A Delaunay approach to interactive cutting in triangulated surfaces

Han-Wen Nienhuys

A. Frank van der Stappen

institute of information and computing sciences, utrecht university

technical report UU-CS-2002-044

www.cs.uu.nl

A Delaunay approach to interactive cutting in triangulated surfaces

Han-Wen Nienhuys and A. Frank van der Stappen*

November 8, 2002

Abstract

We present a method for producing cuts in triangulated surfaces. This method keeps the mesh size low and element quality high. We show the method for triangle meshes in two dimensions, and then generalize it to three dimensional curved surfaces, where bifurcations and annihilations of incisions may occur. This method could be applied to simulating surgery of membrane-like structures, such as veins or intestine.

1 Introduction

This paper is about simulating cuts in triangulated surfaces. The motivation of this work is formed by interactive surgery simulation [2, 8, 11, 15, 16]: computer simulations of surgical procedures can bring down the costs and risks associated with learning to perform difficult operations.

Such simulations combine interactive deformation of virtual tissue with simulations of surgical procedures. The Finite Element Method (FEM) seems suited for computing deformation since it is physically accurate. It is a technique for solving partial differential equations (PDEs) of various types. Meshes form the basis of this technique: the domain of the problem is subdivided in geometric primitives ('elements') such as tetrahedra or triangles. With this subdivision the PDE can be transformed in a system of numerical equations. For the rest of the discussion, we will assume that the PDE is linear and derives from an elasticity problem. The equations can then be discretized into a linear system, where the matrix is called *stiffness matrix*.

Finding a solution is a two step process: first, the stiffness matrix is constructed, then the unknowns (a set of deformations) are computed from the stiffness matrix and given loads. The characteristics of a mesh influence both steps. The accuracy of the discretization process is determined by element shapes. For triangle meshes, large elements and large angles decrease the accuracy of the FEM discretisation (analogously, large dihedral angles do so for tetrahedra). The accuracy of the solution process in finite precision arithmetic is determined by the condition number of the stiffness matrix, where a lower condition number is better. The convergence speed of iterative solution methods also depends on the condition number. This holds for both the conjugate gradient algorithm (a static approach), and for damped dynamic relaxations using explicit time integration. The condition number of a complete stiffness matrix can be bounded by the condition numbers for separate elements. In contrast to the FEM discretisation error, high condition numbers are caused by elements with small sizes and small angles. So, the requirements for optimal accuracy in the discretisation and for optimal speed and accuracy in the numerical process are contradictory. However, it is safe to say that a mesh with 'round' elements, i.e. no angles close to π and 0, and bounded element sizes is a good general purpose mesh.

Iterative algorithms seem well suited as numerical solution strategies for surgery simulations, since they allow on-line mesh changes [8, 16]. If such an iterative approach is used, then mesh quality becomes an issue of vital importance: the maximum size of simulations is largely determined

*{hanwen,frankst}@cs.uu.nl

by mesh characterics: better meshes yield faster convergence, enabling larger and more accurate simulations. It follows that simulated surgical manipulations (such as needle insertions, cuts, and cauterizations) should be designed to keep mesh quality high. Secondly, the manipulations should also keep the complexity of the mesh low, since the cost of a single iteration step is proportional to the size of the mesh.

Simulation of cuts in surgery simulation is related to simulation of other destructive surgical procedures. The first operation to have been simulated on volumetric meshes is cauterization. This was done by removing elements in contact with a virtual cauterization tool [8]. A disadvantage of element removal is that it produces a jagged surface on the virtual tissue.

For cutting, subdivision methods are the norm [2, 11, 15]: elements that are in contact with the scalpel are marked active, and subdivided to produce a cut conforming to the scalpel position. The subdivision moves along with the scalpel during its stay in an active tetrahedron. When the scalpel leaves an active element, the subdivision is entered in the mesh permanently. A 2D example of subdivision is in Figure 12.

Subdivision methods always increase the size of the mesh. Moreover, these methods tend to produce degeneracies. This is caused by the use of an *active region*. Mesh modification is done only within a fixed region of the mesh, and if the scalpel moves close to the boundary of that region, poorly-shaped elements are inevitable.

Ganovelli and O'Sullivan [12] try to counter the the degeneracies caused by subdivision cutting. They deal with these degeneracies by collapsing short edges of the mesh. This approach does improve the quality of the mesh, but this solution does not repair all inconsistencies: not all edges may be contracted, and flat triangles and tetrahedrons, which do not contain short edges but are still degenerate, are not dealt with.

We have previously tried an approach where scalpel nodes are snapped to the trajectory swept by the scalpel [16]. The advantage of this method is that the mesh size remains small, and few short edges are created. However, there are a number of disadvantages: since no new nodes are created, the resolution of the cut is bounded by the mesh resolution. The incision does not reach up to the position of the scalpel, but lags behind it. A more serious problem is that snapping can result in degenerate elements in the mesh. Such degeneracies are dealt with by subdividing flat elements, and collapsing the resulting short edges, effectively removing the flat element. Unfortunately, not all edges can be collapsed. Using existing mesh features as a basis for mesh modification is problematic: when the scalpel is not especially close to a mesh feature, it may not be possible to match the mesh topology to the scalpel path without introducing degeneracies.

In summary, producing high quality cuts in tetrahedral meshes is a difficult problem. For cutting in surfaces, there is an analogous problem, which has the same difficulties as volumetric cutting approaches. Serby et al. [19] propose a method which also relies on a form of node snapping: the scalpel is modeled as a line segment, and nodes from the mesh are projected onto that segment. In a post-processing step, the edge lengths and element volumes are optimized using a particle system. The result is a good looking cut, without a decrease in mesh quality and size. However, in reality, the path of a scalpel is not a large line segment, but a concatenation of several small ones, and their approach does not seem to address this issue. Bruyns and Senger [5] use surface cutting with subdivision in large-scale simulations of various procedures.

In light of the complications of volumetric cutting, we decided to take a step back, and analyze the cutting problem for surface meshes first. In this paper we will present a method that produces cuts in a triangulated surface which does not decrease the mesh quality and keeps the mesh size low. We have analyzed this problem primarily to gain insight into the cutting problem for 3D tetrahedral meshes. Nevertheless, this technique could be applied in surgery simulation for membrane-like structures, such as skin or intestine.

