# Combining finite element deformation with cutting for surgery simulations

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

Department of Computer Science, Utrecht University, PO Box 80089, 3508 TB Utrecht, The Netherlands
{hanwen,frankst}@cs.uu.nl

**Abstract**

*Interactive surgery simulations have conflicting requirements of speed and accuracy. In this paper we show how to combine a relatively accurate deformation model—the Finite Element (FE) method—and interactive cutting without requiring expensive matrix updates or precomputation. Our approach uses an iterative algorithm for an interactive linear FE deformation simulation. The iterative process requires no global precomputation, so runtime changes of the mesh, i.e. cuts, can be simulated efficiently. Cuts are performed along faces of the mesh; this prevents growth of the mesh. We present a provably correct method for changing the mesh topology, and a satisfactory heuristic for determining along which faces to perform cuts. The incision surface will be jagged; this problem is a subject of current research.*

***Keywords:*** finite elements, tissue deformation, simplicial complex, virtual surgery, tetrahedral mesh, cutting

## 1. Introduction

Interactive surgery simulations aim to provide a learning tool for surgeons in training. The core of such a system is a simulation of deformation of human tissue. A realistic tissue simulation should include both elastic deformations, like stretching and shearing, and plastic deformations, like tearing and cutting.

Elastic deformation of soft material, such as tissue, is governed by complicated partial differential equations (PDEs). Exact solutions are generally impossible to derive; hence, we must either turn to approximative methods or alternative models. The most popular approximative methods are Finite Element (FE) methods. Alternative models are models inspired by physical concepts, but they lack a rigorous mathematical and physical justification. They are also called *physics based*.

Plastic deformations include tearing, fracturing, plastic stretching, and cutting. However, a full interactive simulation of these phenomena is currently far beyond reach, so in the context of interactive surgery simulation, plastic deformations are limited to changing the mesh that represents the tissue.

Before we present our solution, we will look at previous work in interactive surgery simulation and establish how the conflicting requirements of interactive speed, realistic deformations and cutting are balanced. We will focus solely on these subjects and ignore related topics such as haptic feedback, non-interactive simulation, display hardware, and effectivity assessment. We conclude with an overview of our approach.

ChainMail[11, 18, 19] was one of the first deformation techniques to simulate cutting in actual surgical simulations. ChainMail relies on a very simple model of deformation, which is not realistic but has low computational costs. ChainMail puts severe constraints on what mesh topology may be used.

Mass-spring models[2, 3, 6, 8, 9, 13, 15, 16] attach springs to the mesh that represents the tissue geometry. The elastic behavior of the springs can be changed to match physical properties of the material. The springs can have linear behavior, but non-linear or volumetric material properties[7, 13] may also be used.

A collection of springs strives to be in a state of locally minimal potential energy. An implementation is evident: one can use numerical methods to minimize the energy function of the system. However, a dynamic formu-

lation (points represent masses, forces cause acceleration, springs are damped) has always been more popular, presumably due to its simple numerical solution method.

Cutting in spring meshes can be implemented in two ways: in both approaches the scalpel is tracked and whenever it intersects with a spring, the mesh is modified. In the first approach, the spring is simply removed[3, 6, 9, 15, 16]. In the second approach, a colliding spring is split. For two-dimensional meshes, this technique has been used in a simulation[16] of a surgical procedure.

A research prototype by Bielser et al.[2] used a splitting technique for three dimensional volumetric meshes. They subdivided any tetrahedron intersected by the scalpel. Cuts add lots of small volume elements, and cause the mesh size to explode. Not only is this an evident disadvantage for interactive simulations, it also causes a deterioration of the numerical stability of the spring relaxation scheme. Additionally, rounding errors can cause artefacts such as dangling nodes.

In contrast to the preceding models, FE methods have a physical and mathetical foundation. FE methods discretize the body and then the discretization is deformed. The deformation of the discretization is an approximation of the exact solution of the PDEs. In its simplest form (static linear elasticity and linear geometry), the computational process of determining an FE solution amounts to solving a linear system $K\mathbf{u} = \mathbf{f}$, where $K$ has size $3n \times 3n$ and $n$ is the number of nodes in the mesh.

Bro-Nielsen et al.[4, 5] used a variety of precomputation techiques—that essentially reduce to precomputing the inverse of the matrix $K$—to achieve interactive performance with the FE method. Cutting was implemented by removing volume elements that collide with a virtual scalpel, and updating $K^{-1}$ to reflect this change. However, the removal violates the physical principle of mass preservation, and updating $K^{-1}$ to reflect a change in the mesh typically costs $\mathcal{O}(n^2)$ operations, which makes cutting prohibitively expensive in large meshes.

To our best knowledge previous work in interactive surgery simulation either uses a non-realistic deformation model, or does not include interactive cutting. Our approach combines an FE model with interactive cutting in large meshes.

