Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2007)
D. Metaxas and J. Popovic (Editors)

Arbitrary Cutting of Deformable Tetrahedralized Objects

Eftychios Sifakis† Stanford University Intel Corporation
Kevin G. Der † Stanford University
Ronald Fedkiw† Stanford University Industrial Light+Magic

Abstract

We propose a flexible geometric algorithm for placing arbitrary cracks and incisions on tetrahedralized deformable objects. Although techniques based on remeshing can also accommodate arbitrary fracture patterns, this flexibility comes at the risk of creating sliver elements leading to models that are inappropriate for subsequent simulation. Furthermore, interactive applications such as virtual surgery simulation require both a relatively low resolution mesh for efficient simulation of elastic deformation and highly detailed surface geometry to facilitate accurate manipulation and cut placement. Thus, we embed a high resolution material boundary mesh into a coarser tetrahedral mesh using our cutting algorithm as a meshing tool, obtaining meshes that can be efficiently simulated while preserving surface detail. Our algorithm is similar to the virtual node algorithm in that we avoid sliver elements and their associated stringent timestep restrictions, but it is significantly more general allowing for the arbitrary cutting of existing cuts, sub-tetrahedron resolution (e.g. we cut a single tetrahedron into over a thousand pieces), progressive introduction of cuts while the object is deforming, and moreover the ability to accurately cut the high resolution embedded mesh.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Physically based modeling I.3.7 [Computer Graphics]: Animation

1. Introduction

Methods for modeling and animating cracks and incisions on rigid or deformable solids have received significant attention in computer graphics, some dating back to the early days of deformable models [TF88a, TF88b]. One class of applications dealt with modeling the phenomena of material failure and fracture (see e.g. [OH99, OBH02]) while other methods focused on incisions that are explicitly placed on a deformable model, for example in the context of a surgical manipulation in a virtual simulation environment.

Virtual surgery in particular has spawned a very active thread of research due to the high visibility and impact of such applications as well as the unique algorithmic requirements it entails. Interactivity in a virtual surgery application mandates the use of a relatively coarse simulation mesh to obtain high frame rates. At the same time, a substantially higher resolution is required on the surface geometry to prevent distorting features that are essential in planning incisions and manipulating the flesh. These requirements place severe limitations on the design of a cutting algorithm. For example, methods that decompose tetrahedra into subelements in order to cut the mesh (e.g. [BMG99, BGTG03]) create many more tetrahedra (essentially making a fine resolution mesh near the cuts) and create ill-conditioned tetrahedra that impose severe time step restrictions (see e.g. [MK00, BG00]) making the simulation impractical even on the low resolution base mesh. Thus, some authors have avoided element decomposition resorting instead to other techniques such as deleting any element touched by the blade [FDA02], or restricting cutting to mesh element boundaries [NvdS00]. Whereas the second option can produce a jagged surface, subsequent snapping of nodes can smooth the cut surface [NvdS01, SHS01]. Of course, this also tends to produce sliver elements, and some authors have proposed the use of meshing algorithms such as edge collapse to remove degenerate tetrahedra [GO01, SHGS06].

Although first proposed in the context of fracture, the virtual node algorithm [MBF04] alleviates many of the aforementioned difficulties. By duplicating elements to obtain new degrees of freedom necessary for modeling the topological change, it preserves the conditioning of the initial mesh while creating a minimal number of new elements. Thus one obtains the goal of a low number of well-conditioned elements. A limitation of the virtual node algorithm is that if the mesh were completely fractured, the smallest possible units would be individual nodes surrounded by their incident material. A tetrahedron can be cut at most once along

† email: {sifakis|kder|fedkiw}@cs.stanford.edu

SCA 2007, San Diego, California, August 04 - 05, 2007
© 2007 ACM 978-1-59593-624-4/07/0008 $ 5.00

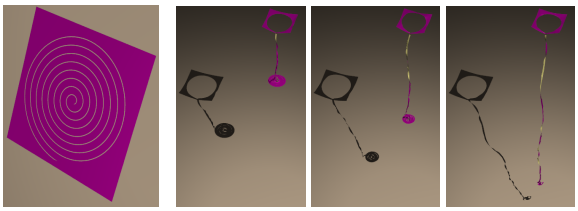

Figure 1: *A spiral cut applied to a piece of cloth leads to the unwinding of a long ribbon. The uncut triangulated surface consists of 128 triangles, while the final embedding mesh contains 587 triangles.*