2 Cutting in 2D

We state the general mesh cutting problem as follows: given a starting mesh, and positions of a user-controlled scalpel, modify the mesh at every moment to show an incision that represents the

past trajectory of the scalpel, and ends exactly at the scalpel. The challenge in this problem is to produce a well-shaped mesh with few elements. The simplest case is the two-dimensional form. Here the mesh is a triangulation in the plane, and the virtual scalpel is a point. The challenge is to produce elements that have no large angles and no short edges.

2.1 Delaunay triangulation

Since we are putting emphasis on the quality of the mesh, we briefly review the Delaunay Triangulation (DT), a popular technique for generating a well-shaped triangulation of a given set of points. The DT of a set of points is defined as a triangulation of that set where the circumcircle of every triangle does not contain any other points from the set. This property is also referred to as the empty-circle property. The empty-circle property can also be defined for edges of the mesh: an edge is called *legal* or *Delaunay* when the circumcircles of its incident triangles do not contain the opposite node of the other triangle. An example of an illegal edge is in Figure 1. Delaunay triangulations and Delaunay edges are intimately related: a Delaunay triangulation only has Delaunay edges.

Non-Delaunay or illegal edges of a triangulation can always be flipped: the two triangles incident with the edge always form a convex quadrilateral, and the diagonal may be switched. The flipped diagonal is always Delaunay, and the minimum angle of the pair of triangles is always increased, thus improving mesh quality. Flipping illegal edges only affects a single part of the mesh, so it can be seen as local improvement strategy. The Delaunay triangulation can be constructed by starting with an arbitrary triangulation and flipping illegal edges until none are left. The final result maximizes the minimum element angle. This is called the maxmin-angle property.

There are various ways to measure element quality: for example, minimum angle, circumradius to shortest edge ratio, circumradius to inscribed radius ratio, etc. In 2D all these measures are equivalent up to constant factors [14], so the maxmin-angle property of the DT means that it is a reasonable meshing technique for virtually all element quality measures. When coupled with algorithms for point insertion it is a basis for many refinement meshing techniques [1, 7, 18].

Figure 1: An illegal edge has a triangle whose circumcircle contains the opposite vertex of a neighboring triangle (left). By reconnecting ('flipping') the edge, the circumcircles become disjoint (right).

2.2 Cutting and Delaunay flips

Our problem, cutting in meshes, is different from the standard meshing problem. A starting mesh is given, and the shape of our domain is variable: as the cut progresses, the shape of the boundary is changed. The Delaunay Triangulation assumes a given set of points, and it is always convex. Cuts result in non-convex boundaries, so the standard DT is not directly usable. Nevertheless, we can retain the idea of using edge flips to locally improve the mesh.

Our approach works as follows: when cutting, the scalpel is attached to a node, the *active node*, so moving the scalpel moves the active node. The active node is always part of the boundary, so it is incident to two boundary edges, the *cut edges*. During a cut, these edges almost coincide geometrically, and form an *incision*. We call triangles incident with the active node *active triangles*. The active node is moved, and after each movement, local remeshing is applied to the active triangles. The technique is demonstrated in Figure 2; a more elaborate example is in Figure 4. The remeshing process consists of the following actions.

- Edges are flipped to improve the element angles.

- If another node is found close to the active node, it is removed.
- The incision is split, thus introducing new nodes to approximate the scalpel path.

During a cut, nodes are removed in front of the scalpel, and inserted behind the scalpel.

The rationale for flipping and removing nodes is that the Delaunay criterion tends to flip away triangles with angles approaching 180°, since these have large circumcircles. Nodes that are close together get connected. Such triangles have a small angle opposite a short edge. These elements are dealt with by removing nodes that are close to the active node. This can only be done for nodes that are not part of the boundary: removing boundary nodes would change the shape of the object represented.

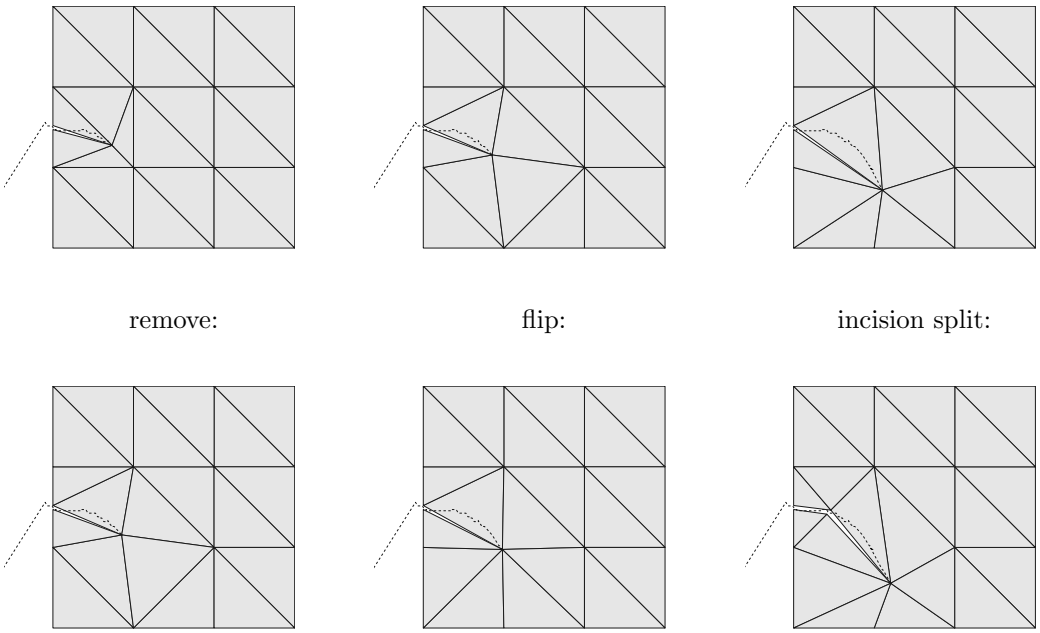

Figure 2: The evolution of a simple 2D cut is depicted from left to right. The key steps are to flip edges (center), to remove close nodes (left), and to insert nodes behind the scalpel (right).

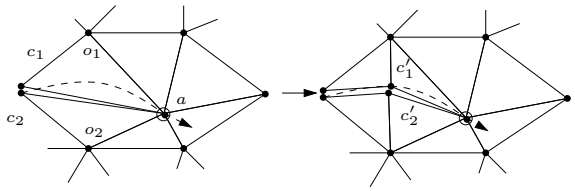

Figure 3: During an incision split, nodes are inserted in the cut edges. This happens when ac, with $c = (c_1 + c_2)/2$, fails the empty-circle criterion as diagonal of aco_1o_2.