Deformation is simulated using an FE method. Contrary to tradition, we use an iterative algorithm to solve the equations. The deformation model and its solution process are detailed in Section 2.

This iterative method does not require global precomputation, so local changes to the mesh require only local updates. Therefore cuts can be simulated efficiently. The finite element method is intrinsically volumetric, so we must deal with cuts in a volumetric mesh. We propose a method that does not inflate the mesh size: we simulate cuts by applying cuts only to faces, as is demonstrated in Figure 1, rightmost picture. This bounds the size of the mesh, and helps keeping the geometry of the mesh uniform. We divide the cutting problem into three subproblems:

1. Given the path of a scalpel, how does one find faces near the path? In Section 3 we introduce a heuristic that has turned out to be satisfactory.
2. Given these faces of the mesh, how does one perform the cut? In Section 4 we give an algorithm for performing the cut. This algorithm can proved to be correct; a sketch of the reasoning behind the proof is presented.
3. How do we prevent the cut surface (faces from the mesh to be cut) from having a jagged look? We hope to remedy this by changing the positions of vertices in the mesh; this will be fully addressed in future work.

Results of a prototype are given in Section 5. Section 6 discusses the results and topics for further research.

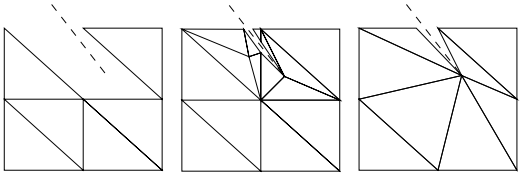

**Figure 1:** *Three approaches to cutting in volumetric meshes demonstrated in 2D. Current approaches: remove volume elements (left) and subdivide volume elements (center). We propose to cut along faces, which are edges in 2D. (rightmost)*

## 2. Deformation

A body made of elastic material subject to external forces will deform until its internal elastic forces counterbalance the external forces. The deformations and external forces satisfy a set of PDEs defined on the body.

The finite element method[20] is a process that approximates the continuous problem defined by the PDEs: the body under consideration is subdivided into a *finite* set of non-overlapping *elements*. If a solution of the PDE is assumed to have a simple form on every element, then the problem is changed to a finite-dimensional one, whose solution is an approximation of the original problem.

The most versatile and simple 3-dimensional mesh type is the tetrahedral mesh. We assume that the displacement function is piecewise linear on every tetrahedron. Hence, the function is uniquely characterized by the values that it assumes on the vertices of the tetrahedra (the *nodes* of the mesh). The forces can also be thought to be applied to the vertices.

The simplest model of material elasticity yields the following set of equations for a single tetrahedron. These equations link the elastic force $\mathbf{F}_j \in \mathbb{R}^3$ in node $j$ for $j = 0, \ldots, 3$

and displacements $\mathbf{u}_i \in \mathbb{R}^3$ of node $i$ for $i = 0,\ldots,3$:

$$\begin{aligned} \mathcal{X} &= (\mathbf{X}_1 - \mathbf{X}_0 | \mathbf{X}_2 - \mathbf{X}_0 | \mathbf{X}_3 - \mathbf{X}_0)\,, \\ \mathcal{U} &= (\mathbf{u}_1 - \mathbf{u}_0 | \mathbf{u}_2 - \mathbf{u}_0 | \mathbf{u}_3 - \mathbf{u}_0)\,, \\ \mathcal{E} &= \mathcal{U}\mathcal{X}^{-1} \\ (\mathbf{F}_1|\mathbf{F}_2|\mathbf{F}_3) &= \mathsf{volume} \cdot \mathcal{X}^{-T}(2\mu\mathcal{E} + \lambda\,\mathrm{trace}(\mathcal{E})\mathcal{I}) \\ \mathbf{F}_0 &= -\sum_j \mathbf{F}_j, \end{aligned} \tag{1}$$

Here, vectors $X_j \in \mathbb{R}^3$ represent the undeformed location of node $j$ for $j = 0,\ldots,3$. The numbers $\mu$ and $\lambda$ are positive constants, called Lamé parameters. They represent elasticity properties of the material and are equivalent to the Young modulus and Poisson's ratio. The trace of a matrix is the sum of its diagonal elements. Strain, a matrix denoted by $\mathcal{E}$, is a quantity that measures local relative deformation. The unit matrix is denoted by $\mathcal{I}$.

The body under scrutiny is composed of tetrahedral elements, and the forces within each element are described by Equations (1). By summing the equations over all elements, we obtain a linear relationship between all displacements and all elastic forces: the elastic forces are described by $K\mathbf{u}$, where $K$ is a matrix of size $3n \times 3n$, $\mathbf{u}$ a vector of size $3n$ and $n$ is the number of nodes in the mesh. The matrix $K$ describes the elastic reaction of the complete body to a displacement and is called the *stiffness* matrix. The vector $\mathbf{u}$ represents the collective displacements of all nodes. In a body in equilibrium, elastic forces $K\mathbf{u}$ balance external forces $\mathbf{f}$, so we have $K\mathbf{u} = \mathbf{f}$.

