Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institute of Scientific Computing

Computer Graphics Research Group

CS Technical Report 309, November 11, 1998

Interactive Cuts through 3-Dimensional Soft Tissue

Daniel Bielser, Volker A. Maiwald, Markus H. Gross

Computer Science Department
ETH Zurich, Switzerland
e-mail: {bielser, grossm}@inf.ethz.ch
http://www.inf.ethz.ch/department/IS/cg

Interactive Cuts through 3-Dimensional Soft Tissue

Daniel Bielser, Volker A. Maiwald, Markus H. Gross

Department of Computer Science
Swiss Federal Institute of Technology (ETH), Zürich, Switzerland
e-mail: {bielser, grossm}@inf.ethz.ch

ABSTRACT

We describe a physically based framework for real-time modeling and interactive cutting of 3-dimensional soft tissue that can be used for surgery simulation. Unlike existing approaches which are mostly designed for tensorproduct grids our methods operate on tetrahedral decompositions giving more topological and geometric flexibility for the efficient modeling of complex anatomical structures. We start from an initial tetrahedralization such as being provided by any conventional meshing method. In order to track topological changes tetrahedra intersected by the virtual scalpel are split into substructures whose connectivity follows the trajectory of the cut, which can be arbitrary. For the efficient computation of collisions between the scalpel and individual tetrahedra we devised a local collision detection algorithm. The underlying physics is approximated through masses and springs attached to each tetrahedral vertex and edge. A hierarchical Runge-Kutta iteration computes the relaxation of the system by traversing the designed data structures in a breadth-first order. The framework includes a force-feedback interface and uses real-time texture mapping to enhance the visual realism.

Keywords: Physically Based Modeling, Surgery Simulation, Soft Tissue, Tetrahedralization, Interactive Cut, Virtual Scalpel, Runge Kutta Method;

1 INTRODUCTION

Surgery simulation is an extremely challenging field of research encompassing the efforts of various disciplines, including Graphics, Vision, Mechanics, Robotics, and Medicine. With the availability of low-cost 3D graphics hardware, systems for surgical training or planning emerged from many research labs and are partly in use as instrumental tools [6]. Besides the development of advanced human computer interfaces for surgery simulation the key research issues relate to the provision of advanced computational models for the real-time representation, deformation, and rendering of soft tissue structures. In most cases, the ultimate goal is to interactively manipulate high resolution 3D models. In order to tackle this problem, we have to address two different aspects: The first one relates to the development of efficient representations of the underlying 3D geometry tolerating topological changes during manipulation. The second one applies to the fast computation of the physics of deformation, which, of course, has to balance real-time performance against computational accuracy.

Due to the fundamental importance of the above issues considerable related work has been done in the Graphics and Vision communities, part of which, however, in different application contexts. The computationally most accurate methods for the modeling of elastic soft tissue mostly use Finite Element procedures to solve the corresponding governing equations. [2], for instance, develops surface-based snakes to represent human organs. [9] conveys a FEM based model for facial surgery simulation and extend it for animating human emotions [10]. Full volumetric soft tissue models over tetrahedral discretizations can be found in [17]. The powerful mathematical setting of FEM procedures, however, comes along with the drawback of computational costs.

In order to make 3D soft tissue modeling real-time, different strategies have been advocated. Most notably, [3] suggests methods to accelerate the conventional FEM setting. Another interesting approach is the 3D ChainMail, as introduced by [5]. Rather than computing physical deformations on the fly, the method uses a two-pass hybrid scheme, where in a first pass, pure 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 [11] employ surface based mass-spring systems for their real-time simulators. Similar Euler type methods on regular grids are reported in [19]. Pure geometric and topological manipulations based on marching cubes techniques can be found in the algorithms of [16] and [15], whose visual quality is amazing. In most approaches, force feedback devices are utilized to implement the interface to the user.

Furthermore, [12] for instance, developed a soft tissue model for facial animation and [13] used mass-spring and particle systems for the representation of human muscles. Other interesting work can be found in [20] who devised an efficient collision detection method for cloth simulation. Lately, [1] optimized implicit numerical solution strategies for efficient use in cloth modeling.

In summary, existing real time cutting approaches are either designed for surface based tissue models or they restrict the discretization of the underlying continuum to a tensor product topology. The major novelty of our framework lies in the fact that it operates on irregular tetrahedral decompositions thus providing much more topological and geometric flexibility for the efficient representation of complex anatomical structures. The required tetrahedralization can be conveyed by any conventional meshing method, including Delaunay or 3D progressive meshes as described in [7] and [18]. In order to track the topological changes of the mesh when cutting through the tissue, we split all intersected tetrahedra into subsets of smaller simplices. The procedure follows the trajectory of the scalpel, where in most cases collision detection can be handled locally. The designed data structures allow a fast update of the representation. Finally, a hierarchical fourth order Runge Kutta relaxation drives the underlying governing equations. The iteration step size is adapted to the distance from the current scalpel position. We currently use a PHANToM® as a 6 degree of freedom interface to the system.

Our paper is organized as follows: In section 2 we present an overview of our approach and describe its conceptual components. Section 3 addresses the problem of efficient collision detection. The procedures required to track and update geometric and topological changes are illuminated in section 4 and our numerical solution strategies are detailed in section 5. Finally, we discuss the performance and limitations of our framework in section 6. [1]

[1] In order to demonstrate the methods we have included some sample software for the reviewers which runs both on Windows PCs and on SGI workstations.

2 Overview

Fig. 1 depicts the fundamental components of our system. In order to decouple the simulation pipeline from the force-feedback device, we run different processes communicating via TCP/IP. After each rendering step the current position and orientation of the scalpel are read by the *Read Force-Feedback* module and transferred into the modeling pipeline. Then, a sequence of individual processing steps has to be performed allowing one to eventually render the scalpel and the soft tissue model. In a first step possible collisions between the trajectory of the scalpel and individual edges of the tetrahedral mesh have to be detected. In case of collisions, all necessary geometric and topological changes of the underlying tetrahedralization must be tracked and solved. The subsequent relaxation step updates the mass-spring system enabling tissue structures to open or deform when manipulated. In the following sections, we will describe the technical details of all components presented below.

