Chapter 3

Combining FEM and cutting: a first approach

In this chapter, we discuss our first approach to interactive surgery simulation.[1] We will show how cuts can be applied to an interactively deformable object. More formally spoken, we will discuss the following problem. Given a tetrahedral mesh of an object, compute elastic deformations that result from forces applied to it by a user, and modify the object to reflect cuts where a user has positioned a virtual cutting instrument. The problem is assumed to come from a simulation of surgical procedures on soft tissue.

The first interactive deformation simulation using a FEM approach was presented by Bro-Nielsen [17]. His prototype uses linear elasticity on a tetrahedral mesh. The elasticity equations yield a linear system of equations. The simulation is static: for a given load, the simulation displays the corresponding equilibrium solution directly. This is achieved by precomputations: internal nodes of the mesh are normally not visible to the user, so these can be eliminated from the equations. This is an expensive preprocessing step that drastically reduces the size of the system. The smaller matrix is then inverted, so displacements for a given load on the boundary can be computed efficiently.

James and Pai [53, 54] have also shown an interactive linear elasticity simulation. Like Bro-Nielsen, they do this by reducing the problem to the boundary. Instead of removing internal nodes after a discretization, the partial differential equation is transformed to an integral equation on the boundary, before the discretization. Discretizing the integral equations yields a linear system whose inverse can be calculated off-line. During a simulation, the response to changing force can be computed within a guaranteed time span. Parts of the boundary may be fixed or loosened during a simulation, and such changes are handled by updating the inverse matrix.

In contrast to static methods, dynamic methods give time a physical meaning. Objects can have inertia and damping. Given the deformation of an object at some time t, the system computes the deformation at time $t + \Delta t$. This process is called *time-*

[1] The results in this chapter were presented at the Medical Image Computing and Computer Assisted Intervention (MICCAI) conference 2001 [73] and EuroGraphics 2000 [72].

integration. In an *explicit integration scheme*, the deformation at time $t + \Delta t$ is computed from the elastic forces at time t. This is in contrast with an *implicit scheme*, where the new deformation is given by predicting elastic forces at time $t + \Delta t$. The deformations are determined by solving for the displacements given the predicted forces.

Cotin et al. [26] and Picinbono et al. [78] have used time-integration for both linear and non-linear FEM deformations in the context of endoscopic liver surgery. They combined a static and dynamic solution method. A part of the mesh is simulated dynamically with the method of central differences, an explicit integration scheme discussed in Section 2.5. This process is very efficient when masses and frictions are concentrated in the nodes of the mesh (*lumping* of mass and damping). No precomputed structures are stored, so the mesh can easily be modified on the fly. On these parts of the mesh, simulated surgical procedures may be performed. The responses of the rest of the mesh are precomputed by inverting the corresponding stiffness matrix.

Zhuang and Canny [101] show a deformable object simulation with dynamic integration, where the mass and damping are not lumped. This requires precomputed inverses of linear combinations of the mass and damping matrices.

Székely et al. [92] show a prototype of an endoscopic surgery simulator. This simulation also uses dynamic FEM for the elastic deformations with an explicit integration scheme. By selecting very small time steps, instabilities due to contact forces and large movements are suppressed. Small time steps are expensive, hence they intend to run this simulation on a massively parallel machine.

Deformable objects are sufficient to simulate inspection procedures. If surgical manipulations, such as biopsies, cuts, cauterizations, etc., must also be addressed then the simulator should also include the notion of a *virtual tool*: a simulated surgical tool that is under user control through an input device such as a joystick, mouse or a force feedback device. A simulated tool can interact in two ways with a deformable model. First, it can exert force on the model. Second, it can modify the mesh representing the object.

One of the earliest examples of such a virtual tool is the cauterization tool simulated by Cotin et al. [26] for the liver surgery simulation mentioned previously. This tool is implemented in a dynamic Finite Element simulation with tetrahedral elements. All mesh elements colliding with the virtual cauterization tool are removed. Elements are also removed if their stresses are beyond a certain threshold. This simulates a simple form of laceration.

Interactive cuts on tetrahedral meshes were first implemented by Bielser et al. [12], using a *subdivision method*. In this technique, every tetrahedron that is in contact with the virtual scalpel is subdivided so the surface swept by the scalpel is represented within the mesh. The initial implementation uses a generic split, that replaces a single tetrahedron by 17 tetrahedra. In later improvements, the incised tetrahedron is replaced by a configuration-dependent number: a tetrahedron that is only cut partially, is subdivided differently from a tetrahedron sliced into two pieces.

Ganovelli et al. [46] embed both cuts and lacerations into a multiresolution framework: both operations are performed on a mesh that is stored at multiple resolutions. At the finest level of detail, they are represented using subdivision methods.

In this chapter, we also use FEM for deformation modeling, and combine that with cutting. Cuts modify the mesh, and invalidate precomputed structures such as ma-

trix inverses. Other systems incorporating cuts use dynamic methods with explicit time integration [12, 26, 46]. Such systems have a physical notion of time, and include time-related effects such as friction, damping and inertia. However, if we assume that surgical procedures are executed with controlled movements, then these effects are not necessary for a visually realistic simulation, and we can use a static formulation. Static methods do not have stability problems, and simulate less behavior, suggesting that they might be more efficient. Cuts have been simulated with subdivision methods in previous work. These can represent incisions accurately, but unfortunately, they always increase the number of elements in the mesh. This brings down the performance of iterative relaxation techniques, such as explicit integration. For this reason, we will present a cutting method that does not increase the mesh size.

3.1 Linear FEM

We will assume the following deformation problem. The deformable object is given as a tetrahedral mesh with nodes 1 to m. At each moment in the simulation, external forces are given for nodes of the mesh. Such forces might include gravity but also those produced by user-controlled instruments. Some nodes have a fixed position. The deformation problem is to compute the displacements resulting from the forces applied. The deformations are governed by linear elasticity: material reacts linearly to stresses, and stresses are linear in the displacements.

We recall Equations (2.22). In linear elasticity, the gradient of the displacement function

$$\mathbf{G} = \frac{\partial \mathbf{u}}{\partial z}, \tag{3.1}$$

is $\mathcal{O}(\varepsilon)$ for some small ε. In this case, the material strain tensor, and first and second Piola-Kirchoff are also $\mathcal{O}(\varepsilon)$. The stress-tensor $\mathbf{S}$ and $\mathbf{T}$ are equal when higher order terms are neglected. They are given by

$$\mathbf{T} = \mathbf{S} = 2\mu\mathcal{E} + \lambda \operatorname{trace}(\mathcal{E})\mathbf{I}. \tag{3.2}$$