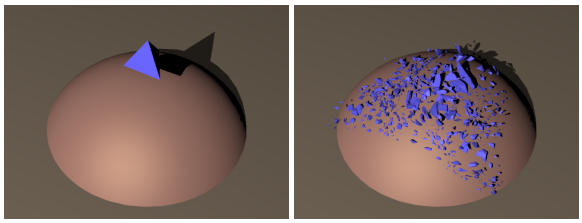

Figure 2: *Our algorithm is used to slice a single tetrahedron into over a thousand pieces placing no restriction on the placement or number of cuts. These pieces are not constrained to lie on nodes or faces, and most of them are completely within the tetrahedron. Many are even too small to visualize. However since they are embedded in a tetrahedron equivalent to the original one, they are efficient to simulate.*

each edge and can only be fractured into four pieces. Thus, the resolution of the simulation mesh effectively limits the resolution of the cutting surface. For a virtual surgery application, where a coarser mesh and precise cuts are required, such restrictions could be detrimental. For example, even a simple V-shaped cut would typically have to be heavily distorted so that the cutting surface only "turns" along element boundaries. More elaborate cuts, such as T-shaped incisions, would incur even more serious distortion or could even lead to spurious material connections. For fracture on the other hand, where finer meshes are typically used and jagged cracks are acceptable, these restrictions are not as severe. An improved version of the virtual node algorithm was proposed in [TSSB*05] for embedding a high resolution tetrahedral muscle geometry into a coarser resolution tetrahedral simulation mesh. They diagrammatically showed that the virtual node concept could be extended to split a tetrahedron into as many pieces as required to embed arbitrary geometry, and used the high resolution tetrahedral muscle geometry mesh to prescribe the connectivity of the coarse embedding mesh. An important limitation of their method, however, is the requirement for a full tetrahedralization of the embedded geometry (rather than a description of its boundary surface), rendering it inapplicable to virtual surgery applications which define a cutting surface without any predetermined notion of volumetric connectivity.

[SOG06] proposed a meshless method for simulating a finite element constitutive model similar to the fracture work in [PKA*05] (see also [WSG05]). However, they stress that the crack itself is better modeled with explicit triangulated geometry rather than with the meshless method. This decoupling of the cutting surface from the simulation geometry (i.e. an explicit triangulated cutting surface combined with a meshless discretization of the simulation geometry) is similar in spirit to the fracture work of [MTG04, MBF04] both of which embedded a triangulated surface into a background simulation mesh. Stressing that this explicit triangulation of the cutting surface permits geometric operations such as subdivision and self-intersection resolution, we devise a novel algorithm that allows for a direct implementation of the conceptual diagram from [TSSB*05] without requiring any predetermined notion of volumetric connectivity (as they do). Thus, our algorithm can also be used as an embedded meshing tool which, using only the boundary surface of an input object, will embed it in an arbitrary simulation mesh duplicating the degrees of freedom of the embedding mesh as needed to resolve the topology and connectivity of the original embedded object. This feature is demonstrated in figure 10 where a high-resolution triangulated surface of a face is used to carve an embedded flesh volume out of a coarser embedding mesh. In this example, embedding tetrahedra touching both the upper and the lower lip are automatically duplicated to allow opening of the mouth. Finally, such embedded models may be further cut if desired, as shown in figure 10.

Our main contribution is a cutting algorithm that yields efficient simulation models while imposing no restriction on the geometry of the cuts. In this aspect it combines the favorable traits of other methods in the literature while alleviating most of their limitations. For example, it provides the versatility of remeshing-based or point-based methods in accommodating arbitrary cuts, while maintaining the simplicity and efficiency of an underlying tetrahedral representation (unlike point-based methods) and preventing stringent timestep restrictions due to ill-conditioned sliver elements (unlike methods based on remeshing). Similar to the virtual node algorithm it creates new degrees of freedom to reflect the topological changes incurred by the cuts while preserving the good conditioning of the initial mesh, but avoids the restrictions on the geometry of the cutting surface imposed by the formulation of [MBF04]. When used as an embedded meshing tool, it provides the freedom to embed an arbitrary high resolution geometry in a coarser simulation mesh in the fashion of [TSSB*05] but only requires a surface representation of the embedded geometry as contrasted to the full volumetric representation required by their approach. Finally, our method supports incremental introduction of cuts as well as cutting an object while it deforms.

2. Previous Work

[NTB*91, HTK98, MMA99, SWB01] broke connections between elements under high stress, [NF99] addressed blast waves, [MG04, MTG04] considered fracture along element

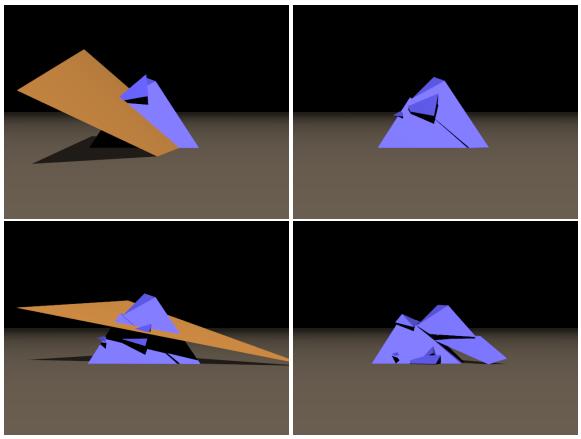

Figure 3: *A single tetrahedron is progressively cut (including through previous cuts) as the fragments are simulated. Each cut consists of a single triangle depicted in gold.*

