Appeared in Proceedings of Pacific Graphics 2000, IEEE Computer Society Press, pp. 116-125

Interactive Simulation of Surgical Cuts

Daniel Bielser and Markus H. Gross

Computer Science Department
ETH Zürich, Switzerland
email: {bielser, grossm}@inf.ethz.ch

Abstract

We present a framework for the interactive simulation of surgical cuts such as being practiced in surgical treatment. Unlike most existing methods our framework is based on tetrahedral volume meshes providing more topological flexibility. In order to keep the representation consistent we apply adaptive subdivision schemes dynamically during the simulation. The detection of collisions between the surgical tool and the tissue is accomplished by using an axis aligned bounding box hierarchy which was adapted for deformable objects. For haptic rendering and feedback, we devised a mechanical scalpel model which accounts for the most important interaction forces between scalpel and tissue. The relaxation is computed using a localized, semi-implicit ODE solver. The achieved quality and performance of the presented framework is demonstrated using a human soft tissue model.

1 Introduction and Related Work

Over the past years surgery simulation has emerged as an fascinating field of research offering a rich repository of challenges for computer graphics. We observe that the worldwide activities in this field spawned various systems for surgical training or planning, partly already being in use as instrumental tools [9]. As a matter of fact, realistic simulation of tissue cutting and deformation belongs to the most important components of a surgical simulator. Therefore research focus is mainly directed onto the design of advanced computational models for the topological and geometric representation, deformation, and rendering of soft tissue structures. In most advanced surgery simulation environments, the ultimate goal is to manipulate high resolution 3D tissue models in real time and at a premium quality visual feedback.

In order to approach this goal we are concerned with the following important issues:

- *Geometric and topological representation of soft tissue:* The mathematical concepts to efficiently represent the underlying 3D geometry form an essential component in any surgery simulation environment. It has to tolerate topological changes of the tissue during manipulation while keeping the computational burden sufficiently low.
- *Collision detection:* The detection and handling of contacts and collisions between the surgical tools and the tissue as well as the inter-tissue collisions are critical to provide realistic visual feedback. While collision detection algorithms have been extensively investigated in robotics and CAD, relatively few concepts exist for surgery simulation.
- *Haptic modeling and rendering*: Tissue and surgical tools form a mechanical system and various interaction forces arise during simulation. For correct mechanical behavior we have to devise a mechanical model for both external and internal forces. Especially the internal tissue forces caused by large deformations quickly lead to nonlinear mechanics. Hence, powerful approximations have to be found.
- *Fast computational schemes:* The fast computation of the underlying deformation and force model is essential. In most applications large deformations only occur in the direct vicinity of the surgical tool. As a consequence, the governing equations can be adapted locally in terms of approximation quality and resolution.

In this paper we will present a framework for building the core components of a surgical simulator which addresses the points raised above.

Due to the importance of these issues considerable related work has been done in the Graphics and Vision communities. The computationally most accurate methods for the modeling of elastic soft tissue mostly use Finite Element procedures to solve the governing equations. Of the rich literature on this subject, we confine our survey to [13] and [14] who convey a FEM-based model for facial surgery simulation and extend it for animating human emotions [15]. Recently, Boundary Element Methods (BEM), like [7] or [12] were proposed, condensing the solution into the domain boundary. Condensation, however, has the disadvantage that the whole stiffness-matrix has to be recomputed when cutting some surface elements. This can be very expensive since BEM matrices are usually not sparse.

In order to make 3D soft tissue modeling real-time, different strategies have been proposed. [6], for instance, develops surface-based snakes to represent human organs. Furthermore, [17] developed a soft tissue model for facial animation and [18] used mass-spring and particle systems for the representation of human muscles.

Another interesting approach is the 3D ChainMail, as introduced by [8]. Rather than computing physical deformations on the fly, the method uses a two-pass hybrid scheme, where in a first pass, mere geometric deformation fields are applied. In the second pass, the tissue is post-relaxed by some iterative solvers. The topology of the discretization is restricted to tensor product grids. Others, like [16] employ surface based mass-spring systems for their real-time simulators. Similar Euler type methods on regular grids are reported in [26]. Pure geometric and topological manipulations based on marching cubes techniques can be found in the algorithms of [22] and [21]. In most approaches, force feedback devices are utilized to implement the interface to the user.

Recently, first concepts were presented to dynamically update volume meshes to keep track of topological changes in the representation. For instance, [19] presents a dynamic splitting procedure of tetrahedra for the modeling of brittle fracture. [3] uses a strategy to determine and duplicate the vertices of the polyhedron that are close to the collision points along the line of cut. A dynamic subdivision algorithm using an operator framework is introduced by some of the authors of this paper in [5]. Eventually, [28] used the concept of Intelligent Scissors for volume cutting along with interactive definitions of a cut contour onto the object's surface.

Related work on collision detection can be found in [27] where an efficient collision detection method for cloth simulation is devised. Lately, [2] optimizes implicit numerical solution strategies for efficient use in cloth modeling.

We will first describe the conceptual framework of our approach. Section 3 addresses an improved subdivision scheme used for tissue representation. Our methods for collision detection are explained in section 4. In section 5 we will elaborate on the algorithms used for relaxation and multiprocessor implementation. Finally, we introduce our concept for force and haptic modeling and present some examples using a human soft tissue model.

2 Conceptual Overview

Our framework is based on an unstructured tetrahedral mesh which can be generated in a preprocess. As an option, the mesh can be aligned to some 3D texture data, such as provided by CT, MRI or other medical imaging devices. Fig. 1 depicts the major components of the simulation framework. The *collision detection* procedures detect any interaction between the surgical tools, such as scalpel or hooks, and the tissue structures. At this point, tissue-tool intersections have to be computed. Topology and *geometry update* procedures carry out the modifications of the tetrahedral mesh enabling one to model incisions. To compute the resulting tissue deformations, a multi-threaded *relaxation* procedure traverses the simplices of the mesh and calculates their positional movements according to the laws of deformation. When it comes to haptics, a separate haptic server manages a physical model of the scalpel and determines the resulting force output. Due to the elaborate haptic perception of the human hand, update rates of at least 1 kHz have to be maintained for haptic rendering. Our framework is designed to enable these tasks to work in parallel. In our current implementation, geometry update and collision detection form a single process, all other computational tasks are programmed as threads. Hence, the operating system distributes them automatically to the available number of processors thus balancing the load. We achieve real-time response for the visual and haptic rendering by actively controlling the processes' priority queue. Hence, the frame-rate is decoupled from other computational tasks and rendering speed depends on the performance of the allocated processor and on the graphics subsystem. A similar concept holds for the force feedback, which connects via TPC/IP in order to make it running on a separate haptic server.

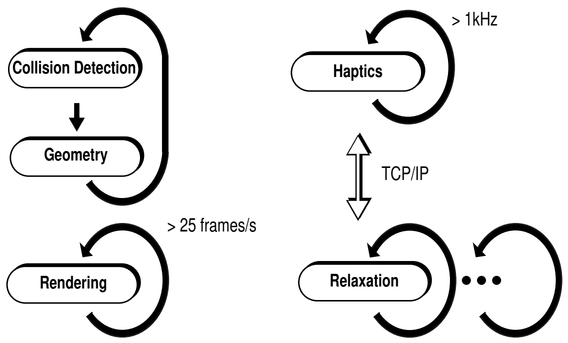


Figure 1: Parallel tasks of the simulation and their real-time requirements

Since different algorithms are working on the same data structure, it is necessary to avoid concurrent interactions. This is solved by defining exclusively atomic operations on the shared data segments.