Here, $\mathcal{E}$ is the linearized material strain tensor, given by

$$\mathcal{E} = \frac{1}{2}\left(\mathbf{G}^* + \mathbf{G}\right). \tag{3.3}$$

In a FEM approximation with linear tetrahedra the gradient of the displacement is constant across a tetrahedron. For a tetrahedron τ, it can be computed by

$$\mathbf{G} = \mathbf{U} \cdot \mathbf{Z}^{-1}.$$

The tensor $\mathbf{Z}$ transforms coordinates from a unit tetrahedron to τ. If nodes 1 to 4 have locations $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_4$, then $\mathbf{Z}$ is defined by

$$\mathbf{Z} = (\boldsymbol{z}_1 - \boldsymbol{z}_4) \otimes \boldsymbol{e}_1 + (\boldsymbol{z}_2 - \boldsymbol{z}_4) \otimes \boldsymbol{e}_2 + (\boldsymbol{z}_3 - \boldsymbol{z}_4) \otimes \boldsymbol{e}_3,$$

and $\mathbf{U}$ is defined analogously to $\mathbf{Z}$, i.e.

$$\mathbf{U} = (\mathbf{u}_1 - \mathbf{u}_4) \otimes \boldsymbol{e}_1 + (\mathbf{u}_2 - \mathbf{u}_4) \otimes \boldsymbol{e}_2 + (\mathbf{u}_3 - \mathbf{u}_4) \otimes \boldsymbol{e}_3.$$

We recall from (2.38) that the elastic forces $\mathbf{f}^{\text{el}}$ on nodes 1 to 4 of a tetrahedron τ are given by

$$\begin{aligned} \mathbf{f}^{\text{el}}_{j,\tau} &= -\nu(\tau)(\mathbf{T}\cdot\mathbf{Z}^{-*})\cdot\mathbf{e}_j, \qquad j=1,2,3, \\ \mathbf{f}^{\text{el}}_{4,\tau} &= -\sum_{j=1}^{3}\mathbf{f}_{j,\tau}. \end{aligned} \tag{3.4}$$

Here, $\mathbf{Z}^{-*}$ is the transpose of the inverse of $\mathbf{Z}$.

For a static problem, elastic forces should balance body forces and surface tractions, which are both condensed into external nodal forces $\mathbf{f}^{\text{ex}}$. For every node $i=1,\dots,m$, we should have

$$\sum_{\tau}\mathbf{f}^{\text{el}}_{i,\tau}(\mathbf{u})+\mathbf{f}^{\text{ex}}_i=\mathbf{0}. \tag{3.5}$$

Since $\mathbf{f}^{\text{el}}$ is linear in $\mathbf{u}$, this can be condensed in a linear system by setting

$$Ku=f, \tag{3.6}$$

where all elastic force relations are represented in the $\mathbb{R}^{3m\times 3m}$ matrix K, called *primitive stiffness matrix*, and all displacements and forces in the $3m$ vectors u and f. The quantity $r=f-Ku$ is called the residual force.

Equation (3.5) determines displacements up to a constant translation and rotation. Additional conditions are necessary to ensure that a unique solution exists. In the discussion preceding Equations (2.25), we noted that $\mathbf{u}$ should be set on a part of the boundary with non-zero area. In this chapter, we will assume that each connected component of the mesh is fixed in at least three non-collinear nodes. In other words, we assume that there is some set of nodes N_{fix} that have a fixed displacement, say

$$\mathbf{u}_j=\bar{\mathbf{u}}_j \qquad \text{for } j\in N_{\text{fix}}. \tag{3.7}$$

This introduces c constraints on all displacements, where $c=3|N_{\text{fix}}|$. The constrained variables from Condition (3.7) may be eliminated from Equation (3.5), yielding a reduced linear system

$$\tilde{K}\tilde{u}=\tilde{f}.$$

We call $\tilde{K}\in\mathbb{R}^{(3m-c)\times(3m-c)}$ the reduced stiffness matrix, $\tilde{u}\in\mathbb{R}^{(3m-c)}$ the reduced displacement vector, and $\tilde{f}\in\mathbb{R}^{(3m-c)}$ is the reduced force or load vector.

The standard technique to solve a FEM problem is to assemble the matrix $\tilde{K}$, and then solve the system using matrix algorithms. In an interactive simulation, the mesh and boundary conditions can change during a simulation, so neither K nor $\tilde{K}$ are constant. We avoid the hassle of updating K to reflect such changes by using a *matrix-free method*. According to Equation (3.5), elastic forces in a node only depend on displacements of neighbor nodes. For a given displacement d, elastic forces for each element are computed, and then summed into an elastic force vector. In effect, the product Kd can be formed without forming K explicitly. This is sufficient to run algorithms such as the Conjugate Gradient algorithm, which is discussed in Subsection 2.4.1. Matrix-free CG was first proposed by Daniel [30] and Kaniel [56]. It is suited for solving very large problems on parallel machines [24].

In this method there is no need to reduce K to $\tilde{K}$. Boundary conditions that fix nodes are handled by maintaining two vector variables for the residual: the elastic force is computed as $r' = -Ku$. The constrained residual r_k, which is used for running the CG loop, is found by zeroing the entries of $r' + f$ that correspond with a fixed node. Since d_k is a linear combination of r_k and d_{k-1} the displacements of these nodes are not changed. In effect, this procedure solves Equation (3.6) on a $(3m - c)$-dimensional affine subspace of $\mathbb{R}^{3m}$. For notational convenience, we will not distinguish between K restricted to this subspace, and matrix $\tilde{K}$ with reduced size. Since fixed nodes are in equilibrium, we can view the constraints as applying forces that exactly balance the elastic forces in the corresponding entries of r'.

3.2 Cuts

In this section we present a method for making cuts in a deformable object; in other words, we shall change the mesh to produce cuts where a user positioned a virtual cutting instrument. It is assumed that every movement of the cutting instrument is represented as a triangle in a surface, and processing the cut involves processing each of those triangles in the order in which they are generated.

More formally spoken, the assumptions for making cuts are as follows. An object is given, represented by a tetrahedral mesh. The mesh is part of a deformation simulation, and the shape of the mesh is available both in reference and deformed configuration. The user controls a virtual scalpel in the shape of a line-segment. A movement of that scalpel sweeps a surface called *scalpel sweep*. We assume that that surface is not strongly curved, so it can be approximated with a triangulation: the scalpel sweep is assumed to be already converted to sequence of connected triangles $\Delta_1, \ldots, \Delta_k$ in a preprocessing step. We assume that earlier parts of the movement are earlier in the sequence. The triangles $\Delta_1, \ldots, \Delta_k$ are called *sweep triangles*. The objective is to approximate the scalpel sweep by faces of the mesh. Since the sweep triangles are ordered in time, we can process an entire movement by processing the sequence one triangle at a time. More formally, given the mesh and the incision resulting from $\{\Delta_1, \ldots, \Delta_{j-1}\}$, modify the mesh such that Δ_j is also approximated by an incision in the deformed configuration of the mesh. This incision should be connected to the existing incision.

In order to the keep mesh size constant, we will perform cuts along existing faces of the mesh. This increases the number of nodes, but keeps the number of elements constant. Cutting along faces involves four separate actions executed in sequence:

1. Faces that will be cut are selected. They form the *cut surface*.

2. Vertices of these faces are moved such that the cut surface approximates the surface $\{\Delta_1, \ldots, \Delta_j\}$.

3. The connectivity of the mesh is changed to reflect the cut. This procedure is called *dissection*, and it is further discussed in Chapter 7.

