patches share a vertex but not the corresponding normal, there is no strategy based only on the vertex and normal information (and respecting the vertex and normal information) local to one patch at a time to generate the same curve tangent for either patch independent of the normal information of the other.

How do they solve this?

Pipeline... of dit moet eerder

It is important to note that the in the time (2001) of the paper by Vlachos et al. not the same programmable shaders existed. In the paper they provide an overview of a pipeline that suggest to do the assembly of the control nets and normals in a preprocessing stage, which is done on the CPU. Today the process could in total be controlled on the GPU by making use of the so called tessellation shaders.

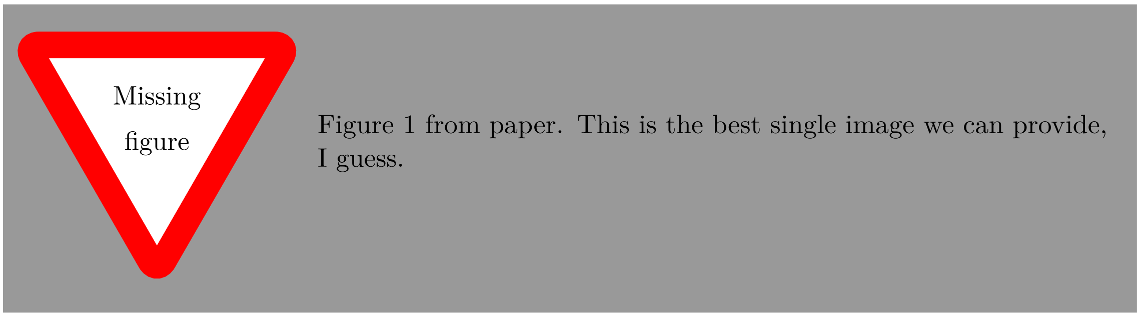

Figure 1 from paper. This is the best single image we can provide, I guess.

Figure 1: *From left to right (a) Input triangulation, (b) Gouraud shaded input triangulation, (c) geometric component of the PN triangles (shaded according to surface normal variation), and (d) curved PN Triangles (shaded with independently constructed quadratically varying normals)*

Conclusion: huilen met de pet op dit.

MEEEEH

References

[1] Alex Vlachos, Jörg Peters, Chas Boyd, and Jason L Mitchell. Curved pn triangles. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 159–166. ACM, 2001.