A New Approach to Cutting Tetrahedral Meshes

Menion Croll

August 9, 2007

1 Introduction

Volumetric models provide a realistic representation of three dimensional objects above and beyond what traditional surface models offer. The finite element method along with a volumetric model of an object can be used to create an effective representation of a deformable object that responds properly to external forces and stimuli. In particular, cutting is an important feature in many application areas for deformable 3D models.

Surgery simulation is one obvious application area in which cutting a deformable 3D model takes on critical importance. In order to properly model the effects of a cut on soft tissue, it is important to model the effects of the cut on tissue accurately in real time. To that end, this paper presents an algorithm which satisfies the unique requirements of cutting in such a model, specifically it must be efficient enough to run interactively, it must not create an excessive number of new tetrahedra which would complicate the finite element method, and it must accurately cut the mesh.

2 Previous Work

The vast majority of work done in interactive cutting has involved surface meshes. The primary idea presented in [2] was to not introduce new nodes when performing a cut on a surface mesh, but rather to rearrange the existing nodes to account for the topological changes. In [3] a Delaunay approach to cutting in triangulated surfaces is presented, which keeps the size of the mesh small and element quality high. [4] proposes an algorithm for cutting 3D surface meshes which accurately tracks the cutting tool, but introduces a large number of additional faces.

A state machine is used in [1] to identify the subdivisions necessary to cut each tetrahedron. While it accurately processes arbitrary intersections in the tetrahedral mesh, it produces many new tetrahedra due to the subdivision scheme, which is undesirable for use in conjunction with the finite element method. The algorithm presented in [5] is similar in concept to the algorithm we propose in that the existing vertices of the tetrahedral mesh are realigned...
Figure 1: The object to be cut is a green cylinder with the red lines representing the start and stop of the cut plane and copied during a cut in order to not introduce extra tetrahedra. The difference between the two algorithms exists primarily in the identification of which vertices will be moved.

3 Cutting Algorithm

Given a tetrahedral mesh and two coplanar lines intersecting the mesh, which represent the previous position of the cutting tool and the current position of the cutting tool, this algorithm creates a cut plane between those two lines. Using this cut plane we then rearranges the vertices of the tetrahedra such that no tetrahedron contains vertices on both sides of the cut plane. This is achieved in five basic steps.

1. Identify the cut plane.
2. Determine which tetrahedra intersect with the cut plane
3. Move the vertices of the intersected tetrahedra to the cut plane
4. Split the vertices which have been moved
5. Remove degenerate tetrahedra

3.1 Identify the Cut Plane

Assuming the two lines representing the position of the cutting tool are parallel, the identification of the cut plane is trivial. Given two points \((p1, p2)\) from one line, and one point \((p3)\) from the other line, the normal of the plane \((n)\) is given by

\[
\mathbf{n} = (p1 - p2) \times (p3 - p2).
\] (1)
Figure 2: The planes which define the cut to be made. The teal plane is the cutplane, while the purple and yellow planes are the back and front planes respectively.

The general equation of the plane is

$$ax + by + cz + d = 0.$$  \hfill (2)

where $a, b, c$ are the components of $n$, and

$$d = -ap_1x - bp_1y - cp_1z.$$  \hfill (3)

It is convenient to use the Hessian normal form \cite{6} of the plane for later calculations, which is given by the unit normal vector

$$\hat{n} = \frac{(p_1 - p_2) \times (p_3 - p_2)}{|(p_1 - p_2) \times (p_3 - p_2)|},$$  \hfill (4)

along with the constant

$$p = \frac{d}{\sqrt{a^2 + b^2 + c^2}}.$$  \hfill (5)

With the final equation of the plane being

$$\hat{n} \cdot x = -p.$$  \hfill (6)

Using the normal of the cut plane, two planes perpendicular to the cut plane can be identified: the plane which borders the back side of the cut (the back plane), and the plane which is on the leading edge of the cut (the front plane). To generate these planes, two points $p_1$, and $p_2$ can be selected from the respective line, while $p_3$ can be generated via the normal vector of the cut plane

$$p_3 = p_1 + n.$$  \hfill (7)

The equation for the front and back plane is generated using the steps outlined above.
3.2 Determine which tetrahedra intersect the cut plane

Each vertex in the model is classified based on its location with respect to the three planes listed above. A vertex may be involved in the cut if it is between the back plane and the front plane. Each of these vertices is further classified based on which side of the cut plane it resides. Since the three planes are already in Hessian normal form, distance from a plane for any vertex $v$ is given by

$$D = \hat{n} \cdot v + p,$$

and the sign of $D$ represents on which side of the plane the vertex resides.

From this stage it is simple to determine which tetrahedra intersect the cut plane. If every vertex of the tetrahedron is on the same side of the cut plane, it does not intersect the cut plane, otherwise it does intersect the cut plane.

3.3 Move the vertices of the intersected tetrahedra to the cut plane

Moving the vertices to the cut plane is a two step process. First we must determine which vertices to move, and second we must decide where on the cut plane to move them.