4. Finally, degenerate elements, created as artefacts of the cutting process are removed from the mesh.

This technique is a variant of a superposition method [96], which has been used before in offline mesh generation. A regular starting mesh (in our case, the mesh originally given) is superimposed onto the boundary of the desired object (in our case, the sweep surface). The grid is then adapted to include the boundary. In our case, we project mesh nodes onto the sweep surface, and remove degeneracies. Projecting mesh nodes onto boundaries has been proposed earlier grids [52,93], but was used for meshing hexahedral off-line. In our case, we use the technique for on-line tetrahedral mesh modification.

3.2.1 Selecting faces

We use the following approach for selecting faces. The sweep triangle is intersected with all edges of the mesh. For every edge intersected, we mark the node that is closest to the intersection point. For a tetrahedron, we can consider the nodes marked from its edges. These marked nodes form a subset of the tetrahedron, and define a node, edge, face, or the entire tetrahedron. If the set of closest nodes contains three elements, then we select the corresponding face for performing a cut. This approach is demonstrated in Figure 3.1. When processing connected sweep triangles, the sweep/edge intersections from different sweep triangles are combined when selecting a surface.

The case that all four nodes are selected does not occur often when the scalpel sweep is a plane; this can only happen if all edges are intersected exactly halfway. To show this, let the supporting plane of the sweep triangle be given by $\{\mathbf{x} \in \mathbb{R}^3 : \mathbf{x} \cdot \mathbf{n} = \alpha\}$ for some $\mathbf{n} \in \mathbb{R}^3$ and $\alpha \in \mathbb{R}$. If this plane intersects an edge $\mathbf{ab}$, selecting $\mathbf{a}$, then there is a $\lambda \geq \frac{1}{2}$ such that

$$\alpha = \mathbf{n} \cdot (\lambda \mathbf{a} + (1-\lambda)\mathbf{b}).$$

We have

$$\begin{aligned} |\alpha - \mathbf{a} \cdot \mathbf{n}| &= |\mathbf{n} \cdot (\mathbf{b} - \mathbf{a})(1-\lambda)| \\ &\leq |\lambda(\mathbf{b} - \mathbf{a}) \cdot \mathbf{n}| \\ &= |\mathbf{b} \cdot \mathbf{n} - \alpha| \end{aligned}$$

If four nodes from a tetrahedron are selected by a sweep plane, then the plane intersects four edges of the tetrahedron, and we can always label the nodes of the tetrahedron with $\mathbf{a}, \mathbf{b}, \mathbf{p}$ and $\mathbf{q}$, such that

$$\begin{aligned} |\alpha - \mathbf{a} \cdot \mathbf{n}| &\leq |\mathbf{p} \cdot \mathbf{n} - \alpha|, \\ |\alpha - \mathbf{p} \cdot \mathbf{n}| &\leq |\mathbf{b} \cdot \mathbf{n} - \alpha|, \\ |\alpha - \mathbf{b} \cdot \mathbf{n}| &\leq |\mathbf{q} \cdot \mathbf{n} - \alpha|, \\ |\alpha - \mathbf{q} \cdot \mathbf{n}| &\leq |\mathbf{a} \cdot \mathbf{n} - \alpha|. \end{aligned}$$

In other words, all distances are equal, and the edges $\mathbf{ap}$, $\mathbf{aq}$, $\mathbf{bp}$, and $\mathbf{bq}$ are intersected precisely halfway.

The resulting cut surface will be connected, since adjoining tetrahedra share their edges and hence their sweep/edge intersections. If a sweep halves an edge, then a consistent choice is made by selecting the node with the lowest index. If two consecutive

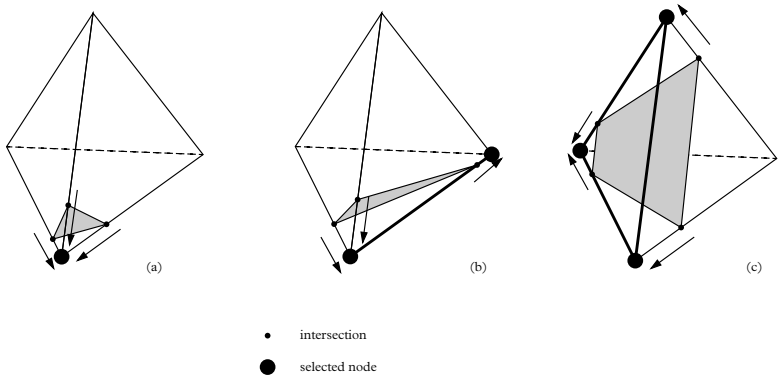

Figure 3.1: Surface selection by closest nodes from edge/sweep intersections. The sweep is colored grey, and the selected feature is marked by bold lines and dots. In Picture (a), (b) and (c) a node, an edge and a face are selected respectively.

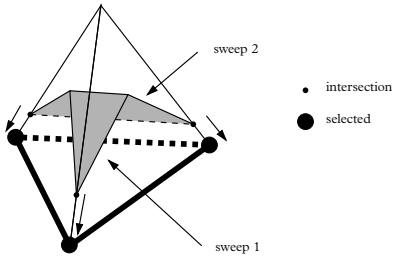

Figure 3.2: When processing multiple sweep triangles, intersections from different sweeps are combined.

sweep triangles have different orientations, it is possible that the sweep intersects edges twice, and all nodes of a tetrahedron would be selected. This is prevented artificially by only considering the first edge/sweep intersection. The cut surface does not necessarily have the same topology as the scalpel sweep. It is possible that the selection process leads to a branching cut surface, as is demonstrated in Figure 3.3

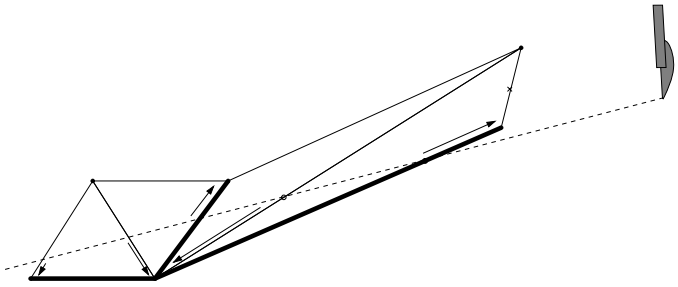

Figure 3.3: Cut surfaces may branch, shown in 2D. Selected faces (edges) are shown bold.

A face can only be selected when three edges of an incident tetrahedron are intersected by the scalpel sweep. This typically happens when the scalpel has already left the tetrahedron. This means there will be a lag between the faces selected from the mesh and the position of the scalpel. This is also demonstrated in Figure 3.4.

