Pixel-Shader-Based Curved Triangles

Michael Schwarz*
University of Erlangen-Nuremberg

Marc Stamminger†
University of Erlangen-Nuremberg

1 Introduction

Since real-world objects often consist of curved surfaces, it is desirable to be able to render such surfaces in real-time. One example of adequate primitives are curved PN triangles [Vlachos et al. 2001], a special kind of cubic triangular Bézier patches.

To render a surface described by PN triangles, they can be subtriangulated such that the resulting mesh approximates the surface reasonable well. While subdivision to a fixed level is easy and supported by some ATI hardware, it is usually ill-suited if the screen-space sizes of the PN triangles vary considerably. Hence a software-based adaptive tessellation needs to be performed, which however incurs a certain runtime overhead—not at least since it has to avoid cracks between adjacent PN triangles subtriangulated to different LODs. An alternative rendering approach is given by directly ray-tracing the surface, which is usually done by either Bézier clipping [Roth et al. 2001] or Newton iteration [Stürzlinger 1998].

In this work, we propose rendering curved PN triangles by drawing their bounding volumes and performing a ray/PN-triangle intersection test for each fragment in the pixel shader. This not only avoids having to adapt the LOD of the surface's tessellation dynamically but also keeps the memory footprint and the CPU load low.

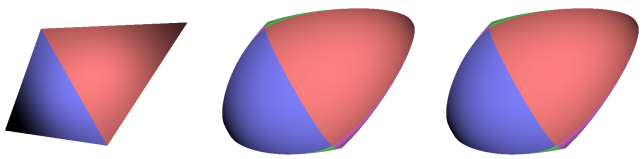

Figure 1: Square dipyramid (left), corresponding PN triangles rendered by subtriangulation (center) and with our approach (right).

2 Method Description

Our pixel shader performs up to four Newton iterations to find an intersection. As starting points for the iterations we take positions near the three corner vertices as well as the center. We project each of these points onto the ray corresponding to the current fragment and sort them by the distance of their projection to the ray's origin (method A). Alternatively, we first sort them by their distance to the ray and then order the first two like above (method B).

A Newton iteration is performed for the first starting position. If it doesn't converge within $N_{\max}$ iteration steps or if the surface is intersected outside the visible parameter domain, a new Newton iteration is initiated for the next starting point. Once a hit is reported, the remaining starting positions are skipped and the lighting is done. In case all four Newton iterations fail, we discard the fragment.

To get a tight but simple bounding volume, we perform a 1-to-4 subdivision of the Bézier triangle and take the triangular prism that encompasses the control points of the four resulting Bézier triangles, exploiting the convex hull property.

*e-mail: schwarz@cs.fau.de
†e-mail: stamminger@cs.fau.de

To take advantage of early depth culling and thus avoid the unnecessary execution of the pixel shader for occluded fragments, we don't adapt a fragment's depth to the actual Bézier surface point but keep the one of the rendered bounding prism's face. Though this approximation might introduce occlusion errors in some rare cases, they can easily be ruled out by a simple preprocessing step.

3 Results and Future Work

We render a single cubic Bézier triangle in arbitrary positions on an NVIDIA GeForce 7800 GT with $N_{\max} = 10$ and a viewport of size 800×600 at constantly more than 20 fps. Figure 2 shows some examples along with the total number of performed Newton iteration steps for both method A (middle) and B (right) of ordering the starting points. Note that the iteration count is rather low for the front-facing regions; thus restricting to them would allow to decrease $N_{\max}$. Moreover, models ranging from simple (cf. figure 1) to more complex ones were successfully tested. However, the frame rates often dropped from real-time to interactive with 5–10 fps.

Currently, the processing of fragments that get discarded because they don't belong to the PN triangle is the most time-consuming part. Their negative impact on performance is even amplified by the GPU's rather coarse thread granularity concerning dynamic flow control. To tackle this problem and increase the frame rate, we intend to investigate the subdivision of highly curved input PN triangles, such that the resulting ones are flat enough to safely select a smaller value for $N_{\max}$. In addition, tighter and more complex bounding volumes should prove favorable.

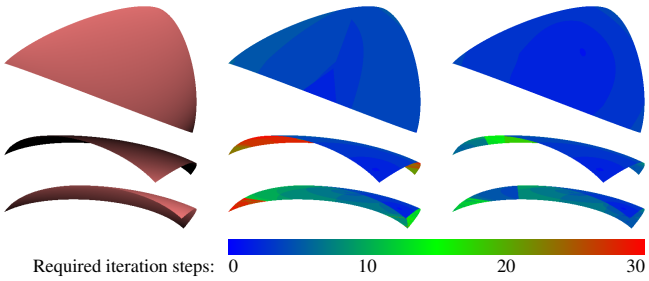

Figure 2: Single cubic Bézier triangle in different positions.

References

ROTH, S. H. M., DIEZI, P., AND GROSS, M. H. 2001. Ray tracing triangular Bézier patches. *Computer Graphics Forum 20*, 3 (Sept.), 422–432.

STÜRZLINGER, W. 1998. Ray tracing triangular trimmed free form surfaces. *IEEE Transactions on Visualization and Computer Graphics 4*, 3 (July–Sept.), 202–214.

VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. L. 2001. Curved PN triangles. In *Proceedings of the 2001 ACM Symposium on Interactive 3D Graphics*, 159–166.