Cutting in deformable objects

Snijden in deformeerbare objecten

(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de Rector Magnificus, Prof. dr. W.H. Gispen, ingevolge het besluit van het College voor Promoties in het openbaar te verdedigen op woensdag 18 juni 2003 des ochtends te 10.30 uur

door

Han-Wen Nienhuys

geboren op 15 januari 1975, te Eindhoven

Promotor: Prof. dr. Mark H. Overmars
Copromotor: dr. ir. A. Frank van der Stappen
Faculteit Wiskunde en Informatica
Universiteit Utrecht

Contents

1 Introduction 8

1.1 Training surgeons 8

1.2 Interactive surgery simulation 9

1.3 Overview of thesis 11

2 Preliminaries 15

2.1 Continuum mechanics 16

2.2 Finite Element Method 22

2.3 Linear tetrahedra for hyperelastic materials 26

2.4 Unconstrained optimization 28

2.4.1 Linear Conjugate Gradient method 29

2.4.2 Non-linear conjugate gradients 32

2.4.3 Truncated Newton 32

2.5 Time integration 33

2.6 Mechanics of soft tissue 36

3 Combining FEM and cutting: a first approach 38

3.1 Linear FEM 40

3.2 Cuts 42

3.2.1 Selecting faces 43

3.2.2 Node snapping 46

3.2.3 Degeneracies 46

3.3 Results 50

3.4 Discussion 53

4 Relaxation algorithms 59

4.1 Convergence of dynamic relaxation 59

4.2 Experimental setup 63

4.3 Hyperelastic compressible materials 65

4.3.1 St. Venant-Kirchoff elasticity 67

4.3.2 Linear geometry approximation 67

4.3.3 Neo-Hookean elasticity 68

4.3.4 Veronda-Westmann elasticity 69

4.4 Relaxation algorithms 70

4.4.1 Dynamic parameters 70
4.4.2 Line search 72
4.5 Results 73
4.5.1 Tuning CG 73
4.5.2 Performance 74
4.5.3 Influence of mesh size 74
4.5.4 Linear elasticity 77
4.5.5 Nonlinear material models 77
4.5.6 Influence of mesh quality 79
4.6 Discussion 79
4.7 Conclusion 83

5 A Delaunay approach to interactive cutting in triangulated surfaces 85
5.1 Introduction 85
5.2 Cutting in 2D 87
5.2.1 Delaunay triangulation 88
5.2.2 Cutting and Delaunay flips 89
5.3 Surface cuts in 3D 91
5.3.1 Single incision 92
5.3.2 Multiple incisions 95
5.4 Results 98
5.5 Discussion 98

6 2D needle insertion 104
6.1 Related work 104
6.2 Needle insertion in 3D 105
6.3 Physical model 106
6.3.1 Elasticity 106
6.3.2 Stick-slip friction 108
6.4 Needle representation 109
6.5 Edge bisection 110
6.6 Solution method 114
6.7 Implementation 115
6.8 Computational experiments 115
6.8.1 Test situation 116
6.8.2 Residual tolerance 117
6.8.3 Mesh density 117
6.8.4 Insertion speed 119
6.8.5 Timings 119
6.8.6 Relocation errors 120
6.8.7 Nonlinearity effects 120
6.9 Discussion 122

7 Mesh representation **128**
7.1 Abstract oriented simplexes 129
7.2 Representing the mesh 131
7.3 Changing the mesh 135
7.4 Interfacing with mesh connectivity 142
7.5 Discussion 142

8 Conclusions **144**

A Tensor calculus **155**
A.1 Tensors 155
A.2 Tensor calculus 157

Index of notation

notation	name	where defined
$\mathbf{a}, \mathbf{b}, \ldots$	1-tensors (“vectors”)	Section A.1
$\mathbf{a} \cdot \mathbf{b}$	euclidian inner product on $\mathbb{R}^3$	Section A.1
$\mathbf{a} \otimes \mathbf{b}$	tensor product of two vectors	(A.2)
$\mathbf{A}, \mathbf{B}, \ldots$	2-tensors (“matrices”)	Section A.1
$\mathbf{I}$	identity 2-tensor	Section A.1
Lin	space of 2-tensors or linear mappings on $\mathbb{R}^3$	Section A.1)
$\operatorname{trace} \mathbf{A}$	Trace of a 2-tensor	(A.3)
$\det \mathbf{A}$	determinant of a 2-tensor	(A.5)
$\iota_1, \iota_2, \iota_3$	3 invariants of a 2-tensor	(A.6)
$\mathbf{A}^*$	transpose or adjoint of a 2-tensor	(A.1)
$\mathbf{A}^{-*}$	inverse of $\mathbf{A}^*$	
$\mathbf{A} : \mathbf{B}$	inner product on Lin	(A.4)
$\mathbf{T}$	first Piola-Kirchoff stress tensor	(2.4)
$\mathbf{S}$	second Piola-Kirchoff stress tensor	(2.4)
$\mathbf{C}$	right Green-Lagrange stress tensor	(2.1)
$\mathbf{E}$	material strain tensor	(2.21)
$\mathcal{E}$	linear approximation to $\mathbf{E}$	(2.22)
$\mathbf{Z}$	tetrahedron shape tensor	(2.32)
ρ	mass density	(2.2)
$\boldsymbol{z}$	coordinates in reference configuration	Section 2.1
$\mathbf{p}(\boldsymbol{z}, t)$	motion	Section 2.1
$\mathbf{u}(\boldsymbol{z}, t)$	displacement	(2.20)
λ	Lamé parameter	(2.17)
μ	Lamé parameter	(2.17)
E	Young’s modulus	(2.18)
ν	Poisson ratio	(2.19)
c	wave speed	
∇f, $\operatorname{grad} f$	gradient of a function f	
d	search direction	Section 2.4.2

r'	discretized elastic forces	Section 2.4.1
r	discretized residual forces	Section 3.1
K	stiffness matrix	(2.29)
C	damping matrix	(2.55)
M	mass matrix	(2.52)
Π	virtual work function	(2.40)
$V(\mathcal{B})$	domain of the partial differential equation	(2.26)
$V_h(\mathcal{B})$	finite-dimensional subset of $V(\mathcal{B})$	(2.27)
W	energy density	(2.14)
$(u, v)_A$	inner product $u^T A v$	(2.44)
$\|u\|_A$	norm associated with A, $\sqrt{(u, u)_A}$	(2.44)
$\mathrm{cond}_2(A)$	2-norm condition number of A	(2.47)
σ, τ	simplexes	Chapter 7
Δ_j	(scalpel) sweep triangles	Section 3.2
$\|\cdot\|_2$	euclidian or L_2 norm	
$\|\cdot\|_\infty$	maximum or supremum norm	

Chapter 1

Introduction

1.1 Training surgeons

Performing surgical operations is traditionally taught in an apprentice/master setting. The surgeon in training watches accomplished surgeons perform an operation, and after sufficient experience, he may perform operations under expert guidance. After much practice, the trainee then becomes an expert in operations.

Although this system of educating surgeons is effective, it has serious drawbacks. Due to lack of experience, trainees take longer to perform procedures, which increases costs. They are also less skilled, which subjects patients to extra risks. This is not a desirable situation, but the alternatives to training within the operating room also have disadvantages. Books and videos describing operations are not interactive. Test animals do not always reflect human anatomy, and their use is expensive. Synthetic phantoms do not reflect mechanic properties of living tissue, and have to be discarded after dissecting them.

These issues are aggravated by the introduction of *laparoscopic* or *endoscopic* surgery. In this technique, the abdomen can be operated on without major trauma. Instruments, such as graspers, scissors, and staplers, are introduced in the body through small holes in the abdomen. These instruments are mounted on long rods, and surgeons can operate them from the outside. The operating site is viewed through a laparoscope, a tube-like device which contains a camera and a lamp. It also inserted through a small incision. Compared to open surgery, the trauma caused by the openings is small. This speeds up patient recovery, and reduces pain and scarring.

For patients, the benefits of laparoscopic surgery are clear. For surgeons however, it opens up a range of new problems. During an intervention the surgeon cannot directly see the operating site, but must rely on a coarse 2D video image of the site. Since the instruments and the camera are introduced through small holes, their movements are restricted. This makes manipulating them awkward. Elastic response of the tissue is relayed through wires to the surgeon, thus reducing kinesthetic feedback. Hence, surgeons have reduced visual and haptic feedback during laparoscopic surgery, and cannot rely on traditional hand-eye coordination. Training a new laparoscopic procedure takes more practice than learning new traditional surgical procedures.

It has been pointed out [8, 9] that computer-assisted training might offer a solution to these problems. If apprentices are initially trained on "virtual" patients, simulated by computers, no costly operating rooms have to be used, and during practice there is space for making errors without consequences. If the simulation is sufficiently advanced, it may be more realistic than animals and phantoms. Students can make errors and learn from them, and they can experiment with different surgical techniques. Moreover, a virtual environment may be used to recreate unusual complications, and train students for situations that occur only sporadically in practice.

A virtual environment is also conducive to experimentation. Virtual environments can be used in experiments to determine skills and techniques that are effective for surgical procedures. This is useful because little is known of the exact nature of surgical skills [94].

1.2 Interactive surgery simulation

In summary, interactive surgery simulations could be a highly useful tool for training surgical procedures. Unfortunately, constructing such simulations is a technically challenging task. Consider such a hypothetical system, consisting of a computer connected with a fancy display and specialized "joystick" that also renders reaction forces (a *force feedback* or *haptic* device) . Such a system lets the student manipulate virtual organs, and feel their reactions. When the student uses the joystick to push, cut or otherwise manipulate simulated organs, the system should respond to these actions with a credible reaction. The time available for computing these reactions is severely limited. Typically, the display must be updated within 40 milliseconds, and reaction forces should be relayed to the haptic device within 2 milliseconds. Producing a realistic deformation of a virtual organs in so little time is a hard task, and it is solely this topic that the rest of the thesis is focused on.

Full-fledged systems for the problem at hand, which include realistic visualization and simulated surgical instruments, already exist. The most popular technique for simulating the soft tissue in these systems are mass-spring-damper systems [12, 13, 20, 28, 46, 67, 70, 71, 76, 83, 95]: these consist of mass points connected by a network of damped springs. Other techniques have also been presented, for example space-filling spheres [91] and ChainMail [83]. The role of such models is to provide deformations that look qualitatively convincing: the result should "look good." The heuristic nature of these models makes it impossible to go beyond looking good. For example, mass-spring-damper networks do not account for the volume in between the mass points, and hence they cannot simulate truly volumetric properties of material, such as volume preservation.

The lack of quantifiable realism in heuristic deformation models has motivated the move towards methods that are based on the physics of deformation. Deformations of an elastic object are described by laws of physics. These laws can be translated into precise mathematical descriptions in the form of partial differential equations, which can be discretized and solved. The solutions thus found approximate reality in a quantifiable manner. Such discretization methods include Finite Element [14, 103], Finite

Difference [64], and Boundary Element Methods [58].

This thesis only considers the Finite Element Method (FEM). The FEM is a well-studied and highly popular solution method to compute solutions to partial differential equations defined on irregularly shaped objects. Among others, it can be applied to problems of heat conduction, elasticity and electromagnetism. The FEM solves such problems by subdividing the object into a collection of geometric primitives, called *elements*. This process is called meshing. These elements, e.g., triangles, tetrahedra or bricks, can only deform in elementary ways. Physical laws of the material specify how neighboring elements of a mesh interact. The relation between neighboring elements can thus be captured in an equation. The aggregate of all elements together leads to an enormous system of equations, which can be solved with numerical techniques. The solution to this discretized system is an approximation to the solution of the continuous system. A schematic example is shown in Figure 1.1.

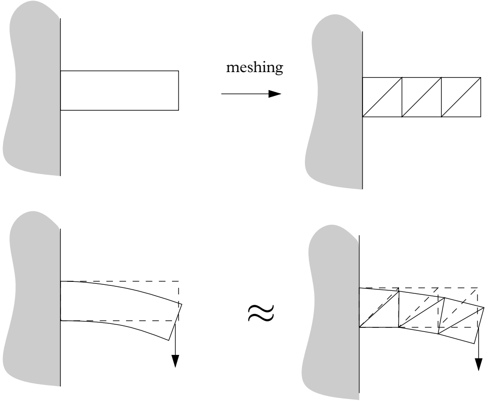


Figure 1.1: In the Finite Element Method, the object simulated (top left) is meshed (top right). The exact solution (bottom left) is then approximated by the deformation of the meshed object (bottom right).

The FEM has been applied before to interactive deformation and surgery simulation. We can distinguish two techniques. Linear elasticity simplifies the equations of elasticity into a linear system. The inverse of this system can be computed beforehand, so that elastic response to external forces can be computed within a guaranteed amount of time. This is a desirable property in interactive simulations, and such precomputation techniques have been used widely in both prototype surgery simulations [18, 37], and generic deformable object simulations [53]. Unfortunately, the linear approximation is only valid when deformations are small, an assumption that is questionable for soft material. In particular, linear elasticity cannot realistically model deformations that involve rotations; an example of linear and nonlinear elasticity is shown in Figure 1.2. Accurately describing these deformations requires nonlinear elasticity, which cannot be used in conjunction with precomputation techniques. Such problems must be solved iteratively: the configuration is moved from some starting configuration to the solution in small steps. We also refer to such a method as a *relaxation method*. The most popular iterative method is explicit dynamic time-stepping, or *dynamic relaxation*. The object is considered to have mass and damping, and the evolution of its movements is described by Newton's laws of motion, which can be numerically computed. Dynamic relaxation has been used for computing solutions to nonlinear elasticity problems in prototype simulation systems [32, 77, 92, 100, 101]. Hybrid systems of static linear precomputed and time-stepping techniques have also been presented [27, 49]. Time-stepping is sometimes also used for linear elasticity, since velocity-dependent friction forces can be integrated in a dynamic model naturally [2].

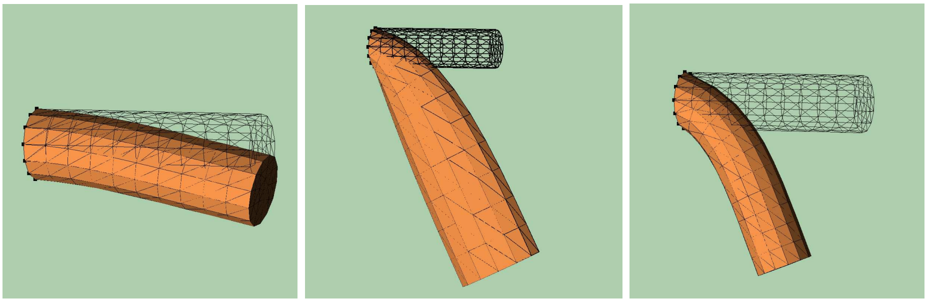


Figure 1.2: Linear elasticity is accurate for small deformations: on the left, an object fixed on the left under gravity load. The original is shown in wireframe. These deformations do not scale linearly: in the center, the same object with force scaled by 10 with linear elasticity. On the right, the same object with a nonlinear elasticity model.

Iterative methods have another advantage over linear elasticity with precomputation. In a surgery simulation, destructive operations such as cuts and cauterizations, can change the mesh. Mesh changes invalidate the precomputed structures that are used in the linear model. By contrast, mesh changes are handled naturally in an iterative method.

Mesh changes can include both simulated surgical procedures and mesh refinements to increase the precision of the FEM solution. In this thesis we will consider both cuts and refinements. Cuts have previously been simulated using subdivision methods: a virtual scalpel slices through an object represented by a mesh, and all elements in contact with the virtual scalpel are subdivided. An example of a subdivision cut in 2D is given in Figure 1.3. In 3D, subdivision techniques for cutting were pioneered by Bielser et al. [12]. Their work has been followed by many other researchers [22, 45, 46, 66]. Subdivision methods can represent cuts accurately. Unfortunately, they increase mesh size substantially.

1.3 Overview of thesis

In this thesis, we will consider nonlinearly deformable objects where the mesh representing the object can be changed by cuts; hence the title "Cutting in deformable objects." More precisely, we will use the FEM for modeling interactive deformation, and solve

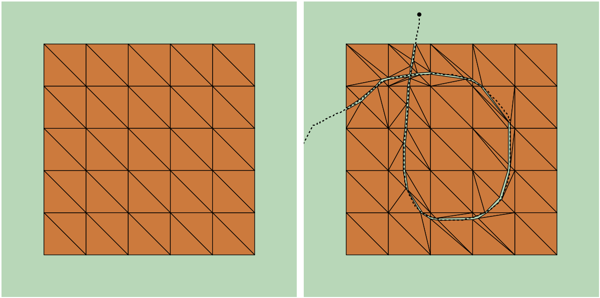


Figure 1.3: A triangle mesh (left), and cut in that mesh produced by a subdivision method (right). The mesh mirrors the scalpel path (dotted) accurately, but uses many small and skinny triangles to do so.

the resulting equations with an iterative method, allowing both the use of nonlinear elasticity models and run-time mesh-changes.

The FEM is a technique traditionally used in engineering: it is a tool to compute solutions to engineering problems with high accuracy. In these cases the analysis proceeds in three separate steps. First, the object is meshed. Then, for each element the local equations are generated and assembled in a large system of equations. Finally, this system of equations is solved. Typically, these three steps are performed by routines that communicate via files on disk or matrices stored in memory.

In an interactive simulation, low response times are very important, while accuracy is not. For quick responses, meshing, equation assembly and system solution should be tightly integrated; structures such as matrices in memory or files on disk cause undesirable overhead. Hence, at the surface, building a deformation simulation seems like a problem in program design: the mesh and the deformations should be stored such that communication between different modules can be done efficiently, while a certain flexibility in the complete system is maintained.

It is true that complex integrated systems can only be implemented when they are designed in a modular manner. However, in the case of FEM computations, the mesh and the solution process must be integrated on a deeper level than program design. Both the accuracy and speed of solution processes strongly depend on the granularity and the quality of the mesh: larger meshes slow down computations, as do meshes with skinny and flat elements. Moreover, flat elements introduce errors in the approximation. Figure 1.4 illustrates this phenomenon. If the mesh is changed on-line, for example due to cuts, care must be taken that the changes do not adversely affect the performance of the total system. So, meshing and deformation are closely coupled, and it is not realistic to treat both problems separately.