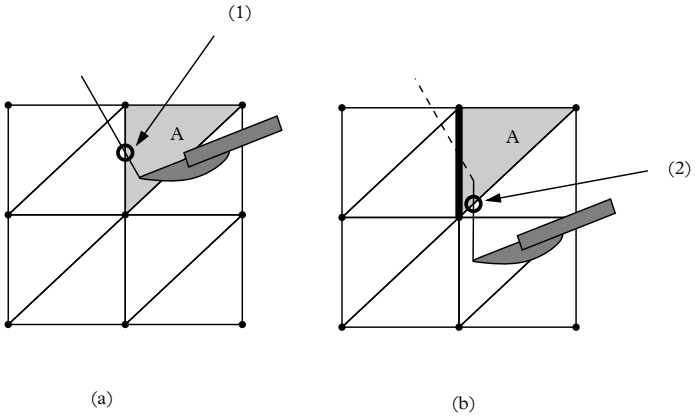

Figure 3.4: Selected faces (bold edges) lag behind the scalpel sweep, demonstrated in 2D. The bold edge in (b) is selected once intersection (2) has been found. This happens after the scalpel has left triangle A. Between steps (a) and (b), intersection (1) must be remembered.

The sweep triangles are processed one by one, so special precautions are needed to ensure that two connected triangles Δ_i and Δ_j will produce a connected cut in the mesh. To this end, the surface selector remembers tetrahedra with incomplete cuts: these are

tetrahedra that contain the boundary of $\{\Delta_1, \ldots, \Delta_{j-1}\}$. Further sweep triangles may extend the cut, thus finishing 'incomplete' tetrahedra, as shown in Figure 3.1.

The output of surface selection is a set of faces of the mesh. When the mesh is dissected along these faces, this produces a crude form of cutting, where the incisions are jagged. A sample from our implementation is shown in Figure 3.5.

3.2.2 Node snapping

A smooth incision in the object is created by repositioning nodes to be on the sweep surface in the mesh. This is done before dissecting the cut. The deformed location an internal node on the cut surface is projected orthogonally onto the plane of triangle Δ_j, yielding a point $\boldsymbol{w}$. If v is on the boundary of the mesh, our choice is more restricted, since moving boundary nodes changes the shape of the surface. For surface nodes, we use an approach that does not change the shape of a flat surface. For each boundary face f incident with v, the location of v is projected orthogonally on the intersection of Δ_j and f, yielding a point $\boldsymbol{w}_f$. If $\boldsymbol{w}_f$ lies in f, then that point is considered for the new location. The $\boldsymbol{w}_f$ closest to the original position of v is picked. This approach is illustrated in Figure 3.6.

Nodes must be repositioned in the reference configuration of the mesh, while the scalpel surface interacts with the deformed mesh. To translate between these configurations, we find the tetrahedron incident with v that contains $\boldsymbol{w}$, and use its strain to transform $\boldsymbol{w}$ back to the reference configuration. If no such tetrahedron is found, then repositioning node v fails.

3.2.3 Degeneracies

The previous subsection has introduced a recipe for relocating nodes of the mesh to generate smooth incisions in deformable material. However, changing node positions alters the shape of mesh elements, a process which affects the deformation computations. In particular, projecting nodes can lead to flat elements if all nodes of an element are selected. The element will be flattened by the projection, and adjacent tetrahedra may be inverted. Examples are shown in Figure 3.7. Flat elements lead to unbounded condition numbers and unbounded discretization errors [87]. This significantly affects the speed and the accuracy of the simulation, so these elements have to be removed before deformation computations can continue.

In this section we present a heuristic approach that collapses flat elements and thus removes them. This is done in multiple passes. All degeneracies in the mesh are collected in a list, and every element on the list is subjected to the collapse procedure described below. After this pass, a new list of degeneracies is made. If this list is shorter than the old one a new pass is made. After the last pass, the remaining degeneracies are considered incollapsible, and they are removed by cutting them free, and removing them. These extra cuts can cause spurious incisions ("cracks") perpendicular to original incision.

Flatness of elements is quantified by their aspect ratio. We define the aspect ratio of a tetrahedron by the minimum height divided by the maximum edge length. If the

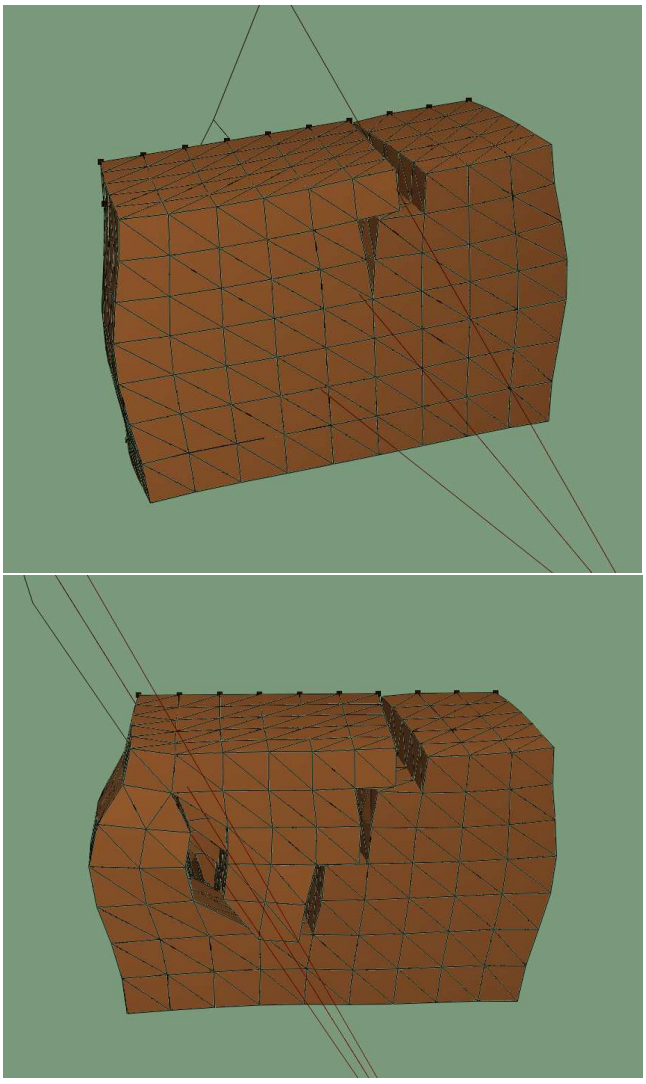

Figure 3.5: A cut in a generated object cube without node repositioning. The last and next sweep triangle are indicated by the two triangles partially penetrating the object.

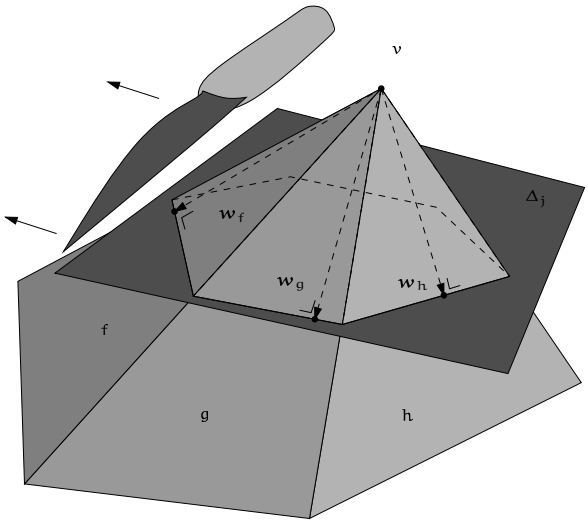

Figure 3.6: When repositioning a boundary node v, it can be projected within faces f, g and h, leading to different points $\boldsymbol{w}_f$, $\boldsymbol{w}_g$ and $\boldsymbol{w}_h$. The closest $\boldsymbol{w}$ is selected.

