Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2014)
Vladlen Koltun and Eftychios Sifakis (Editors)

An Adaptive Virtual Node Algorithm with Robust Mesh Cutting

Yuting Wang[†] Chenfanfu Jiang[‡] Craig Schroeder[†] Joseph Teran[†]

[†]Department of Mathematics [‡]Department of Computer Science
University of California, Los Angeles

Abstract

We present a novel virtual node algorithm (VNA) for changing tetrahedron mesh topology to represent arbitrary cutting triangulated surfaces. Our approach addresses a number of shortcomings in the original VNA of [MBF04]. First, we generalize the VNA so that cuts can pass through tetrahedron mesh vertices and lie on mesh edges and faces. The original algorithm did not make sense for these cases and required often ambiguous perturbation of the cutting surface to avoid them. Second, we develop an adaptive approach to the definition of embedded material used for element duplication. The original algorithm could only handle a limited number of configurations which restricted cut surfaces to have curvature at the scale of the tetrahedron elements. Our adaptive approach allows for cut surfaces with curvatures independent of the embedding tetrahedron mesh resolution. Finally, we present a novel, provably-robust floating point mesh intersection routine that accurately registers triangulated surface cuts against the background tetrahedron mesh without the need for exact arithmetic.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling;

1. Introduction

The virtual node algorithm was developed to model topological changes defined by cuts in a tetrahedron mesh that do not lie on mesh facet boundaries [MBF04]. Although simply splitting a tetrahedron mesh along element faces is the simplest means of changing mesh topology, it limits the paths of cutting surfaces to be aligned with the facets of the original grid. The VNA was designed to generalize this approach to cuts that follow more arbitrary geometric paths. Avoidance of expensive tetrahedral re-meshing approaches that rebuild the mesh to respect propagating cuts was the primary motivation. With the VNA, topological changes are achieved by duplicating mesh elements that intersect the cut. Duplicate copies of mesh elements then contain only a portion of the material being modeled (often referred to as embedded material), but all duplicate mesh elements are copies of original elements and thus have predictable (and ideally high quality) conditioning.

However, as pointed out in [SDF07] the original algorithm does have some obvious limitations. First, it is often desirable to allow a cut to pass through a mesh facet, but the original VNA approach requires all cuts to pass through tetrahedron faces without intersecting the vertices of the mesh. Thus, cuts cannot lie on tetrahedron faces or pass through vertices and must be perturbed to satisfy this constraint. Furthermore, a cut cannot cross a tetrahedron face more than once, thus the resolution of the tetrahedral mesh limits the types of allowable cuts, which is counter to the original motivation of the algorithm. We provide a new approach that removes both of these limitations. First, we redevelop the original VNA in a manner that allows cuts to pass through all mesh facets including vertices, edges and faces. Second, we develop an adaptive approach to the embedding and duplication processes that allows cuts to pass through mesh faces an arbitrary number of times. This prevents limitations on the cutting surface geometry imposed by the original VNA from the requirement that a cut only cross a tetrahedron face in one location. This adaptivity is only used to allow subsequent cuts to a simulation mesh; we do not adaptively refine the elements of the simulation mesh.

[†] e-mail: {yuting,craig,teran}@math.ucla.edu
[‡] e-mail: cffjiang@cs.ucla.edu

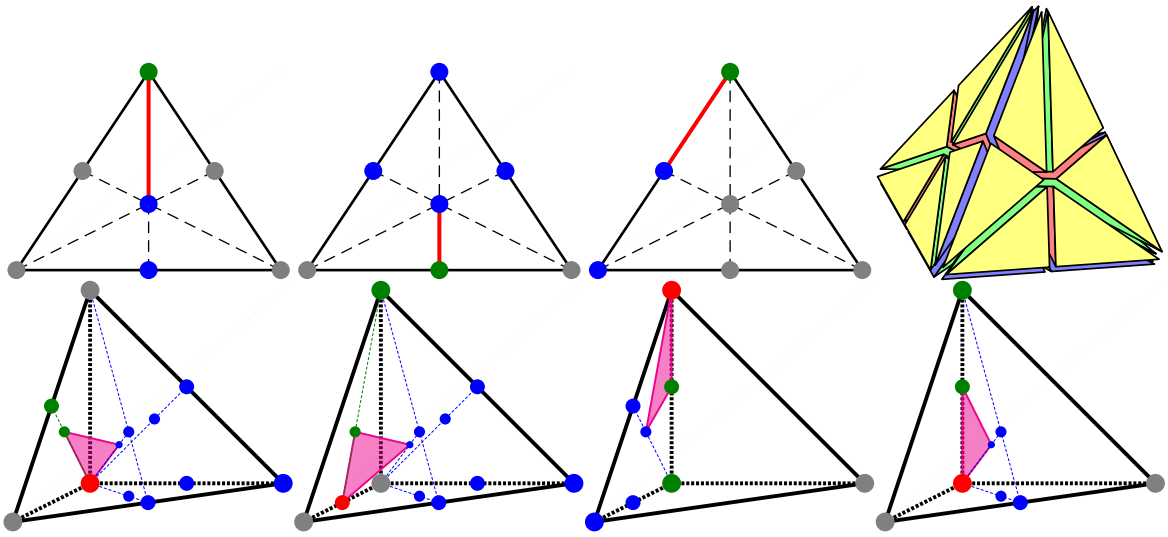

Figure 1: The maximally split configuration in 2D contains three types of cutting flags (top row). The red flag is set if intersections are registered at the green location and one of the blue locations on the triangle. In the maximally split configuration in 3D (top right), each color represents a different type of cutting face, which are shown individually on the bottom row. The correspondence is, from bottom left to bottom right: green, red, yellow, and blue. For each of the four 3D cases (bottom), the shaded flag is set if intersections are registered at the red location, one of the green locations, and one of the blue locations on the tetrahedron.

Lastly, we provide a novel mesh intersection algorithm for robustly defining the intersections of a cutting triangulated surface against a tetrahedron mesh. This surface must be resolved against the geometric primitives used for the duplication process in the VNA, and it can be sensitive to rounding errors. We provide a novel technique for robustly computing the quantities in this process that are sensitive to rounding errors.