3 Crack-free Tetrahedral Subdivision

To our knowledge [5] were the first to present a representation to cut arbitrary tetrahedral meshes. The generalized subdivision scheme presented in this work divides a tetrahedron into a fixed number of 17 smaller simplices that are independent of the topology of the cut. While this scheme works well in practice, it may lead to a rapidly increasing number of tetrahedra for large cuts. In addition, cracks may arise in the physical representation of the tissue.

One possibility for solving the problem of inconsistent internal representations would be to remesh the adjacent tetrahedra, as is done in some other remeshing techniques [4]. Subsequently, we will present an improved subdivision method that solves the above problem without having to update adjacent tetrahedra. Our method confines the newly created simplices to the actual tetrahedron by introducing cut-specific subdivision patterns.

The basic idea of our procedure is to subdivide only those edges and faces that are part of the cut surface. We achieve a consistent subdivision by restricting the permitted subdivisions of a tetrahedral face to the following three cases.

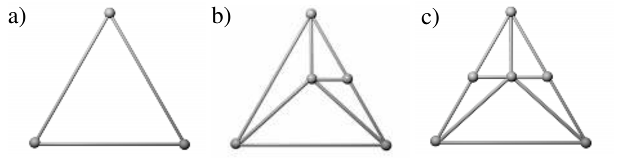


Figure 2: The three permitted face subdivisions

The first case of Fig. 2 represents the trivial situation of an undivided face when no intersection with the cut surface is detected. The second image depicts an intersection of a face affecting one of its edges. This subdivision scheme will be applied whenever a face is partially cut or notched in. The last case represents a face having two edges affected by the cut. It will be used for completely split faces.

As in [5] the cut algorithm is simplified by distinguishing only five topologically different cases. Two of these cases represent a complete split of a tetrahedron, while the other three stand for partially cut tetrahedra. By restricting our subdivision patterns of these five cases to the three face subdivision patterns from above, we obtain the final pattern as presented in Fig. 3.

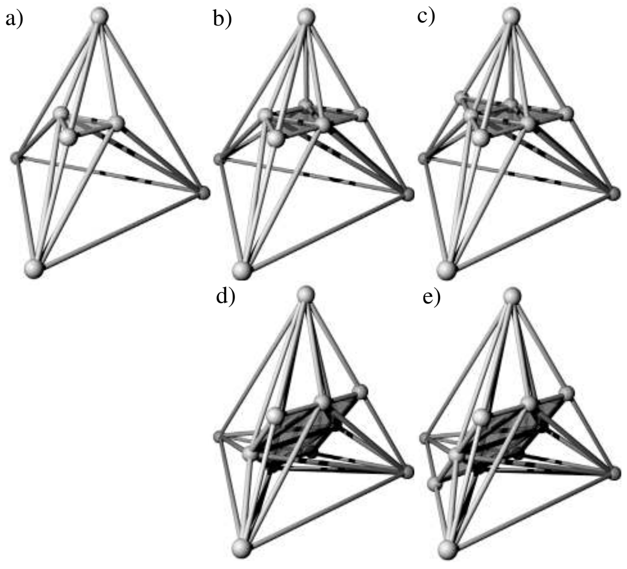


Figure 3: Subdivision patterns for the five cases

The localization of the subdivision is clearly visible. All faces and edges that are not intersected by the cut surface are kept intact - as desired. Note that the additional node which is inserted into every completely cut face (see Fig. 2c) is not necessarily needed. However we found that this scheme provides two advantages: Firstly, it leads to more symmetry in the subdivision patterns, and secondly, it creates additional degrees of freedom for the cut surface improving the results of the subsequent numerical relaxation.

Elastic tissue behavior forces the cut surface to open during the relaxation. In order to provide the necessary degrees of freedom all newly inserted mass nodes, edges and faces of the cut surface that do not belong to the boundary are inserted twice. We extended the look-up-table based approach described in [5] by adding the subdivision rules from above.

As described, the geometrical representation results from replacing new vertices (mass nodes) according to the previously calculated intersections (see also chapter 4). Utilizing the described setup we observed potential self-intersections of the newly inserted tetrahedra which deserves further discussion. To illustrate these possible intersections we show in Fig. 4 two slightly different tetrahedralizations of case c) of Fig. 3.

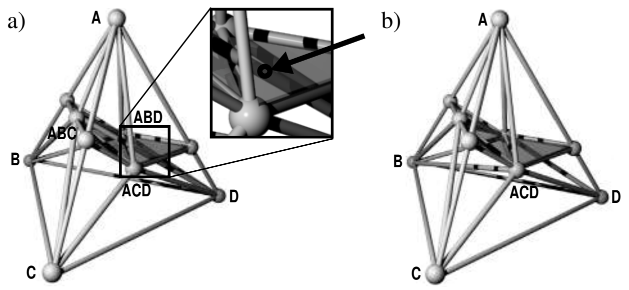


Figure 4: Potential self-intersection of newly inserted tetrahedra

In Fig. 4a the edge which connect the nodes D and ABC intersects the triangle (A, ABD, ACD). As a consequence all tetrahedra containing this edge penetrate the two tetrahedra containing this face. Fig. 4b depicts the same case, however, we inserted edge (B, ACD) instead of (D, ABC) - and thus solved the problem. To this end, for the patterns of Fig. 3b and Fig. 3c the edges (B, ACD), (D, ABC), and (C, ABD) have to be tested against the corresponding faces. This requires two additional intersection tests per tetrahedron in the worst case.

4 Collision Detection

In order to achieve realistic and physically correct visual feedback, collisions between the surgical tool and the tetrahedral mesh have to be detected. The detection of these collisions basically consists of two tasks. The first one is to find the surface (boundary) elements interacting with the surgical tool. The second one deals with finding all tetrahedra inside the tissue that are cut or notched-in by the scalpel. We call the first problem *surface collision detection* and the second one *volume collision detection*.

4.1 Surface Collision Detection

Our earlier work in [5] showed that the calculation of the first interaction of the tool with the object's surface may become a substantial computational burden. As an extension of this approach, we constructed a bounding volume hierarchy over the surface triangles. As opposed to rigid body motion, where the bounding volumes are fixed relative to the objects, the shapes of our bounding volumes follow the deformation of the object. We observe that recently proposed collision detection methods tend to apply more and more sophisticated bounding volumes [11] [1]. Since most of these algorithms are designed for rigid objects the bounding volumes can be calculated in a preprocess. As this is not possible for deformable objects, we decided to use most simple bounding volumes to keep the update overhead as small as possible. We eventually found that *axis aligned bounding boxes* serve best for building a hierarchy over the surface elements of a deformable object.

In case of a deformation the bounding box tree has to be updated. This is done locally by bottom up traversal of the tree starting from each affected surface element. For further decreasing the computational cost for these updates, we added a small tolerance to the size of each bounding box of the surface triangles. This gives the algorithm a certain “grace period” that allows it to skip updating for small positional changes of a child element. In particular a tolerance-box is constructed around every node of a surface triangle with a given size s_{tol}. That is, all bounding boxes around the surface triangles are scaled correspondingly (Fig. 5a). After each positional update of a mesh node, the node is compared to the tolerance box boundaries. Note that this test can be computed very efficiently due to the axis-alignment of the bounding boxes. If the node leaves the tolerance box, the bounding volumes of the adjacent surface triangle have to be updated.