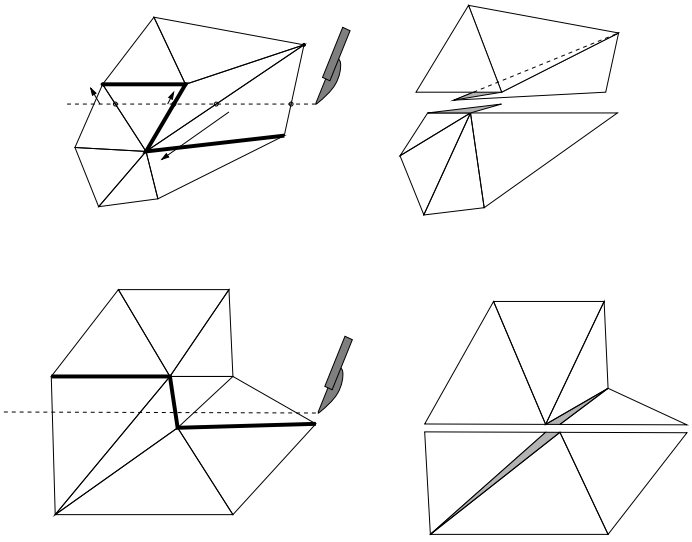

Figure 3.7: Projecting nodes may result in degeneracies, even in 2D. If all nodes of an element (top) are selected, this leads to degenerate and inverted elements. Selecting a face almost perpendicular to the scalpel sweep also leads to degenerate elements

tetrahedron is represented by its set of nodes τ, and the supporting plane of a triangle σ by $\text{plane}(\sigma)$ then

$$\frac{\min_{p\in\tau} d(p, \text{plane}(\tau\backslash\{p\}))}{\max_{p,q\in\tau} d(p,q)}.$$

Flat elements come in different shapes, and they can be classified by their angles [10]. The strategy to collapse an element is determined by its shape, as measured by the number of large dihedral angles it contains.

- In elements without large angles, so-called *needles*, the shortest edge is collapsed.
- In elements with a single large angle, so-called *spindles*, the edge opposite the large angle is split. Then the shortest edge is collapsed.
- In elements with two large angles, so-called *slivers*, both edges containing the large angle are subdivided by introducing a new node. The resulting short edge is collapsed.
- In elements with three large angles, so-called *caps*, a new node is introduced in the face opposite the three angles. The resulting short edge is collapsed.

The shape classification and removal approach are demonstrated in Figure 3.8 and 3.9.

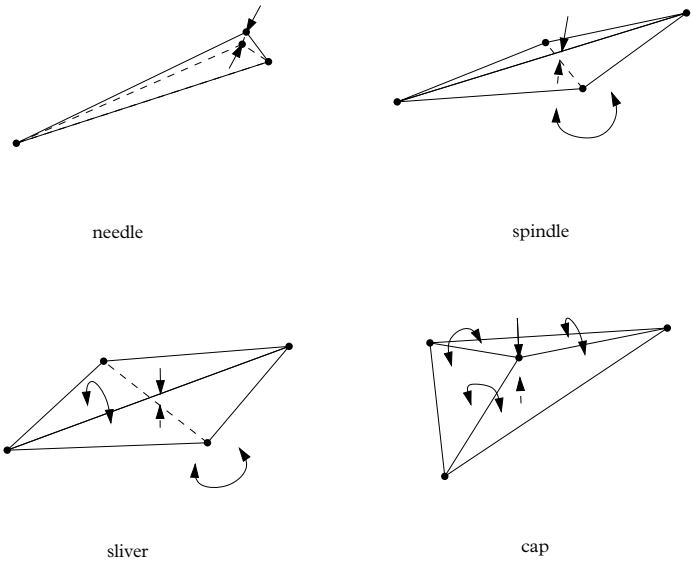

Figure 3.8: Degenerate tetrahedra either have flat triangles or short edges, or they have large dihedral angles.

Edge collapses are substitutions on the simplexes of the mesh. They change the mesh connectivity, and when either node is on the boundary, they may also change the topological type of the object [35]. In a mesh of a physical three-dimensional object,

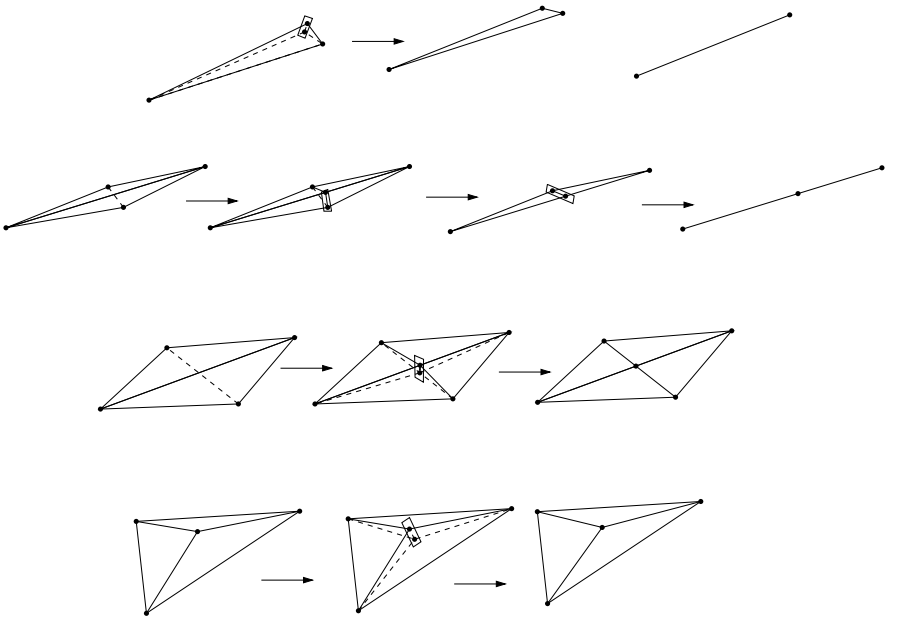

Figure 3.9: Collapsing elements by introducing short edges and collapsing them.

all (oriented) triangles should be in exactly one oriented tetrahedron. The result of an edge collapse does not always satisfy this requirement, and in this case, the edge collapse fails. An example is given in Figure 3.10.

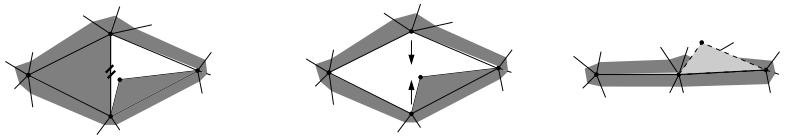

Figure 3.10: An impossible edge collapse in 2D. The edge marked with ticks is collapsed. This involves the triangles incident with the edge, and merging both nodes of the edge. In this case, the result of a collapse is no longer a manifold, as the edge marked in bold is incident with three triangles.