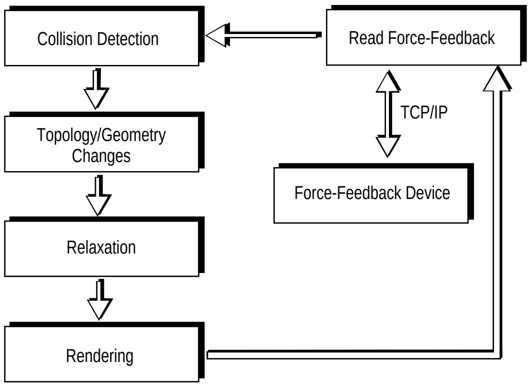


Figure 1: Conceptual components and data flow

3 Collision Detection

The detection of interactions between the virtual scalpel and the tetrahedral mesh is a substantial prerequisite for any further topological or geometric analysis and operates locally on a list of *active tetrahedra*. Here, an active tertrahedron has at least one intersection point with the scalpel, which, for now, is represented by a thin line. The list is updated at each time step Δt. Fig. 2 shows two consecutive scalpel positions over time. All active tetrahedra for the scalpel position at time t_i are drawn in black, whereas those at t_{i+1} are presented in yellow. The active tetrahedron containing the point of the scalpel shall be named *active point tetrahedron* (bold edges in Fig. 2). The swept surface bounded by the scalpel positions at t_i and t_{i+1} and by the line between the two scalpel points is colored in red. Assuming the time step Δt to be sufficiently small and approximating the swept surface by a plane the collision detection reduces to finding a) intersections between the approximating plane and the *active tetrahedra* edges and b) intersections of the trajectory of the scalpel point and one of the faces of the *active point tetrahedron*. As we will see, both computations are needed to determine the topology of the tetrahedral splits explained in Section 4.

Note, that an intersection between the trajectory and one of the adjacent tetrahedral faces only occurs, if the scalpel point leaves the old *active point tetrahedron*. This can be checked by an inside test of the scalpel point. If this test fails we search the new *active point tetrahedron* in the direct neighborhood of the old one by applying similar inside tests. Of course, the success of this strategy is based on the assumption that for sufficiently small time steps Δt the probability of finding the new *active point tetrahedron* in the direct vicinity is very high. Therefore, the algorithm's average

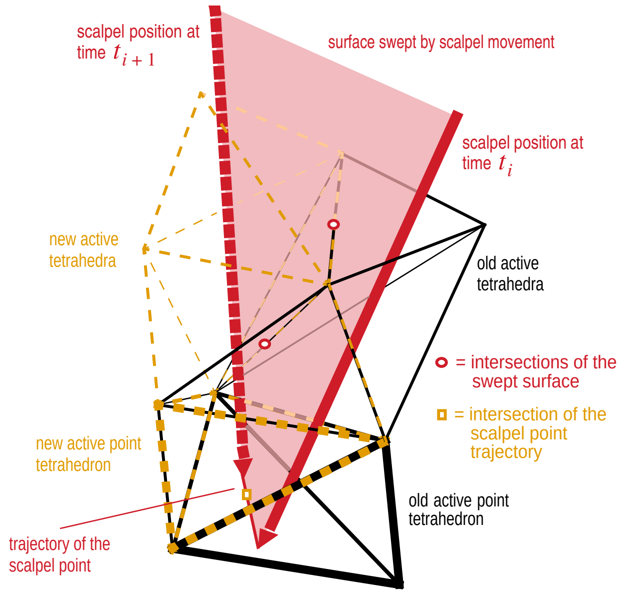


Figure 2: Local collision detection using a list of active tetrahedra

complexity is constant time *O(1)*, albeit a global search is needed to compute intersections in the worst case. After finding the new *active point tetrahedron* we calculate the intersection of the trajectory and the face shared by both. The pseudocode fragment drafted below illustrates the algorithm.

```
determineNewActivePointTetrahedron(active_point_tetra)

    if(InsideTetraTest(active_point_tetra) == TRUE)
        return active_point_tetra
    else
        for all neighbor_tetra of active_point_tetra do
            if(InsideTetraTest(neighbor_tetra) == TRUE)
                return neighbor_tetra
    for all tetra
        if(InsideTetraTest(tetra) == TRUE)
            return tetra
```

Note furthermore, that the global search could be substituted by a recursive breadth-first traversal, however, the high hit rates obtained in our experiments report no significant loss in speed using a global search.

In a second step we determine the cut edges by calculating the intersections between the swept surface and the edges of all old *active tetrahedra.* Subsequently, for each active tetrahedron the intersected edges are marked using a bitcode (*cut-edge-code)* consisting of a bit for each of the six tetrahedral edges. This code is stored for subsequent analysis. Again, a pseudocode fragment illustrates the algorithm:

```
calculateEdgeIntersections()

    for all old_active_tetra do
        for all edge of old_active_tetra do
            if(edge.cut == FALSE)
                if(FaceIntersectionTest(edge) == TRUE)
                    edge.cut = TRUE
```

This procedure enables us to record possible topological changes at time t_i both using active simplices at t_i and positional information at t_i and t_{i+1}.