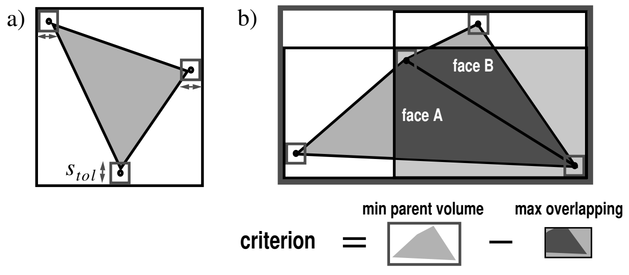


Figure 5: Top view: a) tolerances added to each node within the bounding box, b) determination of the best suited box

Special effort has been put into the construction of the best matching bounding box tree. After computing the bounding boxes on the leaf level of the tree, we use a greedy bottom-up traversal to group pairs of bounding boxes and assign parent boxes to them. The algorithm traverses the direct neighborhood of the surface triangle in breadth first order. This avoids combinations of faces not belonging to the same surface component. We combined two criteria in order to find the best pairs. The first one is the minimal volume of the parent box and the second one accounts for the maximal overlapping volume. We employ the difference of these two volume measures as illustrated in Fig. 5b. Thus, having these two values equal would provide an optimal pair. At run-time new bounding boxes must be computed for groups of new faces. This is optimized by generating predefined box structures.

In order to compute the intersection of a surgical tool with the tissue surface, we perform the bounding box tests hierarchically by top-down traversal of the tree.

4.2 Volume Collision Detection

After detection of the entry points of the surgical tool into the tissue, the tetrahedra affected by the scalpel have to be found for each time step of the simulation. When designing the collision detection algorithms, our goal was to exploit spatial coherency and to confine the search to a local traversal of adjacent tetrahedra. For this reason we have to store the list of tetrahedra currently penetrated by the scalpel at each time step. Following the notation of [5] the tetrahedra at time t_n are named *active tetrahedra* whereas the tetrahedra penetrated at time t_{n-1} are the *previous active tetrahedra*.

Assuming the scalpel being represented by a thin line whose length corresponds to the scalpel's blade, the following situations may occur. The tip of the scalpel $\mathbf{P}_s$ can be either inside or outside the tissue. In addition, one or more segments of the scalpel blade may penetrate the tissue, such as is shown in Fig. 6.

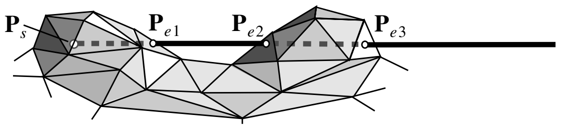
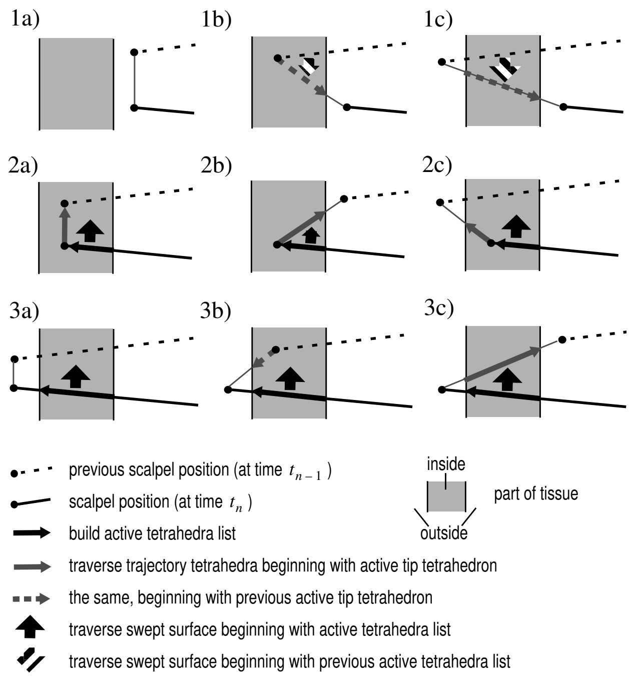


Figure 6: Example of tissue penetration: Scalpel tip $\mathbf{P}_s$ inside tissue, and one segment of penetrated tissue ($\mathbf{P}_{e2}$, $\mathbf{P}_{e3}$)

Analyzing the topology of these penetrations for two consecutive scalpel positions, we obtain 9 different situations depicted in Fig. 7, which have to be handled by the collision detection algorithm.

The rows in Fig. 7 show different possible positions of the scalpel at t_n whereas the scalpel positions at t_{n-1} are altered in every column. To find all possible intersections with the tetrahedral mesh the trajectory of the scalpel tip has to be checked for face intersections. Furthermore, we have to intersect all tetrahedral edges lying between the two consecutive scalpel positions with the swept surface spanned by the scalpel blade and its previous position respectively. Note that this is a significant improvement to [5], which, as a limitation, could only detect directly adjacent tetrahedra.



1a)
1b)
1c)
2a)
2b)
2c)
3a)
3b)
3c)
previous scalpel position (at time t_{n-1})
scalpel position (at time t_n)
build active tetrahedra list
traverse trajectory tetrahedra beginning with active tip tetrahedron
the same, beginning with previous active tip tetrahedron
traverse swept surface beginning with active tetrahedra list
traverse swept surface beginning with previous active tetrahedra list
inside
part of tissue
outside

Figure 7: Cases of different volume collision topologies

The following algorithm manages all the depicted situations. Starting with an entry face, it initially searches all active tetrahedra. The algorithm continues by traversing the trajectory of the scalpel tip, and computes all face intersections of tetrahedra lying between the *active tip tetrahedron* and the *previous active tip tetrahedron.* The pseudo code fragment below describes the recursive algorithm. **findTrajectoryTetra()** is called once with the *active tip tetrahedron* as the argument. If necessary, (see Fig. 7, 1b) and 3b) it can be executed once again taking the *previous active tip tetrahedron* as an input.

```
findTrajectoryTetra(previous_tetra, tetra)
if tetra has a valid entry
    for every face ∈ tetra
        if face ∉ previous_tetra
            findTrajectoryTetra(tetra,
                    face.neighbor_tetra)
```

Note that in the two cases 1c) and 3c) of Fig. 7 there exists no valid *active tip tetrahedron*. Therefore, we use the face intersection algorithm from above to determine a first intersection between the trajectory of the scalpel tip and the object's surface. The tetrahedron belonging to this entry face will then serve as a starting point for the **findTrajectoryTetra** routine.