A complete deformation problem is obtained by imposing boundary conditions. For example we may constrain the coordinates from a set $B$ to be fixed on a certain position. This results in the following linear system:

$$\begin{aligned} K\mathbf{u} &= \mathbf{f} \\ \mathbf{u}(i) &= \mathbf{p}(i) \quad i \in B \subset \{1,\ldots,3n\} \\ & B,\ n,\ K,\ \mathbf{f}, \text{ and } \mathbf{p} \text{ given.} \end{aligned} \tag{2}$$

Entries $(3i+p, 3j+q)$ for $p,q = 0,1,2$ of $K$ represent the elastic force of node $i$ as a result of the displacement of node $j$. Hence, it is non-zero only if node $i$ and $j$ are in some tetrahedron together. Since most combinations of nodes do not share tetrahedra, $K$ is sparse, i.e., mostly filled with zeros.

By substituting the boundary conditions into $K$, we obtain a smaller, strictly positive definite matrix $\tilde{K}$. We want to solve the equation $\tilde{K}\tilde{\mathbf{u}} = \tilde{\mathbf{f}}$, where $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{f}}$ are the displacements and forces of unconstrained nodes.

### 2.1. Solution by matrix decomposition

Traditionally, the matrix $\tilde{K}$, is first calculated from the geometry of the discretized body and its material properties. The solution to Equation (2) is then calculated using techniques from numerical linear algebra: an inverse or a decomposition of $\tilde{K}$ is determined, and then multiple solutions can computed efficiently, so the response to force of the material can be calculated at a interactive rate.

In surgery simulations cuts have to be performed interactively. This implies that the mesh changes in real-time. A matrix $\tilde{K}$ (or worse, $\tilde{K}^{-1}$) that containing information about the mesh, will be invalidated by these changes. Reassembling, recomputing or updating the matrix not only adds significantly to the complexity of the simulation, it can also be very costly: updating the inverse to reflect removal of a node can cost $\mathcal{O}(n^2)$ operations[12].

### 2.2. Solution by iterative processes

Our approach is to solve the system iteratively, i.e., to find a sequence of displacement vectors $\mathbf{u}^1, \mathbf{u}^2, \ldots$, that gradually moves to a configuration with minimal elastic energy. For positive definite systems (like the elasticity equations), the conjugate gradient (CG) algorithm is a popular iterative algorithm[12].

CG determines a new iterand by first finding a search direction $\mathbf{p}$, then finding an optimal step length $\alpha$ and finally setting $\mathbf{u}^{k+1} = \alpha\mathbf{p} + \mathbf{u}^k$. One matrix-vector multiplication $\tilde{K}\mathbf{p}$ is needed for finding $\alpha$. The vector $\tilde{K}\mathbf{p}$ represents the elastic forces on every node (assuming displacements $\mathbf{p}$) so for every iteration we have to calculate $\tilde{K}\mathbf{u}$ once. From Equations (1) it follows that we can calculate the entries of $\tilde{K}\mathbf{p}$ (elastic forces), using local information: for every node we only need to consider information from neighboring nodes. In other words, $\tilde{K}\mathbf{p}$ can be calculated without calculating $\tilde{K}$ itself.

This scheme has been used for parallel implementations of FE analyses[1, 10]: the locality of computing $\tilde{K}\mathbf{p}$ is attractive property for a parallel machine with distributed memory. For surgery simulation, the lack of precomputed structures implies that it is possible to change meshes at run-time —i.e., simulate cuts and stitches—without significant computational penalties.

### 2.3. Performance

The performance of the simulation is determined by two factors:

- the cost of one iteration,
- the number of iterations needed to reach a satisfactory approximation of the solution of Equations (2).

The cost of a single iteration of the conjugate gradient method is directly proportional to the size of the mesh. A 1 Gflop/second computer would have a theoretical peak performance of approximately 1500 iterations per second for a 1000 node cube.

More interesting is the number of iterations needed. Theoretically speaking, the performance of conjugate gradients

is linear[12]: the error diminishes by a constant factor for every iteration. This factor can be bounded by

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}, \qquad \kappa = \|\tilde{K}\|_2 \|\tilde{K}^{-1}\|_2.$$

The quantity $\kappa$ is the condition number of $\tilde{K}$ in the Euclidean norm. This implies that it is advantageous to keep the condition number as low as possible: a lower condition number means a faster decrease in error. It is known that varying element sizes and mesh refinement adversely affect the condition number (see Strang and Fix[20], Chapter 5). Smaller elements increase the bound on the condition number. This motivates the following extra requirement on mesh modifications:

> Mesh modifications should avoid creating more elements, and should specifically avoid creating elements with varying sizes.

In an interactive simulation, the next solution will often be close to solution last computed. So we can expect that only few iterations will be needed. Moreover, in practice CG tends to perform much better than its theoretical bound.