The last step consists of finding the new list of active tetrahedra. Again, the spatial coherency allows us to devise an incremental updating of the old active list given at time t_i. Starting from the *active point tetrahedron* we trace along the scalpel through all penetrated tetrahedra at time t_{i+1}. Those tetrahedra already found in the old active list are kept, (in Fig. 2 only the first one behind the new *active point tetrahedron*), others are removed and the newly found tetrahedra are added. Here, we exploit the adjacency relations encoded in our data structure:

```
buildNewActiveTetrahedraList()
    old_active_tetra = first of old active tetra list

    // recycle entries of the old active tetra list
    while(IntersectionTest(old_active_tetra) == TRUE) do
        new_active_tetra = old_active_tetra
        old_active_tetra = old_active_tetra.next
        new_active_tetra = new_active_tetra.next

    // find and add newly active tetrahedra
    exist_intersected_tetra = TRUE
    while(exist_intersected_tetra == TRUE)
        for all neighbor_tetra of new_active_tetra
            exist_intersected_tetra = FALSE
            if(IntersectionTest(neighbor_tetra) == TRUE)
                exist_intersected_tetra = TRUE
                new_active_tetra = neighbor_tetra
                new_active_tetra = new_active_tetra.next
```

The described procedures are triggered once the scalpel penetrates the surface of the tissue structure.

4 Geometric and Topological Operations

After maintaining a list of *active tetrahedra* and retaining all information about the intersection between the scalpel blade and the edges, appropriate geometric and topological operations on the tetrahedra have to be carried out.

4.1 Cutting Tetrahedra

We start from the observation that there are only five topologically different cases in which a tetrahedron can be cut during one incision. Fig. 3 exemplifies the five cases as occurring during a cut trough a tetrahedral mesh, denoted by A through E. The cases A and B represent a full cut and correspond to those used in the marching tetrahedra algorithm [8]. In addition, we distinguish between three types of slit tetrahedra distinguished by the number of edge intersections (one in C, two in D and three in E).

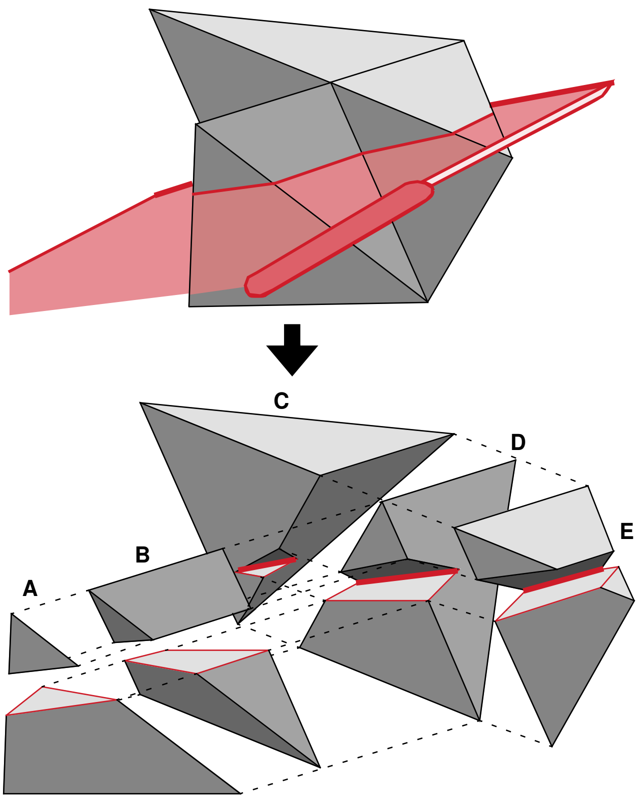


Figure 3: 5 different topologies generated when cutting a tetrahedral mesh

Including all possible rotations and mirroring operations we end up with four different subcases for A, three subcases for B, and six for C, respectively. In addition, we obtain 12 combinations each for cutting the two edges of case D and the three edges of case E. For each of the five cases we first store the set of actions required to establish the new mesh in a lookup-table entry. These actions include the insertion of new massnodes, the assignment of their connectivity, and the insertion of new faces. By rotating and mirroring these five entries we get all possible combinations of edge intersections necessary to complement the lookup-table. The key for the lookup-table, from where all required actions and topological information can be taken, is the *cut-edge-code* described in section 3.

4.2 Splitting Tetrahedra

In order to avoid individual subdivision procedures for each of the five cases and to simplify implementation, we propose to apply a generic 1:17 tetrahedral split, such as presented in Fig. 4. We observe that all possible cases from section 4.1 can be mapped onto this subdivision.

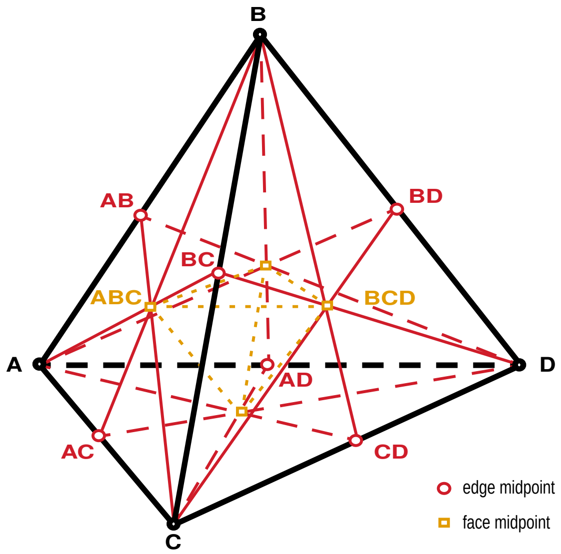


Figure 4: Generic 1:17 split of a tetrahedron

Note that Fig. 4 only depicts the topology of the split. For all edges and faces, the correct geometry of a cut surface is computed by replacing the indicated edge and face midpoints by the current intersection points. Recall that this is exactly, where the results of the collision detection computations from section 3 are used.

By referencing the edge midnodes twice and the face midnodes three times the above subdivision scheme can be implemented as a presplit tetrahedron. Fig. 5 illustrates the five parts of the presplit tetrahedron according to the explained method.