2. Related work

Simulation of topological change in Lagrangian meshes was introduced to computer graphics in the pioneering work of [TF88]. [BSM*02] and [WWD14] provide detailed surveys of applications of such mesh cutting. Early approaches typically made use of simple separation along mesh element boundaries [NTB*91, MMA99, SWB01, MMDJ01, NvdS00] or even element deletion [FG99, CDA00, JBB*10, FDA02]. The available geometric detail in this type of approach was increased somewhat by subdivision of elements in the mesh prior to splitting [MK00, BG00]; however, this tended to introduce elements with poor aspect ratios. More geometrically rich cutting surfaces were generated by allowing separation along more arbitrary paths (albeit with the expense of re-meshing) [NF99, OH99, OBH02]. Recently, such approaches have been used to create some very compelling results for a variety of materials [NKJF09, KMOD09, SSF09, WRK*10, CWSO13, WT08, WTGT09, GBO04, BWHT07]. Embedded methods have been developed to minimize the complexity of re-meshing by embedding material surfaces into the existing mesh [MG04, MBF04, BHTF07, SDF07, GBT07, PO09, HJST13]. Although these works generalized the approach to fracture, the embedding idea goes back at least to free form deformations [SP86, FVDPT97, CGC*02, TSB*05]. Also, particle-based methods can provide flexibility for topology change [PKA*05]. Other interesting models for cut patterns were developed in [MCK13, IO06, IO09, NF99].

3. Modified virtual node algorithm

The original VNA allows for a finite number of embedded cuts in a given element. Specifically, individual tetrahedra in the mesh can only be split into at most four embedded subregions (one associated with each node). This implies a "maximally split" configuration of the mesh in which all possible cuts have been made. In the original algorithm, each disjoint piece in this configuration consists of the elements in the one-ring of the nodes in the original mesh. Every other node in the piece is then a duplicated copy of the original node. Although not originally described this way, the algorithm can be conceived of via manipulations in the material connections of these pieces in the maximally split configura-

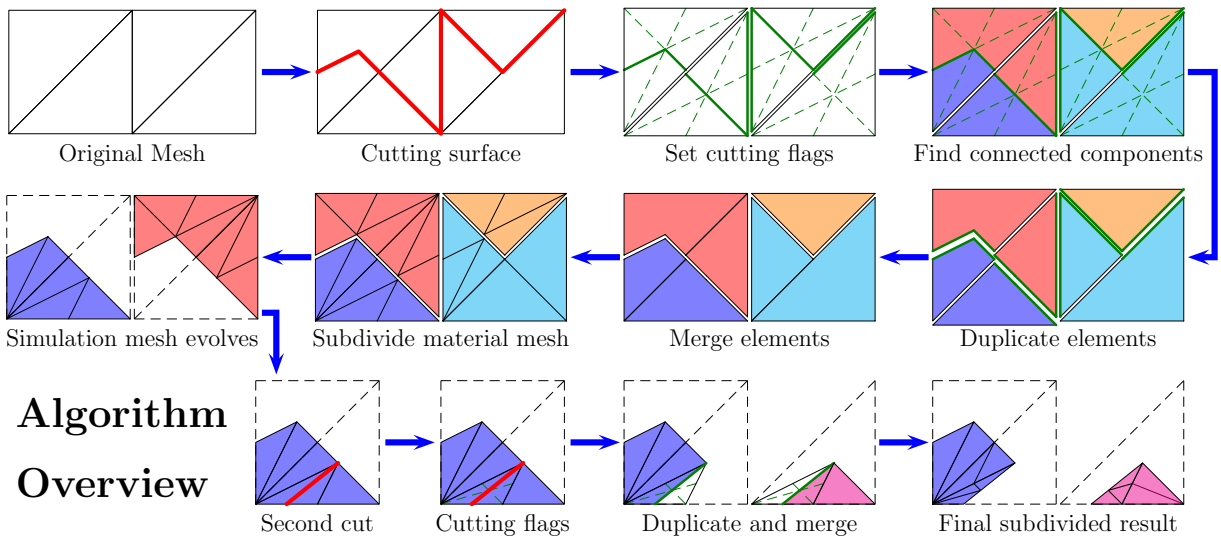

Figure 2: Schematic overview of our cutting algorithm.

tion as in [SDF07]. We will provide this material connection description of our modified algorithm here.

First, we modify the available splits of a tetrahedron from the original four node-associated regions to 24 tetrahedra as shown in the top right of Figure 1. This modification of the maximally split configuration is designed to allow cuts to pass through nodes of the embedding mesh, thus removing a major limitation of the original VNA. We refer to the triangle boundaries separating each of the tetrahedron subregions as cut faces since they will ultimately represent the embedded cut surface geometry. With this view, the first step in the algorithm is to define which cut faces are active. An active cut face means that material is separated along it. Active cut faces for a tetrahedron are identified by considering the way each element of the cutting surface intersects the tetrahedron. There are three cases in 2D and four in 3D, which are summarized in Figure 1. We record this information using cutting flags, which amounts to storing one bit for each possible cut face (12 bits in 2D, 60 bits in 3D) that could be active. Then, we compute connected components of material in the element, where sub-tetrahedra belong to the same component if the flag corresponding to the face between them is not set. Note that unlike in [SDF07], this process has only a finite number of possible cases and is straightforward to implement. Afterwards, a copy of the embedding element is made for each connected component. Each copy has all four of its own nodes distinct from any of the other copies.

Next, tetrahedron copies of an original element created in the first phase are compared to copies of the face-adjacent neighbors of the original element. If any of these copies share a material connection through their neighboring face, the six nodes on the copies of the face are condensed to three, thus sewing the elements back together along the face. Material connection can be established efficiently by virtue of the small number of possible material configurations.

The original VNA did not allow for cut faces on the boundary of the embedding element. However, we show that simply including the faces of the 24 sub tetrahedra that lay on the boundary of the embedding element as cut faces allows the algorithm to let cuts pass through embedding nodes and along embedding faces. This inability in the original algorithm required error-prone perturbation of cut surfaces to prevent these cases. Since we can handle degenerate cuts, we do not require perturbation to avoid them. In fact, we show in Section 5 that the ability to register degeneracies allows intersections to be computed robustly.