Our first venture into the problem of cuts in deformable objects is described[1] in Chapter 3. Here, both problems were treated separately. This has led to the implementation of a system that separates deformation and meshing with the highest degree of

---

[1] Results from Chapter 3 were presented at the Medical Image Computing and Computer Assisted Intervention (MICCAI) conference 2001 [73] and EuroGraphics 2000 [72]

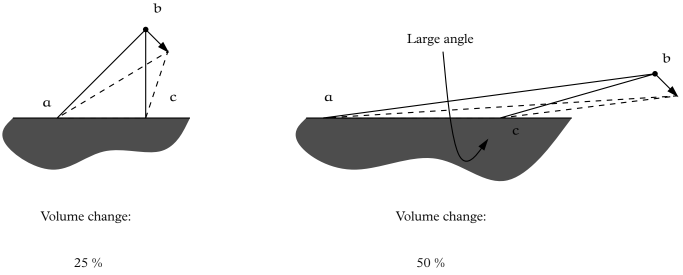


Figure 1.4: Moving a single point of a triangular element. The area of the original element (solid lines) is equal in all cases. The same movement leads to volume reduction of 25% on the left, and 50 % on the right. In the case on the right, an innocuous movement leads to a large volume change, and therefore, large internal forces. The element is stiffer than the real material, and using such flat triangles in the Finite Element discretization yields an inaccurate approximation.

modularity. The system uses linear elasticity using a static iterative relaxation method. A cutting technique was tried that does not increase mesh size like subdivision does. The system succeeds in combining static FEM deformation and interactive cuts on relatively large meshes in a stable manner. However, the cutting method also introduces flat, undesirable elements in the mesh, which must be removed in a separate step.

Chapter 3 uses a *static* approach: an approach where physical time is not simulated. Most other work in deformable object modeling uses dynamic relaxation. In Chapter 4, we take a more in-depth look at static relaxation: this chapter presents a computational study to determine which method performs best on nonlinear problems. Both methods are coded in the same framework. They are compared using a standardized test problem. By timing how long it takes before they reach the final solution, we can determine which one performs best. The conclusion is that the static method is at least as good as the dynamic one: with an optimal choice of parameters, dynamic relaxation is as fast as static relaxation, otherwise it is slower. In this chapter it is also determined how much computational power is needed for running a simulation. The observation that mesh quality influences the performance of an iterative method is confirmed for the nonlinear case as well.

Chapter 3 also proposes a method for cutting in 3D objects represented as tetrahedral meshes. During the development of this method, we have encountered many problems that are not intrinsically three-dimensional, but are partially caused by limitations of both computers and our minds. Computers can only display 2D pictures and we can only see objects from the outside; visualizing what happens inside an object is much harder, both mentally and technically. Therefore, in Chapter 5 we take a step back, and present a method for making cuts in 2D triangle meshes.[2] Existing methods use subdivision, a process that increases the size of the mesh and decreases the quality of the mesh. Both are undesirable, since they make the relaxation more expensive.

---

[2] Chapter 5 was based on a paper accepted for the Fifth International Workshop on Algorithmic Foundations of Robotics (WAFR 2002) [74].

Our approach uses a recognized technique for making high-quality triangle meshes, the Delaunay triangulation. The result is a technique that produces meshes that are measurably better and smaller than those produced with subdivision techniques. The technique is also generalized to curved 3D surfaces, where one scalpel can cause multiple incisions.

In Chapter 6 we turn to a more specific problem, in which the full generality of arbitrary cuts in arbitrary objects is not necessary. This problem concerns simulating how needles are inserted in deforming objects. It has been solved for 2D models with a linear elastic model [36–38], but the extension of this technique to 3D objects and nonlinear elasticity has not been addressed yet. In Chapter 6, we investigate techniques for 2D needle insertion, that might make 3D needle insertion in nonlinear material more tractable. Unlike the work in the preceding chapters, the focus in this chapter is more on quantified accuracy than on visual realism. In this case mesh changes take the form of refinements close to the location of the needle. The resulting simulation has a performance comparable to the work cited, but can also simulate a number of nonlinear models, and readily generalizes to 3D.

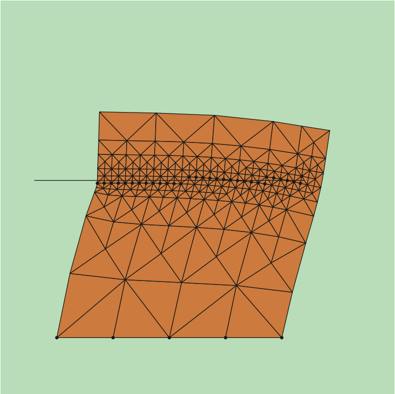


Figure 1.5: Needle insertion in nonlinear material. The mesh is refined close to the needle to increase accuracy.

Finally, Chapter 3 to 6 all manipulate triangle and tetrahedron meshes. For efficiency reasons, the connectivity information of these meshes must be stored explicitly, in the form of pointers that link neighboring triangles and tetrahedra. A side-result of the implementation work of these chapters is a data structure that separates the low-level work of maintaining this connectivity information from the rest of the program, thus making it easy to robustly implement high-level mesh-changes. Chapter 7 describes the data structure, and gives pseudo-code for the low-level operations.

Chapter 2

Preliminaries

The behavior of deforming objects is the topic of continuum mechanics, a branch of mathematics that tries to capture physical phenomena of continuous media in precise mathematical formulations. One branch of continuum mechanics, nonlinear elasticity, provides the mathematical description of how objects deform. Since deformation is fundamental to the topic of the thesis, we will review elasticity in Section 2.1, drawing on the book by Antman [4]. An introduction to the tensor notation used is given in Appendix A.

Continuum mechanics describes materials in terms of partial differential equations. Solving such equations will be done by the Finite Element Method, so this broad category of solution strategies is discussed in Section 2.2. We will work towards the specific technique that we shall use: the Rayleigh-Ritz method for variational problems, using linear interpolation on tetrahedra. This section is loosely based on the introductory chapters of the books by Zienkiewicz and Taylor [103] and Braess [14] on the subject.

The Finite Element Method (FEM) is a discretization method. It transforms a continuous, infinite-dimensional problem into systems of equations with a finite number of variables. For mechanical problems, the FEM discretizes the equations of motion, hence it delivers a system of ordinary differential equations, i.e., equations where time still has a role. There are two ways to deal with these systems: compute the evolution of the system, or try to find the final equilibrium solution directly.

If the final state of the system is all that matters, a *static* method can be used. By assuming that velocity and acceleration are null, the system of differential equations is changed into a normal system of equations. For many mechanical problems, these equations can be stated in terms of finding minimum energy solutions. Hence, we will discuss a number of minimization algorithms in Section 2.4. A more extensive treatment of unconstrained optimization algorithms can be found in the work of Nocedal [75] and Fletcher [41].

If transient effects do matter, then the evolution of the differential equations must be calculated using a *time-integration* method. Section 2.5 discusses a popular time-integration method for mechanical problems, based on the book by Zienkiewicz and Taylor [103].

Basically, our problems come from the simulation of soft tissue. Although simulating the full mechanical characteristics of soft tissue is not possible in an interactive setting, it is instructive to study exactly what kind of characteristics are ignored in our simulations. Section 2.6 briefly discusses a few mechanical properties of living tissue, drawing on the book by Fung [44].

All sections in this chapter tend towards simplification. This is not surprising: the constraints of an interactive simulation do not allow for much sophistication. This chapter also states many results without proving their correctness, and it is in some parts deliberately vague. Since the subjects are too extensive to fit a full discussion in this thesis, the reader is referred to the books mentioned above for more detailed information.

2.1 Continuum mechanics

Continuum mechanics describes the behavior of material objects when they are subjected to loading. The basic assumption is that the objects and their behavior may be described using continous quantities: bodies occupy a continuous region in 3D space, and have continuous motions.

We describe a mechanical or material body as being a subset $\mathcal{B}$ of $\mathbb{R}^3$. We call this set $\mathcal{B}$ the reference configuration. As time progresses, the body may occupy different positions in space: every point of $\mathcal{B}$ moves to some location, depending on the time t. Hence we may describe a deformation by a function $\mathbf{p} : \mathcal{B} \times \mathbb{R} \to \mathbb{R}^3$.

Such a function $\mathbf{p}(\mathbf{z}, t)$ must satisfy at least two requirements to represent a valid motion. First, a body may not penetrate itself: no two points of the body may occupy the same position when deformed. Second, the body can not be folded inside out. In other words, the function $\mathbf{p}(\cdot, t)$ must preserve orientation for all t. Since the volume change of the deformed object is measured by $\det(\mathbf{p}_\mathbf{z}(\mathbf{z}, t))$, we require that this quantity be positive for all t. We shall encounter the derivative $\mathbf{p}_\mathbf{z}$ much more often, and therefore we give it a name and a notation: the 2-tensor field $\mathbf{p}_\mathbf{z}(\mathbf{z}, t)$ is called the deformation gradient, and is denoted by F. It is illustrated for a 2D example in Figure 2.1.

When we talk of deformation, usually we refer to motions $\mathbf{p}$ which locally change the shape of the body. Isometric transformations (rigid movements) do not interest us, therefore we shall use a measure of shape change which filters out these rigid motions. Let

$$\mathsf{C} = \mathsf{F}^* \cdot \mathsf{F}. \tag{2.1}$$

This tensor is called the *(right Cauchy-)Green deformation tensor*, and it is invariant under rotations. This deformation tensor measures the elongation of a fiber running in a particular direction: if we have an infinitesimally long fiber with direction running from $\mathbf{z}$ to $\mathbf{z} + \mathbf{h}$, then its length after deformation is measured by $\sqrt{\mathbf{h} \cdot \mathsf{C} \cdot \mathbf{h}}$. The length change of such a fiber attains extremes when $\mathbf{h}$ is an eigenvector of C. Since C is symmetric, it has three orthogonal eigenvectors, called the *principal axes of strain.* The tensor C is also visualized in Figure 2.1.

Solid bodies resist movement. This property is called inertia, and inertia is measured by the *mass* of a body. We assume that a body $\mathcal{B}$ has a density $\rho(\mathbf{z})$ in each point $\mathbf{z}$, and

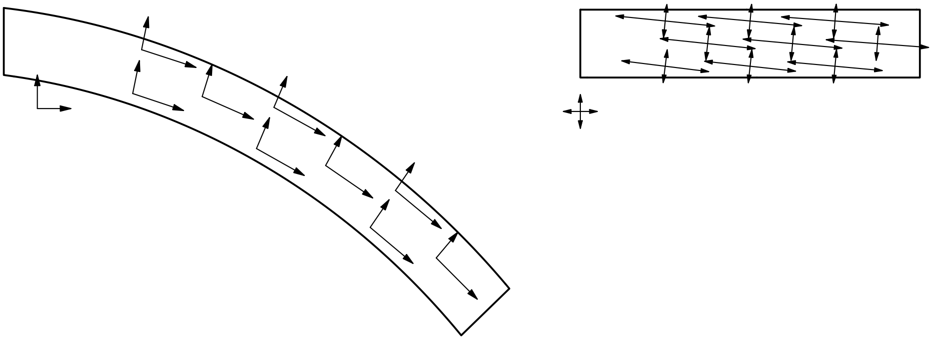


Figure 2.1: This figure shows a deformation of a beam in 2D (reference configuration on the right), with vectors of the deformation gradient (left) and principal axes of strain (right). The identity mapping is represented in the lower left of each picture. The motion used is $\mathbf{p}((x,y)) = -c\mathbf{e}_x + \mathbf{R}(-\alpha x)(y+c)\mathbf{e}_y - \beta x \mathbf{e}_y$, where $\mathbf{R}(\phi)$ is the mapping for rotation over angle ϕ, and $\mathbf{e}_1, \mathbf{e}_2$ is the unit basis in $\mathbb{R}^2$. This corresponds to a combination of bending, dilation and shearing. The deformation gradient $\mathbf{F}$ is really defined in points of the reference configuration, but we show $\mathbf{F}$ in the corresponding deformed points $\mathbf{p}(z)$.

the mass of a some part $\mathcal{A}$ of a body is given by the volume integral

$$\int_{\mathcal{A}} \rho(z)\, dv(z). \tag{2.2}$$

A part $\mathcal{A}$ of a body $\mathcal{B}$ may be subject to force. Forces take the form of either traction on the surfaces or body forces, which are applied to the volume of the body. Surface tractions are denoted by $\mathbf{t}$. Body forces (which can be either electromagnetic or gravity forces) are denoted by $\mathbf{f}$,

The balance of linear momentum, which is also known as Newton's law, postulates that resultant forces equal acceleration. It may be formulated as follows.

$$\begin{aligned} &\int_{\mathcal{A}} \mathbf{f}(z,t) dv(z) + \int_{\partial\mathcal{A}} \mathbf{t}(z,t;\partial\mathcal{A})\, da(z) \\ &\quad = \frac{d}{dt}\int_{\mathcal{A}} \mathbf{p}_t(z,t)\rho(z)\, dv(z), \qquad \mathcal{A} \subset \mathcal{B}. \end{aligned} \tag{2.3}$$

The left side of the equation represents body force plus boundary tractions. The right side of the equation is the derivative of linear momentum. This equation holds for every subset $\mathcal{A}$ of the body $\mathcal{B}$. The traction t on a surface $\partial\mathcal{A}$ depends only on z through the surface normal $\mathbf{n}$ on $\partial\mathcal{A}$. This relation is linear, so there is a 2-tensor $\mathbf{T}$ defined on the body $\mathcal{B}$, such that

$$\mathbf{t}(z,t;\partial\mathcal{A}) = \mathbf{T}(z,t)\cdot\mathbf{n}, \qquad z \in \partial\mathcal{A}, \mathbf{n} \perp \partial\mathcal{A}. \tag{2.4}$$

This theorem is called Cauchy's stress theorem, and the mapping $\mathbf{T}$ is called the *first Piola-Kirchoff stress tensor*. The vector $\mathbf{n}$ is the outward pointing surface normal of $\partial\mathcal{A}$ at point $\mathbf{z}$ in the reference configuration.

Equation (2.3) integrates variables over parts of $\mathcal{B}$, hence it is called *weak*. If $\mathbf{p}$ is continuously differentiable, then the balance of momentum can be rewritten in a *strong*, localized version

$$\operatorname{div}\mathbf{T}(\mathbf{z},t) + \mathbf{f}(\mathbf{z},t) = \rho(\mathbf{z})\mathbf{p}_{tt}(\mathbf{z},t), \qquad \mathbf{z} \in \mathcal{B}. \tag{2.5}$$

The operator div is the divergence of a 2-tensor field. It is defined by $\operatorname{div}\mathbf{T} = \partial\mathbf{T}/\partial\mathbf{z} : \mathbf{I}$, where $\cdot : \cdot$ is the inner product for 2-tensors.

The balance of angular momentum asserts that resultant couples equal angular acceleration. If we assume that no electromagnetic effects take place, then resultant couples are always null, and the tensor $\mathbf{T} \cdot \mathbf{F}^*$ is symmetric. This leads us to introduce the tensor $\mathbf{S}$, called the second Piola-Kirchoff stress tensor, by defining

$$\mathbf{S} = \mathbf{F}^{-1} \cdot \mathbf{T}. \tag{2.6}$$

This tensor is symmetric. If $\mathbf{e}_1$ is the normal of a material plane in $\mathcal{B}$, and $\mathbf{e}_2$ and $\mathbf{e}_3$ span the plane in $\mathcal{B}$, then the matrix entries of $\mathbf{S}$ contain the traction on that plane, $\mathbf{T}\mathbf{e}_1$, but expressed in deformed basis vectors $\{\mathbf{F}\mathbf{e}_1, \mathbf{F}\mathbf{e}_2, \mathbf{F}\mathbf{e}_3\}$.

Constitutive equations define the relations between stress and deformation. A constitutive equation is given by a function $\hat{\mathbf{T}}$, such that

$$\mathbf{T} = \hat{\mathbf{T}}(\mathbf{p}(\cdot,\cdot), \mathbf{z}, t), \qquad \mathbf{z} \in \mathcal{B}. \tag{2.7}$$

The dependency on $\mathbf{p}$ is understood to be causal: the value of $\hat{\mathbf{T}}$ on some time t_0 only depends on values of $\mathbf{p}$ before t_0. The hat on $\hat{\mathbf{T}}$ stresses the difference between the tensor field $\mathbf{T}$, a physical quantity that could be measured in physical realizations of the situations described, and $\hat{\mathbf{T}}$, a function that expresses the mathematical relation between two tensor fields (in this case, the field $\mathbf{T}$ and $\mathbf{p}(\cdot,\cdot)$). For the stress tensor $\mathbf{S}$ we use the same convention.

The description in Equation (2.7) is too general to be of practical use, and therefore we directly restrict ourselves to *simple materials*, where the stress at a point $\mathbf{z}$ only depends on the values of $\mathbf{p}$ in a neighborhood of $\mathbf{z}$, as measured by the first derivatives $\mathbf{p}_{\mathbf{z}}$. We get

$$\mathbf{T} = \hat{\mathbf{T}}(\mathbf{p}(\mathbf{z},\cdot), \mathbf{p}_{\mathbf{z}}(\mathbf{z},\cdot), \mathbf{z}, t), \tag{2.8}$$

for some function $\hat{\mathbf{T}}$. Again the dependency on $\mathbf{p}$ and its derivatives is understood to be causal.

Constitutive equations should be independent of the choice of a time and coordinate system. This property is called *frame-indifference*. For a simple material, this implies that

$$\hat{\mathbf{T}}(\mathbf{F}(\mathbf{z},\cdot), \mathbf{z}) = \mathbf{R}\hat{\mathbf{T}}(\mathbf{U}(\mathbf{z},\cdot), \mathbf{z}), \tag{2.9}$$

$$\mathbf{U}(\mathbf{z},\cdot) = \sqrt{\mathbf{F}(\mathbf{z},\cdot)^* \cdot \mathbf{F}(\mathbf{z},\cdot)}, \tag{2.10}$$

$$\mathbf{R} = \mathbf{F}(\mathbf{z},t) \cdot \mathbf{U}^{-1}(\mathbf{z},t). \tag{2.11}$$

The tensors U and R form the polar decomposition of F: $\mathsf{F} = \mathsf{R} \cdot \mathsf{U}$, where R is a rotation, and U is a positive definite symmetric matrix. The dependency on $\mathsf{U}(z, \cdot)$ is meant to be causal.