3.3 Results

The techniques discussed have been implemented in a prototype written in C++ [89] using OpenGL [88] for visualization.

The FEM implementation was validated using a cantilever beam test problem (See Figure 3.11). A beam is fixed on one end, and loaded with a force F on the other end. Assuming small deformations, the analytical solution for the deflection y of the loose

end is given by [80]

$$y = \frac{4F}{Ed}\left(\frac{l}{h}\right)^3.$$

Here, l, d and h are the dimensions of the beam, E is the Young modulus of the material, and F the load.

We have run the simulation with $l = 1.2\,\mathrm{m}$, $h = 0.2\,\mathrm{m}$, $d = 0.3\,\mathrm{m}$, $E = 10^6\,\mathrm{Pa}$ and Poisson ratio $\nu = 0$. The total load applied at the loose end is 3N. The analytical solution for the deflection of the tip is $8.64 \cdot 10^{-3}\mathrm{m}$. This is small compared to the dimensions of the body, so both linear elasticity and the analytical beam formulas are applicable.

We have an exact solution for this problem, so we can calculate the error in the tip deflection computed by the FEM simulation. Table 3.1 shows how this error depends on the mesh resolution. Both maximum and average errors for the nodes at the tip diminish as the mesh becomes finer, which validates our FEM implementation. The results were computed with a procedurally generated beam (also shown in Figure 3.11), and stopping criterion $\|r\|_2 < 10^{-8}\|f^{ex}\|$.

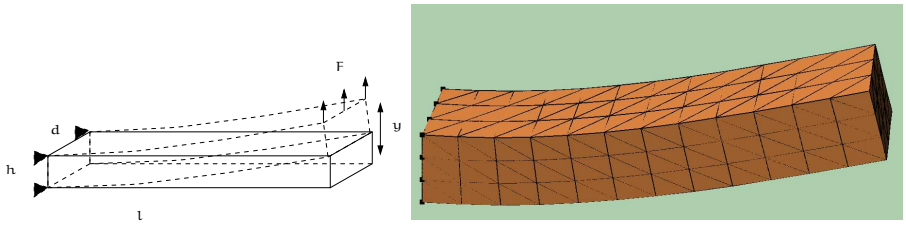

Figure 3.11: The cantilever beam experiment. On the right the result with forces exaggerated by a factor 10.

# elements ($l \times h \times d$)	# nodes	iterations	avg error (%)	max error (%)
$16 \times 4 \times 4$	425	377	29.0	29.3
$24 \times 6 \times 6$	1225	561	15.1	15.3
$32 \times 8 \times 8$	2673	723	8.6	8.8
$40 \times 10 \times 10$	4961	882	5.2	5.3
$48 \times 12 \times 12$	8181	1044	3.2	3.3

Table 3.1: Errors in the deflection of tip nodes of the cantilever beam experiment. The FEM deflections were always less than predicted by the analytical solution.

The visualization loop is interleaved with the relaxation loop, so the course of the CG iteration is visualized during the simulation. It is visible as a quickly damping vibration in the object. In our subjective view, this vibration did not decrease the visual realism of the simulation. At every change in forces or boundary conditions, the CG iteration is restarted.

All further results have been done with the stopping criterion $\|r\| < 10^{-3}\|f^{ex}\|$. The material has Poisson ratio $\nu = 1/4$. On our platform (PIII 1 Ghz), the object shown in Figure 3.12, a procedurally generated object with 1728 nodes and 7986 elements could be smoothly manipulated: the relaxation runs at approximately 55 iterations per second. For this test object between 100 and 300 iterations are needed to reach the solution.

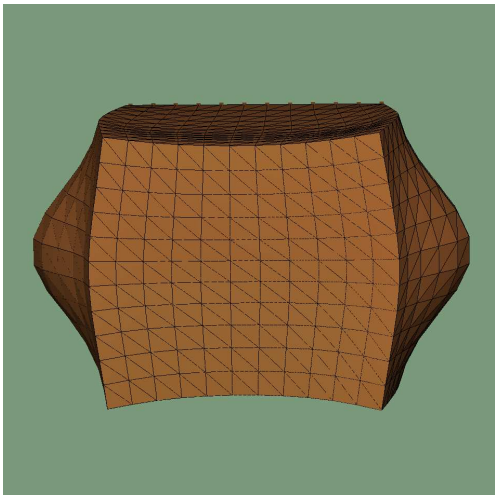

Figure 3.12: A generated cube of 7986 elements, with dilating forces applied to the side.

The cutting simulation places a triangle representing Δ_j under control of the user: its orientation and size can be controlled with the mouse, and a keypress advances the triangle, creating a cut. When processing a sweep triangle Δ_j, the entire boundary of the mesh is tested for collisions. Starting from these collisions and from incomplete tetrahedron intersections, neighboring tetrahedra are tested recursively for collisions.

The degenerate case, where all edges are exactly halved by the sweep triangle, is handled by selecting a random face from the tetrahedron. In practice, we did not observe many of such cases. In a static deformation problem each body must be fixed. Since a cut may split the mesh in different components, the mesh is checked for its connectivity after every cut, and all unfixed components are fixed on an arbitrary face.

Without moving nodes, the cutting results in jagged incisions, as demonstrated in Figure 3.5. By moving nodes, the cutting proces produces smooth incisions. It also introduces various degeneracies, which effectively halt the relaxation loop. Figure 3.14 and 3.15 shows a large cut (108 faces dissected) in a generated cube (3072 elements, 729 nodes), which produces 143 degenerate elements. 2439 iterations are needed to compute the deformation caused by the cut.

The effect of the degeneracy removal process is demonstrated in Figure 3.16. Here the same cut is made, but flat elements are collapsed before resuming the deformation computations. In this example, an element is considered degenerate if its aspect ratio

is less than 0.01 and a dihedral angle is considered large if it exceeds 0.99π. With these thresholds, 99 edges are collapsed, and 13 incollapsible elements are cut free. The resulting mesh contains 3034 elements and 823 nodes, showing that the removal process does not increase mesh size significantly. The deformation requires 236 iterations to converge, a significant reduction. No effort has been spent to optimize the process of degeneracy removal, and in this particular case, the process takes approximately 0.7 seconds on our platform (Pentium III, 1Ghz).

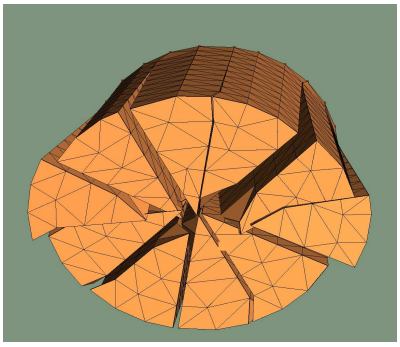

Figure 3.13: Pie-like cuts in a Delaunay Tetrahedralization of a cylindrical point cloud.