4. Adaptive cutting

The maximally split configuration in VNA approach places limitations on the possible geometries of the cutting surfaces. This causes the curvature of the cut surface to be at most at the scale of the tetrahedron resolution in the embedding mesh. This limitation was a large motivation for the work in [SDF07]. However, the generality of the available cutting surfaces in [SDF07] comes at the cost of significant algorithmic complexity. We show that an adaptive approach can be used to add more flexibility in cut surface geometry.

To clarify the steps in the process, we will refer to the original tetrahedron mesh as the simulation mesh. We will

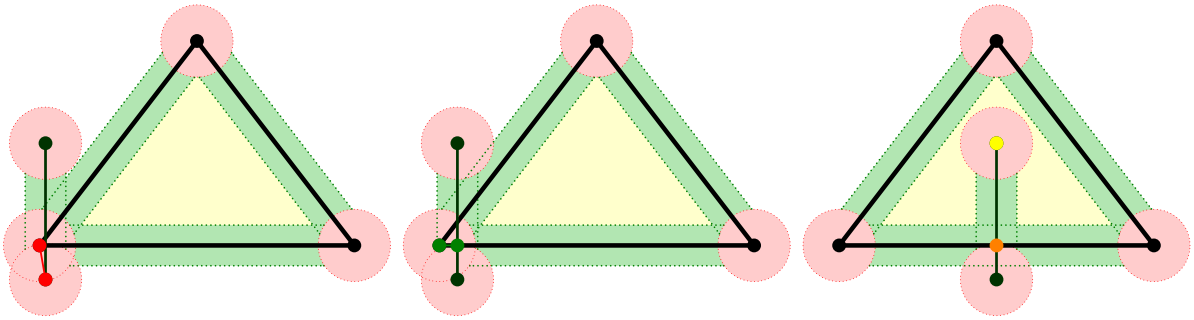

Figure 3: Left: two points within the vertex-vertex tolerance and are registered as intersecting (red). Middle: the two points separate, but the cutting segment remains within the edge-vertex tolerance and the edge-vertex pair is registered as intersecting (green). Right: the cutting segment is far enough from degeneracies that face-vertex (yellow) and edge-edge (orange) intersections are registered.

resolve all material after cuts in another mesh that we refer to as the material mesh. Initially these two meshes would normally be the same. After performing a cutting step, we perform a subdivision step on the embedded material geometry without changing the underlying simulation mesh. We subdivide the material in cut elements by creating an element for each of its maximally split components. Note that this does require that neighboring material mesh elements be split, as shown in the bottom right of Figure 2. While such a subdivision strategy does lead to material elements with unpredictable conditioning, this is not problematic since it does not effect the conditioning of the simulation mesh. This subdivision could clearly have been performed with fewer elements, but this simple strategy worked well for our needs.

The subdivided material mesh is embedded in the simulation mesh. To perform subsequent cuts, we simply cut the the material mesh and then perform duplication and merge steps as if it were the original simulation mesh. However, this process is used to duplicate and merge the simulation mesh with the logic being simply that if a simulation mesh element contains a material element that is duplicated, it is duplicated. Similarly, a merge between two material elements implies a merge between their (duplicated) embedding simulation elements. This can happen in two ways. First, if two material elements with the same embedding simulation element are merged, the corresponding duplicated simulation elements are merged entirely. Second, if two material elements with adjacent embedding simulation elements are merged, then only the faces (determined by adjacency) of their embedding elements are merged. With this adaptive approach, cuts can traverse paths with curvatures arbitrarily higher than the sizes implied by the simulation mesh.

Note that we perform the subdivision step *after* the cutting step. This forces each cut to be resolved at the resolution of the mesh and avoids creating an unnecessarily refined mesh to resolve the details of a complex cutting surface. Rather, the subdivision step afterwards allows cuts to be cut again without introducing large deviations from the cutting surface where cuts meet; deviations are instead limited by the sizes of the tolerances used by the robust intersection routines, which are a small fraction of the size of an element.

When performing incremental cutting, we delay the subdivision step until the entire cut has been made. When an element is split due to the incremental cut, each copy receives a copy of the cutting flags. This allows cutting of the element to resume as more of the incremental cut becomes available while avoiding excessive subdivision on incremental cuts that consist of large numbers of small additions to the cutting surface.

5. Robust intersection computation

Our robust intersection algorithm works by checking for intersections in order of most degenerate to least: vertex-vertex (VV), edge-vertex (EV), face-vertex (FV), edge-edge (EE), face-edge (FE), and finally tetrahedron-vertex (TV). All but the FE and TV cases are degenerate. (In 2D, the cases are VV, EV, FV, and EE, with VV and EV being degenerate cases.) When an intersection is found, it is stored in a hashtable along with the barycentric weights of the closest points on the primitives. A candidate intersection pair is rejected if any degeneracy of the pair is found in the hashtables. Thus, for example, an EV pair is rejected if either VV pair was registered as a degeneracy. Examples of degenerate intersections are shown in Figure 3.

Let $\mathcal{F}[f(x)]$ indicate that the computation of $f(x)$ is being carried out under floating point computation, and let $\mathcal{E}[f(x)]$ be a bound on its error. That is, $|\mathcal{F}[f(x)] - f(x)| \leq \mathcal{E}[f(x)]$. The intersection logic consists of two different types of tests. The primary type of test consists of computing a criterion and testing its sign. These sign checks *must* produce the same result as would be obtained under exact arithmetic. Since we only perform intersection checks for a pair when no degeneracy of the pair was found, we can assume the cri-