### 2.4. Results

The simulation has been tested with a procedural model: a cube uniformly subdivided into tetrahedra. The number of tetrahedra was varied while the Lamé parameters were kept at $\lambda = \mu = 1$. To keep the simulation running, a random force was applied to a random node whenever a solution was found.

The speed in terms of floating point operations cost was comparable with the peak performance of the CPU, with a slight drop for larger meshes. Presumably, this drop is due to the larger models not fitting in the CPU cache.

The number of iterations has also been studied, but showed no evident correlation with mesh size. The algorithm generally reached the solution criterion in 5 to 80 iterations on our test set. However, our mesh had a uniform element size, and as we explained, this is the optimal choice for the CG algorithm.

A cited problem with iterative methods is their lack of reliable convergence[4]: it is not possible to exactly predict how many iterations are needed before reaching a solution. We did not experience this problem in most of the cases we tried.

## 3. Surface selection

Surgery involves cutting, so we have to modify the mesh that represents the tissue at run-time. Incising a tetrahedron yields shapes that are no longer tetrahedra (demonstrated in 2D in Figure 2). This leads to a dilemma: either we must subdivide an incised tetrahedron, or we must restrict cuts to be along faces only.

The former solution has the disadvantage of increasing complexity of the mesh, so we opt for the latter. A cut along faces does not change the number of tetrahedra, so the size of the mesh remains bounded. This scheme does lead to new questions. First, along which faces should the cut be performed? Second, how do we perform the cut? Third, how do we prevent a jagged incision surface? (The incision surface is a set of triangles in the mesh; hence the incision follows the jagged contours of the mesh).

We shall deal with the first question in this section. The second question is the subject of Section 4. The third question is a topic for further research, and will not be answered here.

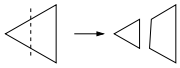

**Figure 2:** *Cutting in a triangular mesh generally does not result in a triangular mesh*

In order to pose the question ("Where to cut?") more precisely, we define some terms. The *scalpel* is the cutting instrument. It is thought to be line-segment-shaped. The entire line segment will act as a blade.

The *scalpel sweep* (or *sweep* for short) is the surface swept by the line segment shaped scalpel. Without loss of generality we may assume that the scalpel sweep has been triangulated.

The *cut surface* is a set of faces where the cut takes place. The *mesh* is a volumetric mesh composed of tetrahedra.

With these terms we can state the question to be answered in this section precisely:

> How do we find an appropriate cut surface in the mesh given a scalpel sweep?

Not every set of faces is acceptable as a cut surface: the surface must be close to the sweep, and its topology must resemble that of the sweep. Typically the scalpel sweep is a connected, non-branching surface, so the cut surface should also be connected and non-branching. Some cut surfaces in 2D that fail to meet these requirements are shown in Figure 3.

We obtained satisfying results with the following method. Consider one edge from the mesh and one face of the scalpel sweep. When we intersect the face with that edge, the vertices closest to the intersection point are put into a *closest-vertex set*. If the sweep and edge do not intersect, then this set is empty. If they do, the set normally contains one vertex. In the degenerate case that both vertices are equally close, the set contains both.

Every tetrahedron in the mesh has six edges, and we can form the union of the closest-vertex sets of those edges. This set determines a feature to select for the cut surface: if it has

zero elements (the sweep does not intersect the tetrahedron at all), then no feature is selected; If it has one, then a node is selected; if it has two, an edge is selected; if it has three, a triangle is selected. The process is demonstrated in Figure 4. The set may also contain four nodes, but this occurs only in a degenerate case: when all intersections are equally close. This is proved in Lemma 1.

In this manner we can select triangles from the tetrahedra that intersect a face of the sweep. These triangles form a cut surface. When we want to process multiple consecutive triangles of the sweep, we add the closest-vertex sets of edge-sweep pairs consecutively.

**Lemma 1** The closest-point set of tetrahedron $t$ and plane $V$ contains no more than three points, except if all points have the same distance to $V$.

*Proof* (Sketch) A plane separates space into two half-spaces. There are three possible configurations: all points of $t$ lie in the same half-space. Then no edge will intersect the plane, and the closest-point set is empty. If the plane separates a single point from the rest of $t$, then the plane will intersect only three edges, and the closest-point set contains at most three points. So assume that the plane separates two pairs of points, and intersects four edges. Then from each edge a different point is selected. This gives us four inequalities for the distance between intersection points and nodes. When combining these, we can conclude that all distances are equal. □

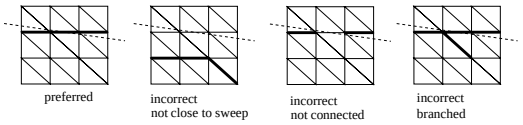


**Figure 3:** *Selecting a cut surface in 2D. Given the sweep (dotted line) of a point shaped scalpel through a triangular mesh, which edges should be selected to cut open the mesh? Shown is a correct solution (left) and cut surfaces that are too far from the sweep, unconnected or branched.*

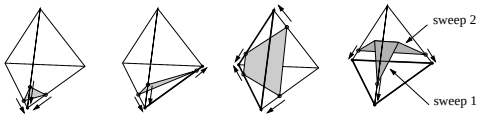


**Figure 4:** *Selecting features to be cut from a single tetrahedron. The scalpel sweep is colored grey, the selected features are marked by bold lines. From left to right: selecting a node, an edge and a triangle, selecting with multiple sweep planes.*

We do not have proof that this approach satisfies all criteria we set up. In fact, in cases like the situation depicted in Figure 5, it fails. However, the following observations make it likely that the approach is satisfying in most cases.