In a second step, all the tetrahedra being intersected by the swept surface have to be traversed and their edge intersections must be calculated. We approximate this swept surface using two triangles and, again, propose a recursive procedure to compute this. In order to handle all cases of Fig. 7 we have to call this procedure both for the *active tetrahedra list*, and for the *previous active tetrahedra list*. It recursively traverses adjacent tetrahedra as long as intersections are detected. To avoid cycles, we tag each visited and intersected tetrahedron and return on the first tetrahedron providing no intersection. The following pseudo code gives an overview of the algorithm.

```
findSwapFaceTetra()
for all tetra
    tetra.state = not visited
for every tetra ∈ pevious active tetra list
    markSwapFace(tetra)
for every tetra ∈ active-tetra list
    markSwapFace(tetra)

markSwapFace(tetra)
if tetra.state ≠ visited
    tetra.state = visited
    for every edge ∈ tetra
        checkFaceIntersection(edge)
    for every face ∈ tetra
        if face contains intersected edge
            tetra.state = intersected
            markSwapFace(face.neighbor_tetra)
```

5 Relaxation

For the computation of the physics of soft tissue, we use a conventional mass-spring system, assigning springs to the edges and masses to the nodes of the tetrahedralization. These relatively simple ordinary differential equations (ODE) have been very well studied in the numerical analysis literature [10]. It is, however, well known, that mass spring systems may lead to stiff ODEs [2] imposing computational constraints onto the solution strategies. In order to better explain our semi-implicit solver, we start with the underlying ODE of type

$$\mathbf{M} \cdot \frac{d^2\mathbf{x}(t)}{dt^2} + \mathbf{D} \cdot \frac{d\mathbf{x}(t)}{dt} + \mathbf{K} \cdot \mathbf{x}(t) = \mathbf{F}_{ext} \qquad (1)$$

The steady state solution of the above system can be computed either by assembling the stiffness matrix $\mathbf{K}$ and calculating the solution vector of the system, or by traversing the different mass nodes and iteratively processing each individual equation. In a simulation environment, one typically wants to achieve a relaxation which is as smooth as possible. Consequently such procedures should have a dense output while still reflecting the dynamics of the system to some extent. Hence, iterative methods are useful even in cases of implicit integration.

5.1 A Semi-Implicit Integration Method

It has been shown that implicit integration methods are superior to explicit ones in cases of stiff equation systems [10][2]. Although a sophisticated implicit solver enables one to use very large time-steps, we have to focus on a simple method to guarantee high update rates for the animation. Interestingly, as we will show subsequently, equation (1) allows us to find a compromise between the stability of an implicit method and the simplicity and density of an explicit iteration.

To simplify notation, we will only consider the governing equation for a single mass m_k. In order to derive the method, we divide the second order differential equation for m_k into a system of two first order differential equations by introducing the velocity function $\mathbf{v}_k(t)$.

$$\left|\begin{aligned} \frac{d\mathbf{x}_k(t)}{dt} &= \mathbf{v}_k(t) \\ \frac{d\mathbf{v}_k(t)}{dt} &= \frac{\mathbf{f}_{ext} - \mathbf{D}\cdot\mathbf{v}_k(t) - F_{spring}(\mathbf{x}(t),\mathbf{K})}{m_k} \end{aligned}\right| \tag{2}$$

The resulting spring force is expressed by an operator $\mathbf{F}_{spring}(\mathbf{x}(t),\mathbf{K})$ which incorporates the spring forces from the adjacent mass nodes. It depends on the position of all affected mass nodes $\mathbf{x}(t)$ and on the stiffness matrix $\mathbf{K}$. It yields as the sum of the spring forces invoked by masses m_j being connected to m_k. The position of a mass node m_i is denoted by $\mathbf{x}_i$.

$$\begin{aligned} &\mathbf{F}_{spring}(\mathbf{x}(t),\mathbf{K}) = \\ &\sum_{j\neq k} k_{jk}\cdot\frac{(l_{jk} - |\mathbf{x}_j(t) - \mathbf{x}_k(t)|)}{|\mathbf{x}_j(t) - \mathbf{x}_k(t)|}\cdot(\mathbf{x}_j(t) - \mathbf{x}_k(t)) \end{aligned} \tag{3}$$

The spring forces are computed by multiplying the elongation from the rest length l_{jk} of the spring with its spring-stiffness k_{jk}.

Let's assume that at time t_0 the equation has the initial values $\mathbf{x}_0 = \mathbf{x}(t_0)$ and $\mathbf{v}_0 = \mathbf{v}(t_0)$. We introduce the notation $\mathbf{x}_{k,n}$ and $\mathbf{v}_{k,n}$ for the position and velocity of the mass node m_k at time t_n respectively. Using an implicit method we describe the calculation of $\mathbf{x}_{k,n+1}$ and $\mathbf{v}_{k,n+1}$ at the time $t_{n+1} = t_n + \Delta t$ depending on the given states $\mathbf{x}_{k,n}$,and $\mathbf{v}_{k,n}$, and the new, unknown states $\mathbf{x}_{k,n+1}$, and $\mathbf{v}_{k,n+1}$. Note that, in general, one has to solve a nonlinear system of equations for the variables $\mathbf{x}_{k,n+1}$ and $\mathbf{v}_{k,n+1}$ at every time-step.

$$\left|\begin{aligned} \mathbf{x}_{k,n+1} &= \mathbf{x}_{k,n} + \Delta t\cdot\mathbf{v}_{k,n+1} \\ \mathbf{v}_{k,n+1} &= \mathbf{v}_n + \Delta t\cdot\frac{\mathbf{f}_{ext} - \mathbf{D}\cdot\mathbf{v}_{k,n+1} - \mathbf{F}_{spring}(\mathbf{x}_{k,n+1},\mathbf{K})}{m_k} \end{aligned}\right| \tag{4}$$

with

$$\begin{aligned} &\mathbf{F}_{spring}(\mathbf{x}_{k,n+1},\mathbf{K}) = \\ &\sum_{j\neq k} k_{jk}\cdot\frac{(l_{jk} - |\mathbf{x}_{j,n+1} - \mathbf{x}_{k,n+1}|)}{|\mathbf{x}_{j,n+1} - \mathbf{x}_{k,n+1}|}\cdot(\mathbf{x}_{j,n+1} - \mathbf{x}_{k,n+1}) \end{aligned} \tag{5}$$

To bypass the computation of this system, we partially substitute $\mathbf{x}_{k,n+1}$ in equation (5) by insertion of the upper equation of (4) and solve the lower equation for the variable $\mathbf{v}_{k,n+1}$. In this case, the solution of the initially implicit approach can be expressed explicitly. With some algebra the equations (4) and (5) are transformed into the following system:

$$\left|\begin{aligned} \mathbf{x}_{k,n+1} &= \mathbf{x}_{k,n} + \Delta t\cdot\mathbf{v}_{k,n+1} \\ \mathbf{v}_{k,n+1} &= \frac{m_k\cdot\mathbf{v}_{k,n} + \Delta t\cdot\mathbf{f}_{ext} - \left(\sum_{j\neq k} E\cdot(\mathbf{x}_{j,n+1} - \mathbf{x}_{k,n})\right)}{m_k + \Delta t\cdot\left(\mathbf{D} + \Delta t\cdot\sum_{j\neq k} E\right)} \end{aligned}\right| \tag{6}$$

$$E = k_{jk}\cdot\frac{(l_{jk} - |\mathbf{x}_{j,n+1} - \mathbf{x}_{k,n+1}|)}{|\mathbf{x}_{j,n+1} - \mathbf{x}_{k,n+1}|}$$

The first equation only depends on the variable $\mathbf{v}_{k,n+1}$ and can therefore be computed after evaluating the second equation. This equation, unfortunately, still depends on $\mathbf{x}_{k,n+1}$. Since the magnitude operator is a nonlinear function, it turns out to be too expensive to extract $\mathbf{x}_{k,n+1}$ at every update step. To this end, we estimate $\mathbf{x}_{k,n+1}$ using an explicit Euler step

$$\mathbf{x}_{k,n+1} = \mathbf{x}_{k,n} + \Delta t\cdot\mathbf{v}_{k,n} \tag{7}$$

This leads to a semi-implicit method that solves for the direction of the mass' motion *implicitly* and for its magnitude *explicitly.*

Sometimes iterative solutions are preferable to achieve a smoother relaxation. Note that in those cases only the values from $\mathbf{x}_{0,n+1}$ to $\mathbf{x}_{k-1,n+1}$ are available for the computation of the new position $\mathbf{x}_{k,n+1}$ of a mass m_k while $\mathbf{x}_{k+1,n+1}$ to $\mathbf{x}_{max,n+1}$ have to be estimated by use of equation (7), as well.