During a cut, the cut edges are in almost in the same location. From a geometric point of view, we can identify both cut edges into a single edge, and check if this edge satisfies the empty-circle criterion. If it does not, new nodes are inserted, where the line connecting nodes opposite the cut edges intersect the scalpel path. We call this procedure a *incision split*, and it is demonstrated in Figure 3. When the scalpel passed that point previously, close nodes were removed, so the split

will not lead to short edges. The newly inserted nodes are dilated slightly, to prevent numerical problems when a path self-intersects. The new nodes always lie on a line that connects two existing mesh nodes, hence the accuracy of the represented trajectory is determined by the resolution of the starting mesh.

When the scalpel enters or exits the mesh, it is close to the boundary of the mesh, so a realistic incision would contain very short edges. To prevent these short edges, special precautions are taken: incisions are postponed and exits are done in advance.

When the scalpel enters the mesh, an active node is created. If the entry happens close to an existing node, we move it and label it active. Otherwise, a new node is inserted at at the entry point. The active node moves along with the scalpel, creating a temporary dent. When the active node is sufficiently far from the entry point, it is "fixed up": nodes are added at the entry point, creating an incision. If the scalpel is *retracting* (moving away) before this fix-up happens, the dent is left in the mesh permanently. Before a fix-up, no incision splits take place.

When exiting, the the cut is finished before the scalpel comes too close to the boundary. At every step the movement of the scalpel is extrapolated. When this extrapolation hits the boundary, and is close to the active node, the cut is finished: a node is inserted at the extrapolation, and the cut is dissected. This exit procedure still leaves relatively short edges permanently in the mesh. To rectify this, these edges are contracted. The boundary edges that are created with this exit operation are added to a list of forbidden edges. These edges are not tested for collisions during ensuing movements. This prevents artifacts when the actual scalpel movement differs from the predicted movement. If the scalpel is sufficiently far from the exit point, they are eligible for collision checking again.

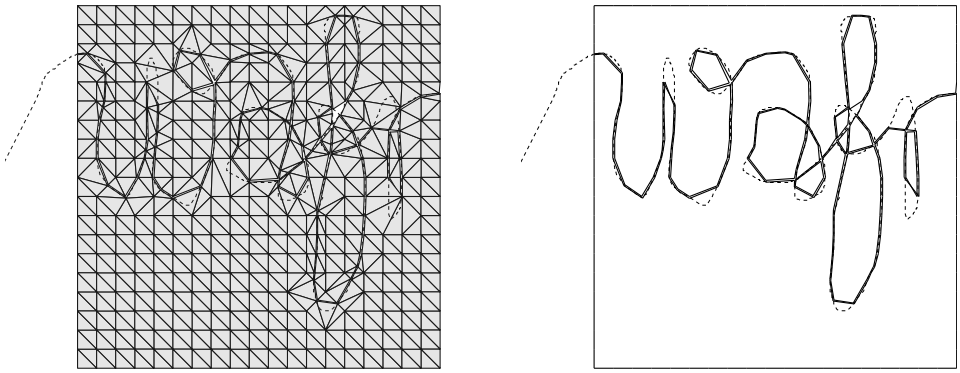

Figure 4: Delaunay cutting on a triangle mesh. Both the mesh itself and the boundary are shown. The scalpel trajectory is indicated with a dotted curve. Notice how strongly curved path segments are cut short in the realized cut. The starting mesh was regular and consisted of 722 triangles. The cut increased mesh size by 41 triangles.

The cutting routines must maintain references to active parts of the mesh, while changes are applied to the mesh. This implies that certain mesh features must be persistent across such change operations. This persistence was enable by using a data structure that resembles standard data structures for subdivision storage, like the half-edge data structure [13]. The difference is that our data structure is only used for manifold simplicial complexes. It also works for tetrahedral subdivisions, and can be manipulated with only two very generic operations, being replace-triangles and change-triangles. Oriented simplexes are associated explicitly with the Edge and Triangle objects representing them. This allows mesh changes to be specified formally in terms of simplexes, and it makes it easy catch programming errors that inadvertently create non-manifold meshes.

Pointers linking Triangle and Edge objects allow for efficient traversal of the subdivision. In this framework, edge flips are specified in terms of replace-triangles, can be performed in constant time, and Edge objects that are also present in the new flipped mesh are retained, making pointers to these objects persistent. Similarly, dissecting internal faces of the mesh can be specified concisely in terms of change-triangles, and pointers to the faces involved are persistent across this change.

3 Surface cuts in 3D

Triangulated surfaces in 3D are a common concept in computer graphics. They have been used for surgery simulations [4, 6]. In these cases elastic behavior was simulated with damped mass-spring systems instead of the FEM. Nevertheless, the concerns for mesh quality continue to hold: flat elements are inverted more easily, and small elements correspond to short springs with small masses. Their high vibration frequencies translate into expensive integration schemes.

In the 2D scheme, the scalpel is a point, and it is attached to a single active node. Triangles are remeshed in the vicinity of the active node. There are two generalizations of the 2D scheme: first, the remeshing process around an active node can be done for curved instead of flat surfaces, assuming a line-shaped scalpel. Second, such a line-shaped scalpel can intersect a curved 3D surface in multiple points, so a consistent model of cutting allows multiple incisions, each with an active node. These active nodes can interact: incisions may meet, leading to annihilations, or incisions may hit folds, leading branches.

We assume that the surface is given as a triangle mesh with a boundary, and that no further information on the surface shape is known. The virtual scalpel is a line segment, and its movement is given by sampled positions of its endpoints. The endpoints are assumed to move with constant velocity between the samples. During a cut, active nodes are part of the boundary of the surface. The two boundary edges incident with an active node again are called cut edges, and define an incision.

3.1 Single incision

A scalpel movement is handled as follows: the line segment representing the new scalpel position is intersected with all active triangles. If a single intersection is found, then the active node is moved to that point. The active triangles are subjected to flipping. The 2D criterion is used to determine whether an edge is flipped: an edge is considered illegal when the two triangles incident to that edge would be illegal in a 2D triangulation. Conceptually, we could say that the incident triangles are unfolded to be coplanar, and then the two dimensional criterion is used. It is not clear whether this flipping criterion leads to a terminating algorithm when applied to the edges of an arbitary 3D surface mesh. In this sense, this technique is now truly a heuristic.