4.3 Data Structures

In our implementation, we use the following data structures: Initially, the two references of the edge midnode (e.g. AB1 and AB2 in Fig. 5) and the three references of the face midnode (e.g. ABC1, ABC2 and ABC3 in Fig. 5) point to the same node and glue the pieces of the tetrahedron together. By defining a *splitEdge* and a *splitFace* operator (Fig. 6) all the cuts can be represented. The *splitEdge* operation is invoked for all edges that have to be cut. The operation subdivides the initial nodal mass m into two submasses m_1, m_2 and assigns them to the corresponding vertices. Similar operations hold for the *splitFace* procedure. A new mass is inserted

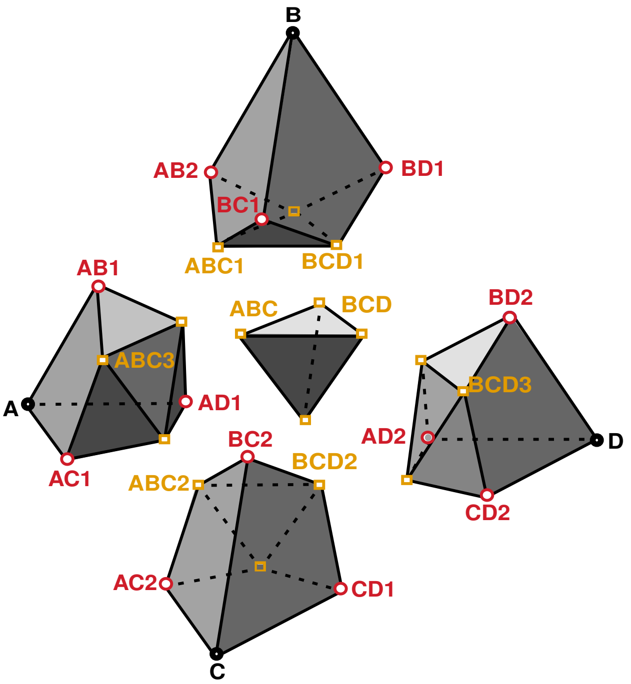


Figure 5: Presplit tetrahedron according to the employed subdivision

and assigned to one of the edges (bold lines in Fig. 6). In both cases we have to redistribute the masses according to the underlying discretization and material settings. Here different algorithms are possible including total mass preservation constraints.

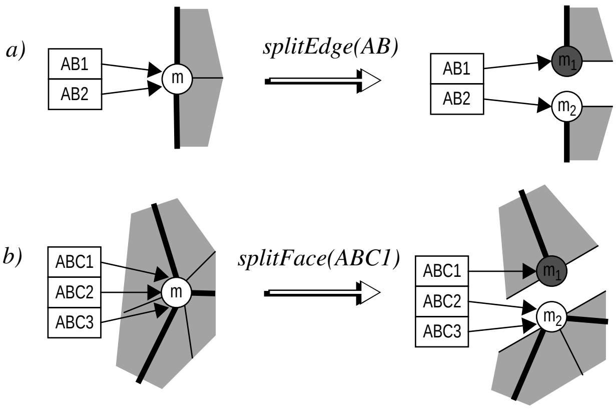


Figure 6: a) *SplitEdge* operation, b) *SplitFace* operation

The described approach enables one to use a single generic subdivision for all five cases including the insertion of all new edges, faces and masses. For efficient representation we store the split operations defined above in a lookup-table, such as described in Section 4.1. In general the operations *splitEdge* and *splitFace* are sufficient to represent all tetrahedral splits. However, since we only insert the visible cut-faces into the geometric representation we have to store additional operations in the lookup-table. Fig. 7 shows a lookup-table entry for case D of Fig. 3.

Comparing the memory consumption of our datastructures to typical tetrahedral meshes, the additional data is primarily required for the representation of the mass-spring system and for the design of efficient access methods.

Some limitations of the generic subdivision procedure deserve further discussion: The first one relates to the potential existence of hanging nodes having no connection to adjacent tetrahedra. This may lead to cracks in the representation. However, due to the spatial coherency of the scalpel trajectory we can avoid major visual artifacts. Although it is possible to solve for all hanging nodes of the representation by splitting adjacent tretrahedra appropriately, we decided to balance accuracy against computational costs and renounced this operation.

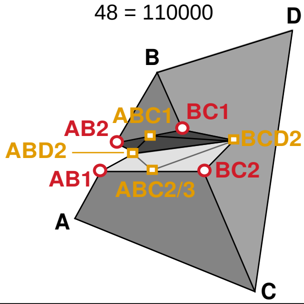
cut-edge-code = AB|BC|BD|AC|CD|AD	action
48 = 110000 	*splitEdge*(AB) *splitEdge*(BC) *splitFace*(ABC1) *insertFace*(BC1, ABC1, BCD2) *insertFace*(AB2, ABC1, ABD2) *insertFace*(ABC1, BCD2, ABD2) *insertFace*(BC2, ABC2, BCD2) *insertFace*(AB1, ABC3, ABD2) *insertFace*(ABC2, BCD2, ABD2)

Figure 7: Example of a lookup-table entry

The second one relates to the fact that the splitting procedure of the tetrahedra is invoked after completing the cut. This might lead to minor positional and visual discontinuities.

5 Relaxation

As already pointed out the underlying physics is modeled by a continuous relaxation of a damped mass-spring system. The numeric solution strategies used to solve the second order differential equations have to find a trade-off between framerate constraints and numerical accuracy. In general, fast convergence requires more computationally expensive schemes and leads to lower framerates assuming one frame for each update cycle. The relaxation algorithm presented in the following section is an adapted fourth order Runge Kutta scheme for second order differential equations. We give an intuitive and geometric derivation of the scheme and obtain correction terms which are similar to the mathematically rigorous treatment in [4].