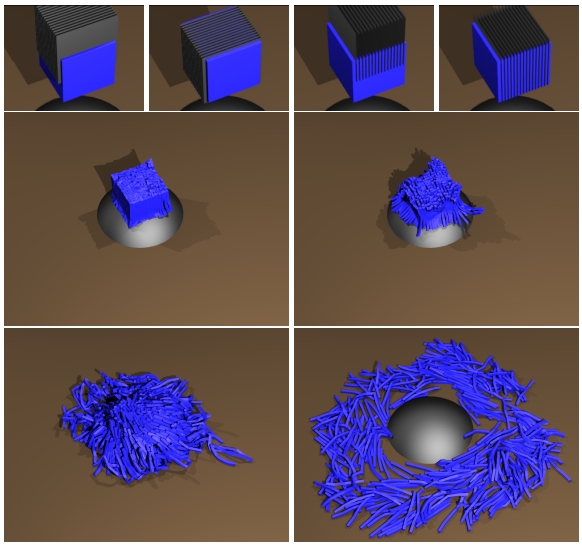

Figure 4: *32 intersecting planes slice a 320 element tetrahedral mesh into 289 sticks. 8K embedding tetrahedra are created after the cutting operation.*

boundaries in an FFD framework, [OH99, YOH00, OBH02] handled fracture events using continuous remeshing, and [MMDJ01] treated fragments as rigid bodies in between collisions. The virtual node algorithm has also been used for melting and burning solids [LIGF06]. [GSH*05, GLB*06] treated the fracture of thin shells, and [BHTF06] addressed the fracture of rigid materials. Modeling of cuts has also been addressed in the context of virtual surgery, especially on facial models [KGC*96, PRZ92, KGPG96] where the cut surface is the result of an explicit surgical manipulation, rather than a physical Rankine condition for fracture.

3. Cutting Algorithm

In our approach there are two simulation primitives, a background tetrahedral simulation mesh and a triangulated cutting surface. When embedding a high resolution material surface mesh into the background tetrahedral mesh, the high resolution surface is considered part of the cutting surface and in fact our cutting algorithm is used to generate this embedded geometry (see Figure 10). Virtual surgery applications progressively add triangles to the cutting surface along the blade's swept front, while fracture applications add triangles based on stress criteria.

3.1. The Cutting Surface

Triangles in the cutting surface may be initially intersecting, e.g. T-cuts are common to many surgical procedures. Leveraging our explicit description of the cutting surface, we resolve self-intersections through subdivision of the cutting triangles. One approach is to compute intersections among all triangles in the cutting surface and then to retriangulate into an intersection-free surface, but we have found that using a polygonal subdivision is more robust and efficient. In particular, one could triangulate our polygonal elements, but subsequent cuts that yield intersections would force further subdivision of these triangles unnecessarily creating more triangles and possibly poorly conditioned triangles. Given a soup of cutting triangles, the polygonal subdivision is unique and can be updated incrementally preserving uniqueness whereas the triangulation is not unique, based on heuristics, and may be found suboptimal when further cutting triangles are added. This means that one would be forced to undo previous triangulations in order to avoid degeneracies and poor conditioning due to newly introduced cuts, hindering the incremental nature of the algorithm.

Our approach is to resolve the cutting triangles into an intersection-free, nonmanifold *polygon mesh*. Each cutting triangle is subdivided into a number of polygons, whose edges are line segments that result from the pairwise intersection of this cutting triangle with all other cutting triangles. When adding new cuts incrementally, these new triangles are resolved with the existing polygons obviating the need to re-examine all past cuts. We will of course eventually triangulate these polygons for collisions and visualization, but the potentially resulting sliver elements are not used by our cutting algorithm and thus do not impact its conditioning.

We compute the polygonization of any triangle in the triangle soup as follows. A node of the polygon mesh is defined in one of three possible ways: a vertex of a triangle in the soup, an intersection between a triangle edge and a face at a point interior to the face, or an intersection of three triangles that occurs at a point interior to all three (see Figure 6). Each triangle is then subdivided into polygons by connecting incident nodes with segments. We place segments between any two incident nodes that are also on a given second triangle. That is, if node 1 and node 2 are both incident on triangle 1 and triangle 2, then a segment is added between

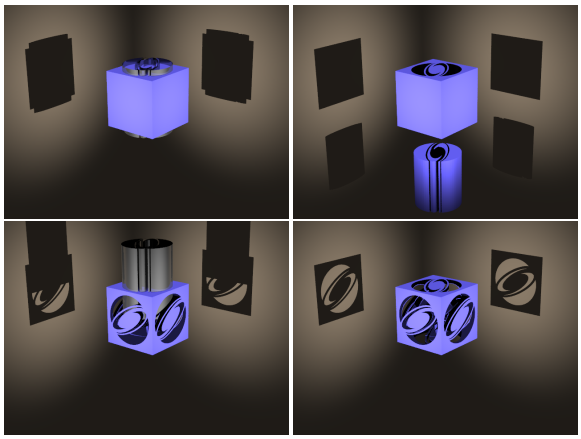

Figure 5: *A cookie cutter surface is used to cut away pieces of a 625 element tetrahedral mesh. Alternating between rotating and cutting allows the pattern to appear on all sides.*

them. We also add segments between any two nodes that lie on the same edge of a triangle from the soup. This definition results in overlapping segments when multiple nodes are collinear, and thus we only construct the minimal set of non-overlapping segments. Finally, the triangle is projected into two spatial dimensions where a boundary tracking algorithm is used to identify each closed polygon (see Figure 6).