Flipping on 3D surfaces is a delicate operation: some flips are topologically impossible (as demonstrated in Figure 5). This means that the all operations must be checked for failure cases.

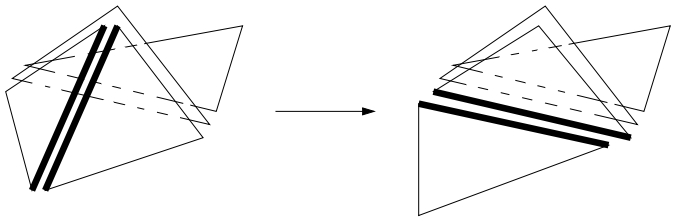

Figure 5: Not all flips in 3D are topologically valid: when flipping the bold edge on the left, the new edge (bold) occurs twice in the resulting complex. The triangles are displayed in an exploded view for clarity.

Incisions are split analogously to the 2D case: during a cut, the last path of the scalpel is stored. When the cut edges violate the 3D empty-circle criterion, new nodes are inserted where

the path is closest to the line connecting the nodes opposite the cut edges.

Node removal is done in a heuristic manner. Suppose that we want to remove a node v that is incident with n triangles. We say that two adjacent triangles incident with v form an *ear*, if the sum of the angles opposite v is less than π. When removing v, all ears are flipped until v is incident with only 3 triangles. Then v is removed, and the involved edges are flipped back if they violate the empty-circle criterion. Ears always exist: suppose that the triangles incident with v are numbered $i = 1, \ldots, n$, and have angles $\alpha_i, \beta_i, \gamma_i$, where γ_i is the angle at v (See Figure 6). Then there must be an $\alpha_i + \beta_{(i+1) \bmod n} < \pi$, since $n\pi = \sum_i (\alpha_i + \beta_i + \gamma_i) = \sum_i (\alpha_i + \beta_{(i+1) \bmod n}) + \sum_i \gamma_i$, and $\sum_i \gamma_i > 0$.

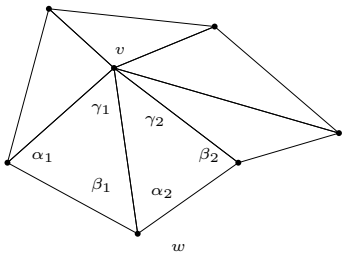

Figure 6: When removing a node, ears are flipped until v is in only 3 triangles. An ear has $\alpha_i + \beta_{(i+1) \bmod n} < \pi$, so the triangles incident with vw form an ear.

When the scalpel is almost parallel to the surface, small movements of the scalpel can result in large movements of an active node. For this reason it is necessary to control large movements. Large movements are subdivided using a maximum distance. This distance is computed as the minimum for all distances between the line spanned by the scalpel and lines spanned by the edge opposite the active node, as indicated in Figure 7.

If the end-points of the scalpel segment cannot move further than $d_{\max}$ as indicated in Figure 7, then the scalpel will not hit that edge. This can be shown as follows: suppose that the scalpel position is given by the line segment $\vec{h}\vec{t}$ for some $\vec{h}, \vec{t} \in \mathbb{R}^3$, and we have a movement $\vec{p}, \vec{q} \in \mathbb{R}^3$, such that $\|\vec{p}\|, \|\vec{q}\| < d_{\max}$. If the movement ends puts the scalpel on the line spanned by the edge, then $\lambda(\vec{p} + \vec{h}) + (1 - \lambda)(\vec{q} + \vec{t})$ is on the edge indicated for some $0 \leq \lambda \leq 1$. By the definition of $d_{\max}$ we have

$$d_{\max} \leq \|(\lambda(\vec{p} + \vec{h}) + (1 - \lambda)(\vec{q} + \vec{t})) - (\lambda\vec{h} + (1 - \lambda))\vec{t}\|$$

On the other hand, the latter expression equals $\|(\lambda\vec{p} + (1 - \lambda)\vec{q}\|$, which be bounded by $\lambda\|\vec{p}\| + (1 - \lambda)\|\vec{q}\| < d_{\max}$. This is a contradiction, so the movement $\vec{p}$, $\vec{q}$ can not hit the edge.

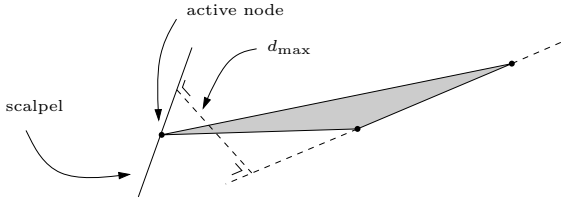

Figure 7: The maximum movement $d_{\max}$ for a scalpel and a single triangle. The total maximum distance is given by the minimum over all triangles incident with the active node.

A similar mechanism for large movement control is also used in our prototype of the 2D mechanism, where it compensates for the slowness of the implementation.

3.2 Multiple incisions

In 3D, the scalpel can interact with the entire surface, and the scalpel may enter the mesh in any place. We can distinguish three cases, as demonstrated in Figure 8: the scalpel hits a boundary

edge of the mesh, the scalpel tip enters through a triangle, or the scalpel hits an internal edge of the mesh. Only the first case also occurs in 2D. It is handled completely analogous to the 2D case.

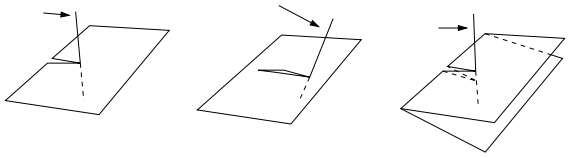

Figure 8: There are three ways for the scalpel to enter: by hitting the boundary, entering with the tip or hitting an exposed internal edge.

In the second case, a new active node is inserted in the incised triangle. When the active node is far enough from the entry point, a node is inserted at the original entry point. The result is an edge that connects the entry node with the active node. During the next incision split a real incision is formed. Until that time, the edge is constrained, so it cannot be flipped away. The third case is when the scalpel hits an exposed edge of the mesh. Then a single active node is inserted in the edge. During the next movement, the cut will branch into two incisions. We now discuss branching and annihilations.