In these generated cubes the frequency of degeneracies strongly correlates with the angle of the cut. This is not surprising, since this mesh only contains six different orientations of tetrahedra. A Delaunay tetrahedralization of a cylindrical point cloud does not have this bias, so for pie-like cuts like those depicted in Figure 3.13, degeneracy counts do not depend on the orientation of the sweep triangle. After applying three large pie-like cuts (45 faces) to such an object (4020 elements and 891 vertices), we find that the number of edge collapses is approximately 5% of the number of the faces dissected (873 faces). A handful of incollapsible degeneracies are encountered.

The simulation does not model a scalpel, let alone physical interactions between the deformable object and a scalpel. As a result, when the body moves by a large amount relative to the next sweep triangle to be processed, due to either external forces or because of deformations caused by a previous cut, the next incision is no longer connected to the previous incision. A similar effect happens when the degeneracy removal procedure changes elements of incomplete cuts: when edges are collapsed, the associated sweep/edge intersections are invalidated as well. Some of these intersections are needed to connect the existing incision to the next cut, hence the degeneracy removal may lead to disconnected incisions. An example of this phenomenon is shown in Figure 3.17. This effect might be mitigated by repeating the cut for the last sweep triangle.

3.4 Discussion

In this chapter we have presented an interactive FEM deformation, using a CG iteration in a matrix-free implementation as a static relaxation algorithm. A matrix-free approach

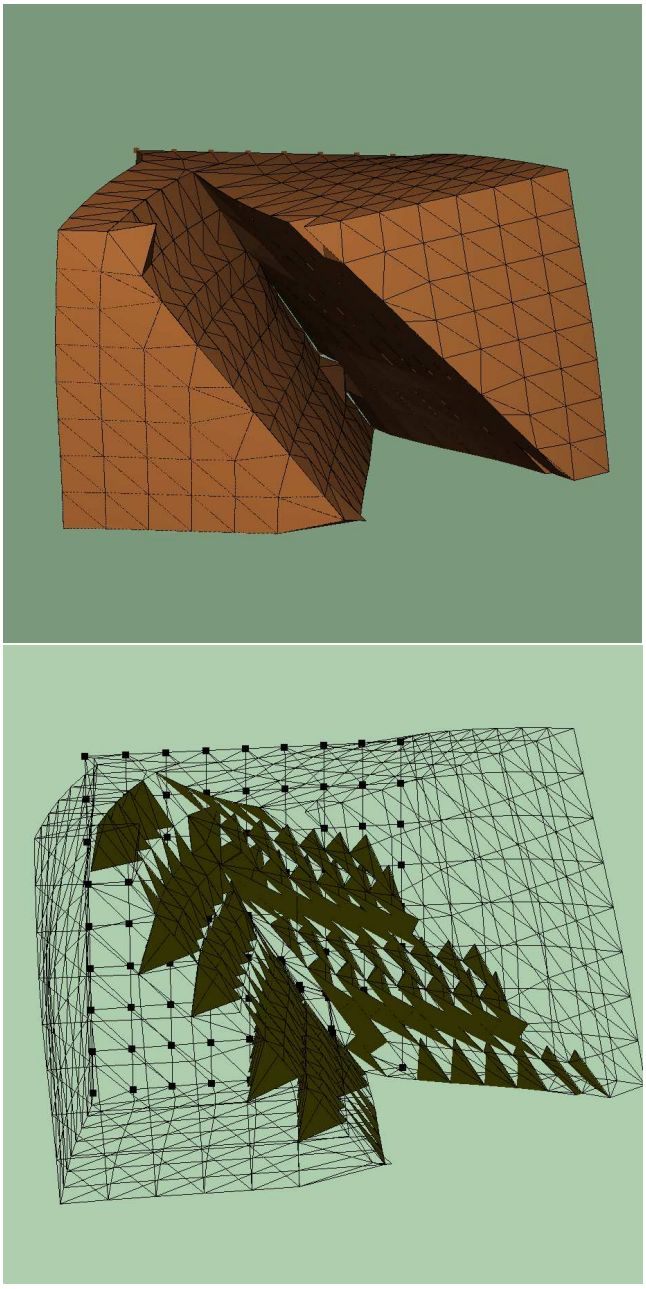

Figure 3.14: A large diagonal cut in a generated cube with back face fixed and dilating forces applied to the side faces. This cut generated 143 degeneracies. On the bottom the mesh is shown in wireframe mode, with only the degenerate tetrahedra filled in. Repositioning failed for two nodes, which are visible as spikes in the incision surface.

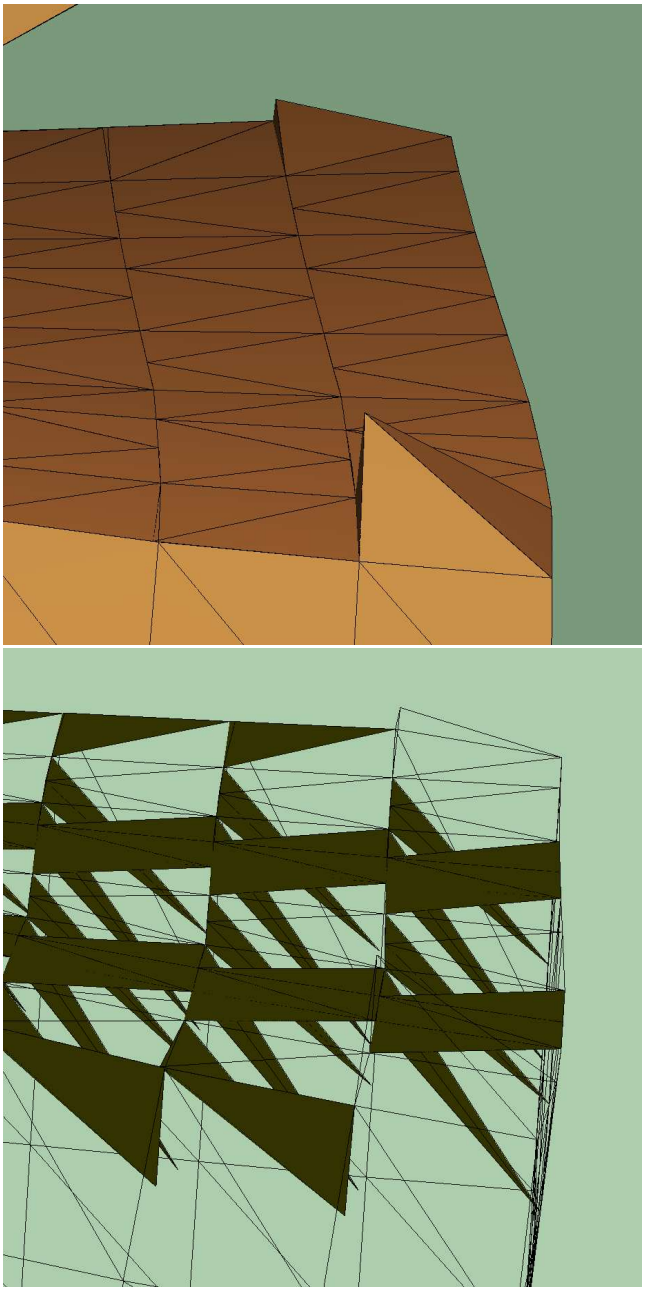

Figure 3.15: The same cut as Figure 3.14 from a different point of view.