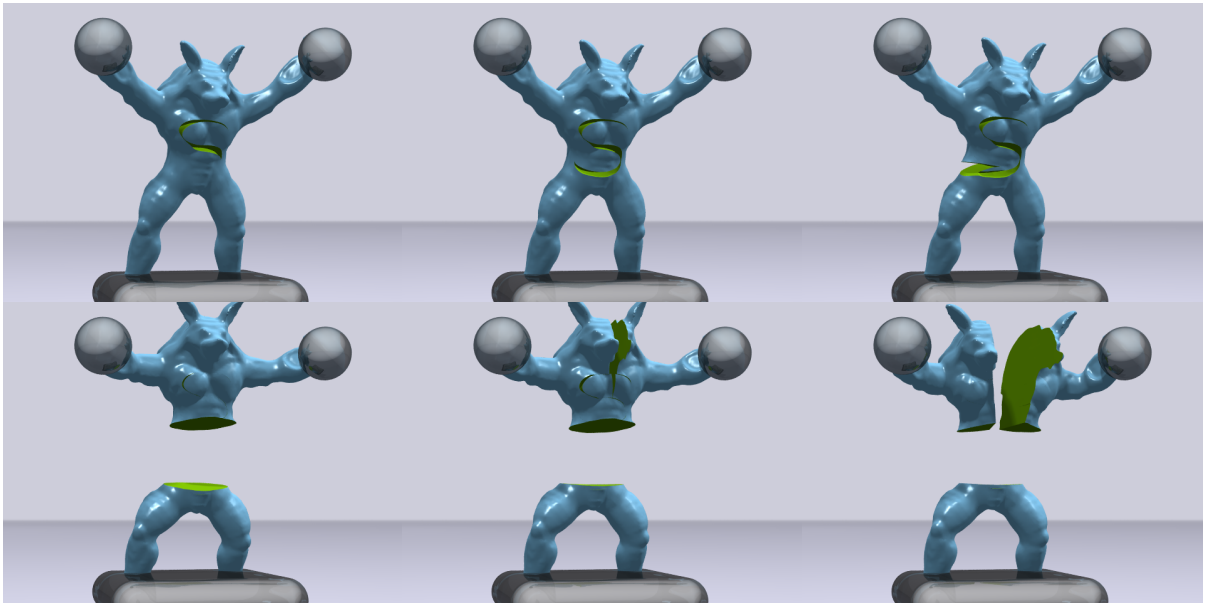

Figure 4: An armadillo mesh with 380K tetrahedra is progressively cut with an "S" and then cut into three pieces.

teria for such degeneracies are not met. The logic that we use differs somewhat from case to case, but in general we use these assumptions to prove $|f(x)| > a$ for some appropriate bound a and then show that $\mathcal{E}[f(x)] < a$. This guarantees that any value that might be computed under floating point has the same sign as the exactly-computed quantity.

The second type of test that we require is to compare some criterion $f(x)$ to a tolerance τ or, equivalently, determine whether $\mathcal{F}[\tau - f(x)] \geq 0$. In this case, we define two bounding tolerances τ_- and τ_+, such that $\tau_- < \tau < \tau_+$ and

$$\tau_- - f(x) \geq 0 \Longrightarrow \mathcal{F}[\tau - f(x)] \geq 0 \Longrightarrow \tau_+ - f(x) \geq 0. \quad (1)$$

If $\mathcal{E}[\tau - f(x)] < |\tau - f(x)|$, then the sign of $\mathcal{F}[\tau - f(x)]$ agrees with the corresponding exact comparison and (1) holds trivially. Consider instead the choices x for which $\mathcal{E}[\tau - f(x)] \geq |\tau - f(x)|$. For these x, we require the stricter condition

$$\tau_- - f(x) \leq \mathcal{F}[\tau - f(x)] \leq \tau_+ - f(x), \quad (2)$$

from which τ_- and τ_+ can be computed given a bound on $\mathcal{E}[\tau - f(x)]$. Note that (2) implies (1). In this way, we obtain guaranteed exact bounds on the criterion even though the tolerance check was performed in floating point. In some cases, we use $\tau_- > 0$ and a tolerance check against τ as an effective means of checking the sign of $f(x)$.

The two comparison strategies described above rely critically on the ability to bound the floating point error in the computation of both the criteria and tolerances. The tolerances are all computed in the same simple way, and their floating point error is computed directly. The criteria have much more complicated computations, making floating point bounds tedious and error prone to compute. Here, we instrumented our code to symbolically track conservative floating point error bounds through the computations in our code. This produces floating point error guarantees that we can use in our derivations. The requirement that the logic above be correct and along with the floating point error bounds obtained symbolically from the code places constraints on the values that may be chosen for the tolerances. The final step is to choose tolerances that satisfy all of the constraints. We have done this in 2D and 3D. Pseudocode for our intersection routines, along with suitable tolerances, are presented in an appendix. If the routines are implemented *exactly* as indicated in the pseudocode using the tolerances provided, the resulting implementation will be provably robust under both IEEE `float` and `double` precision arithmetic. A complete and detailed writeup, including detailed instructions on tracking floating point error, proofs for criterion bounds and correctness, and details on how the tolerance constraints were derived are provided as a supplemental technical document.

6. Examples

In Figure 4, a stretched armadillo is first cut incrementally along an S-shaped path before being diced into pieces. In Figure 7, we peel the skin off a sphere, demonstrating the ability to generate thin slices, even while cutting existing cuts. In Figure 6, we use shaped blades to cut letters into a stretched thin sheet. The sheet fractures and deforms as the blades pass through it.