The scalpel is represented by a line segment, and line segments can interact with curved surfaces in many places: the scalpel may incise the surface in multiple locations, and during a cut a single incision may branch into multiple incisions, as is shown in Figure 10. We could apply the 3D remeshing from the previous section, if we could rule out any interactions between different incisions. Fortunately, this seems possible: we can forbid interactions by ensuring that every triangles is incident with at most one active node. This is achieved by the following restrictions.

- Incisions in edges or triangles that already active are rejected.
- Nodes that separate active nodes cannot be removed.
- Edges that separate two active nodes cannot be flipped.

When an active node moves towards a restricted node, an annihilation is performed: all edges from the restricted node to an active node are dissected. This is demonstrated in Figure 9. When an active node comes close to an edge separating active nodes, then its movement is extrapolated, and a new node is inserted at the extrapolated point. The annihilation now proceeds with the newly inserted node. A normal boundary exit is a special case of an annihilation.

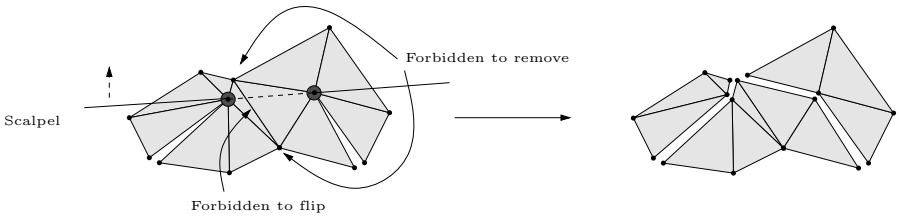

Figure 9: Edges that separate incisions may not be flipped, nodes separating incisions may not be removed. Instead, when an active node comes too close to such a forbidden node or edge, an annihilation is performed (right).

When a scalpel movement is processed, the next position of the active node is determined by intersecting all triangles incident with the new scalpel position. If multiple active triangles are intersected, as demonstrated in Figure 10, then the cut will branch. New incisions are inserted, the old incision is no longer active, and the edges connecting old and new active nodes are dissected. During a branch, the new nodes are close to the original incision. This leads to short edges and degenerate triangles. When the scalpel progresses further, these short edges disappear.

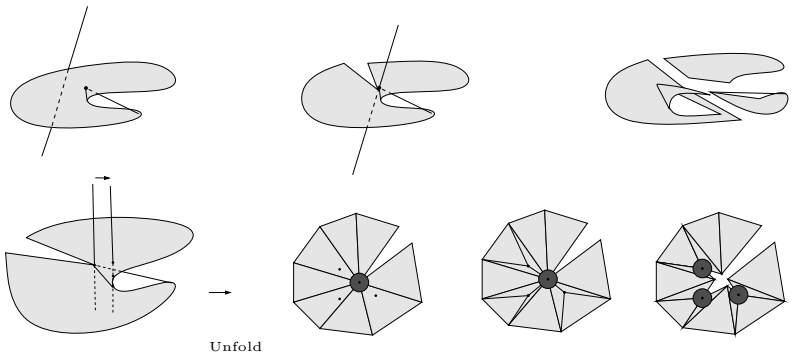

Figure 10: Folded surfaces may lead to branching cuts (top). In such cases, a movement will cause the scalpel to intersect multiple triangles (bottom; intersection points are indicated by dots). When this happens, multiple incisions replace the old active node.

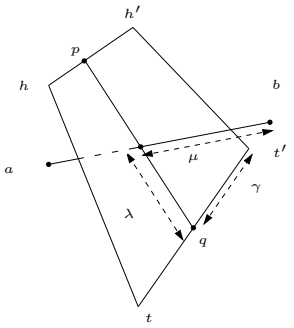

Figure 11: Intersecting a line sweep with an edge

Collisions are computed as scalpel/edge intersections. The procedure for computing such intersections is as follows. We assume that the scalpel moves from $\vec{h}\vec{t}$ to $\vec{h}'\vec{t}'$ for some $\vec{h}, \vec{t}, \vec{h}', \vec{t}' \in \mathbb{R}^3$, and that we want to compute intersections with the line segment $\vec{a}\vec{b}$, for $\vec{a}, \vec{b} \in \mathbb{R}^3$ (See Figure 11). In other words, we want to solve

$$\vec{p} = \gamma\vec{h} + (1-\gamma)\vec{h}'$$
$$\vec{q} = \gamma\vec{t} + (1-\gamma)\vec{t}'$$
$$\lambda\vec{p} + (1-\lambda)\vec{q} = \mu\vec{a} + (1-\mu)\vec{b}$$
$$\lambda, \gamma, \mu \in [0,1]$$

Substituting the first two equations in the last one yields the equation

$$\lambda\gamma((\vec{h}-\vec{t}) - (\vec{h}'-\vec{t}')) + \lambda(\vec{h}'-\vec{t}') + \mu(\vec{b}-\vec{a}) + \gamma(\vec{t}-\vec{t}') = \vec{b}-\vec{t}'.$$

This is may be seen as a linear system in four variables. If we set $x = (\lambda\gamma, \lambda, \gamma, \mu)$, $\vec{c} = (\vec{h}-\vec{t}) - (\vec{h}'-\vec{t}')$ and write the 3×3 matrix $\vec{A}$ for $(\vec{h}'-\vec{t}', \vec{t}-\vec{t}', \vec{b}-\vec{a})$, then we can rewrite the equation as

$$(\vec{c}|\vec{A})x = \vec{b}-\vec{t}', \tag{1}$$

When viewed as a linear system, a solution maybe given as $y + \alpha k$, where $y \in \mathbb{R}^4$ is a particular solution of the system, and $k \in \mathbb{R}^4$ is in the kernel of $(\vec{c}|\vec{A})$. We set $k = (1, -\vec{A}^{-1}\vec{c})$ and $y = (0, \vec{A}^{-1}(\vec{b}-\vec{t}'))$. Using these values, we may derive a quadratic equation for α from Equation (1). We thus obtain triples (λ, γ, μ) that define intersections.

4 Results

The 2D version of this algorithm has been implemented in a small application written in Python. The virtual scalpel is controlled by the mouse. A sample is shown in Figures 4 and 12. The mesh was uniform on a square grid.