5.2 A Parallel Node Traversal Algorithm

From the analysis of various tissue manipulation procedures using different surgical instruments we learned that the major part of the tissue deformation occurs in the direct vicinity of the instrument. In other words, mass nodes further away from this region take a smaller influence on the nodal displacement. This observation suggests an adaptive step size control. For an efficient implementation of this feature, we quantize the possible time steps into values Δt_i, allocate lists to them and store in each list the mass nodes with this time step.

During the relaxation the individual lists are handed over to a scheduler to process the nodes. Lists assigned to smaller time steps are passed over more often than those with larger ones. The scheduler is multithreaded enabling us to handle the mass nodes in parallel.

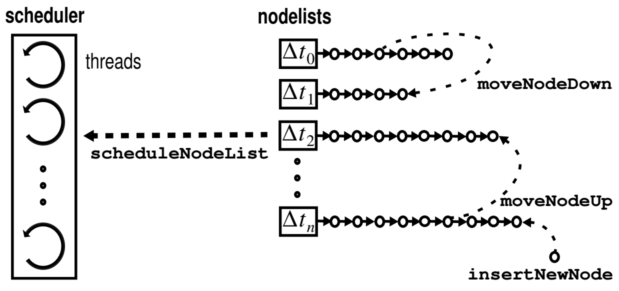


Figure 8: Scheduling data structure and algorithm

A dedicated thread controls reassignments of individual nodes to the different nodelists as well as the distribution of the nodelists to the different threads. For dynamic adaptation of the node's time-step and in order to guarantee a meaningful distribution over the different nodelists, the commands `insertNewNode`, `moveNodeUp`, and `moveNodeDown` were implemented. We suggest to use the averaged acceleration of the node over a fixed number of previous time steps as an assignment criterion. Thus, nodes with a higher acceleration will be updated more often than slower ones.

For fair assignment of the nodelists to available threads, we stamp each list i with the time of the last relaxation t_i. After processing the list i, t_i will be incremented by Δt_i. The scheduler eventually picks the list leading to the smallest increment to the actual time t_{act_n} according to the following relation:

$$t_{act_{n+1}} = \min_{i=0\ldots n} \quad t_i + \Delta t_i \tag{8}$$

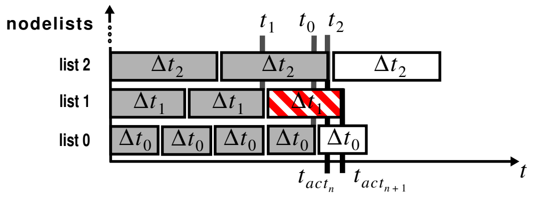


Figure 9: Scheduling of node-lists with different time-steps

Fig. 9 illustrates the scheduling algorithm. The grey boxes stand for the passed relaxation time of each nodelist, whereas the white boxes indicate an estimation of the time required for the next relaxation step. In this example, list 1 will be scheduled next, since it will lead to the smallest increment of the simulation time.

Note that storing the time for each node is an essential prerequisite for moving individual nodes up and down the lists. For computing the next positional update of the same node in a different list j the difference between the actual time t_j and the node's time-stamp serves as the new time-step for the integration.

6 A Haptic Scalpel Model

The haptic interaction with surfaces of arbitrary objects is an essential ingredient of a surgical simulator and has already been treated in several publications [23]. In our work the GHOST API of the PHANToM force-feedback device [24] is applied to model the rigid bone structures. However, we are not aware of any model for describing the haptic behavior of a scalpel cutting through 3D deformable tissue.

In this section, we will present a novel haptics model for a scalpel and its interaction with 3D soft tissue. Due to the high update rates being necessary for realistic haptic rendering our goal was to start with a force model that is independent of the actual tissue representation. In a second step, we account for the interaction with the mesh by communicating entry positions, material parameters, and external forces to and from the haptics server.

6.1 Force Decomposition

Since a scalpel usually cuts only along the direction of its blade, it is admissible to restrict the scalpel's motion to the plane spanned by the blade. To model this behavior we project the external force vector $\mathbf{F}_{ext}$ into this plane. As illustrated in Fig. 10 this is accomplished by decomposition into a force $\mathbf{F}_{\|}$ lying in the plane and a force $\mathbf{F}_{\perp}$ perpendicular to it. $\mathbf{F}_{\|}$ is responsible for the incision whereas $\mathbf{F}_{\perp}$ has to be constrained to avoid movements perpendicular to the cut plane. To simplify the calculation of the forces we further decompose $\mathbf{F}_{\|}$ into a component $\mathbf{F}_{\|a}$ aligned to the blade and into a component $\mathbf{F}_{\|n}$ perpendicular to it.

$$\mathbf{F}_{ext} = \mathbf{F}_{\|} + \mathbf{F}_{\perp} = \mathbf{F}_{\|a} + \mathbf{F}_{\|n} + \mathbf{F}_{\perp} \tag{9}$$

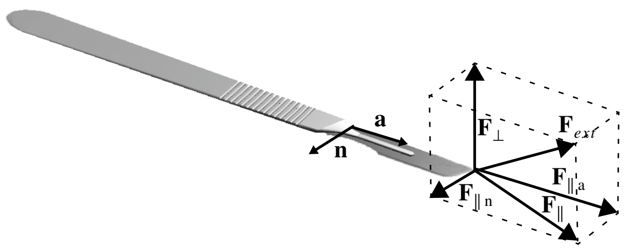


Figure 10: Decomposition of scalpel forces

All further force computations, such as the ones that describe the scalpel's friction during a cut, are carried out separately for $\mathbf{F}_{\|a}$ and $\mathbf{F}_{\|n}$ and added up again to obtain the resulting forces.

To compute an external force $\mathbf{F}_{ext}$ from the positional data of the force-feedback device, we think of the scalpel tip as being connected via a spring with a so-called *proxy* storing the tip's previous position. This turns out to be a meaningful approach which was already used in several related works [23].

6.2 Modeling the Tissue Forces

When the blade of a scalpel enters a piece of tissue, the following two important interactions occur. Firstly, the scalpel separates the tissue and penetrates into the interior. Secondly, the forces imposed by the scalpel deform the tissue. The relation of these two phenomena is established by the friction of the scalpel blade.

The friction occurring during a cut can be separated into two different forces:

- As long as the force applied to the scalpel is smaller than a given tissue-dependent threshold F_{enter}, the scalpel will not open the tissue. We call the force responsible for this effect the *static cut friction*.
- If the magnitude of the force $\mathbf{F}_{\|}$ exceeds F_{enter} the scalpel will start cutting. Note that the friction imposed in this case, the *dynamic cut friction*, is usually smaller than the *static cut friction*.

We observe two physical effects describing the cut friction. The force which cuts open the molecular structure of the tissue is called F_{open}. It is constant in time, however differs in the coefficient μ for the static and dynamic case. Due to the inertia of the displaced tissue the force $F_{displace}$ causing its displacement around the blade to be dependent on the blade's velocity v. We model this process in our setting in terms of a of viscosity force.

In addition, there exists a surface friction $F_{friction}$ arising from the friction of the scalpel blade on the tissue. This friction is dependent on the magnitude of the force $\mathbf{F}_{\perp}$ perpendicular to the cut surface.