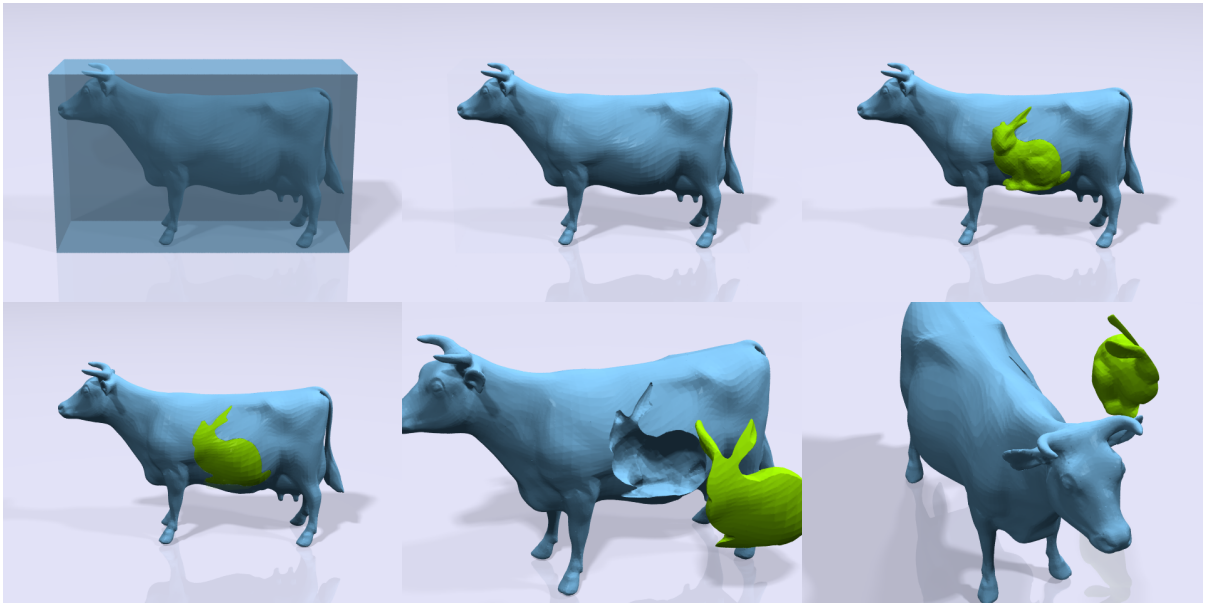

Figure 5: We demonstrate the ability to construct tetrahedralized volumes from triangulated surfaces using our cutting algorithm. We cut a box by a cow surface and then cut it again with a bunny surface, resulting in two separate volumes. Details are accurately resolved in both volumes, and the corners are sharp.

Since our algorithm cuts tetrahedralized volumes using an arbitrary cutting surface, we can use it to perform boolean operations. In Figure 5, we cut a 20k-element cow surface from a $208 \times 128 \times 68$ box, where each cube is divided into six tetrahedra. Our cutting algorithm accurately resolves the delicate geometry of the cow surface, including its horn, ear and tail. The cow volume is cut again by a bunny surface mesh, resulting in two volumes.

7. Discussions

While our approach addresses a number of limitations of the existing VNA, it is not without limitations. First, while our adaptive definition of embedded material does improve the ability to resolve cut surfaces at resolutions independent of the embedding mesh, it does so with a considerable algorithmic complexity compared to the original VNA. However, this complexity is still significantly lower than the approach in [SDF07]. Also, without this adaptive routine the algorithm would have the same complexity as the original VNA while adding the ability to pass through nodes, faces and edges. Second, safe tolerances for the mesh intersection routines are considerably larger in 3D than in 2D, which effectively prevents the use of floats for 3D intersection calculations. Lastly, while our adaptive approach does allow for cutting surfaces with curvatures finer than the scale of the original mesh, our incremental resolution of cutting surface can constrain the cut surface curvature in some cases. However, this could be prevented by initially subdividing the material mesh to resolve curvatures of a desired scale.

8. Acknowledgments

All authors were partially supported by NSF (DMS-0502315, DMS-0652427, CCF-0830554), DOE (09-LR-04-116741-BERA), ONR (N000140310071, N000141010730, N000141210834) and Intel STCVisual Computing Grant (20112360).

References

[BG00] BIELSER D., GROSS M. H.: Interactive simulation of surgical cuts. In *Proc. Pac. Conf. Comp. Graph. App.* (2000), pp. 116–442. 2

[BHTF07] BAO Z., HONG J., TERAN J., FEDKIW R.: Fracturing rigid materials. *IEEE Trans. Vis. Comp. Graph. 13* (2007), 370–378. 2

[BSM*02] BRUYNS C. D., SENGER S., MENON A., MONTGOMERY K., WILDERMUTH S., BOYLE R.: A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools. *Journal Vis. Comp. Anim. 13* (2002), 21–42. 2

[BWHT07] BARGTEIL A., WOJTAN C., HODGINS J., TURK G.: A finite element method for animating large viscoplastic flow. *ACM Trans. Graph. 26* (2007), 19–38. 2

[CDA00] COTIN S., DELINGETTE H., AYACHE N.: A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation. *Vis. Comp. 16* (2000), 437–452. 2

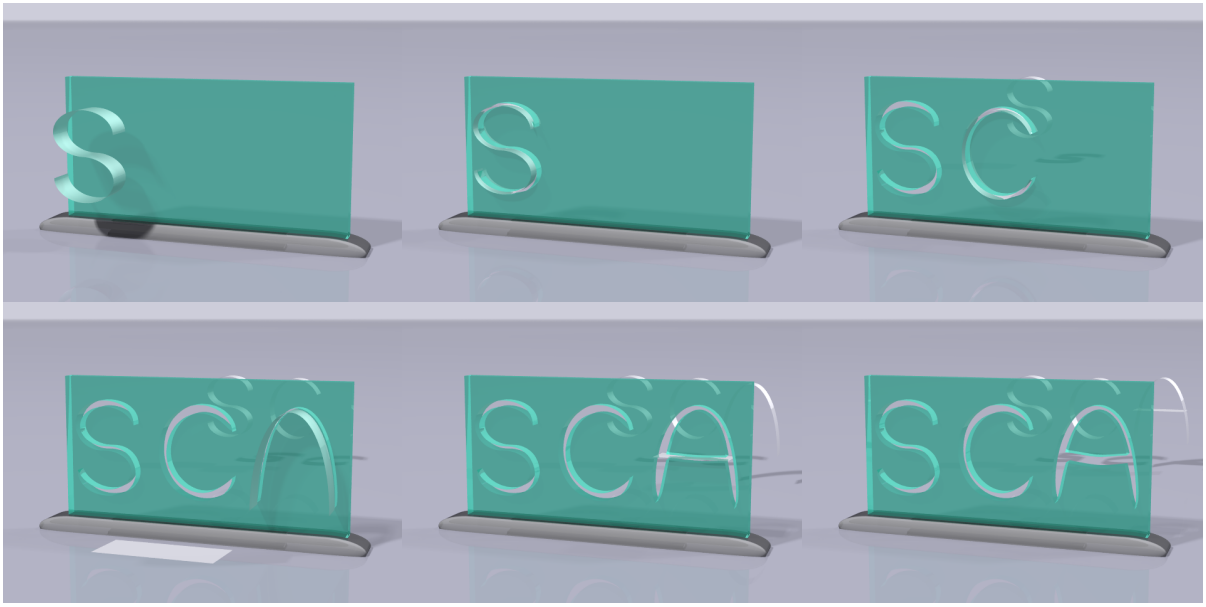

Figure 6: We cut letters out of stretched elastic sheet, demonstrating the ability to mix cutting, re-cutting, and simulation.