By substituting Equation (2.6) into (2.9), we can derive the following description for a frame indifferent constitutive equation which uses symmetric tensors only:

$$\mathbf{S}(z, t) = \hat{\mathbf{S}}(\mathbf{C}(z, \cdot), z).$$

A further simplification can be done if we assume that the material responds to deformations equally in all directions. Then for all rotation mappings $\mathbf{Q}$ we have

$$\mathbf{Q} \cdot \hat{\mathbf{S}}(\mathbf{C}, z) \cdot \mathbf{Q}^* = \hat{\mathbf{S}}(\mathbf{Q} \cdot \mathbf{C} \cdot \mathbf{Q}^*, z).$$

Such material is called *isotropic* or *hemitropic*. If not, then the material is called *aelotropic* or *anisotropic*. Examples of materials that are aelotropic are those that contain networks of regularly arranged fibers.

The tensor $\mathbf{C}$ is symmetric, and can therefore be characterized up to rotations by its three eigenvalues, or equivalently, by its three invariants. If λ_1, λ_2 and λ_3 are the eigenvalues of $\boldsymbol{A}$, then the invariants ι_1, ι_2 and ι_3 are defined as follows:

$$\begin{aligned} \iota_1(\boldsymbol{A}) &= \lambda_1 + \lambda_2 + \lambda_3, \\ \iota_2(\boldsymbol{A}) &= \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1, \\ \iota_3(\boldsymbol{A}) &= \lambda_1\lambda_2\lambda_3. \end{aligned}$$

The invariants can be determined without computing the eigenvalues of $\boldsymbol{A}$. We have

$$\begin{aligned} \iota_1(\boldsymbol{A}) &= \operatorname{trace}(\boldsymbol{A}), \\ \iota_2(\boldsymbol{A}) &= \frac{1}{2}((\operatorname{trace}\boldsymbol{A})^2 - \operatorname{trace}(\boldsymbol{A}^* \cdot \boldsymbol{A})), \\ \iota_3(\boldsymbol{A}) &= \det \boldsymbol{A}. \end{aligned}$$

The trace of a matrix $\mathbf{H}$ can be defined in terms of the inner product for 2-tensors: $\operatorname{trace}(\mathbf{H}) = \mathbf{H} : \mathbf{I}$. In a matrix representation, the trace of a matrix is the sum of the diagonal elements.

The following representation theory should come as no surpise: a hemitropic stress function $\hat{\mathbf{S}}$ depends on $\mathbf{C}$ only through the invariants ι_1, ι_2 and ι_3 of $\mathbf{C}$. For hyperelastic media, this means that the energy density W must also be a function of the invariants:

$$W = \hat{W}(\iota_1, \iota_2, \iota_3, z). \tag{2.12}$$

Some materials have restriction on the ways in which they can deform. The most important example of this is *incompressibility*. A material is incompressible if deformations conserve volume locally. This constraint may be expressed as $\det \mathbf{C} = 1$. Constitutive equations describing incompressible material can be derived by a limit process. This limit process introduces a new variable, the pressure p, that serves to balance any elastic force that tries to violate the constraint. Examples of incompressible material are those containing lots of fluid.

We can distinguish many classes of material by restricting the dependency of T on $\mathbf{p}$ even further. We will restrict ourselves to *elastic* materials. These are materials where T depends only on $\mathbf{p}_z$, and not on t or $\mathbf{p}$ itself. The dependency on $\mathbf{p}_z$ does not take the past history of $\mathbf{p}_z$ into account, but only its value at time t. In other words, this is material that satisfies

$$\mathsf{T} = \hat{\mathsf{T}}(\mathbf{p}_z(z,t), z), \tag{2.13}$$

for some function $\hat{\mathsf{T}}$. Such a material is also called *Cauchy-elastic.* An elastic material is hyperelastic, or *Green-elastic*, if there is a scalar function W called stored energy function, such that

$$\hat{\mathsf{T}}(\mathsf{F}, z, t) = \frac{\partial W(\mathsf{F}, z)}{\partial \mathsf{F}}, \tag{2.14}$$

or equivalently

$$\hat{\mathsf{S}}(\mathbf{C}, z, t) = 2\frac{\partial W(\mathbf{C}, z)}{\partial \mathbf{C}}. \tag{2.15}$$

If the stress is independent of temperature, or temperature is held constant, then the equilibrium response of any elastic material is hyperelastic.

These considerations give us some constraints on functions that can be used as $\hat{\mathsf{T}}$ and W, but not all such functions lead to descriptions that realistically describe existing materials. Mathematical conditions exist that express basic properties of materials, e.g., elastic forces should oppose deformations, and extreme deformations should lead to extreme tensions within the material. This still leaves a lot of freedom in specifying constitutive equations, i.e. selecting W or $\hat{\mathsf{T}}$ functions. Finding a constitutive equation for a given material is far from trivial. A variety of theoretical models exist for different classes of material (e.g. metals and rubbers), but they are usually only validated by fragmentary experimental results [5].

The equations presented so far are inherently nonlinear: the strain tensor $\mathbf{C}$ is nonlinear in $\mathbf{p}$. This ensures that solutions to the equations for $\mathbf{p}$ will not be linear in the given conditions, i.e. the boundary conditions and the body forces. However, a linear approximation to the elasticity does exist. It has three virtues: it is simpler from a conceptual point of view, it is computationally simpler, and relatively uncomplicated proofs of existence and unicity of the solution can be given.

If we assume that deformations are small, then we can assume that the stress strain relation is linear, or equivalently, W is quadratic in $\mathbf{C}$. This is called the *St. Venant-Kirchoff* material model, and it is the simplest material model available. It will be used in Chapter 4. Since W is quadratic in this model, must have the following form.

$$W(\iota_1, \iota_2) = c_0 + c_1\iota_1 + c_2\iota_1^2 + c_3\iota_2. \tag{2.16}$$

The choice of c_0 is irrelevant, so we set $c_0 = 0$. The constants $c_1, \ldots, c_3$ are dependent: the reference configuration should be stress-free, so we can eliminate one constant by imposing $\mathbf{S} = 0$ when $\mathbf{C} = \mathbf{I}$. Two constants remain. We can express the energy density thus obtained as follows:

$$W(\iota_1, \iota_2) = \frac{1}{2}\left(\left(-\mu - \frac{3\lambda}{2}\right)\iota_1 + \left(\frac{\lambda}{4} + \frac{\mu}{2}\right)\iota_1^2 - \mu\iota_2\right). \tag{2.17}$$

The stress tensor is linear in $\mathbf{C}$,

$$\mathbf{S} = \mu(\mathbf{C} - \mathbf{I}) + \frac{1}{2}\lambda(\iota_1 - 3)\mathbf{I}.$$

Hence the tension in the material is proportional to the deformation, as measured by $\mathbf{C}-\mathbf{I}$, with additional tension caused by volume changes, which are measured by $(\iota_1 - 3)$ for small deformations.

The parameters μ and λ are called Lamé parameters. Material properties are usually expressed in parameters that have a macroscopically more intuitive definition: the Young modulus measures the resistance to stretching. Its notation is E, and we have

$$E = \mu\frac{(3\lambda + 2\mu)}{\lambda + \mu}. \tag{2.18}$$

The unit of E is Pascal (N/m^2). The Poisson ratio is the ratio between the transverse contraction and longitudinal stretching. We have

$$\nu = \frac{\lambda}{2(\lambda + \mu)}. \tag{2.19}$$

Since ν is a ratio, it is dimensionless. For physical reasons, we have $0 \leq \nu < 1/2$. If ν tends to $1/2$, then $\lambda \to \infty$, and the material becomes incompressible. The meaning of E and ν is illustrated in Figure 2.2.

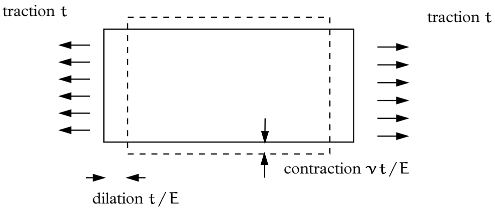


Figure 2.2: A uniformly distributed traction leads to uniform stresses and strains. Material in the direction of the load expands inversely proportional to the Young modulus E. The contraction in the transversal direction is ν times the dilation, where ν is the Poisson ratio.

We introduce the displacement $\mathbf{u}$ of a point. The displacement $\mathbf{u}$ of $\mathbf{z}$ at time t is defined by

$$\mathbf{u}(\mathbf{z}, t) = \mathbf{p}(\mathbf{z}, t) - \mathbf{z}. \tag{2.20}$$

This expression vanishes in the reference configuration. For rotations, $\mathbf{u}$ generally is nonzero. We write $\mathbf{G}$ for $\frac{\partial \mathbf{u}}{\partial \mathbf{z}}$. We can linearize both the definition of strain in Equation (2.1), and the constitutive relations in (2.7). Linearizing the strain tensor is also called the linear geometry assumption. We have

$$\mathbf{C} = \mathsf{F}^* \cdot \mathsf{F} = (\mathbf{I} + \mathbf{G})^* \cdot (\mathbf{I} + \mathbf{G}) = \mathbf{I} + \mathbf{G}^* + \mathbf{G} + \mathbf{G}^* \cdot \mathbf{G}.$$

When we assume that $\mathbf{G}$ is small, then we can neglect the quadratic term $\mathbf{G}^* \cdot \mathbf{G}$, obtaining

$$\mathbf{C} \approx \mathbf{I} + \mathbf{G}^* + \mathbf{G}.$$

Another measure for deformation is the material strain tensor E, which is defined by

$$\mathsf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I}). \tag{2.21}$$

The tensor E disappears in the reference configuration. Let $\mathcal{E}$ be the linear geometry approximation of E, then we have

$$\begin{aligned} \mathcal{E} &= \frac{1}{2}(\mathbf{G} + \mathbf{G}^*), \\ \tilde{W}(\mathcal{E}) &= \mu \operatorname{trace}(\mathcal{E}^*\mathcal{E}) + \frac{1}{2}\lambda(\operatorname{trace}\mathcal{E})^2, \\ \mathbf{S} &= 2\mu\mathcal{E} + \lambda\mathbf{I}(\operatorname{trace}\mathcal{E}). \end{aligned} \tag{2.22}$$

When both strain and material linearizations are combined, stresses are linear in the displacement. If deformations are small, then $\partial\mathbf{u}/\partial\mathbf{z} = \mathcal{O}(\varepsilon)$ for some small $\varepsilon > 0$. In this case $\mathcal{O}(\varepsilon^2)$ terms are neglible and both $\mathcal{E}$ and $\mathbf{S}$ are also $\mathcal{O}(\varepsilon)$. It follows that we can equate $\mathbf{S}$ and T since

$$\mathsf{T} = \mathsf{F} \cdot \mathbf{S} = (\mathbf{I} + \mathcal{O}(\varepsilon)) \cdot \mathbf{S} = \mathbf{S} + \mathcal{O}(\varepsilon^2). \tag{2.23}$$

This combination is called linear elasticity. Suppose that the displacements are fixed on a subset of the boundary with non-zero area, say $\partial\mathcal{B}_0$, and tractions are specified on the rest of the boundary, say $\partial\mathcal{B}_3$ and $V(\mathcal{B})$ is the set of functions with square-integrable derivatives that satisfy the boundary condition on $\mathcal{B}_0$, then then the following problem has a unique solution.

$$\min_{u \in V(\mathcal{B})} \int_{\mathcal{B}} \left(\tilde{W}(\mathcal{E}(\mathbf{u})) - \mathbf{f} \cdot \mathbf{u}\right) \, dv(\mathbf{z}) + \int_{\partial\mathcal{B}_3} \mathbf{t} \cdot \mathbf{u} \, da(\mathbf{z}), \tag{2.24}$$

$$\mathbf{u} = 0, \qquad \text{on } \partial\mathcal{B}_0. \tag{2.25}$$

The solution is weak: the space $V(\mathcal{B})$ consists of functions from $\mathcal{B}$ to $\mathbb{R}^3$ with square-integrable derivatives that satisfy the boundary condition on $\mathcal{B}_0$.

2.2 Finite Element Method

Mathematical modeling processes such as the one in the previous section, yield a collection of partial differential equations. The unknown variable in such an equation is a sufficiently smooth function defined on the domain $\mathcal{B}$ of the equation. In the general case it is not possible to compute a solution in closed form to these problems. Therefore, approximative schemes are necessary. The Finite Element Method (FEM) is an approximative scheme, and it is very popular for mechanical analysis. In this section, we shall briefly explain the essentials of this method, and work towards the simple FEM scheme that we will use (linear tetrahedral elements).

Let us suppose, for the moment, that the partial differential equation is mechanical, set in $\mathbb{R}^3$, and that the equation we wish to solve states that the resultant force $\mathbf{r}$ for the displacement $\hat{\mathbf{u}}$ is zero on a domain $\mathcal{B} \subset \mathbb{R}^3$. In other words,

$$\text{find } \hat{\mathbf{u}} \in V(\mathcal{B}), \text{ such that } \mathbf{r}(\hat{\mathbf{u}}) = \mathbf{0}. \tag{2.26}$$

The set $V(\mathcal{B})$ is a linear space of candidate solution functions with suitable differentiability and boundary conditions. The functions in $V(\mathcal{B})$ represent displacements, so they take on values from $\mathbb{R}^3$. Equilibrium solutions of the equations of motion in (2.5) have such a form.

The space $V(\mathcal{B})$ is infinite-dimensional, and to make finding a solution tractable we search the solution in a smaller, finite-dimensional subspace of $V(\mathcal{B})$. Such subspaces are typically created by interpolating functions $V(\mathcal{B})$ using points from $\mathcal{B}$ spaced at distance h. Since h is a parameter, the notation for the subspace is $V_h(\mathcal{B})$. We obtain the following problem.

$$\text{Find } \tilde{\mathbf{u}} \in V_h(\mathcal{B}), \text{ such that } \mathbf{r}(\tilde{\mathbf{u}}) = \mathbf{0}. \tag{2.27}$$

The space V_h is much smaller than V, so it is unlikely that it contains a solution $\tilde{\mathbf{u}}$ for which (2.27) holds exactly. All we can really hope for is that $\mathbf{r}(\tilde{\mathbf{u}})$ is as small as possible. To measure this, we weigh $\mathbf{r}$ with functions $\boldsymbol{w}$ from a finite-dimensional space W_h, and state the problem as follows.

$$\begin{gathered}\text{find } \tilde{\mathbf{u}} \in V_h(\mathcal{B}), \text{ such that for all } \tilde{\boldsymbol{w}} \in W_h \\ \int_{\mathcal{B}} \tilde{\boldsymbol{w}}(\boldsymbol{z}) \cdot (\mathbf{r}(\tilde{\mathbf{u}}))(\boldsymbol{z})\, dv(\boldsymbol{z}) = 0.\end{gathered} \tag{2.28}$$

Formulations where the equations are integrated over sets in $\mathbb{R}^3$ are also called *weak* formulations. The weak equations of motion only require functions whose derivative is square integrable on $\mathcal{B}$ in the Lesbesgue sense. This explains why a V_h consisting of piecewise linear functions is sufficient for computing a solution: the derivatives of such functions are not defined everywhere, but they are square integrable.

Formulations such as Equation (2.26) set a condition that should hold in every point of the domain $\mathcal{B}$. Equation (2.5) is a second order partial differential equation, which suggests that candidate solutions should be twice differentiable, i.e. $V(\mathcal{B}) = C_2(\mathcal{B})$. Since this is more restrictive than the weak form of the equations, we call such a form *strong*.

The space W_h is finite-dimensional, and hence has a basis $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k\}$ for some $k \in \mathbb{N}$. We may therefore rephrase this equation in terms of $\boldsymbol{w}_j$ as

$$\int_{\mathcal{B}} \boldsymbol{w}_j(\boldsymbol{z}) \cdot (\mathbf{r}(\tilde{\mathbf{u}}))(\boldsymbol{z})\, dv(\boldsymbol{z}) = 0, \quad j = 1, \ldots, k.$$

In some types of electromechanical calculations, and in the linear simplifications of the continuum mechanics of Section 2.1, it may happen that $\mathbf{r}$ is affine linear, say

$$\mathbf{r}(\mathbf{u}) = \mathbf{L}(\mathbf{u}) - \mathbf{f}.$$

Here, $\mathbf{L}$ is a differentiation operator. The space $V_h(\mathcal{B})$ is finite-dimensional, and hence has a basis $\mathbf{u}_1, \ldots, \mathbf{u}_l$ for some $l \in \mathbb{N}$. If we expand $\tilde{\mathbf{u}}$ in this, then the linearity of $\mathbf{r}$ yields the following set of equations:

$$\tilde{\mathbf{u}} = \sum_i \alpha_i \mathbf{u}_i$$
$$\sum_i \alpha_i \int_{\mathcal{B}} \mathbf{w}_j(z) \cdot (\mathbf{L}\tilde{\mathbf{u}}_i)(z)\, dv(z) = \int_{\mathcal{B}} \mathbf{w}_j(z) \cdot \mathbf{f}(z)\, dv(z), \qquad 1 \leq j \leq k$$

If we look closely, we see that the above equation simply is a linear system of size $l \times k$. Let $K \in \mathbb{R}^{l \times k}$, and $f \in \mathbb{R}^k$ be defined by

$$\begin{aligned} K_{ij} &= \int_{\mathcal{B}} \mathbf{w}_j(z) \cdot (\mathbf{L}\mathbf{u}_i)(z) dv(z), \quad i = 1, \ldots, l, j = 1, \ldots, k \\ f_j &= \int_{\mathcal{B}} \mathbf{w}_j \cdot \mathbf{f}\, dv(z), \quad j = 1, \ldots, k, \end{aligned} \tag{2.29}$$

then the solution with zero weighted residual is given by

$$\begin{aligned} \tilde{\mathbf{u}} &= \sum_{i=1}^{l} \alpha_i \mathbf{u}_i, \\ K\alpha = f, &\qquad \alpha = (\alpha_1, \ldots, a_l). \end{aligned}$$

If W_h is large enough then we could find $\tilde{\mathbf{u}}$, for example by solving the least-squares problem

$$K^T K \alpha = K^T f.$$

The matrix K is often called *stiffness matrix*, especially if the original problem is mechanical. This process of finding the solution by weighting the error ("residual") is called the *weighted residual* method, or *Petrov-Galerkin* process. The functions $\mathbf{u}_j$ are called *shape functions*, and $\mathbf{w}_j$ are *trial functions*.