All in all, the entire phenomenon of cut friction is formulated as the sum of the two forces projected onto the unity vectors of $\mathbf{a}$ and $\mathbf{n}$ pointing into the opposite direction than $\mathbf{F}_{\|\mathrm{a}}$ and $\mathbf{F}_{\|\mathrm{n}}$ (see Fig. 11).

$$\mathbf{F}_{cut} = \mathbf{F}_{cut\,\mathrm{a}} + \mathbf{F}_{cut\,\mathrm{n}}\,. \tag{10}$$

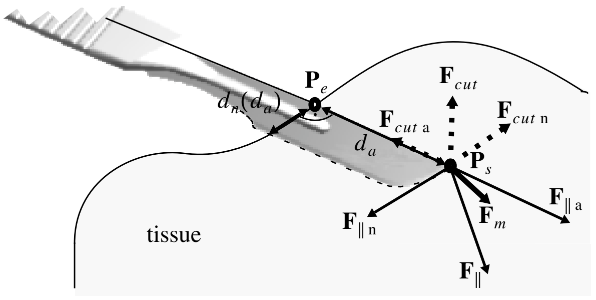

$\mathbf{P}_e$
$d_n(d_a)$
$\mathbf{F}_{cut}$
$\mathbf{F}_{cut\,\mathrm{a}}$
$\mathbf{F}_{cut\,\mathrm{n}}$
d_a
$\mathbf{P}_s$
$\mathbf{F}_{\|\mathrm{a}}$
$\mathbf{F}_{\|\mathrm{n}}$
$\mathbf{F}_m$
$\mathbf{F}_{\|}$
tissue

Figure 11: Calculating the resulting force

Since $\mathbf{F}_{cut\,\mathrm{a}}$ and $\mathbf{F}_{cut\,\mathrm{n}}$ are computed using similar relations, we write the corresponding equation using a generic placeholder x.

$$\mathbf{F}_{cut\,\mathrm{x}} = (F_{open\,\mathrm{x}} + F_{displace\,\mathrm{x}} + F_{friction\,\mathrm{x}}) \cdot \mathbf{x} \tag{11}$$

with $\mathbf{x} = \mathbf{n}$ or $\mathbf{x} = \mathbf{a}$

6.3 Static Cut-Friction

For the following considerations we neglect the tissue displacement. As a result, the resulting static cut friction force $\mathbf{F}_{cut_{static}}$ will compensate the force $\mathbf{F}_{\|}$

$$\mathbf{F}_{cut_{static}} = -\mathbf{F}_{\|} \quad \text{until} \quad F_{enter} = \left|\mathbf{F}_{cut_{static}}\right| \tag{12}$$

In the static case the velocity of the scalpel will be zero, the displacement force $F_{displace}$ vanishes, and the static cut friction is modeled with

$$F_{open\,\mathrm{x}} = \mu_{static} \cdot \eta \cdot c_{edge} \cdot d_{\mathrm{x}} \tag{13}$$

$$F_{friction\,\mathrm{x}} = \alpha_{static} \cdot |\mathbf{F}_{\perp}| \tag{14}$$

μ_{static} : static tissue opening coefficient
η : stiffness of the material
c_{edge} : sharpness of the blade's edge.
(small $c_{edge} \Rightarrow$ sharp edge)
α_{static} : static friction coefficient
d_{a} : penetration depth of the scalpel blade
d_{n} : breadth of the scalpel blade

The values that change during the simulation are in particular d_{x} and possibly the material stiffness η. The penetration depth of the scalpel d_{a} is calculated as the distance between the scalpel tip $\mathbf{P}_s$ and the entry point of the scalpel $\mathbf{P}_e$. The breadth of the scalpel blade d_{n} can be described as some function depending on d_{a}.

6.4 Dynamic Cut-Friction

As already mentioned the dynamic cut friction $\mathbf{F}_{cut_{dynamic}}$ applies as soon as $|\mathbf{F}_{cut}|$ exceeds the value of F_{enter}. During cutting the cut friction $\mathbf{F}_{cut_{dynamic}}$ does not necessarily point exactly into the opposite direction than $\mathbf{F}_{\|}$. The difference between the applied force $\mathbf{F}_{\|}$ and the reacting cut friction force $\mathbf{F}_{cut_{dynamic}}$ results eventually in the acceleration force $\mathbf{F}_m$ of the scalpel.

$$\mathbf{F}_m = \mathbf{F}_{\|} - \mathbf{F}_{cut_{dynamic}} \tag{15}$$

In the case of dynamic cut friction all of the three forces described earlier must be considered. Note that $F_{open\,\mathrm{x}}$ and $F_{friction\,\mathrm{x}}$ differ from the static case only in terms of smaller coefficients whereas $F_{displace\,\mathrm{x}}$ is new.

$$F_{open\,\mathrm{x}} = \mu_{dynamic} \cdot \eta \cdot c_{edge} \cdot d_{\mathrm{x}} \tag{16}$$

$$F_{displace\,\mathrm{x}} = \rho_{dynamic} \cdot \eta \cdot c_{shape} \cdot d_{\mathrm{x}} \cdot d_{\mathrm{d}} \cdot v_{\mathrm{x}} \tag{17}$$

$$F_{friction\,\mathrm{x}} = \alpha_{dynamic} \cdot |\mathbf{F}_{\perp}| \tag{18}$$

$\rho_{dynamic}$: dynamic tissue opening coefficient
c_{shape} : shape of the scalpel's blade
d_{d} : depth of the scalpel blade

6.5 Constraints on the Blade Movement

In order to model a correct interaction between the scalpel and the tissue, the movement of the scalpel's blade must be constrained along the direction of its edge. In fact, we will have to impose two separate constraints.

Firstly, the motion into the direction perpendicular to the blade is penalized through a force pushing the scalpel tip back to the plane. This force is proportional to the distance of the scalpel tip from the plane. We establish this force by restricting the proxy's movement to the plane. The same proxy is used for modeling the static cut friction. To this end, the proxy's position is fixed until F_{enter} is reached.

Secondly, we exploit the fact that a scalpel usually has only one sharp edge. Therefore it is assumed to cut only into the direction of forward motion. All backward motions are hence interpreted as pulling the tool back and, as a consequence, no force will be applied. The distinction between cutting and withdrawing can be obtained from the scalar product between $\mathbf{F}_{\|}$ and $\mathbf{a}$.

7 Results

To obtain models that are both realistic and efficient to compute we generated adaptive tetrahedral meshes. The material properties of the physics-based system as well as the boundary conditions were integrated into the model and the visual quality of the cuts was improved by making use of volume texture mapping.

7.1 Mesh Generation

The Visual Human cryosection data set [20] served us as a data source. In our example, the limp was segmented from the blue color by computing and thresholding each voxel. Subsequently, we applied subsampling and filtering to the initially high resolution data. To obtain the model, we subdivided each volume cell into six tetrahedra. The volume was then extracted and tetrahedralized by a marching tetrahedron-type isovolume algorithm. To achieve a consistent mesh representation, tetrahedra intersected by a surface were subdivided using a scheme that allows for a symmetrical subdivision of a quadrilateral, such as the one shown in Fig. 2c.

To achieve adaptive mesh representations of different resolutions, the initial mesh was further processed with a progressive mesh algorithm [25]. The weighting function was tuned to collapse all the small edges of the interior mesh while preserving the details on the mesh surface. The resulting mesh structure can be seen in Fig. 13 and Fig. 14.