3.2. Clipping to the Simulation Mesh

Besides resolving the cutting surface into non-intersecting polygons rather than triangles, a second key to our approach is to further subdivide the cutting surface in a manner that removes intersections with the faces of the tetrahedral simulation mesh. This allows for a tetrahedron-centric approach to the algorithm facilitating subsequent topological processing. This also preserves the incremental nature of the algorithm, since adding a new cut only affects the intersected tetrahedra. Thus, most importantly, we treat both cutting triangles and tetrahedron faces as elements of a large triangle soup and we resolve *all* of these triangles simultaneously into an intersection-free polygon mesh. As mentioned in the previous subsection, incrementally added cutting triangles are resolved with the the existing polygon mesh where our notion of a polygon mesh now includes polygons on tetrahedron faces as well.

In practice, our tetrahedron-centric algorithm is carried out as follows. First, a tetrahedron intersected by a cutting triangle is assigned a copy of that triangle for independent processing, where the triangle is projected into the tetrahedron's barycentric space. When polygonizing that triangle copy, we do so using a pool of segments collected only from the faces of this tetrahedron and any other cutting triangles that intersect this tetrahedron. The resulting polygons can be independently generated per tetrahedron. Since the faces

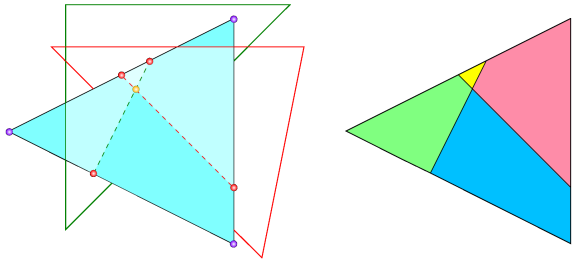

Figure 6: *Left: A triangle (colored blue) in the triangle soup intersects with other triangles. Nodes in the polygon mesh can be vertices of a triangle (shown in purple), intersections of triangle edges with triangle interiors (red), or intersections of three triangles (yellow). Right: We construct segments between nodes to tessellate the triangle into polygons that are intersection-free with all other cutting triangles.*

of the tetrahedron are also in the triangle soup, no polygon segment crosses a tetrahedron boundary, allowing us to trivially prune away polygons that are not on or within the tetrahedron. Note that neighboring tetrahedra that share a face will find the exact same polygons on that common face. See Figure 7, which illustrates a two-dimensional version of our algorithm, where a cutting segment spans three triangles from the background mesh. Since each triangle owns a copy of this cutting segment stored in barycentric coordinates, subsequent deformation of the mesh leads to three non-coincident copies of this cutting segment that only agree at element boundaries.

3.3. Per-Tetrahedron Subdivision

After each tetrahedron has generated its subset of the polygon mesh, we determine the volumetric connected components of material within each tetrahedron. Each connected component is a polyhedron that we compute using boundary tracking on the polygons in the tetrahedron. Then each connected component is embedded in a duplicate copy of the original tetrahedron. Unlike the virtual node algorithm, con-

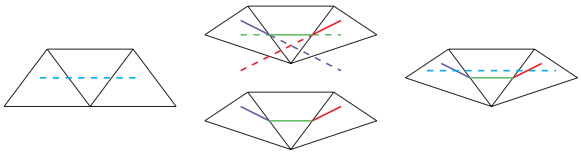

Figure 7: *Clipping the cutting elements in a two-dimensional version of our algorithm. A cutting segment spans three embedding triangles (left). Each triangle receives a copy of the segment. When the embedding deforms, the copies only remain coincident at element boundaries and are clipped to their respective embedding elements (middle). A second cut applied after deformation may intersect the first cut in multiple locations, which is easily resolved with our element-centric approach (right).*