In some applications, the differential operator is self-adjoint: the integral over $\mathcal{B}$ defines an inner product,

$$(\mathbf{x}, \mathbf{y}) = \int_{\mathcal{B}} \mathbf{x}(z) \cdot \mathbf{y}(z)\, dv(z), \qquad \mathbf{x}, \mathbf{y} \in V(\mathcal{B}).$$

If the operator L is self-adjoint, i.e., if $(\mathbf{w}, \mathbf{L}\mathbf{u}) = (\mathbf{L}^*\mathbf{w}, \mathbf{u}) = (\mathbf{L}\mathbf{w}, \mathbf{u})$, then we can take $W_h = V_h$. The weak problem can be translated into a linear system similar to (2.29). In this case, the stiffness matrix will be symmetric. This solution process is called the *Bubnov-Galerkin* method. Additionally, if $\mathbf{L}$ is positive definite, i.e. $(\mathbf{L}\mathbf{u}, \mathbf{u}) \geq 0$ for all $\mathbf{u} \in V(\mathcal{B})$, then so will be the matrix K. Both symmetry and positive-definiteness help in solving the numerical system. For the remainder of this thesis, we assume that $W_h = V_h$.

In many formulations, the residual $\mathbf{r}(\mathbf{u})$ is related to some virtual work functional $\Pi(\cdot)$. For example, the internal energy may equal the work done by the residual force plus work done by the boundary tractions:

$$\Pi(\mathbf{u}) = \int_{\mathcal{B}} \mathbf{u} \cdot \mathbf{r}(\mathbf{u})\, dv(\mathbf{z}) + \int_{\partial\mathcal{B}} \mathbf{u} \cdot \mathbf{t}\, da(\mathbf{z}).$$

In this case, the solution $\mathbf{u}$ that we seek minimizes $\Pi(\mathbf{u})$ over $V(\mathcal{B})$. An approximation of this minimization problem is found again by seeking it in a smaller, finite-dimensional space of V_h. If we are given a basis $\mathbf{u}_1, \ldots, \mathbf{u}_l$ of this space, then the coefficients $\alpha_1, \ldots, \alpha_l$ can be found by solving

$$\min_{\alpha_1,\ldots,\alpha_l} \Pi(\sum_{i=1}^{l} \alpha_i \mathbf{u}_i).$$

This approach is called the *Rayleigh-Ritz* method. We can express the problem as a system of equations by setting the derivative of Π to 0,

$$\frac{\partial \Pi(\sum_i \alpha_i \mathbf{u}_i)}{\partial \alpha_i} = 0, \quad i = 1, \ldots, l. \tag{2.30}$$

If Π is a quadratic form, then its derivative is linear, and (2.30) is exactly the linear system produced by the Bubnov-Galerkin method. Algorithms to solve these systems are discussed in Section 2.4.

We have discussed a general scheme for going from a problem in a space $V(\mathcal{B})$ with infinite dimension to an approximation in a subspace V_h. If we select a V_h and a basis $\mathbf{u}_1, \ldots, \mathbf{u}_l$ to span V_h, we can assemble a system of equations like (2.29) or (2.30).

The support of a function $\mathbf{v} \in V(\mathcal{B})$, denoted by $\operatorname{supp} \mathbf{v}$, is the subset of $\mathcal{B}$ where $\mathbf{v}$ takes on non-zero values. It is advantageous to select shape functions whose supports are as much as possible disjoint. If $\mathbf{u} = \mathbf{0}$ on a region, then we have $\mathbf{L}\mathbf{u} = \mathbf{0}$ on that region, so if $\operatorname{supp} \mathbf{u}_i$ and $\operatorname{supp} \mathbf{u}_j$ are disjoint, then $(\mathbf{u}_i, \mathbf{L}\mathbf{u}_j) = 0$. In other words, having disjoint supports promotes sparsity of the stiffness matrix K.

The most popular approach to obtain V_h—the method that is generally called the 'Finite Element Method'—is to subdivide $\mathcal{B}$ into a collection of geometric primitives such as triangles, quadrilaterals (in 2D) or tetrahedra, hexahedrals or other solids (in 3D). Let the partition be denoted by $\bigcup_j \Omega_j$, and select an integer $p \geq 1$. Every function from V_h is then taken to be a polynomial of degree at most p on every Ω_k. Conditions are set such that all functions have the continuity over the boundaries of each element appropriate for the problem being solved.

There is still ample choice of a basis. Recall that we want to make $\operatorname{supp} \mathbf{u}_j$ as small as possible. One way to enforce this, is to select a set points in the elements, called nodes, that uniquely determine the value of the polynomial on that element. For example, if we take the piecewise linear functions defined on mesh of triangles, then these functions form a finite-dimensional space V_h. The vertices of the triangles form nodes for this finite element basis. Elements that include higher order polynomials, might also have nodes located on the midpoints of edges, or in the interior of the elements.

If we are given function values in each node, the piecewise polynomial is uniquely determined on each element. Since nodes on the boundaries of edges are shared by adjoining elements, continuity of the functions is ensured by construction. Continuity in the derivatives is not required for the weak problem formulation of the elasticity equations.

By pinpointing nodes in a mesh, we can form a natural basis of V_h, which is called the *nodal* basis. It is the collection of functions that take the value 1 on a single node, and 0 on all other nodes. Such a function is identically zero on all elements that do not contain the non-zero node. The hat-functions from Figure 2.3 form the nodal basis for triangles in 2D.

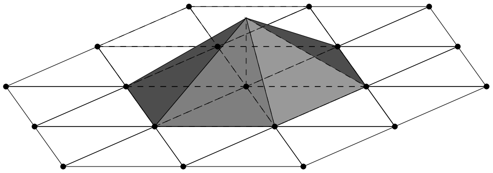


Figure 2.3: Hat functions are a nodal basis for linear triangles

If the derivatives in the differential equation have constant coefficients, i.e. if they are independent of the spatial coordinate z, then the matrices in Equation (2.29) and the derivatives in (2.30) be calculated analytically, so numerical integration methods are not needed.

The relative simplicity of finite elements comes at a price: the difficulty in obtaining a solution is transferred to the problem of subdividing $\mathcal{B}$. The quality of the FEM approximation, as well as the speed and quality of a numerical method depends on the quality of the tessellation of $\mathcal{B}$: subdivision should contain nicely shaped elements. For example, in a triangle subdivision, the triangles should have neither very small nor very large angles [87]. In general, FEM approximation can only be successful if we can generate good meshes for the domain of the problem, and for complex 3D shapes, the task of generating good meshes is a rich source of interesting problems.

We have been sloppy in discussing the traction boundary conditions on purpose; in a displacement-based FE formulation, boundary conditions are added as body forces for boundary nodes, and both can be treated integrally.

2.3 Linear tetrahedra for hyperelastic materials

In the remain of this thesis, we will use linear elements. The displacement function $\mathbf{u}$ is interpolated with a piecewise linear approximation. The interpolation conditions are put at the vertices of the tetrahedral mesh. In other words, we want to have a linear function, and it should map $z_j \in \mathbb{R}^3$ to $\mathbf{q}_j \in \mathbb{R}^3$ for $j = 1, \ldots, 4$. These conditions uniquely determine a linear affine function. It takes the form

$$\mathbf{z} \mapsto \mathbf{A} \cdot \mathbf{z} + \mathbf{b}.$$

The following function satisfies our interpolation requirements

$$\boldsymbol{z} \mapsto \mathbf{Q} \cdot \mathbf{Z}^{-1} \cdot (\boldsymbol{z} - \boldsymbol{z}_4) + \mathbf{q}_4, \tag{2.31}$$

$$\mathbf{Z} = (\boldsymbol{z}_1 - \boldsymbol{z}_4) \otimes \boldsymbol{e}_1 + (\boldsymbol{z}_2 - \boldsymbol{z}_4) \otimes \boldsymbol{e}_2 + (\boldsymbol{z}_3 - \boldsymbol{z}_4) \otimes \boldsymbol{e}_3, \tag{2.32}$$

$$\mathbf{Q} = (\mathbf{q}_1 - \mathbf{q}_4) \otimes \boldsymbol{e}_1 + (\mathbf{q}_2 - \mathbf{q}_4) \otimes \boldsymbol{e}_2 + (\mathbf{q}_3 - \mathbf{q}_4) \otimes \boldsymbol{e}_3. \tag{2.33}$$

Here, $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$ is taken to be an orthonormal base of $\mathbb{R}^n$.

Where the tensor product $\mathbf{a} \otimes \mathbf{b}$ is the 2-tensor defined by $(\mathbf{a} \otimes \mathbf{b}) \cdot \boldsymbol{x} = \mathbf{a}(\mathbf{b} \cdot \boldsymbol{x})$. The derivative of this function with respect to $\boldsymbol{z}$ is constant across an element, and is given by

$$\mathbf{Q} \cdot \mathbf{Z}^{-1}. \tag{2.34}$$

The deformations are given in terms of displacements at every node of the mesh, so in effect, we use a linear interpolation for the displacements. For such an interpolation, we can interpret the derivatives of the elastic energy as elastic forces concentrated at nodes of the mesh. We can view the elastic force in a single node as the compound result of the elastic forces of individual tetrahedra incident with that node.

We take a hyperelastic model as a starting point for deriving stresses. This means that we should choose a function $W(\mathbf{C})$, and find its derivative to obtain the stress:

$$\mathbf{T} = \mathbf{F} \cdot \mathbf{S} = \mathbf{F} \cdot \frac{\partial W}{\partial \mathbf{C}}. \tag{2.35}$$

We can formulate weak equations of motion from the strong version presented in Equation (2.5). Let $\boldsymbol{w}$ be a test function, then in every point of $\mathcal{B}$ we have

$$(\operatorname{div} \mathbf{T}) \cdot \boldsymbol{w} + \mathbf{f} \cdot \boldsymbol{w} = \rho \mathbf{p}_{tt} \cdot \boldsymbol{w}. \tag{2.36}$$

Since the divergence satisfies

$$\operatorname{div}(\mathbf{A} \cdot \mathbf{b}) = \mathbf{A} : \operatorname{grad} \mathbf{b} + \operatorname{div}(\mathbf{A}) \cdot \mathbf{b}, \qquad \mathbf{A} \in \mathrm{Lin}, \mathbf{b} \in \mathbb{R}^3,$$

we can integrate (2.36) over $\mathcal{B}$ and obtain

$$\int_{\mathcal{B}} \operatorname{div}(\mathbf{T} \cdot \boldsymbol{w})\, dv(\boldsymbol{z}) - \int_{\mathcal{B}} \mathbf{T} : \operatorname{grad} \boldsymbol{w}\, dv(\boldsymbol{z}) + \int_{\mathcal{B}} \mathbf{f} \cdot \boldsymbol{w}\, dv(\boldsymbol{z}) = \int_{\mathcal{B}} \rho \mathbf{p}_{tt} \cdot \boldsymbol{w}\, dv(\boldsymbol{z}).$$

The first term may be rewritten using Green's theorem from Equation (A.7), yielding

$$\int_{\partial\mathcal{B}} \mathbf{t} \cdot \boldsymbol{w}\, dv(\boldsymbol{z}) - \int_{\mathcal{B}} \mathbf{T} : \operatorname{grad} \boldsymbol{w}\, dv(\boldsymbol{z}) + \int_{\mathcal{B}} \mathbf{f} \cdot \boldsymbol{w}\, dv(\boldsymbol{z}) = \int_{\mathcal{B}} \rho \mathbf{p}_{tt} \cdot \boldsymbol{w}\, dv(\boldsymbol{z}). \tag{2.37}$$

If we discretize the shape functions and test functions with the same finite element space, then the second term may be computed element by element. For a linearly interpolated tetrahedron $\mathbf{F}$ (and therefore $\mathbf{T}$) and $\operatorname{grad} \boldsymbol{w}$ are constant across a tetrahedron, so we have

$$\int_{\tau} \mathbf{T} : \operatorname{grad} \boldsymbol{w}\, dv(\boldsymbol{z}) = v(\tau)(\mathbf{T} : \operatorname{grad} \boldsymbol{w}),$$

where $v(\tau)$ is the volume of τ in the reference configuration.

Let $\mathbf{q}$ be an element of a nodal basis: $\mathbf{q} = \mathbf{e}_k$ in node j of the tetrahedron and 0 in others. According to (2.34), we have grad $\mathbf{q} = \mathbf{Q} \cdot \mathbf{Z}^{-1}$. We get

$$\begin{aligned} \mathbf{T} : \text{grad}\, \mathbf{q} &= \mathbf{T} : \mathbf{Q}\mathbf{Z}^{-1} \\ &= \mathbf{T} \cdot \mathbf{Z}^{-*} : \mathbf{Q} \\ &= \begin{cases} \mathbf{T} \cdot \mathbf{Z}^{-*} : \mathbf{e}_k \otimes \mathbf{e}_j & j \leq 3 \\ -\mathbf{T} \cdot \mathbf{Z}^{-*} : \mathbf{e}_k \otimes (\mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3) & j = 4. \end{cases} \end{aligned}$$

Here $(\mathbf{Z}^{-1})^*$ is denoted with $\mathbf{Z}^{-*}$. We can interpret this as follows: the elastic forces $\mathbf{f}_1, \ldots, \mathbf{f}_3$ on nodes $1, 2$ and 3 are given by the columns of the matrix representation of

$$-\nu(\tau)(\mathbf{T} \cdot \mathbf{Z}^{-*}) \tag{2.38}$$

with respect to the unit basis. The elastic force on node 4 is $-\mathbf{f}_1 - \mathbf{f}_2 - \mathbf{f}_3$, which implies that the tetrahedron is in equilibrium.

When we assume a body made of hyperelastic material in equilibrium, then we can put $\mathbf{p}_{tt} = 0$. The left hand side of Equation (2.37) is the directional derivative of the potential energy of the system. If we set $\hat{W}(\boldsymbol{z}) = W(\mathbf{C}(\boldsymbol{z}), \boldsymbol{z})$, then that equation is equivalent to

$$\frac{\partial}{\partial \varepsilon} \left(\int_{\mathcal{B}} \hat{W}(\mathbf{p} + \varepsilon \boldsymbol{w})\, dv - \int_{\mathcal{B}} (\mathbf{p} + \varepsilon \boldsymbol{w}) \cdot \mathbf{f}\, dv - \int_{\partial \mathcal{B}} (\mathbf{p} + \varepsilon \boldsymbol{w}) \cdot \mathbf{t}\, dv \right) = 0. \tag{2.39}$$

In other words, the potential energy ("virtual work") is stationary in $\mathbf{p}$, since moving by an infinitesimal motion $\varepsilon \boldsymbol{w}$ does not change the potential energy of the system.

2.4 Unconstrained optimization

Equation (2.39) describes the solution of a static problem as the stationary point of a virtual work function. Assuming that these stationary points are stable, it follows that we can find equilibrium solutions to mechanical problems by finding the minimum of a virtual work function Π. Inspired by Equation (2.39), we define the potential energy of the system discretized by a FEM basis $\mathbf{u}_1, \ldots, \mathbf{u}_n$ as follows.

$$\begin{gathered} \Pi(\alpha_1, \ldots, \alpha_n) = \int_{\mathcal{B}} \hat{W}(\tilde{\mathbf{u}})\, dv - \int_{\mathcal{B}} (\tilde{\mathbf{u}}) \cdot \mathbf{f}\, dv - \int_{\partial \mathcal{B}} (\tilde{\mathbf{u}}) \cdot \mathbf{t}\, dv, \\ \text{where } \tilde{\mathbf{u}} = \sum_{j=1}^{n} \alpha_j \mathbf{u}_j. \end{gathered} \tag{2.40}$$

The static deformation problem may simply by formulated as finding the solution to

$$\min_{x \in \mathbb{R}^n} \Pi(x).$$

The field of unconstrained optimization studies the algorithms that are used to solve such systems. In this section we discuss three algorithms. The most basic problem is a quadratic Π, and the standard algorithm is the Conjugate Gradient algorithm. For more complex functions, the nonlinear Conjugate Gradient algorithm and Truncated Newton methods may be used, which are discussed in the following sections.

2.4.1 Linear Conjugate Gradient method

For linear elasticity problems, the stiffness matrix K is symmetric positive definite, and for such problems, the most popular optimization method to use is the conjugate gradient method (dubbed CG throughout this thesis). Suppose that we have the following functional $\Pi : \mathbb{R}^n \to \mathbb{R}$,

$$\Pi(x) = \frac{1}{2} x^T K x - b^T x, \tag{2.41}$$

$$\min_{x \in \mathbb{R}^n} \Pi(x), \tag{2.42}$$

where $K \in \mathbb{R}^{n \times n}$ is a given symmetric positive definite matrix, and $b \in \mathbb{R}^n$ a given vector. This corresponds with the potential energy in a linear elastic FEM problem. The minimum is given by an x for which the gradient $\partial\Pi/\partial x = Kx - b$ vanishes. The direction of steepest descent is the negative gradient $b - Kx$, which we call *residual.* It is denoted by r.

We define the following inner product and associated norm on $\mathbb{R}^n$. Let $H \in \mathbb{R}^{n \times n}$ be a positive definite symmetric matrix, then let

$$(x, y) = x^T y, \qquad (x, y)_H = (x, Hy), \tag{2.43}$$

$$\|x\| = \sqrt{(x, y)}, \qquad \|x\|_H = \sqrt{(x, y)_H}. \tag{2.44}$$

The norm $\| \cdot \|_K$ is also called the energy norm.

One basis for an iterative algorithm is the line-search model: starting from some approximation $x_k \in \mathbb{R}^n$, we obtain a new solution x_{k+1} by selecting a search direction $d_k \in \mathbb{R}^n$, and searching in that direction, i.e.

$$x_{k+1} = x_k + \alpha_k d_k,$$

The choice for α_k more or less follows from the search direction d_k chosen, so an iterative optimization algorithm is characterized by its choice for d_k.