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. 21* (2002), 586–593. 2

[CWSO13] CLAUSEN P., WICKE M., SHEWCHUK J. R., O'BRIEN J. F.: Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Trans. Graph. 32* (2013), 17:1–15. 2

[FDA02] FOREST C., DELINGETTE H., AYACHE N.: Removing tetrahedra from a manifold mesh. In *Proc. Comp. Anim.* (2002), pp. 225–229. 2

[FG99] FRISKEN-GIBSON S. F.: Using linked volumes to model object collisions, deformation, cutting, carving, and joining. *Trans. Vis. Comp. Graph. 5* (1999), 333–348. 2

[FVDPT97] FALOUTSOS P., VAN DE PANNE M., TERZOPOULOS D.: Dynamic free-form deformations for animation synthesis. *IEEE Trans. Vis. Comp. Graph. 3* (1997), 201–214. 2

[GBO04] GOKTEKIN T., BARGTEIL A., O'BRIEN J.: A method for animating viscoelastic fluids. *ACM Trans. Graph. 23* (2004), 463–468. 2

[GBT07] GISSLER M., BECKER M., TESCHNER M.: Constraint sets for topology-changing finite element models. In *Virt. Real. Inter. Phys. Sim.* (2007), pp. 21–26. 2

[HJST13] HEGEMANN J., JIANG C., SCHROEDER C., TERAN J. M.: A level set method for ductile fracture. In *Proc. Symp. Comp. Anim.* (2013), pp. 193–201. 2

[IO06] IBEN H. N., O'BRIEN J. F.: Generating surface crack patterns. In *Proc. Symp. Comp. Anim.* (2006), pp. 177–185. 2

[IO09] IBEN H., O'BRIEN J.: Generating surface crack patterns. *Graph. Mod. 71* (2009), 198–208. 2

[JBB*10] JEŘÁBKOVÁ L., BOUSQUET G., BARBIER S., FAURE F., ALLARD J.: Volumetric modeling and interactive cutting of deformable bodies. *Progress Biophs. Mol. Bio. 103* (2010), 217 – 224. 2

[KMOD09] KHAREVYCH L., MULLEN P., OWHADI H., DESBRUN M.: Numerical coarsening of inhomogeneous elastic materials. *ACM Trans. Graph. 28*, 3 (2009), 51:1–51:8. 2

[MBF04] MOLINO N., BAO Z., FEDKIW R.: A virtual node algorithm for changing mesh topology during simulation. In *ACM SIGGRAPH* (2004), pp. 385–392. 1, 2

[MCK13] MÜLLER M., CHENTANEZ N., KIM T.-Y.: Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Trans. Graph. 32*, 4 (2013), 115:1–115:10. 2

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proc. Graph. Int.* (2004), pp. 239–246. 2

[MK00] MOR A. B., KANADE T.: Modifying soft tissue models: Progressive cutting with minimal new element creation. In *Proc. MICCAI* (2000), pp. 598–607. 2

[MMA99] MAZARAK O., MARTINS C., AMANATIDES J.: Animating exploding objects. In *Graph. Int.* (1999), pp. 211–218. 2

[MMDJ01] MÜLLER M., MCMILLAN L., DORSEY J., JAGNOW R.: Real-time simulation of deformation and fracture of stiff materials. In *Proc. Eurographics Workshop Comp. Anim. Sim.* (2001), pp. 113–124. 2

[NF99] NEFF M., FIUME E.: A visual model for blast waves and fracture. In *Proc. Graph. Int.* (1999), pp. 193–202. 2

[NKJF09] NESME M., KRY P. G., JEŘÁBKOVÁ L., FAURE F.: Preserving topology and elasticity for embedded deformable models. In *ACM SIGGRAPH* (2009), pp. 52:1–52:9. 2

[NTB*91] NORTON A., TURK G., BACON B., GERTH J., SWEENEY P.: Animation of fracture by physical modeling. *Vis. Comp. 7* (1991), 210–219. 2

[NvdS00] NIENHUYS H.-W., VAN DER STAPPEN A. F.: Combining finite element deformation with cutting for surgery simulations. In *Eurograph. Short Present.* (2000), pp. 43–52. 2

[OBH02] O'BRIEN J., BARGTEIL A., HODGINS J.: Graphical

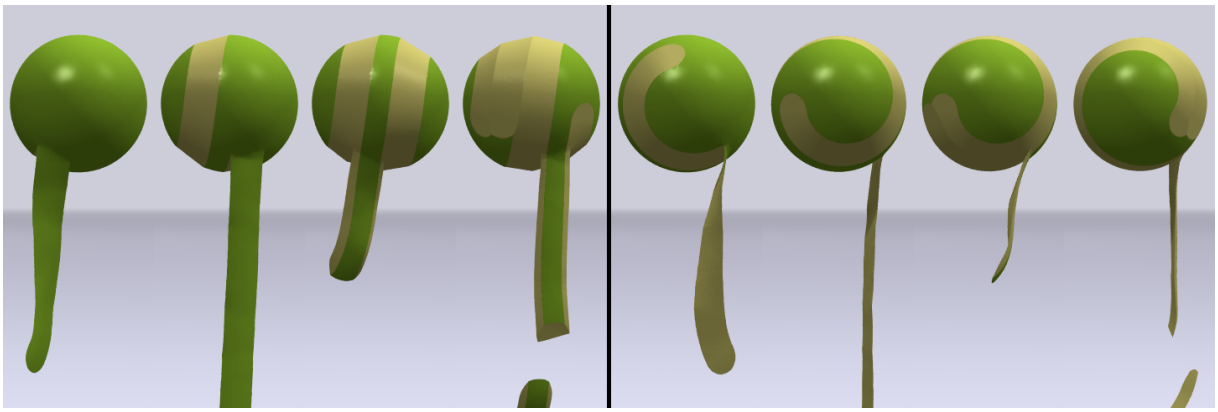

Figure 7: An apple is peeled demonstrating the ability to do incremental cuts and produce thin geometry, shown from the front (left) and the side (right). The skin thickness is $1/60$ of the diameter of the ball.