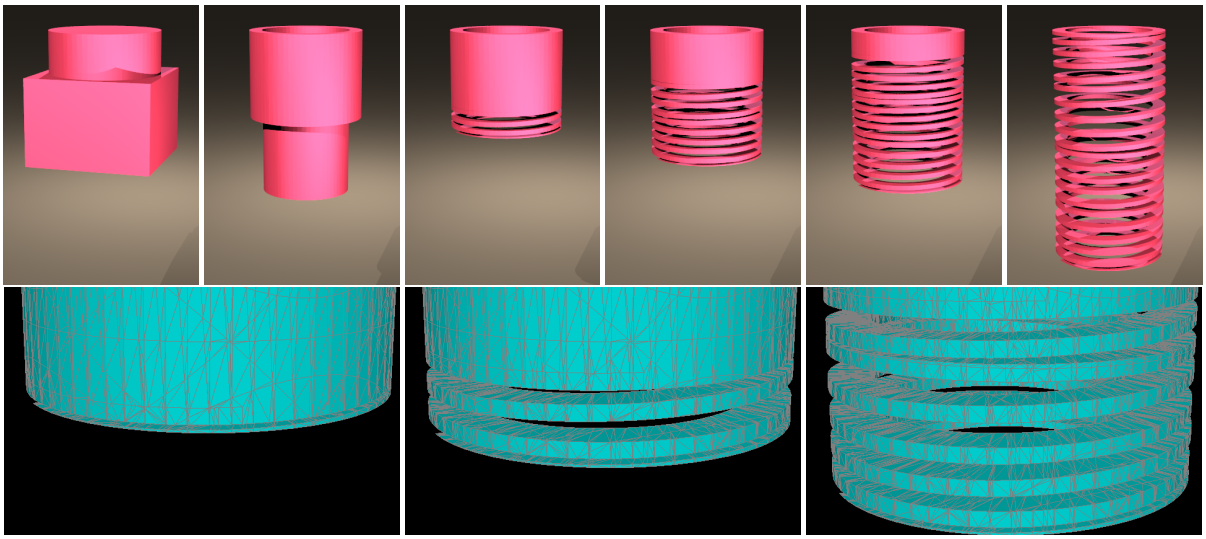

Figure 8: *Top: Using two cylindrical surfaces and a helical surface, we progressively carve a slinky out of a 320 element tetrahedral mesh, which expands and oscillates as it is cut. Bottom: A wireframe rendering reveals the automatic re-triangulation of the material surface as the cutting surface is spiraling up the deforming cylinder. The embedding mesh of the fully cut slinky contains 4713 tetrahedra.*

nected components may be completely interior to a tetrahedron and need not be incident on a node or a face (see Figure 2). When constructing the polygon mesh, tetrahedra that are only partially filled with material require special consideration. They are handled by constructing only the subset of the polygon mesh that is on or within actual material. This is straightforward since the material boundary is already a subset of the previously generated polygon mesh.

3.4. Determining Material Connectivity

Once each tetrahedron has been partitioned into its connected components, we identify pairs of connected components across adjacent tetrahedra as materially contiguous. We do so by examining each pair of tetrahedra that share a common face. Each tetrahedron has been duplicated into identical copies, each containing a single connected component of material embedded within it. The goal is to determine if a copy of tetrahedron A contains material contiguous to material in some copy of tetrahedron B, which we do by individually examining each possible pair of tetrahedron copies. We denote the copies as contiguous if their shared face contains a common polygon (see Figure 9). Finally, we collapse nodes on common faces of materially contiguous tetrahedra to obtain the final simulation mesh.

4. Examples

After cutting, we triangulate the resulting material surface for collisions, self-collisions, and rendering. We determine the material surface simply by keeping only the polygons incident on exactly one tetrahedron, removing those from the material's interior. The boundary polygons that remain are subsequently triangulated. We employ the hybrid framework of [SSIF07] to simulate the embedded material geometry, allowing us to use an arbitrary finite element constitutive model while resolving object collisions and self-collisions on a *soft constrained* copy (see [SSIF07]) of the material surface.

Figure 1 shows a two-dimensional version of our algorithm used to make a spiral-shaped cut into a suspended piece of cloth. Figure 3 illustrates that we can progressively cut a single tetrahedron into multiple pieces as it deforms. Figure 2 shows a single tetrahedron cut into over a thousand pieces by fifty cutting triangles, many of which only partially intersect the tetrahedron rather than slicing completely through it. Figure 4 shows 32 intersecting planes cutting a cube into 289 sticks. In addition, our cutting algorithm can be used to sculpt objects with high surface detail embedded

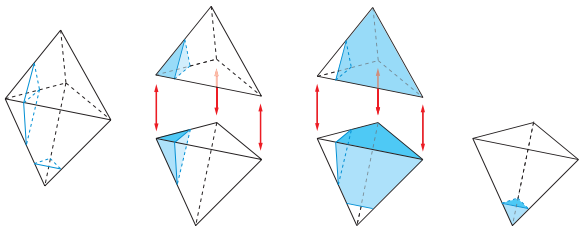

Figure 9: *A polygon mesh resulting from two cuts (left). Each connected component in a tetrahedron is embedded in a duplicate tetrahedron identical to the original. Two pairs of tetrahedra copies are found to be materially contiguous (two middle figures), while one copy of the lower tetrahedron remains materially disjoint (right).*