The conjugate gradient algorithm is the most popular algorithm for the case that K is positive definite. It computes the search direction d_k as a combination of the last search direction d_{k-1} and the current residual. The CG algorithm is given by the following pseudo code, which assumes that a starting solution x_0 is known.

```
$k \leftarrow 0$
$r_0 \leftarrow b - Kx_0$
while $\|r_k\|$ too large:
  if $k = 0$:
    $\beta_k \leftarrow 0$
  else:
    $\beta_k \leftarrow \|r_k\|^2 / \|r_{k-1}\|^2$
  $d_k \leftarrow r_k + \beta_k d_{k-1}$
  $\alpha_k \leftarrow \|r_k\|^2 / (d_k, Kd_k)$
  $x_{k+1} \leftarrow x_k + \alpha_k d_k$
```

```
    $r_{k+1} \leftarrow r_k - \alpha_k K d_k$
    $k \leftarrow k+1$
```

Notice that we can find α_k and update the residual using only one matrix-vector product. The vectors are indexed for clarity, but it is not necessary to store the vectors for each step separately. The CG algorithm can be implemented with four vectors of storage.

The convergence characteristics of CG is a well-researched [97]. Here, we sketch the convergence analysis of CG following the exposition by Axelsson and Barker [6]. The sequence of r_k, d_k and x_k generated by this algorithm satisfies the following properties:

- The residuals are orthogonal: if $i \neq j$, then $(r_i, r_j) = 0$.
- The search directions are K-orthogonal, or K-*conjugate*: if $i \neq j$ then $(d_i, d_j)_K = 0$.
- The sequence of r_k is optimal in the following sense: let

  $$W_k := \operatorname{span}\left\{K^1 r_0, \dots, K^k r_0\right\},$$

  then $\|r_k\|_{K^{-1}} = \min_{r \in r_0 + W_k} \|r\|_{K^{-1}}$, or equivalently, x_k minimizes $\|Kx - b\|_{K^{-1}}$ over $x \in x_0 + K^{-1} W_k$.

  Let $\hat{x}$ be the exact solution to the problem, then $\|x_k - \hat{x}\|_K = \|r_k\|_{K^{-1}}$, so CG minimizes the solution error in the energy norm in each step.
- In exact arithmetic, the algorithm converges in at most n steps, otherwise the set $\{r_0, \dots, r_n\}$ would be a set of $n+1$ orthogonal non-zero vectors.

Since $K^j r_0$ is a polynomial of K applied to r_0, the space W_k can be written in terms of polynomials. Let $\mathbb{P}_l$ be the space of polynomials of degree at most l, then we have

$$W_k = \{\, q(K) K r_0 : q \in \mathbb{P}_{k-1} \,\}.$$

Similarly, we can rephrase the fact that r_k is chosen optimally from $r_0 + W_k$ as follows:

$$\|r_k\|_{K^{-1}} = \min_{r \in r_0 + W_k} \|r\|_{K^{-1}} \tag{2.45}$$

$$= \min_{p \in \mathbb{P}_k, P(0)=1} \|P(K) r_0\|_{K^{-1}}. \tag{2.46}$$

The matrix K is symmetric and positive definite, so it has orthonormal eigenvectors v_i for $i = 1, \dots, n$ with eigenvalues $0 < \lambda_1 \leq \lambda_2 \cdots \leq \lambda_n$. We can expand r_0 in eigenvectors,

$$r_0 = \sum_i (v_i, r_0) v_i,$$

and rewrite (2.46) as

$$\min_{p \in \mathbb{P}_k, p(0)=1} \sqrt{\sum_i (v_i, r_0)^2 p(\lambda_i)^2 / \lambda_i}.$$

If we can bound $|p(\lambda_j)|$ for $j = 1, \ldots, n$, then this minimum is also bounded: suppose that $|p(\lambda_j)| \leq M$ for $j = 1, \ldots, n$, then

$$\sqrt{\sum_i (v_i, r_0)^2 p(\lambda_i)^2 / \lambda_i} \leq M \sqrt{\sum_i (v_i, r_0)^2 / \lambda_i} = M \|r_0\|_{K^{-1}}.$$

In the general case, the eigenvalues of K are distributed in the interval $[\lambda_1, \lambda_n]$, and Chebychev polynomials give a bound on $|p(\lambda)|$ for $\lambda \in [\lambda_1, \lambda_n]$. This leads to the following estimate of $\|r_k\|_{K^{-1}}$:

$$\|r_k\|_{K^{-1}} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|r_0\|_{K^{-1}}, \qquad \kappa = \lambda_n / \lambda_1.$$

If we want to improve the starting solution x_0 by a factor ε, then the number of steps necessary is less than

$$\frac{1}{2} \sqrt{\kappa} \ln \frac{2}{\varepsilon} + 1, \qquad \kappa = \lambda_n / \lambda_1. \tag{2.47}$$

The number κ is the condition number of the matrix K, which also bounds the accuracy of a solution computed with finite precision arithmetic. A method to speed up the CG iteration is *preconditioning*. This method transforms the iteration so it runs on the related problem

$$\begin{aligned} E^{-1} K E^{-T} y &= c, \\ c &= E^{-1} b, \\ y &= E^T x. \end{aligned}$$

The condition number for this problem is $\text{cond}(E^{-1} K E^{-T}) = \text{cond}((EE^T)^{-1} K)$. If this condition number is lower than cond(K) then solving this problem requires less iterations. During every step, a system of the form $(EE^T)x = y$ must be solved, so if (EE^T) has a sufficiently simple form, then this will improve the performance of the CG algorithm.

If K has less than n eigenvalues, say $k < n$, then a kth degree polynomial q exists such that $q(\lambda_j) = 0$. The optimal r will be null, and $\|r_k\|_{K^{-1}} = 0$; in other words: the algorithm will converge within k steps. This observation illustrates that the exact distribution of the eigenvalues of K plays a large role in the convergence behavior. We give a final example which will become relevant in Chapter 4. Suppose that all eigenvalues but the m largest of K are bounded by some constant γ, then we have

$$\|r_k\|_{K^{-1}} \leq 2 \left(\frac{\sqrt{\kappa'} - 1}{\sqrt{\kappa'} + 1} \right)^{k-m} \|r_0\|_{K^{-1}},$$

and where $\kappa' = \gamma / \lambda_1$. The number of steps necessary for a factor ε reduction is less than

$$\frac{1}{2} \sqrt{\kappa'} \ln \left(\frac{2}{\varepsilon} \right) + m + 1.$$

In other words, the magnitude of a few extremely large but isolated eigenvalues does not affect the performance of CG.

2.4.2 Non-linear conjugate gradients

The basic iteration of CG consists of three steps: determining a new search direction, determining the optimal step, and finding the residual in the new point. There is nothing inherently quadratic about the last two steps, so if some search direction is given, then the CG algorithm can also be performed for non-quadratic Π. We refer to this algorithm as the nonlinear CG algorithm. It can be expressed as follows.

```
$k \leftarrow 0$
$r_0 \leftarrow -\operatorname{grad}\Pi(x_0)$
while $\|r_k\|$ is too large:
  select $d_k$
  find $\alpha_k$ such that $\Pi(x_k + \alpha_k d_k)$ minimal
  $x_{k+1} \leftarrow x_k + \alpha_k d_k$
  $r_{k+1} \leftarrow -\operatorname{grad}\Pi(x_{k+1})$
```

For selecting d_k, the conjugate gradient algorithm uses

$$d_k = \begin{cases} r_k, & k = 0, \\ r_k + \beta_k d_{k-1}, & k > 0. \end{cases}$$

For selecting β_k in the nonlinear case, a number of different recipes have been proposed: Fletcher-Reeves,

$$\beta_{FR} = \frac{\|r_k\|^2}{\|r_{k-1}\|}, \tag{2.48}$$

and Polak-Ribière

$$\beta_{PR} = \max\{0, \frac{r_k^T(r_k - r_{k-1})}{\|r_{k-1}\|}\}. \tag{2.49}$$

The convergence theory of linear CG algorithms is fairly complete. In contrast, little is known on the convergence for non-quadratic Π. Both β-selection strategies are equivalent with linear CG when applied to a quadratic Π, and they are known to converge to a stationary point if the line search for α_k is sufficiently precise [75]. In practice, Polak-Ribière is known to perform better than Fletcher-Reeves. This has been attributed to the fact that an unsuccessful Polak-Ribière step yields $\beta_{PR} \approx 0$, which in effect resets the search direction to the direction of steepest descent. However, a convincing explanation of the success of Polak-Ribière is still lacking [42].

2.4.3 Truncated Newton

A function attains an unconstrained optimum at a stationary point, i.e. when $\partial\Pi/\partial x = 0$. This is a nonlinear system of equations, and root-finding techniques can be used to solve it. The Newton-Raphson iteration is a classic method for finding roots of equations [41]. Let $K(x)$ denote the Hessian of the energy function Π in the point $x \in \mathbb{R}^n$, i.e.

$$K(x) = \frac{\partial^2\Pi}{\partial u^2}(x). \tag{2.50}$$

The algorithm can then be expressed as follows.

```
$k \leftarrow 0$
$r_k \leftarrow (\partial\Pi/\partial x)(x_0)$
while $r_k$ is too large:
   solve $K(x_0)d_k = r_k$      $(*)$
   $x_{k+1} \leftarrow x_k + d_k$.
   $r_{k+1} \leftarrow (\partial\Pi/\partial x)(x_{k+1})$
```

The solution step in $(*)$ can be done by a linear CG algorithm if $K(x_0)$ is a positive definite matrix. For elasticity problems, this is normally the case around stable equilibrium solutions. Newton-Raphson with CG as an inner loop is also called the *Truncated Newton* or *Truncated Conjugate Gradient* method. The precision of the solution for the inner loop is of minor importance, so low tolerances can be used [42].

The process has a quadratic convergence, i.e. if $\hat{x}$ is the exact solution to a problem, then

$$\|\hat{x} - x_{k+1}\| \leq c\|\hat{x} - x_k\|^2,$$

for some positive constant c: the process converges superlinearly. This is attractive when high precision of the solution is required: close to a solution, the number of correct digits doubles at each step. The Newton-Raphson iteration is characteristic of superlinear convergence: any algorithm that converges superlinearly must have search directions that approximate Newton search directions [33]. In Chapter 4, the Truncated Newton method will be used to compute solutions with high precision.

2.5 Time integration

Equation (2.37) involves $\mathbf{p}_{tt}$, so it is time-dependent, and $\mathbf{u}$ also depends on time. When an approximation $\tilde{\mathbf{u}}$ of $\mathbf{u}$ is expanded in shape functions $\boldsymbol{w}_i$, for $i = 1, \ldots, n$, then the coefficients are also time dependent:

$$\tilde{\mathbf{u}}(\boldsymbol{z}, t) = \sum_i u_i(t)\boldsymbol{w}_i(\boldsymbol{z}).$$

The Finite Element method transforms the PDE into a system of ordinary differential equations (ODEs), and the solution process requires us to find the evolution of u over time:

$$s(u(\cdot), t) + f^{ex}(t) = M\ddot{u}. \tag{2.51}$$

The function $f^{ex}(t)$ represents body forces and tractions combined, and the $\mathbb{R}^{n\times n}$ matrix M is called *mass matrix*. For shape functions $(\boldsymbol{w}_j)_j$, it follows from

$$M_{ij} = \int_{\mathcal{B}} \boldsymbol{w}_i \cdot \boldsymbol{w}_j \rho \, dv. \tag{2.52}$$

The function s in (2.51) represents internal forces within the material. In a general mechanical formulation these forces may depend on the history of u. This leads to a *viscoelastic* problem. In hyperelastic materials internal stresses do not depend on history of u. Internal forces are elastic forces f^{el}, that depend on u at time t. Energy is generally dissipated by adding a friction term f^{fr} that depends on $\dot{u}$, yielding

$$f^{el}(u(t)) + f^{fr}(\dot{u}(t)) + f^{ex}(t) = M\ddot{u} \tag{2.53}$$

For the linear case (where both elastic and frictional forces are linear in u), a closed form analytic solution to (2.53) exists. For large problems or nonlinear problems, computing that solution is not possible or practical. In these cases, numerical methods must be used. A *numerical integration scheme* or *time integration scheme* computes the evolution of an approximate solution numerically. The process proceeds by advancing the time variable by an increment Δt, called the *time step*. By using Equation (2.51), the configuration at time $t + \Delta t$ is estimated given the situation at time t. If the recipe specifies $u(t+\Delta t)$ and $\dot{u}(t+\Delta t)$ as an unknown in a system of equations involving $f^{el}(u(t+\Delta t)$ then we call the method *implicit*. An *explicit* method uses the forces at time t to predict $u(t + \Delta t)$.

Computing the next result in an implicit method involves solving a large system, which is costly. Advancing a time step in an explicit methods requires much less calculations. The price paid for this simplicity is *conditional stability*. If the dynamic system includes phenomena which evolve quicker than the approximate solution itself, these will be mistaken for exponentially increasing components of the solution. This is called *instability*, and typically results in blow-up of the solution. Conditional stability for mechanical problems is expressed through the Courant-Friedrichs-Lewy criterion: the time step must be smaller than the critical time step Δt_{crit}:

$$\Delta t \leq \Delta t_{crit} \sim h/c \tag{2.54}$$

Here c is the wave speed in the medium, and h the element size. The quantity h/c is the time that a wave needs to propagate across an element of size h. The proportionality constant depends on the problem and the integration scheme used.

During the rest of the discussion, we will consider the linearized FEM equations. These can be specified as

$$M\ddot{u} + C\dot{u} + Ku - f^{ex} = 0, \tag{2.55}$$

The $\mathbb{R}^{n\times n}$ matrix C is the damping matrix. Measuring physically realistic values for C is hard, therefore C is set often set to a linear combination of M and K. This is called *Rayleigh-damping*.

We follow Chapter 17 from Zienkiewicz and Taylor [103] for discussing popular second order schemes for integrating FEM systems. These are the so-called Generalized Newmark (GN) methods and the related *weighted residual* (denoted by SS). Both approaches expand u in a truncated Taylor series in t, and estimate the highest order term using the differential equation. Both methods have similar precision and stability properties, and both can be formulated for j-th order problems, estimating the Taylor series up to the p-th derivative; the methods are denoted as $SSpj$ (for the weighted residual form) and $GNpj$ (for the Generalized Newmark).

The SS22 method is applicable to second order ODEs resulting from a FEM discretization. It estimates the coefficients of a Taylor series expansion of u up to order 2 around the chosen time t, and the accumulated error in u is $\mathcal{O}(\Delta t)$. It starts with estimating weighted averages of u and its derivative at time t

$$\bar{u} = u(t) + \theta_1 \Delta t \dot{u}(t), \tag{2.56}$$
$$\bar{\dot{u}} = \dot{u}. \tag{2.57}$$

The number θ_1 is a weighting parameter. The differential equation produces an equation that estimates the second derivative, weighed by θ_2,

$$(M + \theta_1 \Delta t C + K\frac{1}{2}\theta_2 \Delta t^2)\bar{\ddot{u}} + (C\bar{\dot{u}}) + K\bar{u} + \bar{f} = 0. \tag{2.58}$$

The estimated derivatives can then be used to find $u(t + \Delta t)$

$$u(t + \Delta t) = u(t) + \Delta t \dot{u}(t) + \frac{1}{2}\Delta t^2 \bar{\ddot{u}}, \tag{2.59}$$
$$\dot{u}(t + \Delta t) = \dot{u}(t) + \Delta t \bar{\ddot{u}}. \tag{2.60}$$

The properties of this method are controlled by the values of θ_1 and θ_2. If $\theta_2 = 0$, then we call the method explicit. This explicit method is only stable on the condition that $\theta_1 \geq 1/2$. When $\theta_1 > \frac{1}{2}$, then the $\Delta t_{\text{crit}} = \mathcal{O}(h^2)$, where h is the shortest edge length. This is undesirable, since small h values are necessary to obtain an accurate discretization. If $\theta_1 = \frac{1}{2}$, then

$$\Delta t_{\text{crit}} = \frac{2}{\sqrt{3}}\frac{h}{c}, \tag{2.61}$$

where $c = \sqrt{\frac{\lambda + 2\mu}{\rho}}$ is the wave speed in the medium.

The central difference method of integration has exactly the same stability and error properties as the explicit SS22 method with $\theta_1 = 1/2$. Instead of adding the velocity as an extra variable, it adds the position at time $t - \Delta t$ as a variable: it is a *multi-step* method. Derivatives in Equation (2.55) are replaced by explicit differences, yielding the equation:

$$M\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta t^2} + C\frac{u_{i+1} - u_{i-1}}{\Delta t} + Ku_i + f_i = 0, \tag{2.62}$$

where $u_k \in \mathbb{R}^n$ is approximation for the value of u at time $t + k\Delta t$. It is less trivial to change the value of the time-step during a simulation.

Both explicit SS22 and central differences are particularly efficient if M and C are diagonal. In this case, the mass and damping are redistributed to be concentrated in the nodes: node j has one quarter of the mass of all the tetrahedrons incident with j. This process is called *mass lumping*. If it is applied to the damping matrix, this is called *lumped damping*. Mass lumping is not strictly realistic, but it has been shown to yield precise results, and enlarges the critical time step. The SS22 scheme with lumped masses and lumped damping for nonlinear elasticity problems is examined in Chapter 4.

2.6 Mechanics of soft tissue

In the rest of the thesis we wil concentrate on compressible hyperelastic aelotropic material models. For living tissue this is a simplification of reality. In this section, we briefly explain what effects are neglected by this simplification. It is partially based on the book by Fung [44].

The term soft tissue includes a variety of tissue types in the body. The mechanical characteristics of this tissue are determined by connective tissue. The materials that contribute to the mechanics of the tissue include the following: *Elastine* is a rubbery biological material. Its loading and unloading cycles are almost equal, meaning that it is almost perfectly elastic. This material is found in elastic tissues, such as skin, artery walls, lung tissue. It also helps keep the skin smooth; elastin production stops after puberty. *Collagen* is a biological construction material. It forms the load bearing material in soft and hard tissue. It is a major component of tendons, bone, skin, and blood vessels. Collagen and elastine can form fibers. If these fibers have some dominant orientation, the tissue will behave differently in different directions. The material then is *aelotropic* or *anisotropic*. The fibers are suspended together with cells in a watery gel called *ground substance*. Since water is incompressible, many tissue types are also (nearly) incompressible.