Fix-up, node removal and exits depend on points being close to or far from the active node. These notions are expressed in global constant thresholds for the distance. These thresholds are denoted by $\varepsilon_{\text{edge}}$ for node removal, $\varepsilon_{\text{entry}}$ for entry fixup, and $\varepsilon_{\text{exit}}$ for predicting exits. They were set to fraction of the average global edge length $\bar{h}$. For Figure 4 we used $\varepsilon_{\text{edge}} = \bar{h}/3$, and for Figure 12 we used $\bar{h}/3$ and $\bar{h}/8$. The entry and exit tolerances were set to $\bar{h}/6$. This choice is somewhat arbitrary, but if $\varepsilon_{\text{edge}}$ is set larger than $\bar{h}$, mesh complexity will be *reduced* during a cut. If it is larger than $\bar{h}/2$, then most nodes in the vicinity of the scalpel will be removed. Lower values than $\bar{h}/2$ set a threshold for edge length. This does not directly control the accuracy of the cut trajectory represented in the mesh: the example in Figure 12 shows that lowering the threshold can decrease accuracy.

In Figure 12 the new cutting scheme for 2D is compared with a subdivision approach, where each sliced triangle is replaced by three triangles. Elements shapes and edge lengths are much worse in the subdivision mesh, as evidenced by the histograms of minimum edge length and minimum triangle angle in Figures 13. The price for this better and smaller mesh is a decrease in resolution: the subdivision cut follows the scalpel path more closely.

The 3D version of this algorithm has been implemented in a small application written in C++. It also uses the mouse to obtain scalpel movements. Samples are shown in Figures 14 and 15. The 3D version assumes that all events are strictly ordered in time, and does not take special precautions for degenerate cases. When the scalpel enters or exit in a movement parallel to the surface in, multiple events happen simultaneously, which leads to various failures.

Flips can be implemented in constant time. Node removals can be accomplished in $\mathcal{O}(d \log d)$, where d is the degree of the node [9]. The expected value of d in surface meshes is 6, so the total remeshing process for a single movement can be implemented in constant time, which guarantees that the remeshing process itself is scalable to larger meshes. However, in the 3D version the scalpel can interact with all parts of the mesh at every step. In practice, efficient collision detection will be needed for a scalable implementation.

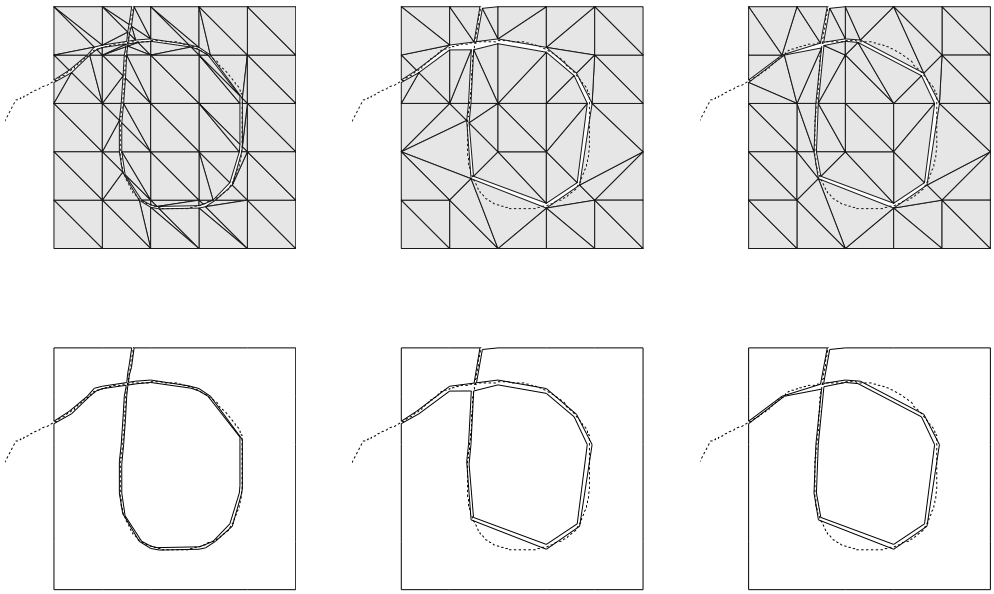

Figure 12: The same cut performed with both subdivision (left) and Delaunay cutting (center, $\varepsilon_{\text{edge}} = \bar{h}/3$ and right, $\varepsilon_{\text{edge}} = \bar{h}/8$). The starting mesh contains 50 triangles. The subdivision cut increases size by 62 triangles, the Delaunay cut in the center by 10, and on the right by 12 triangles. The scalpel trajectory is indicated with a dotted line. Notice that the mesh in the center picture matches the trajectory better than the mesh on the right, although $\varepsilon_{\text{edge}}$ was smaller.

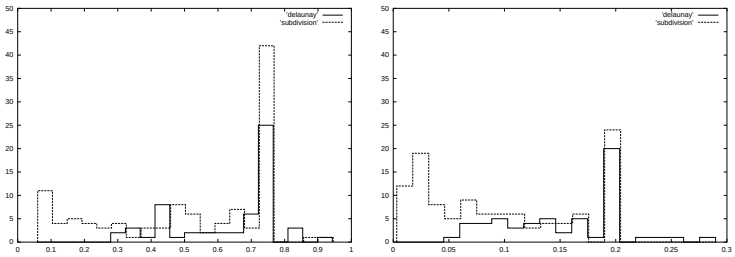

Figure 13: Histogram of minimum element angle (left) and minimum element edge for the subdivision and $\bar{h}/3$ Delaunay cut from Figure 12. The peak corresponds to the unaltered triangles (minimum angle $\pi/4$, minimum length 0.2). The Delaunay cut (solid line) yields larger minimum angles, and longer edges.

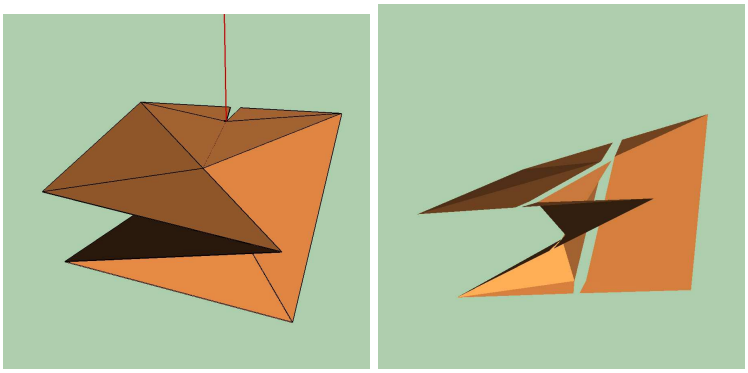

Figure 14: In principle, our approach also works for a cut branching into three cuts. The mesh has very sharp angles and a limited resolution, so edge flips produce an odd end result.