5.1 The Two-Level Runge Kutta Method

Mass-spring systems can be computed by any numerical method solving the governing system of ordinary second order linear differential equations of type

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

Given n nodes the equation establishes the equilibrium of forces for each of the m_k of the diagonal mass matrix $\mathbf{M} \in \Re^{3n \times 3n}$ with $\mathrm{diag}(\mathbf{M}) = [m_1, m_1, m_1, m_2, m_2, m_2, \ldots, m_n, m_n, m_n]$ and describes their positional movement $\mathbf{x}(t) = [\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots, \mathbf{x}_n(t)]$ over time t. $\mathbf{D}$ denotes the damping and $\mathbf{K}$ the stiffness matrix. $\mathbf{D}$ and $\mathbf{K}$ are sparse matrices, element of $\Re^{3n \times 3n}$ consisting of n matrices $\mathbf{D}_k \in \Re^{3 \times 3}$ and $\mathbf{K}_k \in \Re^{3 \times 3}$, respectively. $\mathbf{F}_{ext}$ is a vector of dimension $3n$ and represents the vector of external forces. It can be divided into n 3-dimensional vectors $\mathbf{f}_k$. The above system is solved by iterative stepwise processing of each individual equation. To simplify notation, we will only consider the governing equation for a single mass m_k and omit all indices k for the position $\mathbf{x}_k(t)$.

In order to derive the method we divide the second order[2] differential equation for a single mass m_k into a system of two first order differential equations by introducing the velocity function $\mathbf{v}(t)$.

$$\left| \begin{array}{c} \dfrac{d\mathbf{x}(t)}{dt} = \mathbf{v}(t) \\ \dfrac{d\mathbf{v}(t)}{dt} = \dfrac{\mathbf{f}_k - \mathbf{D}_k \cdot \mathbf{v}(t) - \mathbf{K}_k \cdot \mathbf{x}(t)}{m_k} \end{array} \right| \tag{2}$$

Note that the vector-valued functions can be separated and solved by calculating three scalar differential equations for each component of $\mathbf{x}(t)$ and $\mathbf{v}(t)$.

We assume that at time t_0 the equation has the initial values $x_0 = x(t_0)$ and $v_0 = v(t_0)$. Given the position $x_n = x(t_n)$ and velocity $v_n = v(t_n)$ at time t_n our goal is to compute the values x_{n+1} and v_{n+1} respectively at the time $t_{n+1} = t_n + \Delta t$. We observe that there are two interdependent integration levels which can be integrated using standard Runge Kutta steps, such as known from [14]. Fig. 8 gives a pictorial representation of the different curves to be integrated during relaxation using the variables x, v and t as independent axes of a 3D Euclidean space. Consequently, the curve $x(t)$ lies in the bottom plane of the chart. An approximation of this function requires the values of the tangent vector. It can be obtained by the velocity (gradient) function v(t) lying in the back plane of Fig. 8 with $\vec{v}_n = [1, v(t)]^T$. Likewise, an approximation of $v(t)$ can be calculated by using the velocity gradient $\vec{a}_n = [1, a(t)]^T$ which in turn requires values of $x(t)$.

The basic idea of the following method is to combine the two integration levels by making use of the geometric correspondencies illustrated in the chart.

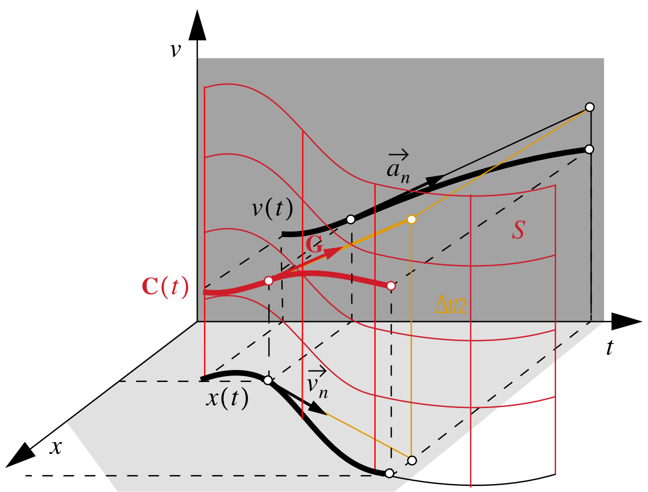


Figure 8: Geometric interpretation of the Runge-Kutta method adapted to second order differential equations

Therefore, we construct the spatial parametric curve $\mathbf{C}(t)$ by projecting the $v(t)$ curve from the back plane onto the surface S. This surface is traced out by translation of $x(t)$ along the v-axis. Note that $\mathbf{C}(t)$ defines the position $x(t)$ and the velocity $v(t)$ of one spatial coordinate of the mass for each time t.

Now, in order to solve the equation system (2), we integrate the "3D" curve **C**(t) over time.

To this end, we introduce the gradient $\mathbf{G}$ of $\mathbf{C}(t)$ given by

$$\mathbf{G}(t) = \begin{bmatrix} 1 & v(t) & a(t) \end{bmatrix} \tag{3}$$

The method starts with the computation of the first intermediate value by multiplying $\mathbf{G}(t_n)$ with half the stepsize ($\Delta t / 2$). The intermediate value for $\mathbf{C}(t_n + \Delta t/2)$ is given correspondingly:

$$\begin{bmatrix} t_n + \dfrac{\Delta t}{2} & x_n + \dfrac{\Delta t}{2} \cdot v_n & v_n + \dfrac{\Delta t}{2} \cdot a_n \end{bmatrix}^T \tag{4}$$

We compute a new intermediate guess for the gradient, say $\mathbf{G}'$, by evaluating the acceleration at this position and continue in the same fashion, we know well from Runge Kutta for first order equations [14].