modeling and animation of ductile fracture. *ACM Trans. Graph. 21* (2002), 291–294. 2

[OH99] O'BRIEN J. F., HODGINS J. K.: Graphical modeling and animation of brittle fracture. In *ACM SIGGRAPH* (1999), pp. 137–146. 2

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ P., GROSS M., GUIBAS L.: Meshless animation of fracturing solids. *ACM Trans. Graph. 24* (2005), 957–964. 2

[PO09] PARKER E., O'BRIEN J.: Real-time deformation and fracture in a game environment. In *Proc. Symp. Comp. Anim.* (2009), pp. 165–175. 2

[SDF07] SIFAKIS E., DER K., FEDKIW R.: Arbitrary cutting of deformable tetrahedralized objects. In *Proc. Symp. Comp. Anim.* (2007), pp. 73–80. 1, 2, 3, 6

[SP86] SEDERBERG T., PARRY S.: Free-form deformation of solid geometric models. In *ACM SIGGRAPH* (1986), pp. 151–160. 2

[SSF09] SU J., SCHROEDER C., FEDKIW R.: Energy stability and fracture for frame rate rigid body simulations. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), pp. 155–164. 2

[SWB01] SMITH J., WITKIN A., BARAFF D.: Fast and controllable simulation of the shattering of brittle objects. *Comp. Graph. Forum 20* (2001), 81–90. 2

[TF88] TERZOPOULOS D., FLEISCHER K.: Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *ACM SIGGRAPH* (1988), pp. 269–278. 2

[TSB*05] TERAN J., SIFAKIS E., BLEMKER S., NG-THOW-HING V., LAU C., FEDKIW R.: Creating and simulating skeletal muscle from the visible human data set. *IEEE Trans. Vis. Comp. Graph. 11* (2005), 317–328. 2

[WRK*10] WICKE M., RITCHIE D., KLINGNER B., BURKE S., SHEWCHUK J., O'BRIEN J.: Dynamic local remeshing for elastoplastic simulation. *ACM Trans. Graph. 29* (2010), 49:1–49:11. 2

[WT08] WOJTAN C., TURK G.: Fast viscoelastic behavior with thin features. In *ACM SIGGRAPH* (2008), pp. 47:1–47:8. 2

[WTGT09] WOJTAN C., THÜREY N., GROSS M., TURK G.: Deforming meshes that split and merge. *ACM Trans. Graph. 28* (2009), 76:1–76:10. 2

[WWD14] WU J., WESTERMANN R., DICK C.: Physically-based simulation of cuts in deformable bodies: A survey. In *Eurograph. State-of-the-Art Report* (2014). 2

Appendix A: pseudocode

Figure 8 provides pseudocode for the routines we used in 2D and in 3D. They should be implemented as is, though changes that do not affect the floating point computations are of course fine (e.g., delaying computations so they will only be computed if required). The EV cases involve a vector normalization, which is written as $u \leftarrow \frac{\hat{u}}{\|\hat{u}\|}$. Sufficient floating point error has been assumed in the analysis so the computation can instead be performed with a single division as $u \leftarrow \frac{1}{\|\hat{u}\|}\hat{u}$. The operation $m^2 \leftarrow \|r\|^2$ in the 3D EE and FV cases do not involve a square root; only m^2 itself will be required. The quantity ε is the machine epsilon, which will be different for `float` or `double` precision.

Algorithm 1 Intersection routines for 2D

function COMPUTE_TOLERANCES(A,B)
$L_a \leftarrow$ maximum bound box edge length of mesh A
$L_b \leftarrow$ maximum bound box edge length of mesh B
$s \leftarrow \sqrt{\varepsilon}$; $L \leftarrow \frac{1+5\varepsilon}{1-7s}(L_a+L_b)$; $t = sL$
$\sigma = 6.5t$; $\tau = 4.5t$; $\hat{\sigma} = 5.5t$; $\kappa = 21\varepsilon L^2$
end function

function VERTEX_VERTEX(A,B)
return $\|A-B\|^2 \leq \sigma^2$
end function

function EDGE_VERTEX(A,B,P)
$\hat{u} \leftarrow A-B$; $m \leftarrow \|\hat{u}\|$; $u \leftarrow \frac{\hat{u}}{m}$
$w \leftarrow P-A$; $\hat{a} \leftarrow u \cdot w$; $\bar{a} \leftarrow m-\hat{a}$
if $m \leq \hat{\sigma}$ **or** $|u \times w| > \tau$ **or** $\hat{a} < 0$ **or** $\bar{a} < 0$ **then**
return (FALSE, 0)
return (TRUE, $\frac{\hat{a}}{m}$)
end function

function EDGE_EDGE(A,B,P)
$a_A \leftarrow (A-P)\times(Q-P)$; $a_B \leftarrow (B-P)\times(Q-P)$
$a_P \leftarrow (P-A)\times(B-A)$; $a_Q \leftarrow (Q-A)\times(B-A)$
if $|a_P| \leq \kappa$ **or** $|a_Q| \leq \kappa$ **or** $\text{sgn}(a_P) = \text{sgn}(a_Q)$ **or**
$|a_A| \leq \kappa$ **or** $|a_B| \leq \kappa$ **or** $\text{sgn}(a_A) = \text{sgn}(a_B)$ **then**
return (FALSE, 0, 0)
return (TRUE, $\frac{a_A}{a_A-a_B}, \frac{a_P}{a_P-a_Q}$)
end function

function TRIANGLE_VERTEX(A,B,P)
$a_A \leftarrow (B-P)\times(C-P)$; $a_B \leftarrow (P-A)\times(C-A)$
$a_C \leftarrow (B-A)\times(P-A)$
if $|a_A| \leq \kappa$ **or** $|a_B| \leq \kappa$ **or** $|a_C| \leq \kappa$ **or**
$\text{sgn}(a_A) \neq \text{sgn}(a_B)$ **or** $\text{sgn}(a_B) \neq \text{sgn}(a_C)$ **then**
return (FALSE, 0, 0, 0)
return (TRUE, $\frac{a_A}{a_A+a_B+a_C}, \frac{a_B}{a_A+a_B+a_C}, \frac{a_C}{a_A+a_B+a_C}$)
end function