5 Discussion

We have presented an approach to cutting in triangulations that produces measurably smaller and better-shaped meshes than previous methods. The method is based on a model of a point-shaped scalpel that moves an active node through a static mesh. During movements, the area around the active node is remeshed. The approach generalizes to 3D surfaces. The 3D surface model supposes a line-shaped scalpel, and it is possible to generalize the single-incision technique to multiple incisions consistently.

The technique bears some resemblance to interactive mesh dragging, a technique proposed by Suzuki et al. [21] to allow dragging operations on triangle surfaces in interactive geometric modelers.

The technique that we have described is heuristic: it employs more or less arbitrary constant tolerances $\varepsilon_{\text{edge}}$, $\varepsilon_{\text{exit}}$ and $\varepsilon_{\text{entry}}$. This implies that the technique is only usable for meshes with a constant density, and that suitable values must be computed beforehand. In retrospect, it might have been wiser to consider an edge too short when the opposite angle in incident triangles is shorter than some threshold. This criterion is also local but scale independent.

We have used a static model for mesh cutting. In the context of surgery simulation and deformable models, this is not realistic: surgical instruments always interact with the deformed mesh, and exert force on the tissue that is cut. A full-fledged simulation would be able to respond to movements with reaction forces, which could be relayed to a force-feedback device. However, the remeshing processs is not affected by the presence of deformation. Both the 2D and 3D algorithm include a step where the new position of the scalpel is intersected with parts of the mesh. When deformation is present, these intersections are done with the deformed mesh. The deformation of the triangles involved can be used to translate the point back to the reference situation. Our

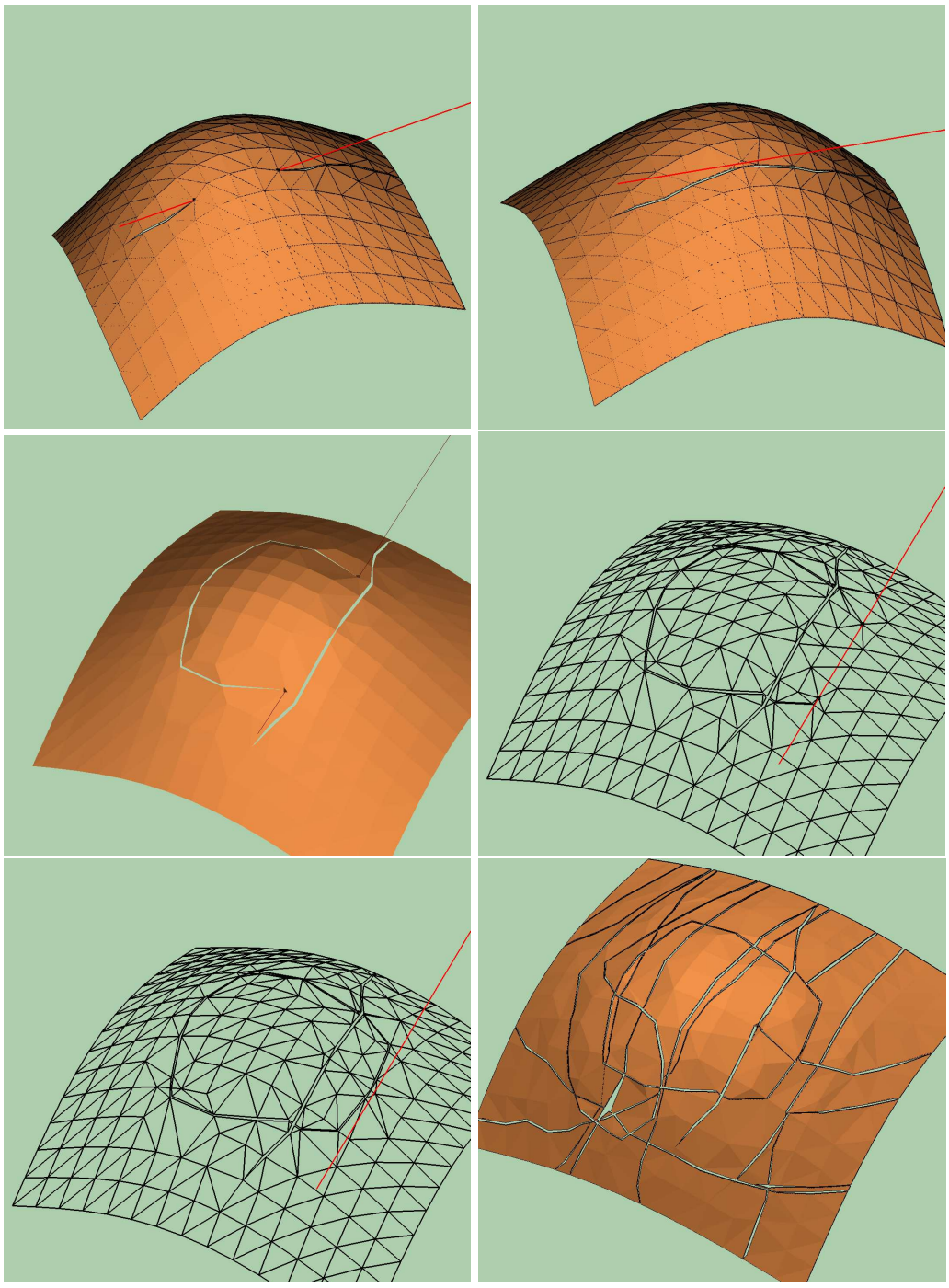

Figure 15: Cuts in a 392 triangle 3D surface mesh of a a Gaussian bell-curve. The first 5 images show the evolution of two cut movements. The final image demonstrates more self-intersecting paths.

technique can proceed as described using reference locations.

Entry and exit are changed by the presence of deformation: surgical instruments must overcome a threshold in force to puncture membranes covering organs [3, 10]. Cutting requires less force than puncture, so after the instrument enters the tissue, it will make an incision immediately, ensuring that entry does not lead to arbitrary small incision depths. This is a physical variant of our entry-fixup, and it would make our own entry-control superfluous. A similar observation might also hold for a scalpel finishing a cut. Removing entry control is attractive, since our approach depends on a heuristically chosen quantitity $\varepsilon_{\text{entry}}$, and it is possible to create a deadlock situation: if a scalpel entry has to be fixed up, the active node is part of the boundary. It is not always possible to preclude element inversion when moving boundary nodes, so the step-size control for large movements may halt further movements in some cases.