The relation between load and deformation (stress and strain) follows the path shown in Figure 2.4. The stress response of biological tissue can be divided into three trajectories: at small loads (OA), the stress is exponential in the strain. For larger loads, the stress is linear in the strain (AB). Finally, in the third trajectory (BC), the tissue is almost stressed to failure, and reacts nonlinearly. Normal tissue loads fall into the first region.

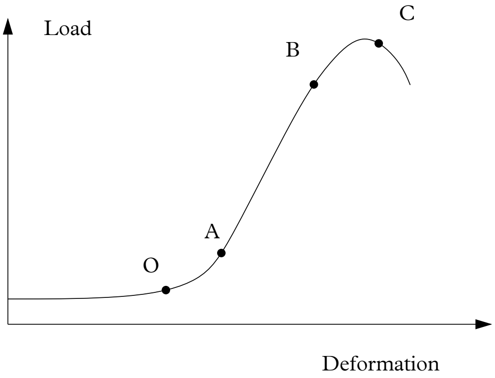


Figure 2.4: Stress response of a rabbit limb tendon, after Fung [44].

When tissue is stretched it offers more resistance than during a following unload. This phenomenon is called *hysteresis*, and it is an example of a viscoelastic effect: stresses in the material depend on the history of the deformation. When tissue is stressed with a constant load, then after the initial elastic response, the tissue will slowly distend further. This process is known as *creep*. A related phenomenon is *stress relax-*

ation: when a tissue specimen is loaded and then held at a constant elongation, stresses within the tissue decrease. This process is rather slow, taking minutes to many hours before a steady state is reached.

When tissue is loaded and then unloaded, its elastic properties change: the tissue becomes softer. When this cyclical loading is repeated often enough, the difference between the cycles disappears, and the deformation converges. The tissue is now said to be *preconditioned.* This processes is illustrated in Figure 2.5.

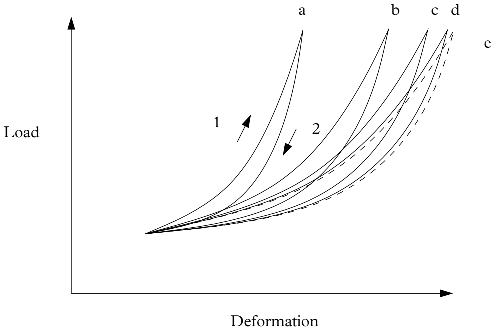


Figure 2.5: During loading (1), tissue offers more resistance than during unloading (2). If such a cycle is repeated, tissue will become softer (a, b, c, d) until the response converges (e).

The first completely 3D description of soft tissue elasticity was given by Veronda and Westman [98]. They used some simplifications to overcome the difficulties posed by the complex mechanical behavior of tissue. They have proposed constitutive equations of soft-tissue on the basis of measurements on cat skin. Anisotropy and viscoelastic effects were handled by measuring only the loading step during a uniaxial stretching test. This gave sufficient experimental data to derive an isotropic hyperelastic constitutive equation. Material parameters were extracted from the same data set by fitting the derived stress/strain curves to the data. The Veronda-Westmann material model will be used in Chapter 4.

A more accurate description is quasi-linear visco-elasticity [44], which accounts for visco-elastic effects by modeling tissue as a superposition of materials with different relaxation times. Kauer [57] used this model to measure tissue elasticity of ex-vivo pig kidney and in- and ex-vivo human uteri. This was done using aspiration experiments: during surgery, a tube was placed over the tissue to be measured. A partial vacuum was created which caused a bulge to form inside the tube. The evolution of this bulge was recorded with a camera, and stored. The conditions of the experiment were repeated in FEM simulation. Material parameters were determined by searching for those settings that recreated the experiment accurately.

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 matrix 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

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

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

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

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 $z_1, \ldots, z_4$, then $\mathbf{Z}$ is defined by

$$\mathbf{Z} = (z_1 - z_4) \otimes \boldsymbol{e}_1 + (z_2 - z_4) \otimes \boldsymbol{e}_2 + (z_3 - z_4) \otimes \boldsymbol{e}_3,$$

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

$$\mathbf{U} = (u_1 - u_4) \otimes \boldsymbol{e}_1 + (u_2 - u_4) \otimes \boldsymbol{e}_2 + (u_3 - u_4) \otimes \boldsymbol{e}_3.$$

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

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

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 $\mathbf{f}^{ex}$. For every node $i = 1, \ldots, m$, we should have

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

Since $\mathbf{f}^{\mathrm{el}}$ 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 $\mathbb{R}^{3m\times3m}$ 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

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

This introduces c constraints on all displacements, where $c = 3|N_{\mathrm{fix}}|$. 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 $\left\{ \mathbf{x} \in \mathbb{R}^3 : \mathbf{x} \cdot \mathbf{n} = \alpha \right\}$ 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

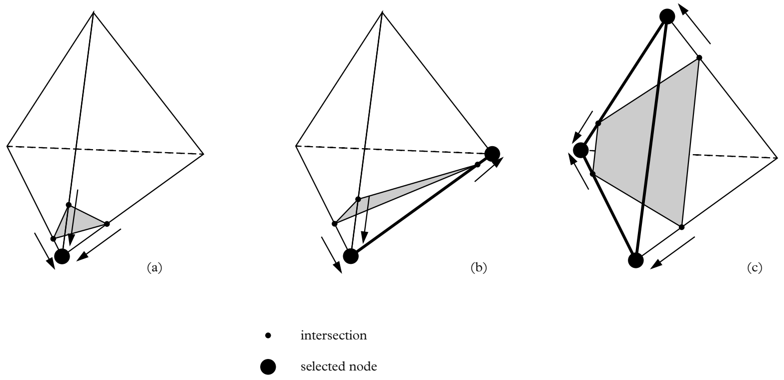


Figure 3.1: Surface selection by closest nodes from edge/sweep intersections. The sweep is colored grey, and the selected feature is marked by bold lines and dots. In Picture (a), (b) and (c) a node, an edge and a face are selected respectively.

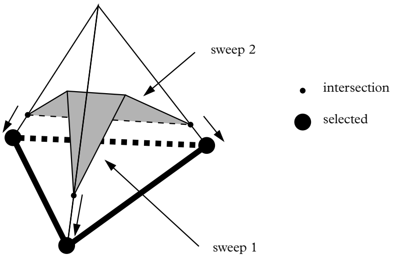


Figure 3.2: When processing multiple sweep triangles, intersections from different sweeps are combined.

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

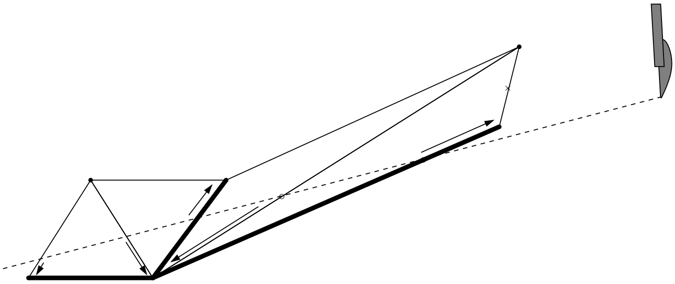


Figure 3.3: Cut surfaces may branch, shown in 2D. Selected faces (edges) are shown bold.

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.

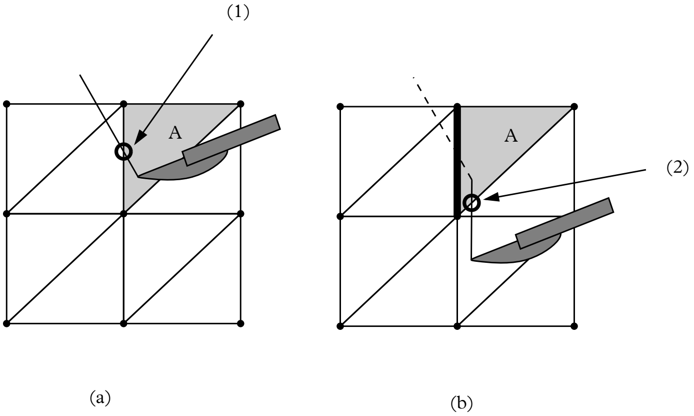


Figure 3.4: Selected faces (bold edges) lag behind the scalpel sweep, demonstrated in 2D. The bold edge in (b) is selected once intersection (2) has been found. This happens after the scalpel has left triangle A. Between steps (a) and (b), intersection (1) must be remembered.

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 ν 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 ν, the location of ν 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 ν 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 ν 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 ν 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

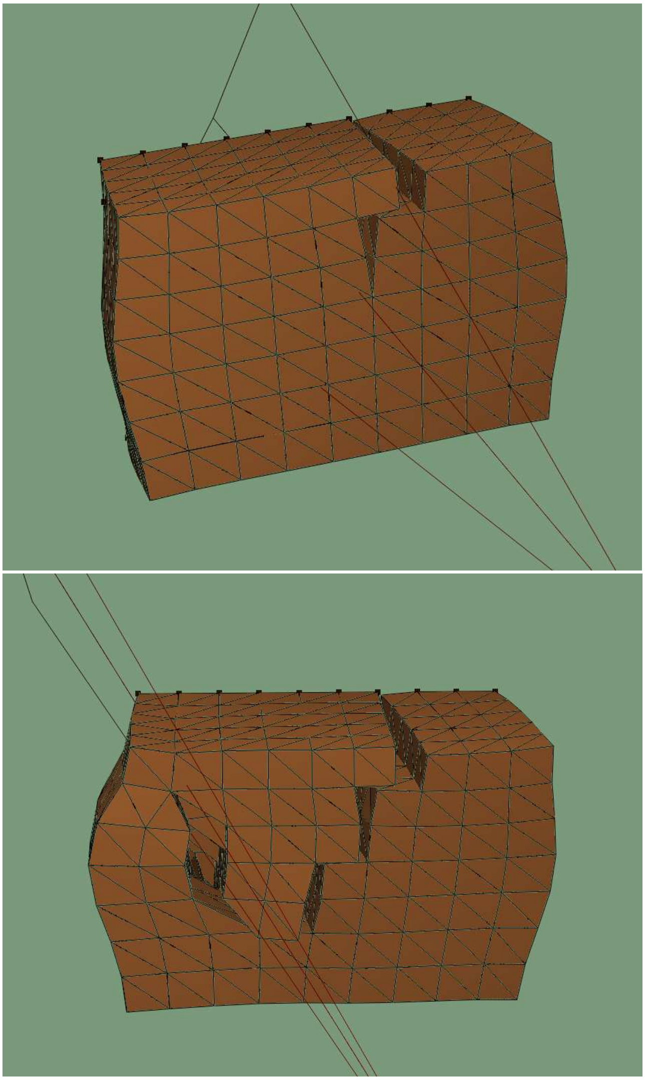


Figure 3.5: A cut in a generated object cube without node repositioning. The last and next sweep triangle are indicated by the two triangles partially penetrating the object.

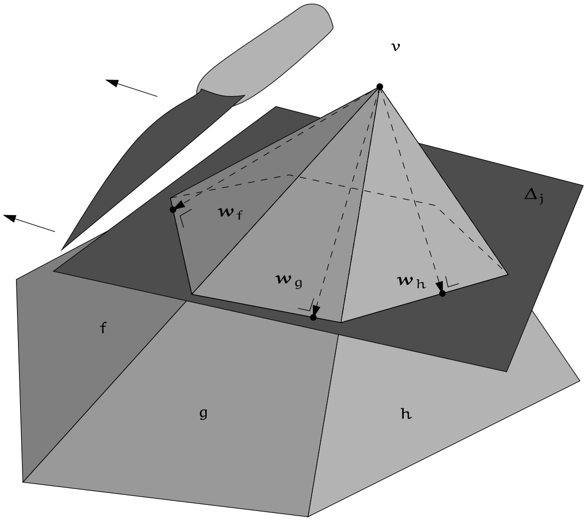


Figure 3.6: When repositioning a boundary node ν, it can be projected within faces f, g and h, leading to different points $\boldsymbol{w}_f$, $\boldsymbol{w}_g$ and $\boldsymbol{w}_h$. The closest $\boldsymbol{w}$ is selected.

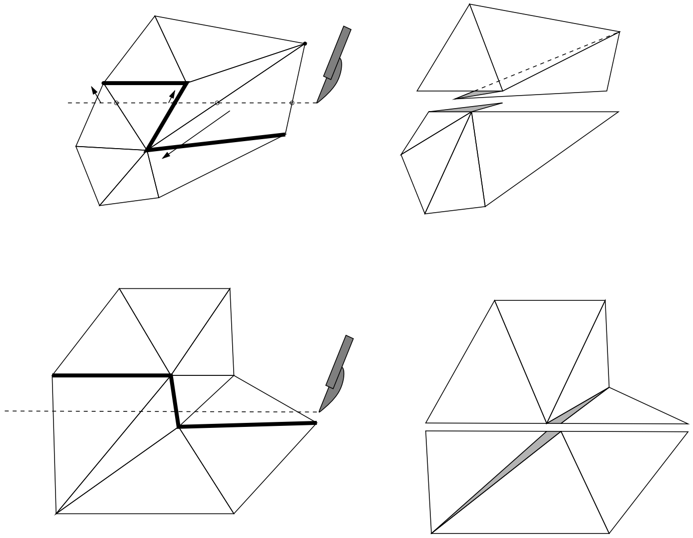


Figure 3.7: Projecting nodes may result in degeneracies, even in 2D. If all nodes of an element (top) are selected, this leads to degenerate and inverted elements. Selecting a face almost perpendicular to the scalpel sweep also leads to degenerate elements

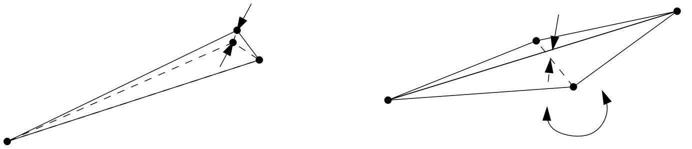
tetrahedron is represented by its set of nodes τ, and the supporting plane of a triangle σ by plane(σ) then

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

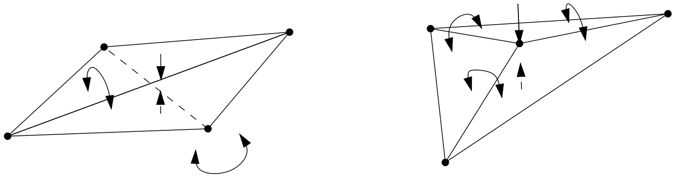
Flat elements come in different shapes, and they can be classified by their angles [10]. The strategy to collapse an element is determined by its shape, as measured by the number of large dihedral angles it contains.

- 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.



needle spindle



sliver cap

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,

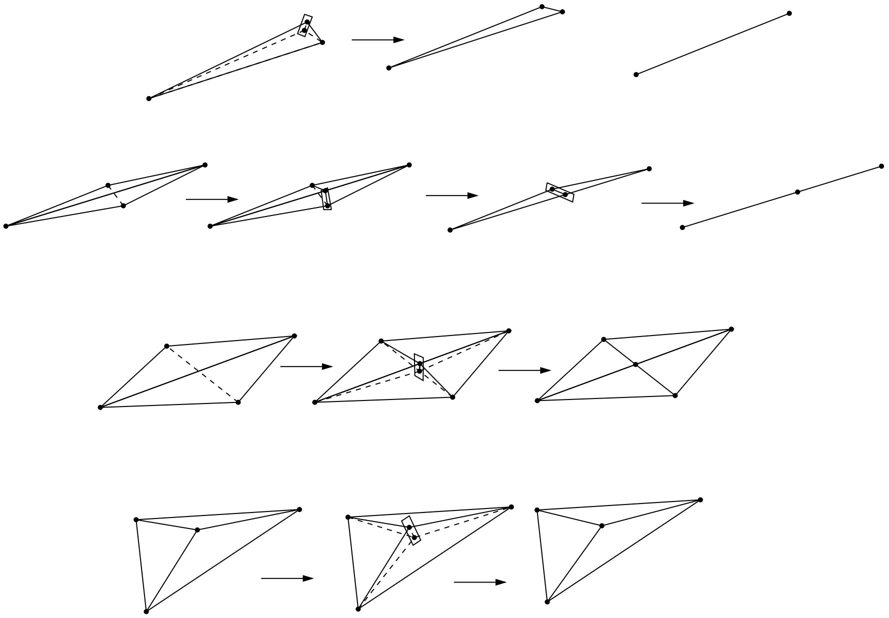


Figure 3.9: Collapsing elements by introducing short edges and collapsing them.

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.

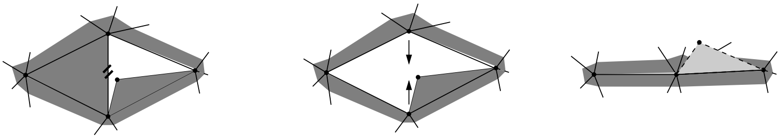


Figure 3.10: An impossible edge collapse in 2D. The edge marked with ticks is collapsed. This involves the triangles incident with the edge, and merging both nodes of the edge. In this case, the result of a collapse is no longer a manifold, as the edge marked in bold is incident with three triangles.

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 $\|r\|_2 < 10^{-8}\|f^{ex}\|$.

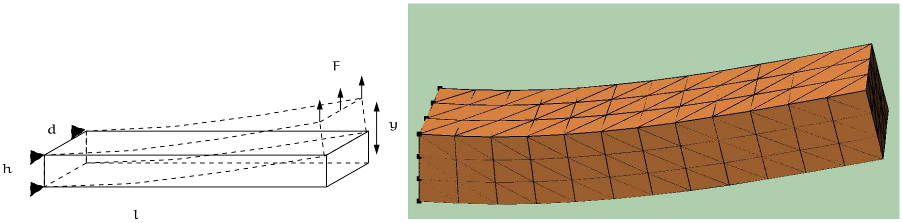


Figure 3.11: The cantilever beam experiment. On the right the result with forces exaggerated by a factor 10.