Switching back from the scalar representations $x(t)$, $v(t)$ and $a(t)$ into the vectorial world $\mathbf{x}(t)$, $\mathbf{v}(t)$ and $\mathbf{a}(t)$ the complete set of equations that make up the scheme can be summarized as follows:

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{v}_n \\ \mathbf{l}_1 &= \mathbf{a}(t_n, \mathbf{x}_n, \mathbf{v}_n) \\ \mathbf{k}_2 &= \mathbf{l}_1 \Delta t \\ \mathbf{l}_2 &= \mathbf{a}\left(t_n + \frac{\Delta t}{2}, \mathbf{x}_n + \frac{\Delta t}{2}\mathbf{k}_1, \mathbf{v}_n + \frac{\Delta t}{2}\mathbf{l}_1\right) \\ \mathbf{k}_3 &= \mathbf{l}_2 \Delta t \\ \mathbf{l}_3 &= \mathbf{a}\left(t_n + \frac{\Delta t}{2}, \mathbf{x}_n + \frac{\Delta t}{2}\mathbf{k}_2, \mathbf{v}_n + \frac{\Delta t}{2}\mathbf{l}_2\right) \\ \mathbf{k}_4 &= \mathbf{l}_3 \Delta t \\ \mathbf{l}_4 &= \mathbf{a}(t_n + \Delta t, \mathbf{x}_n + \Delta t \mathbf{k}_3, \mathbf{v}_n + \Delta t \mathbf{l}_3) \end{aligned} \tag{5}$$

Here, the $\mathbf{k}_i$ stand for the gradients in the bottom plane providing the desired estimations for the velocity, whereas the $\mathbf{l}_i$ represent the gradients of the back plane function approximating the acceleration.

The new position and velocity at time t_{n+1} are calculated straightforwardly by integration using the standard correction terms from [14]:

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n + \frac{\Delta t}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + \frac{\Delta t}{6}(\mathbf{l}_1 + 2\mathbf{l}_2 + 2\mathbf{l}_3 + \mathbf{l}_4) \end{aligned} \tag{6}$$

5.2 Hierachical Breadth-First Traversal

Using the relations derived above the system is solved by iteratively calculating each mass position. As with standard Euler schemes this procedure requires quadratic computational expense. In order to further reduce the complexity of the method we added the following two performance enhancements:

- exponential decay of the number of update cycles based on topological distance
- upper recursion bounds leading to local relaxation procedures

The traversal of the massnodes operates in a breadth-first manner, starting from the current scalpel position, called *focus_of_traversal*. The number of updates of the relaxation is decreased exponentially with the topological distance of the asso-

[2] The reader might distinguish from the "order" of the differential equation to be solved (=2) and the "order" of the Runge Kutta method (=4).

ciated node from the focus. This strategy is justified by the observation that due to the physics of deformation, most significant changes of the continuum occur on average in the direct vicinity of the applied external forces. Fig. 9 gives an illustration of the traversal method.

Let s indicate the topological distance of a massnode from the *focus_of_traversal* measured in terms of the smallest number of edges between them. The function $i(s)$ defines the number of iteration steps for the massnode as a function of s and is set to decrease exponentially.

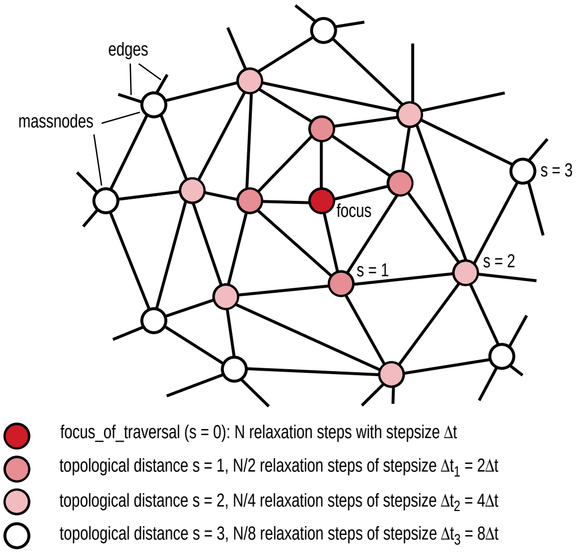


Figure 9: Hierachical breadth-first traversal strategy for efficient relaxation

Furthermore, we fix a maximum iteration depth s_{max} to bound the recursion. Starting with I iteration steps for the *focus_of_traversal* we receive

$$i(s) = \begin{cases} I \cdot 2^{-s} & s \le s_{max} \\ 0 & s > s_{max} \end{cases} \tag{7}$$

At each relaxation step a global time variable is increased by a given increment Δt. Storing a time stamp along with each node its time increment is computed by subtraction of the stamp from the global time variable. Consequently, those nodes are exposed to time increments of size

$$\Delta t_s = \frac{\Delta t}{i(s)} \tag{8}$$

For the calculation of the spring forces, the positions of the neighbor nodes must be considered. These positions can be estimated by using their velocity vector.

The computation of the nodal position includes a damping constant which can slow down the convergence of the system for inappropriate settings. Therefore, we support penalty based damping as well.

6 Results

The sequence of images presented in Fig. 10 shows four frames of a cut through a grid of initially 576 tetrahedra. To enhance the visual realism of the tissue, we applied texture mapping both for exterior and for the interior faces generated during the cut. Displacement boundary conditions were set along the left- and right-hand sides of the volume. Fig. 11[3] gives a more detailed presentation of the topological changes of the mesh by depicting the wire frame representations of the first and of the last frame of Fig. 10. We observe that the tetrahedral splits follow the trajectory of the scalpel. The number of simplices after the procedure grows to 2446.

To demonstrate the ability of our approach to support arbitrary cutting trajectories on even coarse initial meshes Fig. 12 depicts two additional examples based on 2x2x2 and 7x7x7 grids. In order to simplify the construction of the initial meshes the presented examples are based on tetrahedralizations of regular structures using six tetrahedra per cell. However we emphasize, that the presented algorithms impose no restrictions on the initial mesh which can be arbitrary.