- The cut surface is close the scalpel sweep, since features are only selected from tetrahedra where the sweep passes through.
- The cut surface is unlikely to branch, because at most one triangle is selected from each tetrahedron; this limits the amount of triangles in a cut surface.
- The cut surface is unlikely to be disconnected: if the sweep passes through two adjacent tetrahedra, these tetrahedra will also share the sweep/edge intersections of their shared face, so they will share parts of the cut-surface;

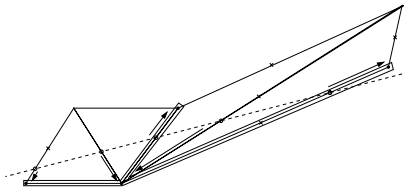

**Figure 5:** *Varying element sizes result in branched cut surfaces in 2D. The scalpel path is dotted. Selected edges are boxed.*

Besides the failure in Figure 5, this technique has a couple more disadvantages: the surface of the incision will be rough, as it coincides with the contours of the mesh. An incision is only made after the scalpel has left the tetrahedron, so the incision that is visible always lags behind the position of the scalpel. We hope to remedy these disadvantages by combining surface selection with a snap operation: this operation will move nodes of the cut surface onto the scalpel sweep. Then the shape of cut surface will match the shape of the scalpel sweep, and a node will be moved to the scalpel before the scalpel completely traverses a tetrahedron, hopefully eliminating the lag between the scalpel and cut realized.

There is a final, different problem: the algorithm assumes that the mesh remains still, and this assumption is questionable for a deformable object. This problem manifests itself in two ways: first, the deformation process has to be halted, so the mesh will be frozen while faces are selected. Secondly, moving the mesh while the scalpel remains steady does not cause a cut surface to be selected. The real impact of these problems depends on the size of the deformations and scalpel movements compared to the size of the mesh, and these should be measured in real applications.

## 4. Topology of cutting

In the previous section, we contemplated where to perform cuts. This leads us to the question of this section:

How to perform cuts?

The surface selection process does not give us a cut surface with guaranteed geometrical or topological properties;

as seen in Figure 5 we must be prepared for branching surfaces. The geometry of the surface is undetermined: cut surfaces coincide with the mesh, and approximate the scalpel sweep, both of which do not have a predetermined shape. Therefore we cannot rely on the geometry of the mesh and cut surface. All we have left is their topology: the abstract structure of nodes, edges, faces, and tetrahedra. In this section, we will explain how the topology of the mesh is used to determine which parts of the mesh will be separated by the cut surface, and which will not.

We start with a slightly more formal definition of the mesh operation that is needed.

**Definition 2** Given a tetrahedral mesh and a set of triangles in that mesh, the unglue operation modifies the mesh such that these and only these triangles will be added to the boundary of the mesh.

This definition uses the concepts 'tetrahedral mesh' and 'boundary', so for a full definition of the operation we have to decide how to represent the mesh. We shall do this after informally specifying the algorithm.

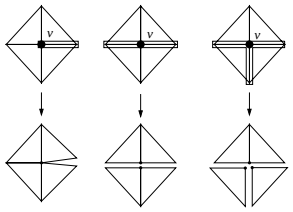


**Figure 6:** *A cut surface can partition the set of triangles containing vertex v into one, two or more parts. The number of components (triangles connected by faces not in the cut surface) equals the number of nodes corresponding to v after the cut.*

As shown in Figure 6, neighboring tetrahedra play an important role in determining how a cut is performed. From left to right we see how a cut-surface divides the tetrahedra containing node $v$ into one, two, and three face-connected components. After the cut, the node $v$ is correspondingly replaced by one, two and three copies. This suggests the following algorithm for performing cuts.

**Algorithm 3** Given a tetrahedral mesh and a cut surface $C$, then the following algorithm modifies the mesh to put triangles in $C$ on the exterior of the mesh.

1 for each node $N$ within the mesh
2 $T :=$ the set tetrahedra incident to $N$
3 $K :=$ the set of components $T$ is divided into by $C$
4 for each component $c$ in $K$
5 $N_c :=$ a copy of $N$
6 substitute $N$ with $N_c$ in all tetrahedra of $c$.