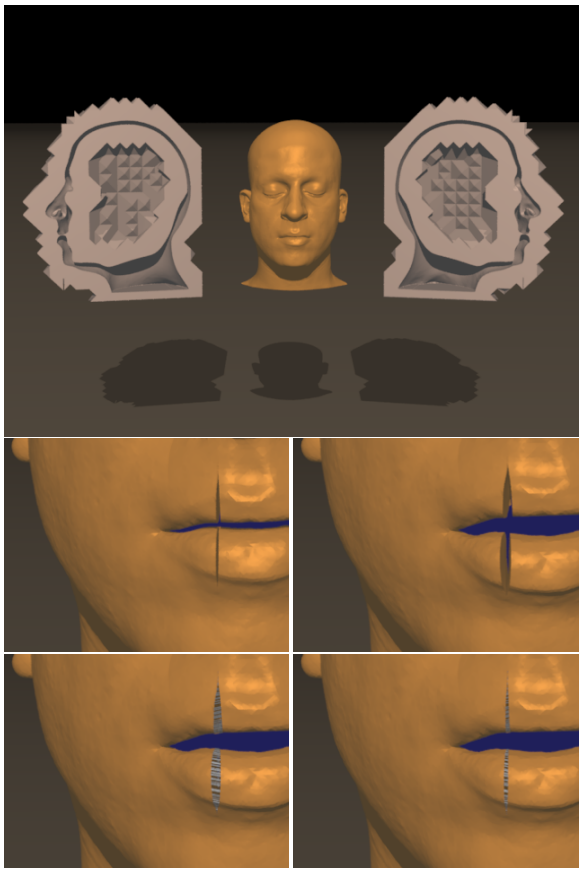

Figure 10: *Top: Using a triangulated skin surface as the cutting geometry, we carve an embedded flesh volume out of a BCC simulation mesh. Middle: An additional incision across the lips is made on the embedded mesh (left). The cut is simulated by pushing the flesh outwards from inside the mouth (right). Bottom: Applying springs across the cut closes the incision.*

within coarse simulation geometry. In Figure 5, we apply a cookie-cutter normal to each face of a cube. In Figure 8, we progressively sculpt a slinky out of a block. Figure 10 (top) shows our algorithm used as a meshing tool to embed a 250k triangle scan of the human face into an 850k tetrahedral simulation mesh. The resulting face model itself can then be further cut, as shown in Figure 10 (middle).

5. Discussion and Limitations

Despite the versatility of our cutting algorithm, a number of notable limitations exist. Our formulation assumes that cuts are introduced in the form of triangulated surfaces. Although this accommodates processes such as material fracture or explicit cutting (e.g. surgical incisions) by discretizing the cracks or incisions into a triangulated surface, there may be certain cases (e.g. elaborate scalpel models, embedding of geometries represented by spline or subdivision surfaces) where this additional triangulation is inconvenient. We note however that, with adequate refinement, a triangulated surface can represent any cut and even an increased triangle count on the cutting surface will not have a negative impact on the complexity or conditioning of the embedding simulation mesh.

Embedding the high resolution material surface in a coarser simulation mesh prevents CFL restrictions associated with small or sliver elements. Nevertheless, our framework may still create small or ill-conditioned triangles on the material surface which can still have a negative effect on collision handling. This is less of a problem for object collisions and more important for self-collisions, highlighting the need for robust collision handling schemes. Finally, although our algorithm can process any arbitrary cutting surface, a cut that partially intersects a tetrahedron without separating it into disconnected fragments will not allow the material to separate within that embedding tetrahedron. However, subsequent cuts can extend this cut into one that fully slices through the tetrahedron. Such partial cuts are geometrically resolved by our algorithm even if the embedding constraint does not allow separation of the material surface on either side of the cut. A subelement elasticity model (such as the soft binding formulation of [SSIF07]) could enable separation of these two surfaces, if desired.

6. Conclusion

We proposed a new cutting algorithm that allows for arbitrary cutting of tetrahedral meshes including those that possess a higher-resolution embedded material surface. We illustrated the efficacy of our algorithm on a number of examples ranging from cutting a single tetrahedron into over a thousand pieces to modeling and cutting a high-resolution face model embedded in a BCC simulation mesh.

Appendix A: Implementation Notes

In section 3 we defined our geometrical algorithm without descending into the implementation details of each geometric operation. This appendix provides specific details on the data structures and code organization in our implementation.

Our principle data structures are:

Per-tetrahedron Triangle List. Each embedding tetrahedron stores all triangles that border the material contained inside it. Prior to any cuts, each tetrahedron's triangle list contains just its four faces. After subsequent cutting, the triangles list will also contain triangles of the cutting surface. Barycentric coordinates of the triangle vertices with respect to the tetrahedron are also stored.

Per-tetrahedron Polygon Mesh. Each tetrahedron stores the polygonized boundary of its embedded material.