Finally, Table 1 gives an idea of the algorithm's performance broken down into the different tasks. The time consumption of the collision detection and of the geometric/topological operations is relatively small compared to the relaxation and rendering procedures. We can clearly verify that the hierarchical and localized relaxation algorithms of section 5 lead to gracefully decreasing framerates as a function of the initial mesh size.

number of cells	initial number of tetrahedra	final number of tetrahedra	average frame-rate	percentage			
				RX	RD	GC	CD
$2 \times 2 \times 2$	48	354	3.78	82.1	11.4	1.7	0.3
$2 \times 4 \times 3$	144	960	2.24	88.9	7.8	0.4	0.4
$4 \times 6 \times 4$	576	2446	1.14	92.2	4.9	0.1	0.5

Table 1: System performance on a SGI Indigo 2, Impact with R10000, 180 MHz. The framerate is averaged over the cut. Notation: relaxation (RX), rendering (RD), topology/geometric changes (GC) and collision detection (CD).

Although the proposed method enables users to freely cut soft tissue structures, there are two restrictions which have to be mentioned:

Since the collision-detection is based on the scalpel point position, it must be the first part to enter the tissue. Likewise, the scalpel-point has to be the last one leaving the tissue.

In addition, no backward movement is supported in the current setting, because each tetrahedron can only be processed once.

Conclusions and Future Work

We presented a framework for the representation and physically based manipulation of volumetric soft tissue that is based on tetrahedral decompositions of the underlying continuum. The framework encompasses several algorithms allowing the efficient computation of all necessary steps including geometry, topology and numerics. Thus, users can freely cut through three dimensional soft tissue along almost arbitrary paths.

Our future work is primarily targeted at finding efficient solutions for the described limitations. Specifically, the full avoidance of cracks and visual discontinuities is an important direction. Further research will be conducted towards a reduction of the substantial increase of simplices during cutting. Additionally, the migration from the line representation of the scalpel to a volumetric representation and the involved force-feedback simulation are topics, we are interested in. Finally, alternative numeric solution strategies will be subject of future research.

Acknowledgment

This research has been supported in part by the Swiss National Science Foundation under grant No. 21-49247.96.

[3] We encourage the reviewers to verify the tetrahedral splits interactively by using the "F", "G" and "C" options of our demo software.

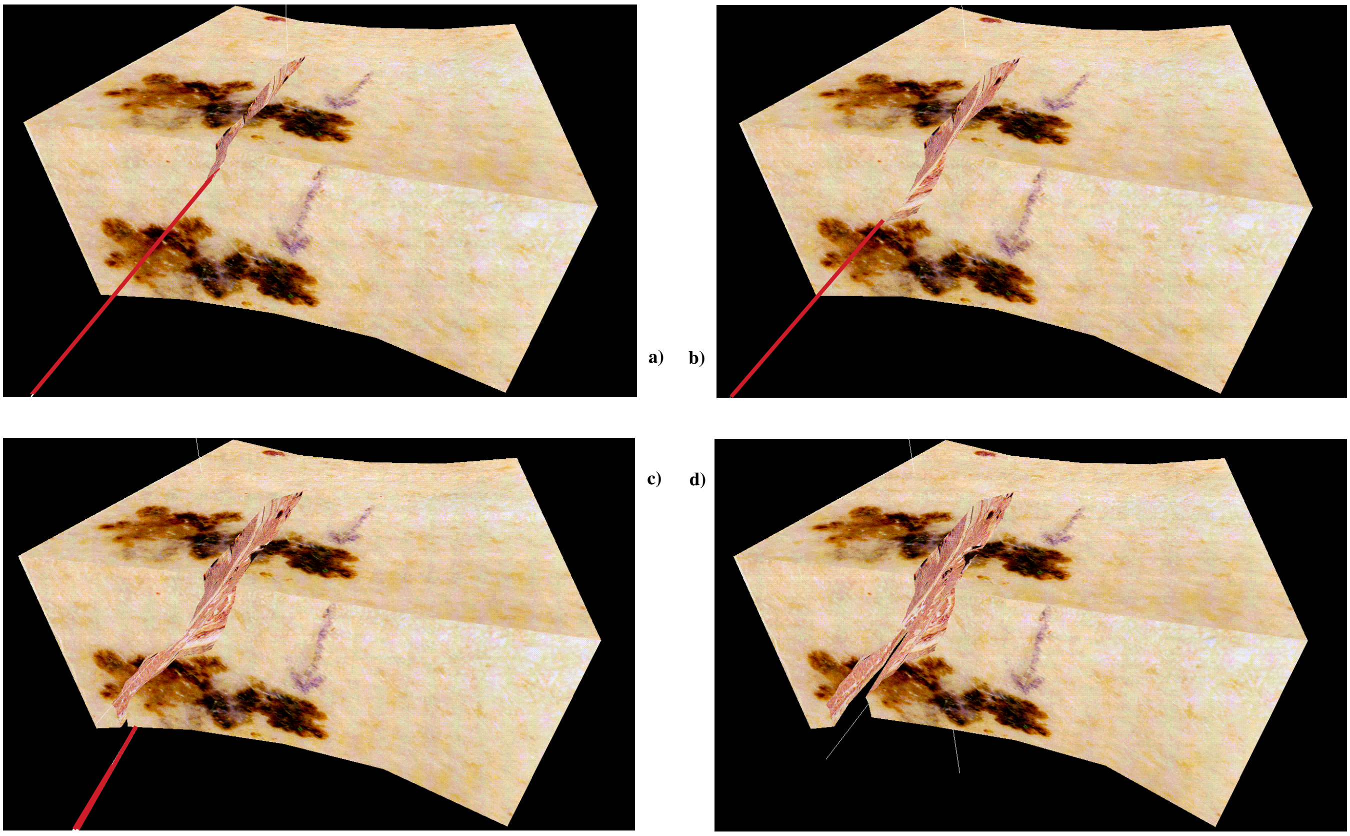


Figure 10: Four frames of an interactive cut through a grid of $4 \times 6 \times 4$ cells