The actual implementation of this algorithm hinges on what data structure is used to represent the mesh. Variants of popular solid modeling data structures, such as radial edge, winged edge, etc.[14,21], have been tried but turned out to be unwieldy for reliable use: we found that maintaining, let alone proving, topological consistency of such structures is difficult.

Basing the data structure on *abstract simplicial complexes*, a well-known theory from algebraic topology, made the implementation of unglue very straightforward. The rest of this section is devoted to an informal overview of simplicial complexes as far as relevant, and a sketch of the implementation. Correctness proofs can be found in a forthcoming note[17].

In a simplicial complex, a mesh is constructed from *nodes* (or *vertices*). Features like edges, triangles, and tetrahedra are represented as sets of vertices. These sets are called *simplexes*, and the defining qualities of a simplex are the following.

- Any set of one vertex is a simplex.
- Any non-empty subset of a simplex is also a simplex. In other words, if a simplicial complex contains a set of four nodes (a 3-simplex or tetrahedron), its subsets of size three (2-simplexes or triangles), size two (1-simplexes or edges) are also in the complex.

A tetrahedral mesh is a simplicial complex that consists entirely of of 3-simplexes and their subsets. Furthermore, 2-simplex (a triangle) in such a complex can be in at most two 3-simplexes (tetrahedra, one on each side). A 2-simplex that is in only one 3-simplexes is an exterior face, i.e., part of the boundary. We call such a complex a 3-dimensional pseudo-manifold.

With these definitions we can prove that Algorithm 3 has the following properties.

- It is well defined on tetrahedral meshes.
- It maps a 3-dimensional pseudo-manifold (a tetrahedral mesh) into a new 3-dimensional pseudo-manifold.
- It puts most faces of the cut surface on the boundary. As shown in Figure 7, some faces cannot be put on the boundary.

There is one caveat. The algorithm as listed above may still change the mesh, even if there is no cut surface, i.e., if $C = \emptyset$. Figure 8 shows such a situation in 2D. To remedy this, we require our complexes to be *cut-regular*:

**Definition 4** A tetrahedral mesh is cut-regular if for every node the tetrahedra incident to $v$ form one face-connected component.

If this holds, then the set $K$ in Step 3 of Algorithm 3 has only one member (assuming $C = \emptyset$). The new mesh will be isomorphic to the old one.

In the implementation, we chose to represent triangles and tetrahedrons not only as sets, but also as instances of 'topological objects': each 3-simplex or 2-simplex in the complex

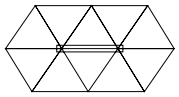

**Figure 7:** *Not all faces of a cut surface end up on the boundary. The both nodes of the boxed edge have only one connected component in their incident tetrahedra. An unglue with this cut surface does not change the mesh.*

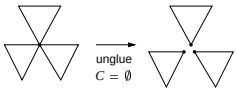


**Figure 8:** *If the tetrahedra incident to one node form multiple components, the unglue algorithm will split the node, even if is the cut-surface is empty.*

corresponds to a unique object. Triangle objects contain a reference to the tetrahedrons that they are part of. The mesh is a data structure which contains a set of nodes, and a lookup table that maps oriented simplexes to objects in memory. Instantiating objects for triangles and tetrahedrons allows the program to refer to topological structures with pointers that remain valid even after the node-substitution in Step 6 of Algorithm 3.

We used *oriented* simplicial complexes. In an oriented complex, each set is ordered, and the parity of the ordering determines the orientation of the edge, triangle, or a tetrahedron. In the oriented 3-dimensional pseudo-manifold, each triangle is part of precisely one tetrahedron. An oriented triangle is on the boundary if its *mate* (the corresponding triangle with opposite orientation) is not in the simplicial complex. Oriented pseudo-manifolds have an important advantage over an unoriented ones in the implementation: the unglue operation does not create any new triangles, so no objects need to be instantiated or destroyed. Moreover, an oriented triangle has a unique surface normal; this makes visualization slightly easier.

With this data structure, neighbors of a given tetrahedron can be found efficiently with Algorithm 5. This is sufficient for finding all tetrahedra incident to a node $v$ (Step 2 of Algorithm 3). We start with a tetrahedron that contains node $N$, and recursively find all its neighbors that contain $N$. If the simplicial complex is cut-regular, then this proces reaches all tetrahedra containing $N$.

**Algorithm 5** Given a tetrahedron object $t$ and an index $j$, then the following algorithm finds the $j$th neighbor tetrahedron of $t$.

1 $s :=$ the simplex corresponding with object $t$
2 $s_j :=$ the simplex obtained by removing the $j$th node of $s$
3 $s'_j :=$ the mate of $s_j$ (i.e., $s_j$ with its parity flipped)
4 $u :=$ the triangle object corresponding to $s'_j$
5 return the backlink stored in $u$
(the tetrahedron $u$ is part of)

Finally, we may speed up this process by caching the triangles that are contained in a tetrahedron, thus eliminating Steps 2 and 3 in Algorithm 5. If the triangle objects also store references to their mates, we can eliminate the lookup in Step 4, thus making the entire procedure a constant-time operation.