Intersection Registry. As mentioned in section 3.1 the points referenced by our algorithm are either vertices of the polygon soup, intersections of an edge and a triangle, or intersections of three triangles. In fact, we represent all such

points as the intersection of three triangles using the following process. If a segment pq is common to triangles pqr and pqs, then the intersection between pq and a different triangle xyz is encoded as the intersection of the three triangles pqr, pqs and xyz. If pqr is the only triangle incident on pq, we introduce a virtual triangle $pq\emptyset$ (where $\emptyset$ denotes an unspecified auxiliary point) such that the same point is encoded as the intersection of pqr, $pq\emptyset$, and xyz. Similarly, a vertex p of the triangle soup is represented as the common point of three triangles pqr, pst, and puv. If no such three triangles exist, e.g. if p only appears on pqr, the point may be described as the intersection of pqr, $pq\emptyset$, and $pr\emptyset$, where the last two virtual triangles have been introduced accordingly. The intersection registry stores which triangles intersect at each polygon point (which may be more than three) and determines whether the intersection of three given triangles has already been registered (possibly as the intersection of three different triangles). Additionally, we store the barycentric coordinates of each intersection point with respect to each of the triangles defining the intersection. Note that virtual triangles pose no problem as the barycentric weight of the virtual vertex will be identically zero.

An uncut tetrahedralized volume is initialized with the triangle list containing the faces of each tetrahedron, the polygon mesh containing the same faces as the polygonal (triangular) boundaries of each uncut tetrahedron, and the intersection registry containing all the vertices of the uncut volume as the intersection of all embedding faces incident on each of them.

Each new cutting triangle T_{new} added to the triangle soup is processed according to the following steps.

Intersection with embedding volume. T_{new} is assigned a unique index. We compute the set of embedding tetrahedra that intersect T_{new} (in world space as the embedding volume may have deformed) and create a copy T^k_{new} for each of them assigning each copy a unique index. Each T^k_{new} is appended to the triangle list for its corresponding tetrahedron, converted to barycentric coordinates with respect to it, and keeps a record of the unique index of its parent T_{new}.

Computation of new intersections. Each new triangle T^k_{new} is intersected against all possible pairs of old triangles in the list of its respective tetrahedron. If the intersection already exists, the intersection registry is updated to include T^k_{new} as a triangle incident on this intersection. Otherwise, a new intersection is registered and a unique index is assigned.

Update of old polygons. Any old polygon edge in the polygon mesh for a tetrahedron intersected by T_{new} that contains a newly introduced intersection point is bisected into two collinear edges, and the polygon mesh is updated accordingly. Additionally, any newly created segments on the plane of an existing polygon are tested for intersection against that polygon. If they are found to be intersecting, the old polygons are split to include the new segment.

Creation of new polygons. Each triangle T^k_{new} is clipped to the material contained in the embedding tetrahedron to yield a new polygon, which is added to the polygon mesh.

Computation of connected components. The polygon mesh for each tetrahedron is used to compute the new connected components of material, as in section 3.3. Each component is defined as a polyhedral volume and the tetrahedron is split into as many copies as distinct material fragments.

Topology update based on connectivity. The process of section 3.4 is used to rebuild the topology of the embedding volume. All of our data structures are updated to reflect the topological changes.

This process may be iterated for each new triangle added to the cutting surface. In practice, we have generalized the previous steps to accommodate the introduction of any number of new cutting triangles at once.

Acknowledgments

Research supported in part by an ONR YIP award and a PECASE award (ONR N00014-01-1-0620), a Packard Foundation Fellowship, ONR N0014-06-1-0393, ONR N00014-05-1-0479 for a computing cluster, ONR N00014-02-1-0720, ONR N00014-06-1-0505, ARO DAAD19-03-1-0331, NSF CCF-0541148, NSF ITR-0205671, NSF IIS-0326388, NSF ACI-0323866 and NIH U54-GM072970. Special thanks to Joseph Teran, Court Cutting and Aaron Oliker for invaluable feedback on developing this algorithm in the context of a surgical simulator.

References

[BG00] BIELSER D., GROSS M.: Interactive simulation of surgical cuts. In *Pacific Graph.* (2000), pp. 116–125.

[BGTG03] BIELSER D., GLARDON P., TESCHNER M., GROSS M.: A state machine for real-time cutting of tetrahedral meshes. In *Pacific Graph.* (2003), pp. 377–386.

[BHTF06] BAO Z., HONG J., TERAN J., FEDKIW R.: Fracturing rigid materials. *IEEE Trans. Viz. Comput. Graph.* (2006). (in press).

[BMG99] BIELSER D., MAIWALD V. A., GROSS M. H.: Interactive cuts through 3-dimensional soft tissue. In *Eurographics* (1999).

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

[GLB*06] GUO X., LI X., BAO Y., GU X., QIN H.: Meshless thin-shell simulation based on global conformal parameterization. *IEEE Trans. on Vis. and Comput. Graph. 12*, 3 (2006), 375–385.

[GO01] GANOVELLI F., O'SULLIVAN C.: Animating cuts with on-the-fly re-meshing. In *Eurographics 2001, Short Presentations Programme* (2001).

[GSH*05] GINGOLD Y., SECORD A., HAN J., GRINSPUN E., ZORIN D.: A discrete model for inelastic deformations of thin shells. In *Poster, Eurographics/ACM SIGGRAPH Symp. on Comput. Anima.* (2005).