# 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.

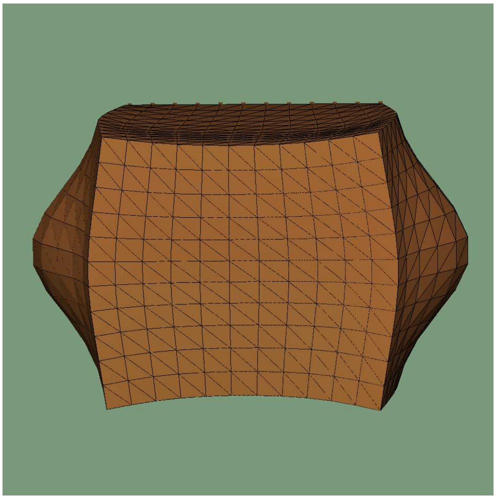


Figure 3.12: A generated cube of 7986 elements, with dilating forces applied to the side.

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).

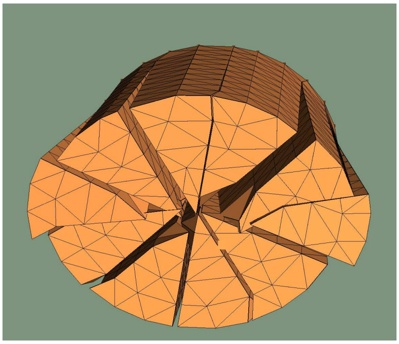


Figure 3.13: Pie-like cuts in a Delaunay Tetrahedralization of a cylindrical point cloud.

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

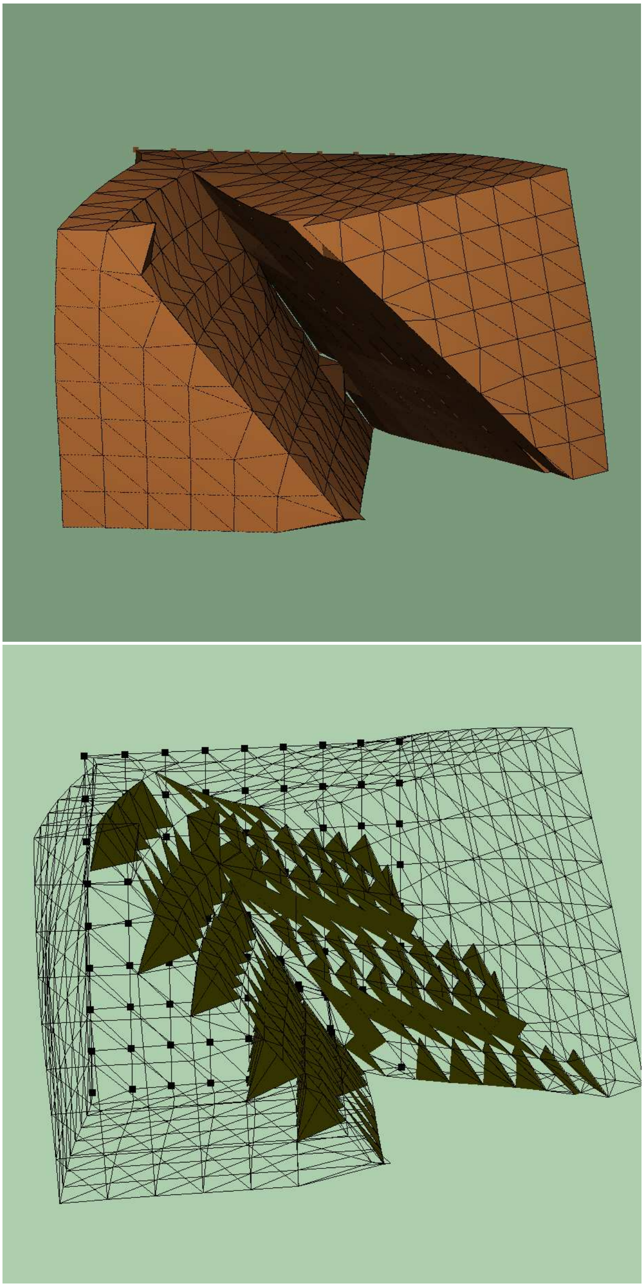


Figure 3.14: A large diagonal cut in a generated cube with back face fixed and dilating forces applied to the side faces. This cut generated 143 degeneracies. On the bottom the mesh is shown in wireframe mode, with only the degenerate tetrahedra filled in. Repositioning failed for two nodes, which are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

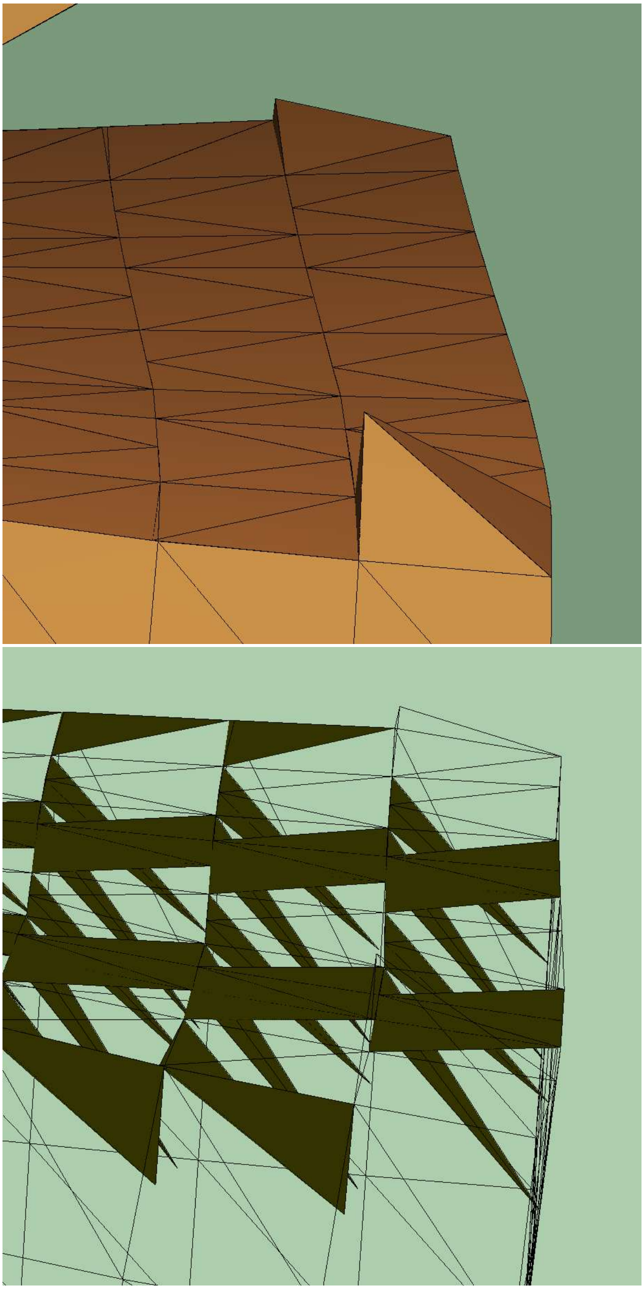


Figure 3.15: The same cut as Figure 3.14 from a different point of view.

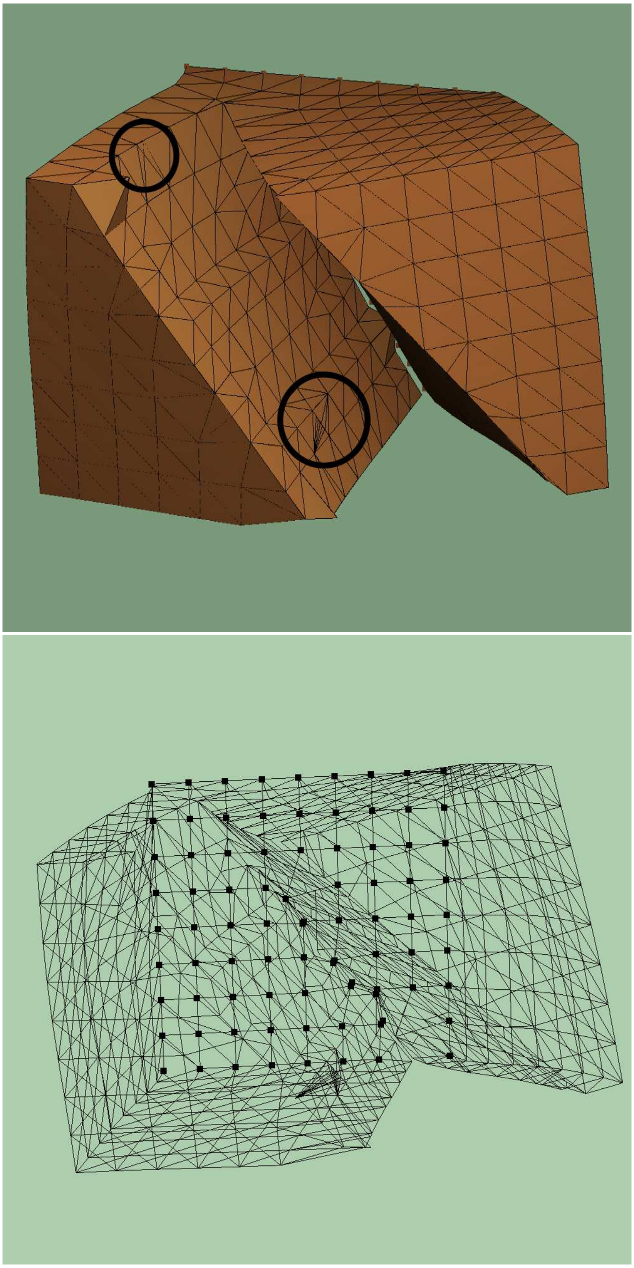


Figure 3.16: The same cut as Figure 3.14, but with degeneracies removed. Two artefacts of the removal process are circled. At the top we see an inverted element. At the bottom, an incollapsible degeneracy has lead to spurious subdivision before it was dissected. The incollapsible degeneracy has also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

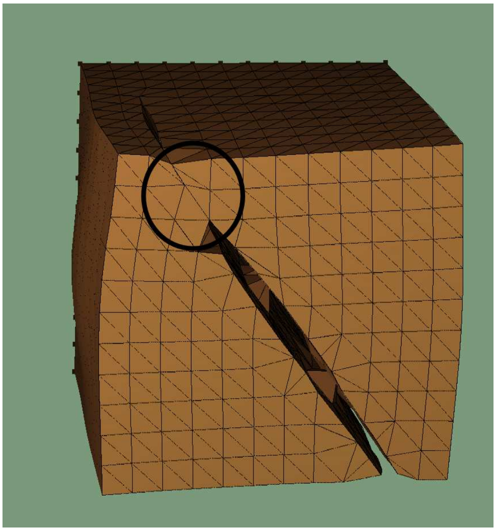


Figure 3.17: The same cut as Figure 3.14, but made in several small steps. Notice how both sides of the cut remain attached.

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.

Chapter 4

Relaxation algorithms

This chapter is intended as an expansion of the work of Chapter 3, where we have described our first steps into interactive deformation modeling. Our first approach is a completely linear model with an iterative solution based on the Conjugate Gradient algorithm. We have shown how mesh modification and deformation are easily combined with this method. However, linear elasticity has its limits: it assumes that deformations remain small. This assumption is questionable for soft material, such as soft tissue.

Other work in deformable objects primarily uses dynamic methods to compute deformations. Such methods compute the evolution of deformations over time as they move to a steady state. They are perhaps easier to understand than iterative static methods, since all intermediate results have a physical interpretation. Since they compute more physically relevant information, one could also expect that they are more expensive than a static method.

This chapter addresses both the extension to nonlinear material and convergence speed in more detail. We will extend the deformation framework of the previous chapter to include nonlinear deformations and a dynamic formulation. Using this framework, we benchmark the convergence speed of a static algorithm by comparing it to a dynamic method applied to the same problem. The rest of this chapter starts with detailing theoretical convergence of a dynamic method, then it introduces the convergence experiment, material models, and finally it shows and discusses the results.

4.1 Convergence of dynamic relaxation

The theoretical convergence speed of Conjugate Gradients (CG) has been analyzed extensively in literature, and was discussed in Section 2.4. In this section, we briefly analyze the convergence speed of dynamic relaxation in the case of linear elasticity. We will show how quickly a dynamic method will settle into a steady state, and find that the convergence speed of the dynamic problem is similar to that of CG.

We recall from (2.55) that the PDE for linear elasticity can be discretized into the following n-dimensional differential equation for the function $u(t) \in \mathbb{R}^n$

$$M\ddot{u} + C\dot{u} + Ku + f^{ex} = 0.$$

Here f^{ex} represents the external force, $M \in \mathbb{R}^{n \times n}$ is the mass matrix, representing the inertia of the object, and $C \in \mathbb{R}^{n \times n}$ the damping matrix, and $K \in \mathbb{R}^{n \times n}$ the stiffness matrix. The integration methods in (2.60) and (2.62) require diagonal M and C matrices for efficient time-stepping, so we use lumped masses. Since C must also be diagonal, we take $C = \eta M$ for some constant $\eta > 0$. This is a form of Rayleigh damping.

This dynamic solution has two parameters: η controls the amount of damping, and Δt is the time step of the integration scheme. Both parameters influence the speed of convergence towards the steady state. We want to determine how quickly this dynamic method reaches the steady state, so that the parameters η and Δt have to be chosen optimally. We determine these optimal parameters by analyzing the evolution of the solution error e over time. We define the error e in a solution u as being the difference between u and a static solution u_{static}. We have $Ku_{static} = f^{ex}$, so the error $e = u - u_{\text{static}}$ satisfies the homogeneous differential equation

$$M\ddot{e} + \eta M\dot{e} + Ke = 0. \tag{4.1}$$

Solutions of this equation are expressed in terms of generalized eigenvalues of M and K, i.e. solutions to

$$Kw = \lambda Mw$$

Both K and M are symmetric and positive definite, so this generalized eigenvalue problem has M-orthogonal eigenvectors with positive eigenvalues. There are eigenpairs (w_i, λ_i) from $\mathbb{R}^n \times \mathbb{R}^+$, such that $Kw_i = \lambda_i M w_i$ for $i = 1, \dots, n$. Since K and M are symmetric, the eigenvectors can be chosen to be M-orthogonal. In addition, the eigenvectors w_j can be normalized, so that we have

$$(w_i, w_j)_M = \begin{cases} 0 & i \neq j, \\ 1 & i = j. \end{cases}$$

This expression uses the notation from Equation (2.44).

The vectors w_i represent normalized undamped vibration modes of the body, and form an M-orthonormal basis of $\mathbb{R}^n$. Therefore, we can decompose e into the eigenvectors w_i, writing

$$e(t) = \sum_j y_j(t) w_j, \qquad y_i(t) = (e(t), w_i)_M$$

Since K and M are positive definite, the eigenvalues are positive. We order the eigenvalues, so $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

Analogous to Subsection 2.4.1, we analyse the error in the energy norm, which is given by $\|e\|_K$. Due to the M-orthogonality of the w_j, we find

$$\|e\|_K = \sqrt{\sum_j y_j(t)^2 \lambda_j}.$$

In other words, the solution error can be decomposed in its modal components. By taking the M-inner product of (4.1) and a vibration mode w_j, we get the following differential equation for the component y_j:

$$\ddot{y}_j(t) + \eta \dot{y}_j(t) + \lambda_j y_j(t) = 0. \tag{4.2}$$

This is a differential equation with constant coefficients. There are three cases for the general solution: the vibration is either underdamped, critically damped or overdamped. Let

$$\mu = -\eta/2,$$
$$\omega_j = \frac{1}{2}\sqrt{|4\lambda_j - \eta^2|}.$$

If $\eta < 2\sqrt{\lambda_j}$, then the system is underdamped, and solutions take the form of

$$y_j(t) = c_{1j} e^{\mu t} \sin(\omega_j t + \varphi_j), \qquad c_{1j}, \varphi_j \in \mathbb{R}.$$

If $\eta = 2\sqrt{\lambda_j}$, then the system is critically damped, and solutions take the form of

$$y_j(t) = (c_{1j} + c_{2j} t) e^{\mu t}, \qquad c_{1j}, c_{2j} \in \mathbb{R}.$$

If $\eta > 2\sqrt{\lambda_j}$, then the system is overdamped, and solutions take the form of

$$y_j(t) = c_{1j} e^{(\mu + \omega_j)t} + c_{2j} e^{(\mu - \omega_j)t}, \qquad c_{1j}, c_{2j} \in \mathbb{R}.$$

Graphs of these three cases are shown in Figure 4.1.

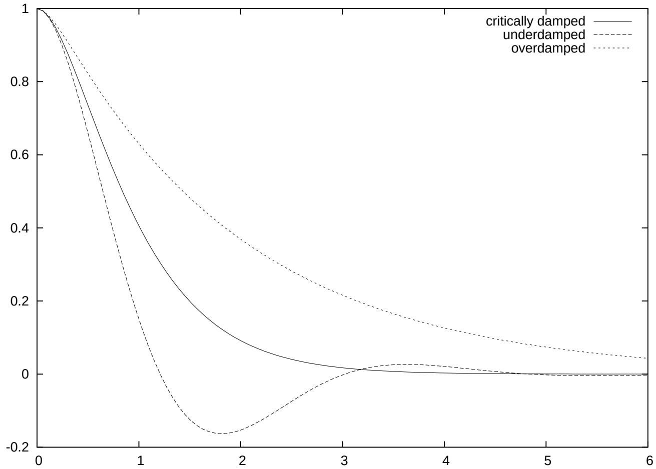


Figure 4.1: Three types of damping demonstrated for Equation (4.2), with critical damping, and $\eta = 1/2\eta_{\text{crit}}$ and $\eta = 2\eta_{\text{crit}}$. Begin values are $\lambda_j = 4$, $y_j(0) = 1$, $\dot{y}_j(0) = 0$.

We see that all modal components of the error diminish over time by $e^{-\eta t/2}$ in the underdamped and critically damped case. If η is larger than $2\sqrt{\lambda_j}$ for any j, then that mode is overdamped, and the corresponding error component will diminish by $e^{-(\eta/2-\omega_j)t}$, which is slower than $e^{-\eta t/2}$. Therefore, the quickest convergence is attained when η is as large as possible, but no mode is overdamped. This is when $\eta = 2\sqrt{\lambda_1}$.