## 5. Prototype

A simulator has been written that implements the techniques we discussed. The design is multithreaded: one thread performs the CG iteration, while another thread deals with visualizing the results and tracking the scalpel. Figures 9 through 12 show the prototype in action.

## 6. Discussion

We have demonstrated how to create a linear static FE deformation simulation that does not require expensive global precomputations, and how to separate topological and geometric aspects when performing cuts in the meshes involved. This research opens a new road to interactive surgery simulations based on finite element analysis.

Theoretically speaking, an FE model is more realistic than a physics based model, such as the mass-spring model. However, it does not follow that this also holds in practice. We believe that an evaluation of FE simulations compared to mass-spring-systems is needed. Furthermore, it should be investigated whether viscosity, incompressibility, non-linear materials, and non-linear deformation would add significant realism to an FE simulation. An additional complication is that it is not clear whether these simulations can at all be implemented efficiently and without global precomputations.

A small test showed that the iterative solver can be fast enough for interactive use. However, a much more thorough theoretical and experimental analysis of the convergence behavior is needed to assess how material properties and size and shape of the mesh may be chosen to guarantee interactive speeds.

The cutting technique is novel, and its current implementation is a proof-of-concept. The cut surface coincides with mesh features, and this causes the cut surface to be jagged, which is probably not convincing enough for actual surgery simulations. We hope to remedy this by using vertex snapping: when a cut is executed, the vertices involved are moved onto the scalpel path.

Finally, interactions between cutting and deformation have not been researched at all: moving material while the scalpel remains steady should also cause a cut, and the tissue should exert a friction force on the scalpel.

**References**


1. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.
2. Daniel Bielser, Volker A. Maiwald, and Markus H. Gross. Interactive cuts through 3-dimensional soft tissue. In *Eurographics '99*, 1999.
3. F. Boux de Casson and C. Laugier. Modelling the dynamics of a human liver for a minimally invasive surgery simulator. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 1999.
4. Morten Bro-Nielsen. *Medical Image Registration and Surgery Simulation*. PhD thesis, Dept. Mathematical Modelling, Technical University of Denmark, 1997.
5. Morten Bro-Nielsen and Stephane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 3(15):57–66, 1996.
6. Morten Bro-Nielsen, David Helfrick, Bill Glass, Xiaolan Zeng, and Hugh Connacher. Vr simulation of abdominal trauma surgery. In *Medicine Meets Virtual Reality 6*, pages 117–123, San Diego, California, 1998. IOS Press.
7. Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. Efficient linear elastic models of soft tissues for real-time surgery simulation. Technical report, INRIA, October 1998.
8. Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. Real-time elastic deformations of soft tissues for surgery simulation. Technical report, INRIA, October 1998.
9. Steven A. Cover, Norberto F. Ezquerra, James F.O'Brien, Richard Rowe, Thomas Gadacz, and Ellen Palm. Interactively deformable models for surgery simulation. *IEEE Computer Graphics and Applications*, pages 68–75, November 1993.
10. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. *Solving problems on concurrent processors*. Prentice Hall Inc., 1988.
11. Sarah F.F. Gibson. 3d chainmail: a fast algorithm for deforming volumetric objects. In *1997 Symposium on Interactive 3D Graphics*, pages 149–154. ACM, 1997.
12. Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, Maryland, 1983.
13. Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic modeling for facial animation. In *SIGGRAPH'95*, 1995.
14. Martti Mäntylä. *An introduction to solid modeling*. Computer Science Press, 1988.
15. K. Mori, Y. Seki, J.-I. Hasegawa, J.-I. Toriwaki, H. Anno, and K. Katada. A method for shape deformation of organ and its application to virtualized endoscope system. In H.U.Lemke, M.W. Vannier, and K. Inamura, editors, *Computer Assisted Radiology and Surgery*, pages 189–194. Elsevier Science B.V., 1997.
16. Paul F. Neumann, Lewis L. Sadler, and Jon Gieser M.D. Virtual reality vitrectomy simulator. In *MICCAI98*, pages 910–917, 1998.
17. Han-Wen Nienhuys. A topological foundation for cutting in tetrahedral meshes. Unpublished.
18. Markus A. Schill, Sarah F. F. Gibson, H.-J. Bender, and R. Männer. Biomechanical simulation of the vitreous humor in the eye using an enhanced chainmail algorithm. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, pages 679–687, 1998.
19. Markus A. Schill, Clemens Wagner, Marc Hennen, Hans-Joachim Bender, and Reinhard Männer. Eyesi — a simulator for intra-ocular surgery. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 1999.
20. Gilbert Strang and George J. Fix. *An analysis of the Finite Element Method*. Prentice-Hall, 1973.
21. K. J. Weiler. *Topological Structures for Geometrical Modeling*. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, 1986.