[HTK98] HIROTA K., TANOUE Y., KANEKO T.: Generation of crack patterns with a physical model. *The Vis. Comput. 14* (1998), 126–187.

[KGC*96] KOCH R. M., GROSS M. H., CARLS F. R., VON BUREN D. F., FANKHAUSER G., PARISH Y. I. H.: Simulating facial surgery using finite element models. *Comput. Graph. 30*, Annual Conf. Series (1996), 421–428.

[KGPG96] KEEVE E., GIROD S., PFEIFLE P., GIROD B.: Anatomy-based facial tissue modeling using the finite element method. In *Proc. of Visualization* (1996), pp. 21–28.

[LIGF06] LOSASSO F., IRVING G., GUENDELMAN E., FEDKIW R.: Melting and burning solids into liquids and gases. *IEEE Trans. on Vis. and Comput. Graph. 12*, 3 (2006), 343–352.

[MBF04] MOLINO N., BAO Z., FEDKIW R.: A virtual node algorithm for changing mesh topology during simulation. *ACM Trans. Graph. (SIGGRAPH Proc.) 23* (2004), 385–392.

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

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

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

[MMDJ01] MÜLLER M., MCMILLAN L., DORSEY J., JAGNOW R.: Real-time simulation of deformation and fracture of stiff materials. In *Comput. Anim. and Sim. '01* (2001), Proc. Eurographics Wrkshp., Eurographics Assoc., pp. 99–111.

[MTG04] MÜLLER M., TESCHNER M., GROSS M.: Physically-based simulation of objects represented by surface meshes. In *Proc. Comput. Graph. Int.* (June 2004), pp. 156–165.

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

[NTB*91] NORTON A., TURK G., BACON B., GERTH J., SWEENEY P.: Animation of fracture by physical modeling. *Vis. Comput. 7*, 4 (1991), 210–219.

[NvdS00] NIENHUYS H.-W., VAN DER STAPPEN A. F.: Combining finite element deformation with cutting for surgery simulations. In *Eurographics 2000, Short Presentations Programme* (2000).

[NvdS01] NIENHUYS H.-W., VAN DER STAPPEN A. F.: *Supporting cuts and finite element deformation in interactive surgery simulation*. Tech. rep., Utrecht University, Institute for Information and Computing Sciences, 2001.

[OBH02] O'BRIEN J., BARGTEIL A., HODGINS J.: Graphical modeling of ductile fracture. *ACM Trans. Graph. (SIGGRAPH Proc.) 21* (2002), 291–294.

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

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ P., GROSS M., GUIBAS L.: Meshless animation of fracturing solids. *ACM Trans. Graph. (SIGGRAPH Proc.) 24*, 3 (2005), 957–964.

[PRZ92] PIEPER S., ROSEN J., ZELTZER D.: Interactive graphics for plastic surgery: A task-level analysis and implementation. In *Proc. of Symp. on Int. 3D Graph.* (1992), ACM Press, pp. 127–134.

[SHGS06] STEINEMANN D., HARDERS M., GROSS M., SZEKELY G.: Hybrid cutting of deformable solids. In *Proc. of the IEEE Virtual Reality Conference* (2006), pp. 35–42.

[SHS01] SERBY D., HARDERS M., SZÉKELY G.: A new approach to cutting into finite element models. In *MICCAI* (2001), pp. 425–433.

[SOG06] STEINEMANN D., OTADUY M. A., GROSS M.: Fast arbitrary splitting of deforming objects. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2006), pp. 63–72.

[SSIF07] SIFAKIS E., SHINAR T., IRVING G., FEDKIW R.: Hybrid simulation of deformable solids. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim. (in press)* (2007).

[SWB01] SMITH J., WITKIN A., BARAFF D.: Fast and controllable simulation of the shattering of brittle objects. In *Comput. Graph. Forum*, Duke D., Scopigno R., (Eds.), vol. 20(2). Blackwell Publishing, 2001, pp. 81–91.

[TF88a] TERZOPOULOS D., FLEISCHER K.: Deformable models. *The Vis. Comput. 4*, 6 (1988), 306–331.

[TF88b] TERZOPOULOS D., FLEISCHER K.: Modeling inelastic deformation: viscoelasticity, plasticity, fracture. *Comput. Graph. (SIGGRAPH Proc.)* (1988), 269–278.

[TSSB*05] TERAN J., SIFAKIS E., SALINAS-BLEMKER S., NG-THOW-HING V., LAU C., FEDKIW R.: Creating and simulating skeletal muscle from the visible human data set. *IEEE Trans. on Vis. and Comput. Graph. 11*, 3 (2005), 317–328.

[WSG05] WICKE M., STEINEMANN D., GROSS M.: Efficient animation of point-sampled thin shells. In *Proc. of Eurographics* (2005), vol. 24.

[YOH00] YNGVE G. D., O'BRIEN J. F., HODGINS J. K.: Animating explosions. In *Proc. of ACM SIGGRAPH 2000* (2000), pp. 29–36.