During a cut, nodes are added and removed; the nodes are added on line segments connecting existing nodes, which implies that the overall resolution of the mesh does not increase. Some interactive simulations of deformable objects refine meshes on demand to provide more accurate results in the region of interest [10, 17, 22]. Delaunay refinement algorithms [7, 18] seem to fit our framework of using Delaunay Triangulations, however more research is needed before this can be used in practice. Refinement algorithms need input geometries without small angles. Moreover, in 3D we are operating on a discretization of a curved surface, so it is not clear where to insert new points. Existing surface meshing algorithms, such as Chew's guaranteed quality surface Delaunay refinement [7], assume that a smooth description of the surface is available.

We have demonstrated an extension of our single incision 3D approach to multiple incisions. The extension is consistent with our model of the scalpel as a moving line segment. It remains to be seen how branching cuts should be combined with deformations.

Unfortunately, both the rationale for Delaunay and our heuristics do not readily generalize to tetrahedral meshes. Delaunay tetrahedralizations of well-spaced points admit *slivers*, flat tetrahedrons that have four nearly planar and nearly cocircular vertices (see Figure 16). Moreover, the higher dimensional equivalent of edge flipping (face flipping), does not always work: there exist configurations of non-Delaunay faces which can not be flipped. This implies that flipping as a local improvement strategy does not always work. If the tetrahedralization is already Delaunay, then flipping non-Delaunay faces during insertion or removal of points in a mesh is always possible. Since cuts are non-convex, this suggests that a tetrahedral generalization must build a constrained or conforming Delaunay tetrahedralization with a moving boundary. A first step towards enabling such cuts would be to extend the current 3D surface cutting approach to cuts of closed surfaces: when cutting a closed surface, surface triangles are added to the inside of the incision, so the surface remains closed during cuts. This surface could then be used as a basis for making a tetrahedralization. An additional complication is that a constrained Delaunay tetrahedralization only exists if edges have no non-incident nodes close by (the surface should be *ridge-protected*) [20].

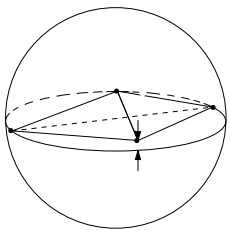

Figure 16: Slivers have an excellent circumcircle to shortest-edge ratio, so they can be present in Delaunay tetrahedralizations of well-spaced point sets. They are degenerate nevertheless.

References

[1] M. Bern and D. Eppstein. *Computing in Euclidian Geometry*, chapter Mesh generation and optimal triangulation, pages 47–123. World Scientific, 1995.

[2] D. Bielser and M. H. Gross. Interactive simulation of surgical cuts. In *Proc. Pacific Graphics 2000*, pages 116–125. IEEE Computer Society Press, October 2000.

[3] P. Brett, T. Parker, A. Harrison, T. Thomas, and A. Carr. Simulation of resistance forces acting on surgical needles. *J. of Engineering in Medicine*, 211(13):335–347, September 1997.

[4] M. Bro-Nielsen, D. Helfrick, B. Glass, X. Zeng, and H. Connacher. VR simulation of abdominal trauma surgery. In *Medicine Meets Virtual Reality 6*, pages 117–123. IOS Press, 1998.

[5] C. Bruyns and S. Senger. Interactive cutting of 3-D surface meshes. *Computer & Graphics*, 25(4):635–642, 2001.

[6] C. D. Bruyns, S. Senger, A. Menon, K. Montgomery, S. Wildermuth, and R. Boyle. A survey of interactive mesh-cutting techniques and a new method for implementing generalized mesh cutting using virtual tools. *The journal of visualization and computer animation*, 13:21–42, 2002.

[7] L. P. Chew. Guaranteed quality mesh generation for curved surfaces. In *Annual ACM Symposium on Computational Geometry*, pages 274–280, 1993.

[8] S. Cotin, H. Delingette, and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. *The Visual Computer*, 16(8):437–452, 2000.

[9] O. Devillers. On deletion in delaunay triangulations. In *Annual ACM Symposium on Computational Geometry*, June 1999.

[10] S. DiMaio and S. Salcudean. Needle insertion modelling and simulation. In *IEEE International Conference Robotics and Automation (ICRA)*, 2002.

[11] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. *Computer Graphics Forum*, 19(3):271–282, 2000.

[12] F. Ganovelli and C. O'Sullivan. Animating cuts with on-the-fly re-meshing. In J. C. Roberts, editor, *EuroGraphics Short Presentations*, 2001.

[13] M. Mäntylä. *An introduction to solid modeling*. Computer Science Press, College Park, MD, 1988.

[14] G. Miller, D. Talmor, S. Teng, N. Walkington, and H. Wang. Control volume meshes using sphere packing: Generation, refinement and coarsening. In *Fifth International Meshing Roundtable*, pages 47–61, October 1996.

[15] A. B. Mor and T. Kanade. Modifying soft tissue models: Progressive cutting with minimal new element creation. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, number 1935 in LNCS, pages 598–607. Springer-Verlag, 2000.

[16] H.-W. Nienhuys and A. F. van der Stappen. A surgery simulation supporting cuts and finite element deformation. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, number 2208 in LNCS, pages 153–160. Springer-Verlag, October 2001.

[17] C. Paloc, F. Bello, R. I. Kitney, and A. Darzi. Online multiresolution volumetric mass spring model for real time soft tissue deformation. In T. Dohi and R. Kikinis, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, number 2489 in LNCS, pages 291–226. Springer-Verlag, 2002.

[18] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[19] D. Serby, M. Harders, and G. Székely. A new approach to cutting in finite element models. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, number 2208 in LNCS, pages 425–433. Springer-Verlag, October 2001.

[20] J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained delaunay triangulations. In *Annual ACM Symposium on Computational Geometry*, 1998.

[21] H. Suzuki, T. Kanai, Y. Sakurai, and F. Kimura. Interactive mesh dragging with adaptive remeshing technique. In *Pacific Graphics*, 1998.

[22] X. Wu, M. S. Downes, T. Goktekin, and F. Tendick. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. *Computer Graphics Forum*, 20(3), 2001.