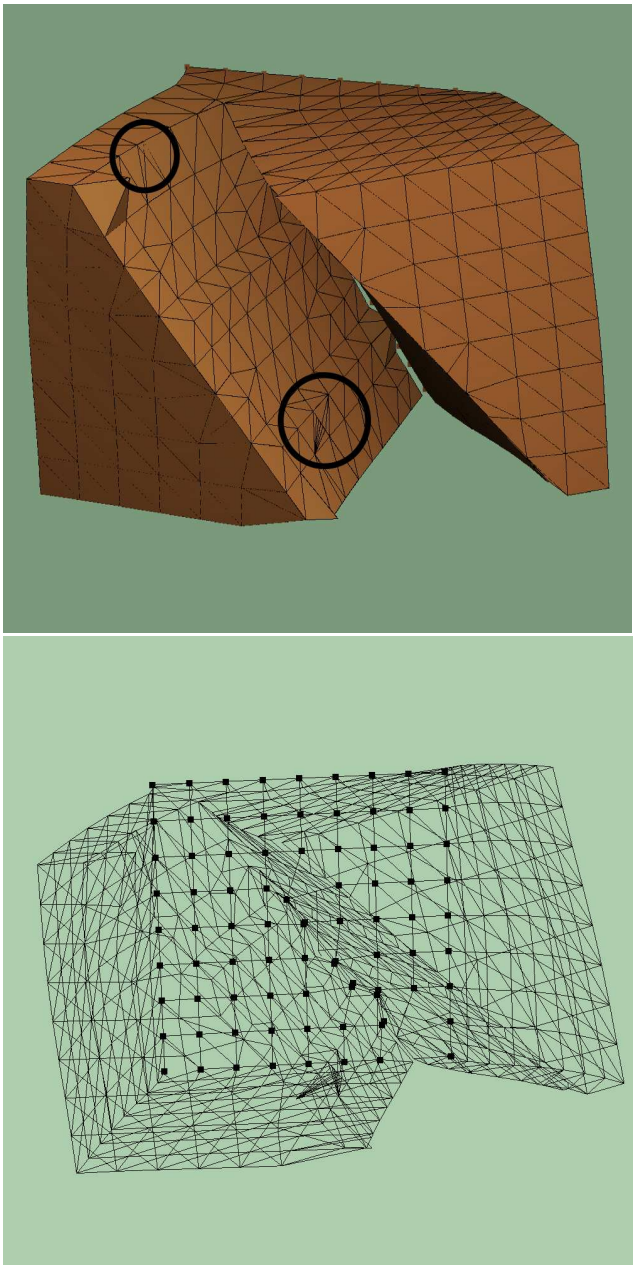

Figure 3.16: The same cut as Figure 3.14, but with degeneracies removed. Two artefacts of the removal process are circled. At the top we see an inverted element. At the bottom, an incollapsible degeneracy has lead to spurious subdivision before it was dissected. The incollapsible degeneracy has also caused a "crack" in the incision, also visible in the wireframe at the bottom.

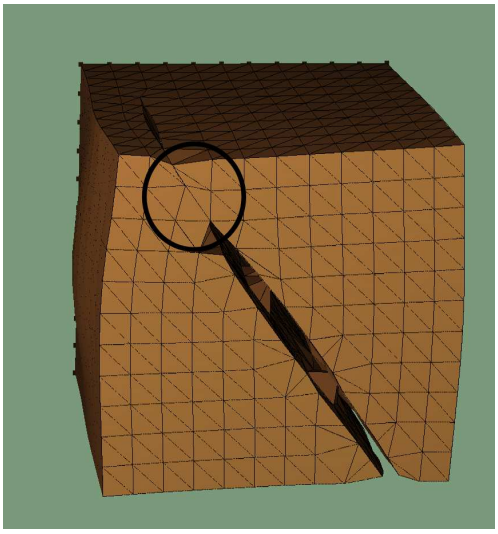

Figure 3.17: The same cut as Figure 3.14, but made in several small steps. Notice how both sides of the cut remain attached.

uses minimal storage, at the cost of a relatively small computational overhead. The deformation simulation seems satisfactory for visual inspection, and it does not have stability problems. This deformation model is the simplest that can be realized in 3D, so we should investigate to what extent this technique can be applied in a more general setting. Specifically, the following questions are open.

- To what degree can we consider this method interactive or real-time?
- How do static methods and dynamic methods compare, assuming that dynamic effects are not strictly necessary?
- Can the approach be extended to nonlinear models?

These questions will be explored in more detail in Chapter 4 and Chapter 6.

We have also presented a cutting method that moves existing mesh faces and performs cuts along these faces. The purpose of this method is to produce cuts without increasing mesh size. We have succeeded in that goal. Unfortunately, this cutting technique has a number of disadvantages as well. We do not place any restrictions on the starting mesh, so the projection technique produces more or less inverted elements and arbitrary, degenerate element shapes. The inverted elements might be prevented by simply forbidding node relocations in those cases. The degeneracies pose a more serious problem. Because of their arbitrary nature, we are forced to use heuristic methods to remove. In their current form, these heuristics may change the topology of the object. They seem to work for most degeneracies that occur, but not for all cases, and the technique does not address inverted elements. Although heuristics based on local topological reconfiguration are accepted techniques for offline meshing [43, 55], it is

not clear whether they can be applied to on-line remeshing, especially when there are stringent real-time constraints. We performed limited experiments that indicate that more work is needed to make this scheme practically useful.

The cutting module handles sweep triangles one by one, and remembers sweep/edge intersections of previous sweep triangles to produce connected incisions. This implies that it retains references to large parts of the mesh. Each mesh change, due to either dissection or degeneracy removal, must also be applied to references in the cutting module. This double administration is wasteful and error prone. Moreover, degeneracy removal can cause disconnected cuts to appear.

These last problems are symptoms of a fundamental limitation of our cutting approach. The underlying assumption is that the scalpel sweep may be represented by a triangulated surface $\{\Delta_1, \ldots, \Delta_k\}$, and that we can reduce the cutting problem to processing single triangles Δ_j incrementally. First, this ignores physical interactions between material and the scalpel. Second, the speed of computing hardware limits the resolution of the mesh. Assuming that we have uniform meshes, tetrahedra are likely to be much larger than the scalpel movements to be represented. Since a mesh face is scheduled for dissection only when the sweep has sliced through an incident element, the incision effected in the mesh will always lag with the cut made by the user. This might be ameliorated by moving mesh nodes to be on the boundary of the scalpel sweep. However, given the problems that we encountered with moving mesh nodes onto the plane of the scalpel sweep, this is likely to be fraught with even more degeneracy problems than the current approach.

In summary, the approach to cutting that we have presented satisfies our initial desire for small mesh sizes, but does not match the intended application well. It is clear that the cutting problem should be reconsidered completely. A start of this will be made in Chapter 5.