Heuristic Backface Culling of Animated Subdivision Surfaces

by Tomas Möller and Carlo H. Séquin, University of California at Berkeley

Subdivision surfaces are gaining more widespread use in the computer graphics community due to their desirable properties, such as 1) generating smooth surfaces from meshes with arbitrary topology, 2) avoiding cracks easily (also during animation), 3) simplicity and elegance. However, algorithms for efficient rendering of animated subdivision surfaces, where the vertices of the initial mesh can move freely from frame-to-frame, seem to be non-existing. For such surfaces, all points of the final mesh are usually recomputed for each frame.

We present our exploration of a simple heuristic technique that not only culls backfaces, but also avoids computing a large number of points that do not contribute to any polygon that appears in the final image. The basic idea is illustrated in 2D in figure 1, where we consider the backfacing line segment A. Since both its neighbors (B & C) also are backfacing, we can prove[1] that the subdivided line segments A′ & A″ also will be backfacing, and thus we need not compute them. C″ & B′ are also provably backfacing, but they are needed to properly subdivide their neighbors C′ & B″, which may become frontfacing. In the 3D case,

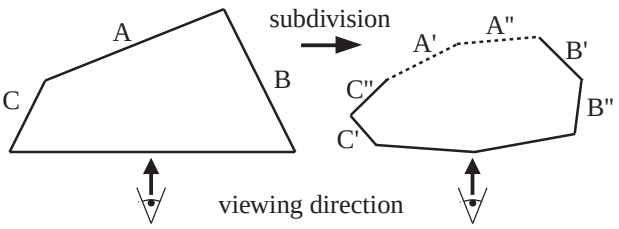

Figure 1: 2D example.

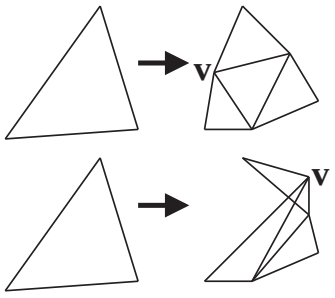

Figure 2: Top: what normally happens. Bottom: triangle orientation flip.

we concentrate on Loop's subdivision scheme [2, 3]. Looking at a triangle Δ in a triangle mesh, we call the triangles that share one edge with Δ *e-neighbors*, and the triangles that share only one vertex with Δ

[1]This obviously depends on which subdivision scheme is used. For this example, we used cubic uniform B-spline refinement.

v-neighbors. The heuristic that we use is: If a triangle Δ and all of its neighbors (v & e) are backfacing, then all new subdivided triangles "inside" Δ are also assumed to be backfacing. We call such triangles *strongly backfacing*, and a triangle with all neighbors being strongly backfacing is called *totally backfacing*. This two-level scheme of isolating the "irrelevant" back zone is needed because of the reach of the influence of Loop's subdivision scheme based on nearest neighbors. In each subdivision step we compute the status (front, back, strong, totally back) of all triangles that have not yet been culled. Triangles that are totally backfacing are excluded from all further computations. This conservative culling is necessary because part of the invisible neighborhood of the front side is needed for correct subdivision, i.e., strongly backfacing triangles must be subdivided because the first "row"of backfacing triangles need their neighbors.

An example of the algorithm in action is shown in figure 3. To get even better performance, we use

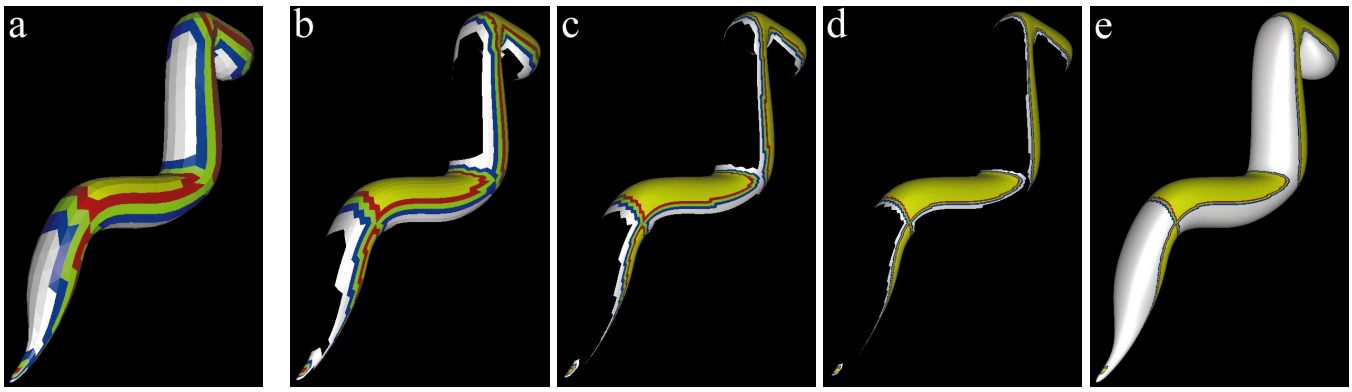

Figure 3: The camera that does backface culling is located in the upper right back. $a)$ initial worm mesh (color codes: white=totally back, blue=strong, green=back, red=front, yellow=totally front), $b)$ after one subdivision step: the white triangles from the previous step have been discarded, $c)$ after two subdivision steps, $d)$ after three subdivision steps (which is what we send to the graphics pipeline), $e)$ what normally is sent to the graphics pipeline without our technique.

the same idea for frontface determination, i.e., we label frontfacing triangles with all their neighbors front-facing as *totally frontfacing*. The new subdivided triangles of a totally frontfacing triangle are all totally frontfacing, and the status computation can be omitted.

Mathematically, there are no guarantees that meshes with grossly varying facet sizes cannot produce "flipped" facets with orientations almost opposite to that of their neighbors (figure 2). Such sign reversals,

if not present in the initial control mesh, are clearly unwanted artifacts that should be suppressed, and that is exactly what our heuristic does.

Results: For a high enough degree of subdivision, e.g., when each resulting primitive must be smaller than half a pixel [1], we have observed reductions in rendered polygon count close to the theoretical limit of 50% and also total speed-ups of close to 100%. The memory overhead for status book-keeping is one byte per triangle, which is typically negligible, and well worth it, since we now need to store only 50% of the triangles. The code overhead is 5-10% of the total execution time.

Acknowledgement: Thanks to Jordan Smith for providing the SLIDE implementation environment, and to Denis Zorin and Maryann Simmons for useful dicsussions.

References

[1] DeRose, T., M. Kass, and T. Truong, “Subdivision Surfaces in Character Animation”, *Computer Graphics (SIGGRAPH 98 Proceedings)*, pp. 85–94, July 1998.

[2] Hoppe, H., T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, “Piecewise Smooth Surface Reconstruction”, *Computer Graphics (SIGGRAPH 94 Proceedings)*, pp. 295–302, July 1994.

[3] Loop, C., *Smooth Subdivision Based on Triangles*, Master’s Thesis, Department of Mathematics, University of Utah, August 1987.