To determine which vertex to move, for each of the affected tetrahedra, we find which side of the cut plane the tetrahedron primarily resides. There are several metrics which could be used to identify this. We chose to use the side which contains the point which is the greatest distance from the cut plane. The
reason for this is to minimize the stretching of the tetrahedra which contain that vertex, thereby reducing the number of degenerate tetrahedra created.

Identification of which side of the cutplane a tetrahedron should reside is given by

$$\alpha = \text{sign}(D_i) \text{if} \forall j \neq i |D_i| > |D_j|.$$  \hfill (9)

Therefore, a vertex $v$ must be moved if $\text{sign}(v) \neq \alpha$. From here there are two cases.

- If this vertex is on the interior of the object, it is moved orthogonally to the cut plane. For a vertex $v$,

$$v = v + n \cdot D$$ \hfill (10)

where $n$ is the normal of the plane and $D$ is the distance from the plane.

- If this vertex is on the surface of the object, we need to keep it on the surface plane to prevent inaccurate distortions of the object. Therefore we find the line of intersection between the surface plane and the cut plane, and move the vertex to the closest point on that line. The line can be represented by a point on the line and a vector. The vector for the line, $a$, is given by

$$a = s \times c.$$ \hfill (11)

where $s$ is the normal vector for the surface on which the point resides, and $c$ is the normal vector of the cutplane. A point, $p1$, on the line can be found by finding the intersection of a line in the surface plane with the cutplane. A second point, $p2$, can be found by adding the vector of the line to $p1$,

$$p2 = p1 + a.$$ \hfill (12)

The parameter $t$ of the point on the line closest to the point to be moved is given by

$$t = \frac{(p1 - v) \times (p2 - p1)}{|p2 - p1|^2}.$$ \hfill (13)

The vertex can now be moved, to the point on the line described by $t$,

$$v = p1 + t(p2 - p1).$$ \hfill (14)

At this point there is a seam in the object where the appropriate vertices have been moved to the cut plane, but if it were left at this point the edges of the cut would be ragged. To correct this, the tetrahedra that intersect either the front or back plane of the cut have any points which have been moved to the cut plane moved a second time to the respective front or back plane. The procedure for this second move is achieved through the exact process outlined above.
3.4 Split the vertices which have been moved

In order to produce the actual cut, all of the vertices which have been moved are split in two. Each tetrahedron which contains a moved vertex is then associated with one of the copies, based on which side of the cut plane it resides. Effectively, every tetrahedron on the positive side of the cut plane keeps the original vertex, and every tetrahedron on the negative side of the cut plane is connected to the copy of the vertex.

3.5 Remove degenerate tetrahedra

Due to the number of vertices that are rearranged, some will become degenerate, which is undesirable. These tetrahedra are identified and removed. Any tetrahedra which has every vertex moved to the cutplane is considered degenerate and removed from the model.

4 Results

As shown, the algorithm creates a straight cut between two points. An analysis of the complexity of the algorithm as well as experimental results follows.

4.1 Algorithm Complexity

Given a model with \( n \) vertices and \( m \) tetrahedra, identification of the cutplane can be done in linear time for each segment of the cut. Determining which tetrahedra intersect the cutplane requires \( 3n \) distance calculations in the worst case to determine how far each vertex is from each of the 3 planes involved in
the cut. Additionally, determining that 4 numbers are not all equal requires 3 comparisons in the worst case. Comparisons must be made to determine which tetrahedra straddle the cut plane. Each point is moved a maximum of two times, once to the cut plane, and once to either the front or back plane of the cut. A single move can be done in constant time, resulting in 2n additional calculations. The moved vertices are then split, which incurs an additional cost of at most n constant time calculations. Finally, removing degenerate tetrahedra requires m calculations.

Summing the costs for each phase of the cut results in a complexity of

\[ f(n) = 3n + 3m + 2n + n + m \]

\[ f(n) = 6n + 4m. \]

Therefore,

\[ f(n) \in O(n + m). \]

4.2 Run Times

All tests were conducted on a machine running Windows XP Professional, using an Intel Core2 Quad CPU @ 2.66GHz, 4GB of RAM, and an NVIDIA GeForce 8800 GTX video card.

On a model consisting of 3234 vertices and 15272 tetrahedra, this algorithm required .047 seconds to make the cut. A model with 1720 vertices and 7844 tetrahedra required .016 seconds to cut.

5 Future Work

This algorithm allows for a reasonably accurate cut or series of cuts to be made on an object. As currently proposed, the run time increases linearly with the size of the object being cut. By using a more intelligent strategy to identify which tetrahedra are affected by the cut, it is possible to create an algorithm in which the run time increases with the size of the cut instead.

Interactive cuts, specifically a series of very small cuts less than the diameter of a tetrahedron may cause problems for this algorithm, especially for cases where the front plane and back plane both intersect the same tetrahedron. In order to be effective for simulations using haptic devices or other high frequency input devices this must be resolved.

As it stands, this algorithm requires that the lines representing the front and back edges of a cut be coplanar, which is unrealistic for most application areas. In order to resolve this, we propose that instead of a single cut plane, no less than two planes should be used for cutting. One plane generated from 2 points on the back line and 1 point from the frontline, with the second plane generated from 1 point on the back line and 2 points from the frontline. This would double the complexity of the algorithm, but would allow more interesting cuts of the object.
References