Algorithm 2 Intersection routines for 3D (part I)

function COMPUTE_TOLERANCES(A,B)
$L_a \leftarrow$ maximum bound box edge length of mesh A
$L_b \leftarrow$ maximum bound box edge length of mesh B
$s \leftarrow \sqrt{\varepsilon}$; $a \leftarrow \sqrt{s}$; $L \leftarrow \frac{1+5\varepsilon}{1-7a}(L_a+L_b)$
$b \leftarrow sa$; $c = aL$; $d = L^2$; $e = bLd$; $f = \varepsilon d^2$
$\sigma = 6.5c$; $\tau = 4.5c$; $\delta = 2.25c$; $\gamma = 2.25c$
$\hat{\sigma} = 5.5c$; $\mu = 24e$; $\rho = 56e$; $\zeta = 1317f$
$\lambda = 1215f$; $\phi = 470f$; $\nu = 6844.5f$; $\xi = 56e$
end function

function VERTEX_VERTEX(A,B)
return $\|A-B\|^2 \leq \sigma^2$
end function

Algorithm 3 Intersection routines for 3D (part II)

function EDGE_VERTEX(A,B,P)
$\hat{u} \leftarrow A-B$; $m \leftarrow \|\hat{u}\|$; $u \leftarrow \frac{\hat{u}}{m}$
$w \leftarrow P-A$; $\hat{a} \leftarrow u \cdot w$; $\bar{a} \leftarrow m-\hat{a}$
if $m \leq \hat{\sigma}$ **or** $\|u \times w\|^2 > \tau^2$ **or** $\hat{a} < 0$ **or** $\bar{a} < 0$ **then**
return (FALSE, 0)
return (TRUE, $\frac{\hat{a}}{m}$)
end function

function EDGE_EDGE(A,B,P,Q)
$u \leftarrow B-A$; $v \leftarrow Q-P$; $r \leftarrow u \times v$; $m^2 \leftarrow \|r\|^2$
$w \leftarrow P-A$; $\hat{d} \leftarrow r \cdot w$; $n \leftarrow r \times w$; $\hat{a} = n \cdot v$
$\hat{b} = n \cdot u$; $\bar{a} = m^2 - \hat{a}$; $\bar{b} = m^2 - \hat{b}$
if $m^2 \leq \lambda$ **or** $\hat{d}^2 > \gamma^2 m^2$ **or**
$\hat{a} \leq \phi$ **or** $\hat{b} \leq \phi$ **or** $\bar{a} \leq \phi$ **or** $\bar{b} \leq \phi$ **then**
return (FALSE, 0, 0)
return (TRUE, $\frac{\hat{a}}{m^2}, \frac{\hat{b}}{m^2}$)
end function

function TRIANGLE_VERTEX(A,B,C,P)
$u \leftarrow B-A$; $v \leftarrow C-A$; $r \leftarrow u \times v$; $m^2 \leftarrow \|r\|^2$
$w \leftarrow P-A$; $\hat{d} \leftarrow r \cdot w$; $n \leftarrow r \times w$; $\hat{b} = n \cdot v$
$\hat{c} = -n \cdot u$; $\hat{a} = m^2 - \hat{b} - \hat{c}$
if $m^2 \leq \nu$ **or** $\hat{d}^2 > \delta^2 m^2$ **or**
$\hat{a} \leq \zeta$ **or** $\hat{b} \leq \zeta$ **or** $\hat{c} \leq \zeta$ **then**
return (FALSE, 0, 0, 0)
return (TRUE, $\frac{\hat{a}}{m^2}, \frac{\hat{b}}{m^2}, \frac{\hat{c}}{m^2}$)
end function

function TRIANGLE_EDGE(A,B,C,P,Q)
$a \leftarrow A-Q$; $b \leftarrow B-Q$; $c \leftarrow C-Q$
$p \leftarrow P-Q$; $v_P \leftarrow ((A-P)\times(B-P)) \cdot (C-P)$
$v_A \leftarrow (b \times c) \cdot p$; $v_B \leftarrow (c \times a) \cdot p$
$v_C \leftarrow (a \times b) \cdot p$; $v_Q \leftarrow (a \times b) \cdot c$
if $|v_A| \leq \mu$ **or** $|v_B| \leq \mu$ **or** $|v_C| \leq \mu$ **or**
$\text{sgn}(v_A) \neq \text{sgn}(v_B)$ **or** $\text{sgn}(v_B) \neq \text{sgn}(v_C)$ **or**
$|v_P| \leq \xi$ **or** $|v_Q| \leq \xi$ **or** $\text{sgn}(v_P) = \text{sgn}(v_Q)$ **then**
return (FALSE, 0, 0, 0, 0)
return (TRUE, $\frac{v_A}{v_A+v_B+v_C}, \frac{v_B}{v_A+v_B+v_C}, \frac{v_C}{v_A+v_B+v_C}, \frac{v_P}{v_P-v_Q}$)
end function

function TETRAHEDRON_VERTEX(A,B,C,D,P)
$v_A \leftarrow ((B-P)\times(C-P)) \cdot (D-P)$
$v_B \leftarrow ((P-A)\times(C-A)) \cdot (D-A)$
$v_C \leftarrow ((B-A)\times(P-A)) \cdot (D-A)$
$v_D \leftarrow ((B-A)\times(C-A)) \cdot (P-A)$
$s \leftarrow v_A + v_B + v_C + v_D$
if $|v_A| \leq \rho$ **or** $|v_B| \leq \rho$ **or** $|v_C| \leq \rho$ **or** $|v_C| \leq \rho$ **or**
$\text{sgn}(v_A) \neq \text{sgn}(v_B)$ **or** $\text{sgn}(v_B) \neq \text{sgn}(v_C)$ **or**
$\text{sgn}(v_C) \neq \text{sgn}(v_D)$ **then**
return (FALSE, 0, 0, 0, 0)
return (TRUE, $\frac{v_A}{s}, \frac{v_B}{s}, \frac{v_C}{s}, \frac{v_D}{s}$)
end function

Figure 8: Algorithms for robust intersection under floating point.