7.2 3D Texturing

We believe that with increasing texture memory on graphics boards 3D textures will become manageable at adequate resolutions. Therefore, we applied 3D texture mapping to enhance the realism and the visual quality of the simulation. Initially, each node has canonic texture coordinates being associated with its spatial position in the 3D data set. Since filtering and isosurface mesh construction generate positional drifts of the nodes, we have to register the 3D texture dataset explicitly after the mesh construction. This step has to be performed only once and during preprocessing. Then, correctly aligned 3D textures can be displayed throughout the simulation.

7.3 Material Properties and Boundary Conditions

Besides the texture volume, our framework supports volumetric material properties and boundary conditions. The material volume is generated by segmenting the texture volume and by assignment of different stiffness values to individual tissue types. The detailed description of this nontrivial problem is beyond the scope of this paper, however, some instrumental tools were proposed by some of the authors in [13]. The rigid bone structures as well as the left and right boundaries of our knee model (red nodes in Fig. 14) are defined interactively as displacement boundary conditions. The tissue between the boundaries is anisotropically prestressed accounting for the fact that the leg muscles have a strong longitudinal orientation.

7.4 Examples

Fig. 12 shows 4 frames of a sequence representing an interactive surgical cut into the knee model which was performed using the PHANToM force-feedback device. The relaxation thread ran in parallel to the computations of the geometric updates. We observe that the tissue separates immediately after the scalpel force exceeds the described thresholds. This is due to the prestressing and the boundary conditions. To illustrate the performance of the simulation the elapsed time is displayed in every frame. The example was simulated on a Silicon Graphics Onyx2 Infinite Reality with eight R10000, 200 MHz processors. With the initial model consisting of 1381 tetrahedra we achieved 30 frames per second rendering speed.

To further elucidate the geometric processing and updates Fig. 13 illustrates the initial surface triangulation of our model and Fig. 14 depicts the underlying mesh of the last image of Fig. 12.

In order to demonstrate the performance of the presented methods Fig. 15 illustrates the capabilities concerning topological modifications and Fig. 16 the geometrical accuracy. Both examples initially consitst of 2980 tetrahedra.

8 Conclusions and Future Work

We have presented a system for interactively cutting soft tissue models that efficiently represents and tracks arbitrary cut trajectories with a high degree of detail. Besides collision detection and fast and stable relaxation methods our system features a haptical model of the interaction between the scalpel and the tissue.

Future work will comprise further improvements of the physical model including the representation of incompressibility and refinements of the intersected tetrahedra without insertion of additional simplices.

Acknowledgments

We would like to thank Remo Ziegler for the implementation of the surface collision detection algorithms and Alexander Beck for the integration of the haptical scalpel model.

References

[1] S. G. Arthur Gregory, Ming C. Lin and R. Taylor. "A framework for fast and accurate collision detection for haptic interaction." In *Proceedings of the IEEE Virtual Reality*, 1998.

[2] D. Baraff and A. Witkin. "Large steps in cloth simulation." In *SIGGRAPH Proceedings*, pages 43–54, 1998.

[3] C. Basdogan, Chih-Hao, and M. A. Srinivasan. "Simulation of tissue cutting and bleeding for laparoscopic surgery using auxiliary surfaces." In *Medicine Meets Virtual Reality*, pages 38–44. IOS Press, 1999.

[4] J. Bey. "Tetrahedral grid refinement." In *Computing*, volume 55, pages 355–378, 1995.

[5] D. Bielser, V. A. Maiwald, and M. H. Gross. "Interactive cuts through 3-dimensional soft tissue." In *Proceedings of the Eurographics '99*, volume 18, pages C31–C38, 1999.

[6] M. Bro-Nielsen. "Modelling elasticity in solids using active cubes - application to simulated operations." In N. Ayache, editor, *Computer Vision, Virtual Reality and Robotics in Medicine*, Lecture Notes in Computer Science. Springer-Verlag, Apr. 1995. ISBN 3-540-59120-6.

[7] M. Bro-Nielsen and S. Cotin. "Real-time volumetric deformable models for surgery simulation using finite elements and condensation." *Computer Graphics Forum*, 15(3):C57–C66, C461, Sept. 1996.

[8] S. F. Gibson. "3d chainmail: a fast algorithm for deforming volumetric objects." In *Proceedings 1997 Symposium on Interactive 3D Graphics*, pages 149–154, Apr. 1997.

[9] M. Gross. "Graphics in medicine: From visualization to surgery simulation." In *ACM Computer Graphics*, volume 32, pages 53–56, 1998.

[10] E. Hairer, S. P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations*. Springer, 1991.

[11] T. He. "Fast collision detection using quospo trees." In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 55–62, 1999.

[12] D. L. James and D. K. Pai. "Accurate real time deformable objects." In *SIGGRAPH Proceedings*, pages 65–72. ACM Press, 1999.

[13] R. Koch, M. Gross, D. von Bueren, G. Frankhauser, Y. Parish, and F. Carls. "Simulating facial surgery using finite element models." In *Proceedings of SIGGRAPH 96*, pages 421–428, 1996.

[14] R. Koch, S. Roth, M. Gross, A. Zimmermann, and H. Sailer. "A framework for facial surgery simulation." *Technical Report No. 327, Computer Science Department, ETH Zurich*, 1999.

[15] R. M. Koch, M. H. Gross, and A. A. Bosshard. "Emotion editing using finite elements." In *COMPUTER GRAPHICS Forum*, volume 17, pages C295–C302, 1998.

[16] U. Kuehnapfel, C. Kuhn, M. Huebner, H. Krumm, H. Maafl, and B. Neisius. "The karlsruhe endoscopic surgery trainer as an example for virtual reality in medical education." In *Minimally Invasive Therapy and Allied Technologies*, volume 6, pages 122–125. Blackwell Science Ltd., 1997.

[17] Y. Lee, D. Terzopoulos, and K. Waters. "Realistic face modeling for animation." In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 55–62. ACM SIGGRAPH, Addison Wesley, Aug. 1995.

[18] L. P. Nedel and D. Thalmann. "Real time muscle deformations using mass-spring systems." *Computer Graphics International*, pages 156–165, 1998.

[19] J. F. O'Brien and J. K. Hodgins. "Graphical modeling and animation of brittle fracture." In *SIGGRAPH Proceedings*, pages 137–146. ACM Press, 1999.

[20] N. L. of Medicine. "The visible human project." http://www.nlm.nih.gov/research/visible/visible_human.html.

[21] K. D. Reinig, H. L. Pelster, V. M. Spitzer, T. B. Johnson, and T. J. Mahalik. "More real-time visually and haptic interaction with anatomical data." In K. M. et al, editor, *Medicine Meets Virtual Reality*, pages 155–158. IOS Press, 1997.

[22] K. D. Reinig, C. G. Rush, H. L. Pelster, V. M. Spitzer, and J. A. Heath. "Real-time visually and haptically accurate surgical simulation." In S. H. H. Sieburg and K. Morgan, editors, *Health Care in the Information Age*, pages 542–546. IOS Press, 1996.

[23] D. Ruspini, K. Kolarov, and O. Khatib. "The haptic display of complex graphical environments." In *SIGGRAPH Proceedings*, pages 345–352, 1997.

[24] SensAble-Technologies. "Ghost software developer's toolkit." http://www.sensable.com, 1999.

[25] O. G. Staadt and M. H. Gross. "Progressive tetrahedralizations." In *Proceedings of IEEE Visualization '98*, pages 397–402, 1998.