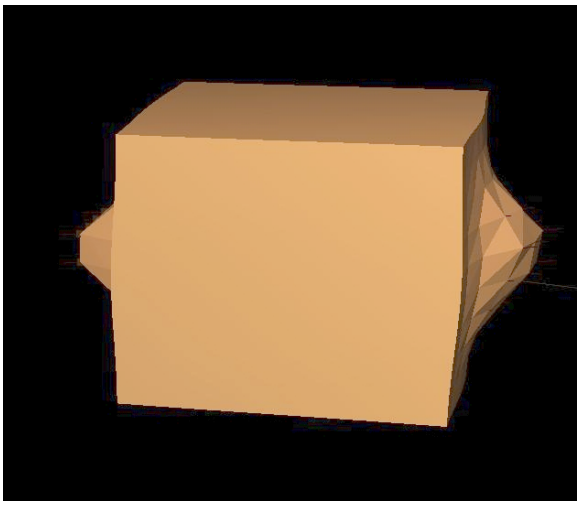

**Figure 9:** *Dilating force applied to the left and right parts of the boundary of a cube of 1000 nodes.*

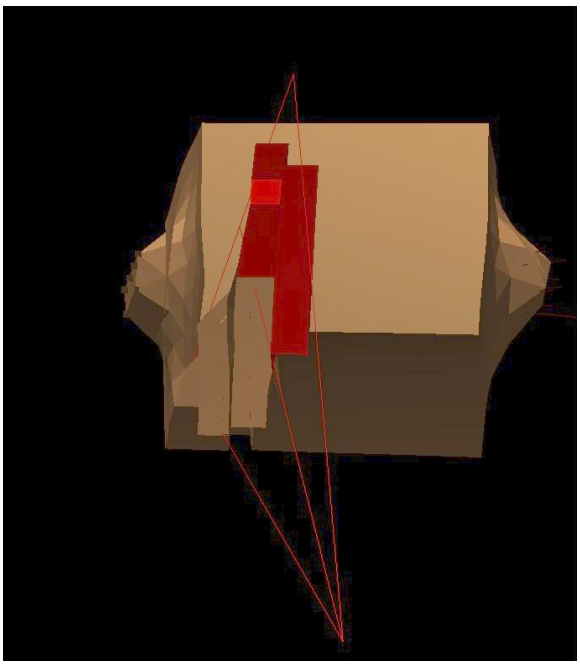

**Figure 10:** *A previously planned incision has already been performed. The cut surface for the next cut is shown in red. Because of the dilating force, the incision is folded open. All front faces have been culled.*

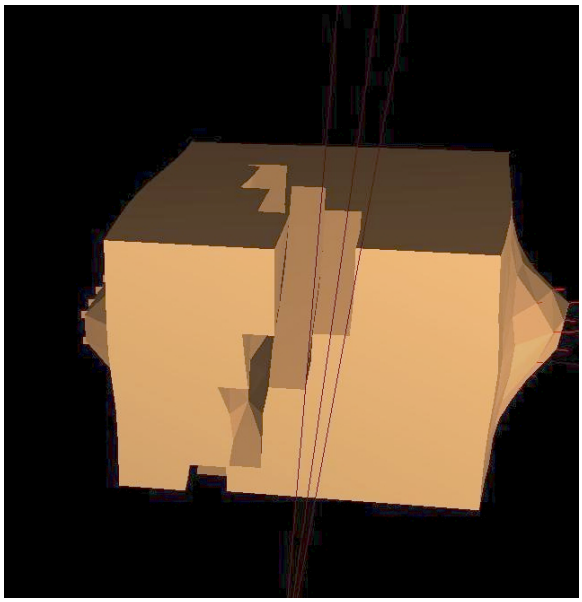

**Figure 11:** *The cut is finished. A view from the outside.*

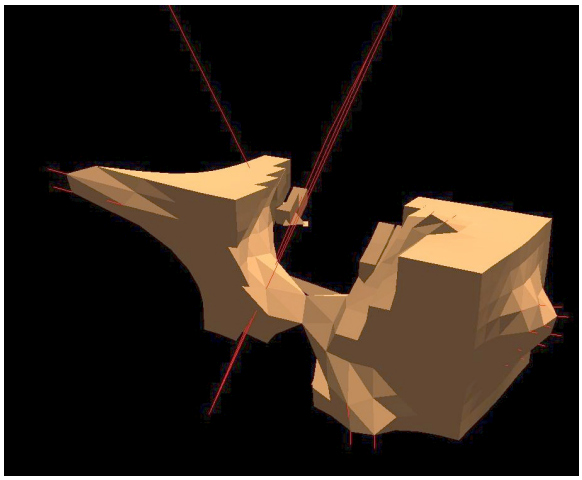

**Figure 12:** *The same block of material fully torn apart and butchered by multiple incisions. The line protuding from the model at the top left visualizes an instrument for applying force.*