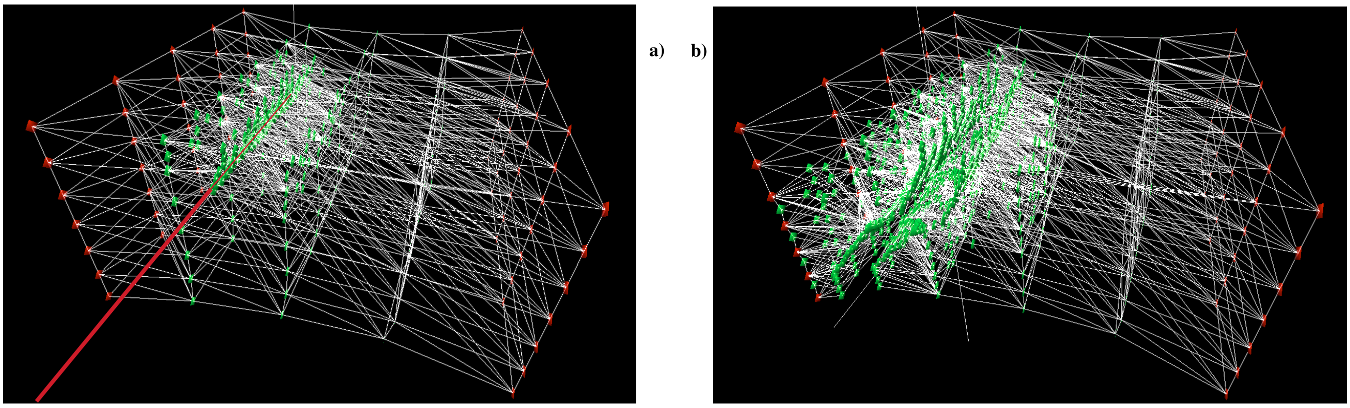


Figure 11: Mesh structures corresponding to frames a) and d) of Fig. 10

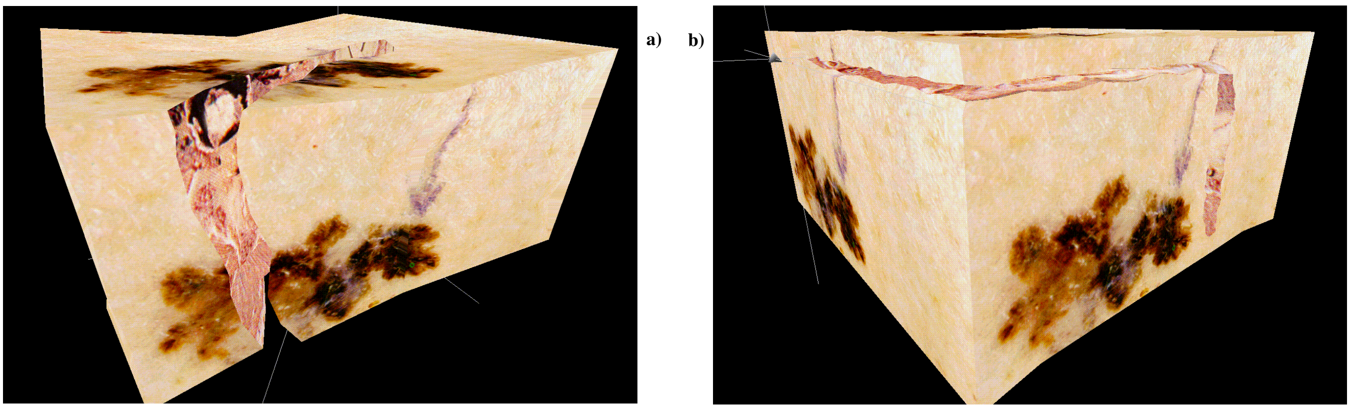


Figure 12: Examples of cuts: a) $2 \times 2 \times 2$ grid, b) $7 \times 7 \times 7$ grid

References

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

[2] 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, pages 535–541. Springer-Verlag, Apr. 1995. ISBN 3-540-59120-6.

[3] 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.

[4] L. Collatz. *The Numerical Treatment of Differential Equations*, chapter 2, pages 61–73. Springer, 1966.

[5] 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.

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

[7] M. H. Gross, O. G. Staadt, and R. Gatti. “Efficient triangular surface approximations using wavelets and quadtree data structures.” In *IEEE Transactions on Visualization and Computer Graphics*, volume 2, pages 130–143, 1996.

[8] P. S. Heckbert. *Graphics Gems IV*, chapter 4, page 329. 1994.

[9] R. M. Koch, M. H. 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.

[10] 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.

[11] U. Kühnapfel, C. Kuhn, M. Hübner, 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.

[12] 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.

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

[14] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*, chapter 16, pages 710–714. Cambridge University Press.

[15] 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*, pages 155–158. IOS Press, 1997.

[16] K. D. Reinig, C. G. Rush, H. L. Pelster, V. M. Spitzer, and J. A. Heath. *Real-Time Visually and Haptically Accurate Surgical Simulation*, chapter 60, pages 542–546. IOS Press, 1996.

[17] S. H. M. Roth, M. H. Gross, and F. R. C. S. Turello. “A bernstein-bezier based approach to soft tissue simulation.” In *COMPUTER GRAPHICS Forum*, volume 17, pages C285–C294, 1998.

[18] O. G. Staadt and M. H. Gross. “Progressive tetrahedralizations.” In *Proceedings of IEEE Visualization ’98,* pages 397-402, 1998.

[19] 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.

[20] 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.