[26] N. Suzuki, A. Hattori, S. Kai, T. Ezumi, and A. Takatsu. "Surgical planning system for soft tissues using virtual reality." *Medicine Meets Virtual Reality*, pages 159–163, 1997.

[27] P. Volino, M. Courchesne, and N. M. Thalmann. "Versatile and efficient techniques for simulating cloth and other deformable objects." In *SIGGRAPH Proceedings*, pages 137–144, 1995.

[28] K. C.-H. Wong, T. Y.-H. Siu, P.-A. Heng, and H. Sun. "Interactive volume cutting.", *Technical Report*, Department of Computer Sicence and Engineering, Chinese University of Hong Kong, 1998.

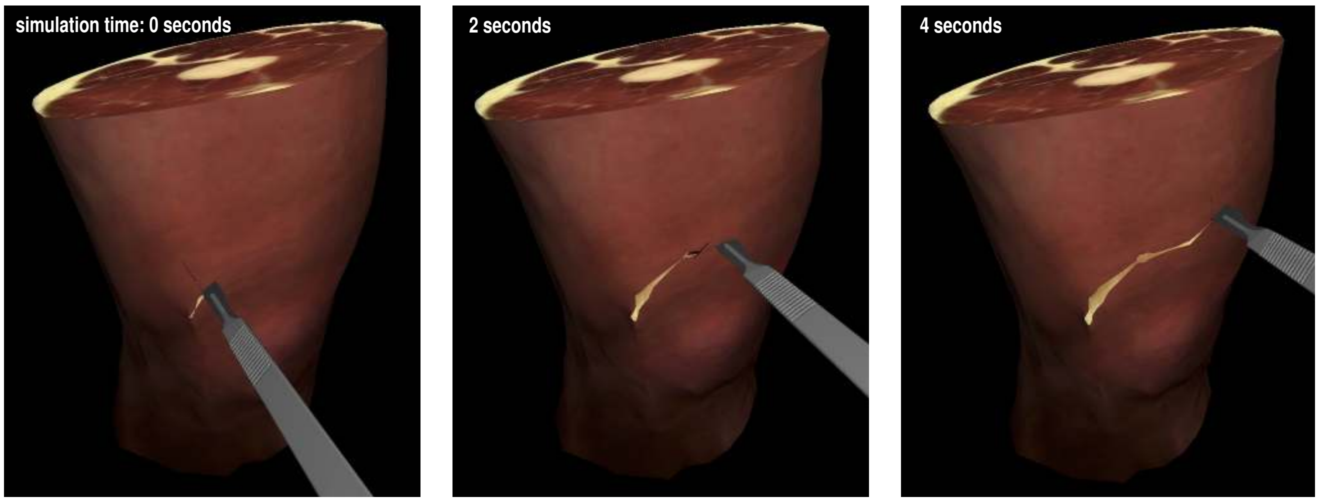
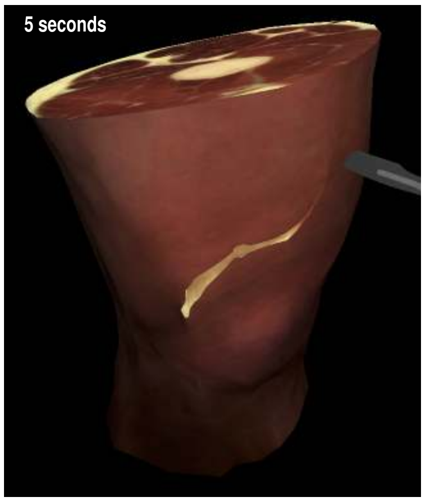


Figure 12: 4 frames of a sequence representing an interactive surgical cut



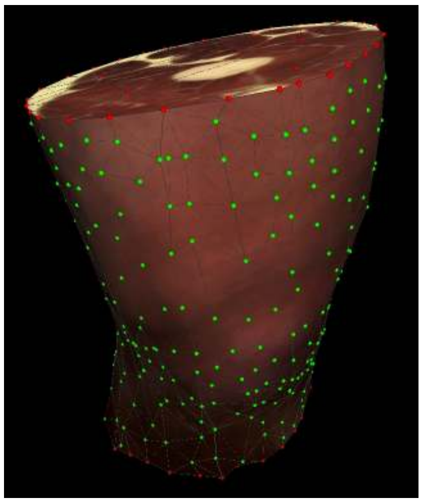


Figure 13: Initial triangulation

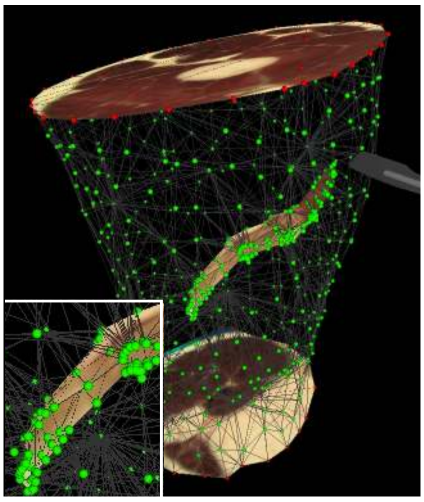


Figure 14: Mesh of Fig. 12d

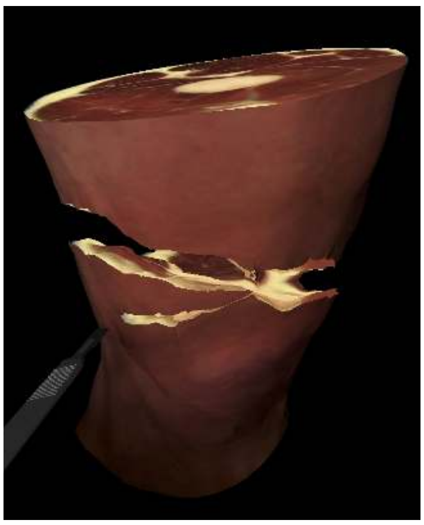


Figure 15: Topolog. modification

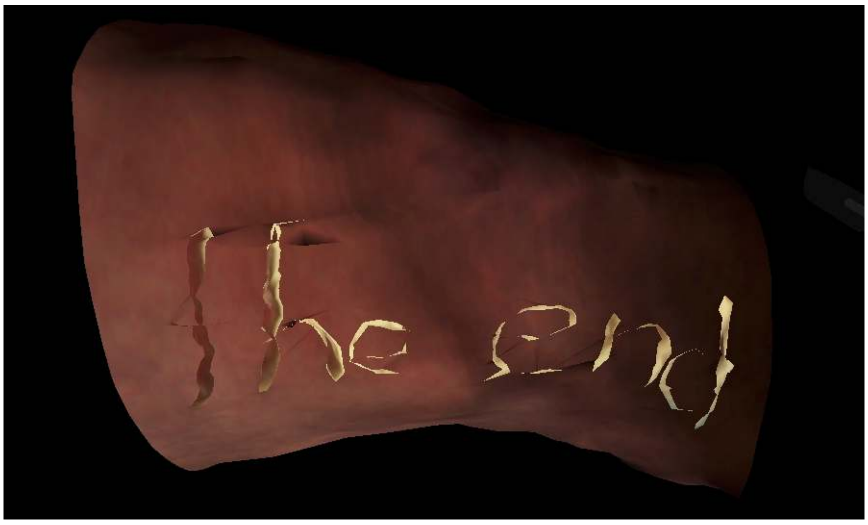


Figure 16: Geometrical accurancy