In this case, we have

$$\|e\|_K = e^{-\eta t/2}\sqrt{\sum_j \lambda_j \tilde{y}_j^2(t)}, \qquad \tilde{y}_j(t) = e^{\eta t/2} y_j(t)$$

The contents of the square root are $\mathcal{O}(t)$, so the error is dominated by the exponential term $e^{-\eta t/2}$. Hence, when the equation is integrated over a time span T, then the magnitude all modal components decreases by $e^{-\eta T/2}$. For a reduction ε in error, we have to integrate over a fixed time span

$$T = \frac{-2\ln\varepsilon}{\eta} = \frac{-\ln\varepsilon}{\sqrt{\lambda_1}}.$$

The stability condition of the SS22 and related explicit second order integration methods for (4.2) is given by Zienkiewicz [103]: the time step Δt must satisfy

$$\Delta t^2 \le \frac{4}{\lambda_j}, \qquad j = 1, \dots, n.$$

The highest frequency mode is given by the largest eigenvalue λ_n, and this mode must also be stable, so we have

$$\Delta t \le \frac{2}{\sqrt{\lambda_n}}.$$

If a modal component y_j is to decrease by a factor ε, then this takes at least N time steps, where

$$\begin{aligned} N &= \frac{T}{\Delta t} \\ &\ge \frac{-\ln(\varepsilon)\sqrt{\lambda_n}}{2\sqrt{\lambda_1}} \\ &= \frac{1}{2}\ln\left(\frac{1}{\varepsilon}\right)\sqrt{\kappa}, \qquad \kappa = \lambda_n/\lambda_1 = \mathrm{cond}_2(M^{-1}K) \end{aligned}$$

Recalling Equation (2.47), we see that CG and dynamic relaxation offer similar performance in the linear case: the condition number of K determines the convergence speed. The effect of the mass matrix M is that of a preconditioner: if M were variable, and could be selected to decrease $\mathrm{cond}_2(M^{-1}K)$, then larger time steps could be taken, leading to more rapid convergence. This "preconditioning" has a physical interpretation: when a discretisation has both small and large elements, increasing nodal masses of small elements decreases their vibration frequencies, thus it brings down λ_n. For a system with lumped masses, M is diagonal, so if we view M as a preconditioner, then increasing nodal masses is analogous to preconditioning with a diagonal matrix.

parameter	notation	value
gravity	g	$9.8\ \mathrm{m/s^2}$
density	ρ	$1000\ \mathrm{kg/m^3}$
Young modulus	E	$1.0 \cdot 10^4 \mathrm{Pa}$
Poisson ratio	ν	0.3
Material nonlinearity	γ	8

Table 4.1: Material parameters and constants for the experiments.

4.2 Experimental setup

Subsection 2.4.1 and 4.1 show that on theoretical grounds CG and dynamic relaxation have the same convergence speed. However, the estimate for CG is not tight. Moreover, the linear analysis does not necessarily extend to nonlinear problems. In order to assess the speed of both algorithms in practice, their convergence in terms of computational cost has to be measured when applied in a practical situation. In this section we will discuss the experimental setting and how convergence and computational cost are measured.

The test object is a horizontal cylinder of very soft material, fixed on one end. At the start of the experiment, the gravity force is applied, and the object moves to a new equilibrium state. We measure how quickly it reaches that state. Material parameters and constants are in Table 4.1. These parameters are in the same order of magnitude as a very soft tissue [7, 57]. The undeformed configuration of the cylinder is shown in Figure 4.2. Cantilever beams of soft material easily lead to large deformations, so they test the performance on nonlinear problems. Moreover elongated structures are also present in the human body, for example, in skeletal muscles and tendons.

The object is meshed using a Delaunay tetrahedrization [69] of cylindrical point clouds. We use two meshes, a coarse mesh of 1230 elements and a more fine grained mesh of 9300 elements. Properties of the meshes used are listed in Table 4.2. The meshes are very well-shaped: they have no extreme element sizes, and no extreme angles. It is unlikely that this quality can be maintained for unstructured meshes during online changes. To assess the impact of mesh quality deterioration, we will examine the influence of edge lengths on relaxation

The computational cost of the solution process iteration is measured in *flops*, floating point operations. During the computation, a flop count is maintained. The flop count per tetrahedron was manually determined for every material model. The resulting counts are shown in Table 4.2. During the computation, these numbers are added in a global variable. This flop count is independent of machine, compiler and timer resolution, and is not affected by any overhead of measuring the performance. Multiplications, divisions, sums and differences were counted as one flop, and 1 MFLOP = 10^6 flop. Compared to these counts, the exponential function was measured to take approximately 50 flops. Another instance of the program runs the same experiment with statistics turned off and maximum optimization settings, to determine the speed of the program in flops per second. By combining both numbers, the computional cost

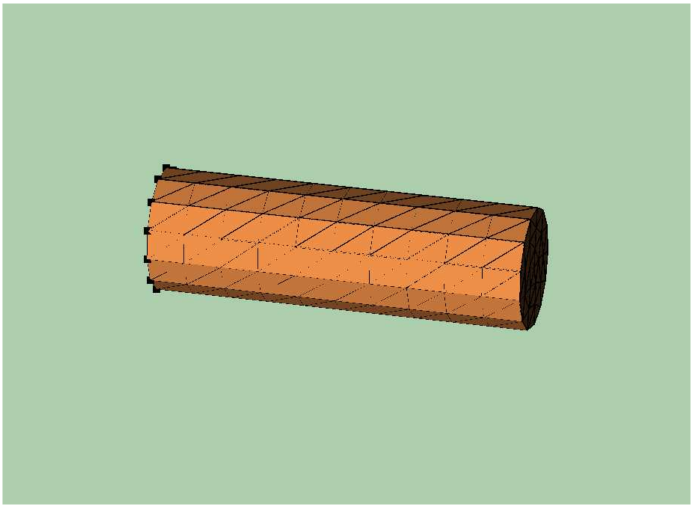


Figure 4.2: The cantilever beam, in undeformed configuration

	small mesh	large mesh
Mesh type	Delaunay	idem
Rod length	0.1m	idem
Rod radius	0.03m	idem
Elements	1230	9300
Nodes	308	1911
Edge lengths	0.05 –0.013m	0.0067 – 0.0025m.
Dihedral angles	20° – 140°	idem

Table 4.2: Geometry of the test input

can be expressed in seconds of computation time.

The rate of convergence was determined by comparing the approximation with an "exact" solution, a solution computed with a smaller error tolerance. This solution was obtained in a two step process first, a nonlinear CG iteration was used to find an approximate solution, such that the residual r satisfies $\|r\|_2 \leq 10^{-2}\|f^{ex}\|_2$. Then a truncated Newton-Raphson algorithm (discussed in Section 2.4.3) was used to obtain a solution such that $\|r\|_2 \leq 10^{-8}\|f^{ex}\|_2$ (except for the linear problem, where the tolerance was set at 10^{-12}).

Suppose that the exact solution of the problem is $\hat{u} \in \mathbb{R}^n$, and at some point, the error is $u - \hat{u} = e \in \mathbb{R}^n$. Hyperelastic mechanical problems are energy minimization problems, so we measure the error with the energy difference between the approximation and the 'exact' solution, i.e. the *energy error* $\Pi(u) - \Pi(\hat{u})$. When e is small, then we can rewrite this to

$$\begin{aligned}\Pi(u) - \Pi(\hat{u}) &= \Pi(\hat{u} + e) - \Pi(\hat{u}) \\ &= ((\partial\Pi/\partial u)(\hat{u}), e) + (K(\hat{u})e, e) + \mathcal{O}(\|e\|^3) \\ &= (e, e)_{K(\hat{u})} + \mathcal{O}(\|e\|^3).\end{aligned}$$

The first term of the last expression is an approximation of the energy difference. Since this expression is less susceptible to rounding errors, we will use it for measuring the convergence.

4.3 Hyperelastic compressible materials

Previous work in soft tissue modeling and deformable object simulation shows a variety of different models in use, both for off-line and on-line simulation. Therefore we use a number of different material models, which are discussed in this section. All of these are compressible, isotropic, hyperelastic models. We recall from Equation (2.12) and the discussion surrounding it, that hyperelastic models are defined by an energy density W, which depends on the three invariants ι_1, ι_2 and ι_3 of the Green deformation tensor $\mathbf{C}$. Some forms of anisotropy can also be added to hyperelastic models, by introducing other types of dependencies in W [51, 78].

We recall from (2.15) that the second Piola-Kirchoff stress tensor $\mathbf{S}$ for hyperelastic materials is given by

$$\mathbf{S} = 2\frac{\partial W}{\partial \mathbf{C}},$$

and elastic forces for the nodes of a tetrahedron are given in (2.35): they are represented in the 2-tensor

$$-\mathsf{T} \cdot \mathsf{Z}^{-*} = -\mathsf{F} \cdot \mathbf{S} \cdot \mathsf{Z}^{-*}. \tag{4.3}$$

In this expression Z is the tensor defined in (2.32). It represents the shape of the tetrahedron.

For Newton-Raphson methods we will also need the derivative of the nodal forces, relative to the tensor U representing node displacements. The derivative of the nodal forces can be expressed as a 4-tensor, a linear map that takes 2-tensors to 2-tensors.

It can be computed from (4.3) by applying the product rule, leading to the following derivative.

$$\mathsf{H} \mapsto \left(\mathsf{H} \cdot \mathsf{Z}^{-1} \cdot \mathsf{S} \cdot + \mathsf{F} \cdot \left(\frac{\partial \mathsf{S}}{\partial \mathsf{u}} : \mathsf{H}\right)\right) \cdot \mathsf{Z}^{-*}, \qquad \mathbf{H} \in \mathrm{Lin}\,. \tag{4.4}$$

The derivative of $\mathbf{S}$ is given by

$$\frac{\partial \mathsf{S}}{\partial \mathsf{u}} = \frac{\partial \mathsf{S}}{\partial \mathsf{C}} : \frac{\partial \mathsf{C}}{\partial \mathsf{u}},$$

and

$$\frac{\partial \mathsf{C}}{\partial \mathsf{u}} : \mathsf{H} = (\mathsf{H} \cdot \mathsf{Z}^{-1})^{*} \cdot \mathsf{F} + \mathsf{F}^{*} \cdot (\mathsf{H} \cdot \mathsf{Z}^{-1}).$$

Equation (4.4) includes $\mathbf{S}$, so when both the forces from (4.3) and their derivative from (4.4) are required, the calculations can be combined. Calculating both is only slightly more expensive than calculating the derivative only.

We assume that the reference configuration of the object is in a stress-free state, so $\mathbf{S} = \mathsf{0}$ when $\mathbf{C} = \mathbf{I}$. The function W represents potential energy, so we arbitrarily set $W = 0$ for $\mathbf{C} = \mathbf{I}$. We introduce the following models.

- St. Venant-Kirchoff material
- St. Venant-Kirchoff material with the linear geometry approximation
- neo-Hookean material
- Veronda-Westmann

The cost of computing an elastic force from the deformation of a tetrahedron varies across these models. The costs are listed in Table 4.3.

Model	Force	Derivative
Linear material/strain	129	129
St. Venant-Kirchoff	235	421
neo-Hookean	277	595
Veronda-Westmann	347	797

Table 4.3: Cost in flops of computing elastic forces and their derivatives in a single tetrahedron, measured by counting operations in the formulas.

For small deformations, all these models reduce to the second model, which allows the computations to be verified using the deformation test of Chapter 3. We express the material parameters for all models using the Lamé constants λ and μ.

4.3.1 St. Venant-Kirchoff elasticity

St. Venant-Kirchoff elasticity addresses the linear geometry approximation. It was used by Zhuang and Canny [101] in a dynamic simulation with non-lumped damping, by Picinbono et al. [78] in a dynamic simulation with lumped mass and damping, and by Debunne et al. [32] in a dynamic simulation with adaptive mesh resolutions.

We recall Equation (2.17) for the St. Venant-Kirchoff model, discussed in Section 2.1.

$$\begin{aligned} W(\iota_1, \iota_2) &= \frac{1}{2}\left(\left(-\mu - \frac{3\lambda}{2}\right)\iota_1 + \left(\frac{\lambda}{4} + \frac{\mu}{2}\right)\iota_1^2 - \mu\iota_2\right), \\ \mathbf{S} &= \mu(\mathbf{C} - \mathbf{I}) + \frac{\lambda}{2}(\iota_1 - 3)\mathbf{I}, \\ \partial\mathbf{S}/\partial\mathbf{C} : \mathbf{H} &= \frac{\lambda}{2}\operatorname{trace}(\mathbf{H})\mathbf{I} + \mu\mathbf{H}. \end{aligned} \tag{4.5}$$

The result of applying the St.Venant-Kirchoff model to our test object is shown in Figure 4.3. The energy function does not have an energy term that prevents material inversion. This is reflected in the result: elements are inverted near the attachment point of the rod.

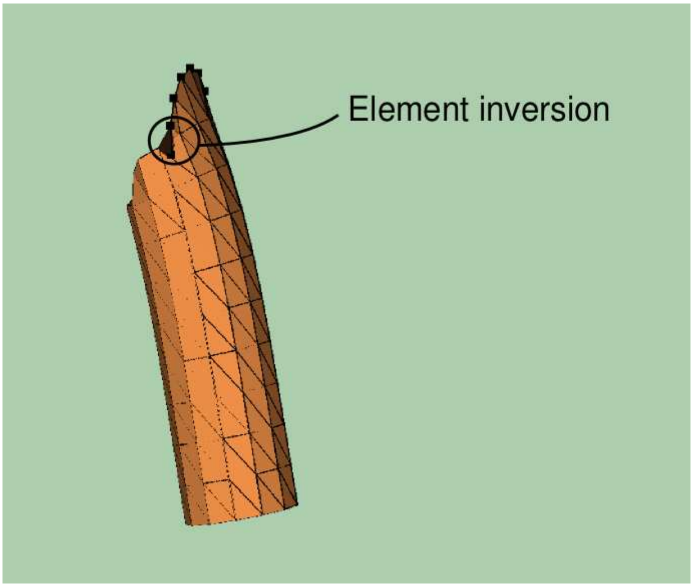


Figure 4.3: St. Venant-Kirchoff elasticity. Elements are inverted where the beam is fixed at the left.

4.3.2 Linear geometry approximation

If this model is combined with the linear geometry approximation, then we obtain linear elasticity, which was discussed earlier in Section 3.1. Linear elasticity was prevalent in early work in surgery simulation [18, 27, 49]. It is also used when high update rates are required. When using the Boundary Element Method [53] or static condensation [18, 36] it is possible to precompute all deformations of an object in advance. With this technique, the high update rates required for haptic interaction can be achieved.

The linear geometry approximation is shown in Figure 4.4. Evidently, the assumption of small deformations does not hold in this situation.

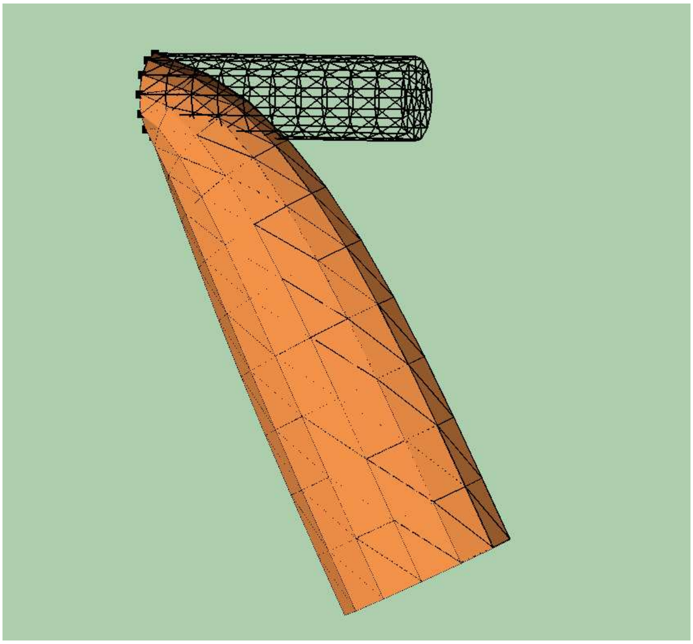


Figure 4.4: The result of applying the linear model to our standard test. The undeformed configuration is shown as a wire frame mesh.

4.3.3 Neo-Hookean elasticity

The compressible neo-Hookean elasticity model is a generalization of the St.Venant-Kirchoff model, and it is used for describing rubbery materials. It has also been used as a material model for interactive deformation by Székely et al. [92] and Wu et al. [100]. The energy density function that we use is given by [65, 102].

$$W(\iota_1, \iota_3) = \frac{1}{2}\left(\mu(\iota_1 - 3) - \mu\ln(\iota_3) + \lambda(\sqrt{\iota_3} - 1)^2\right). \tag{4.6}$$

The stress and its derivative are as follows:

$$\begin{aligned}\mathbf{S} &= (\mu\mathbf{I} + (-\mu + \lambda(\sqrt{\iota_3} - 1)\sqrt{\iota_3})\mathbf{C}^{-1}) \\ \partial\mathbf{S}/\partial\mathbf{C} : \mathbf{H} &= -(\lambda(\sqrt{\iota_3} - 1)\sqrt{\iota_3} - \mu)\,\mathbf{C}^{-1}\cdot\mathbf{H} \\ &\quad + \frac{\lambda}{2}(2\sqrt{\iota_3} - 1)\sqrt{\iota_3}(\mathbf{C}^{-1} : \mathbf{H})\mathbf{I}\cdot\mathbf{C}^{-1}\end{aligned}$$

Compression makes ι_3 tend to zero, so the logarithm tends to minus infinity: the material resists inversion, which is visible in the result shown in Figure 4.5.

For small strains, we have $\mathbf{C} \approx \mathbf{I}$, so $\mathbf{C} - \mathbf{I} = \mathcal{O}(\varepsilon)$ for some small $\varepsilon > 0$. In a linear approximation, we have

$$\begin{aligned}\sqrt{\iota_3} &= 1 + \mathrm{trace}(\mathbf{C} - \mathbf{I})/2 + \mathcal{O}(\varepsilon^2), \\ \mathbf{C}^{-1} &= \mathbf{I} - (\mathbf{C} - \mathbf{I}) + \mathcal{O}(\varepsilon^2).\end{aligned}$$

For small deformations, this reduces to the stress for linear elasticity in Equation (4.5).

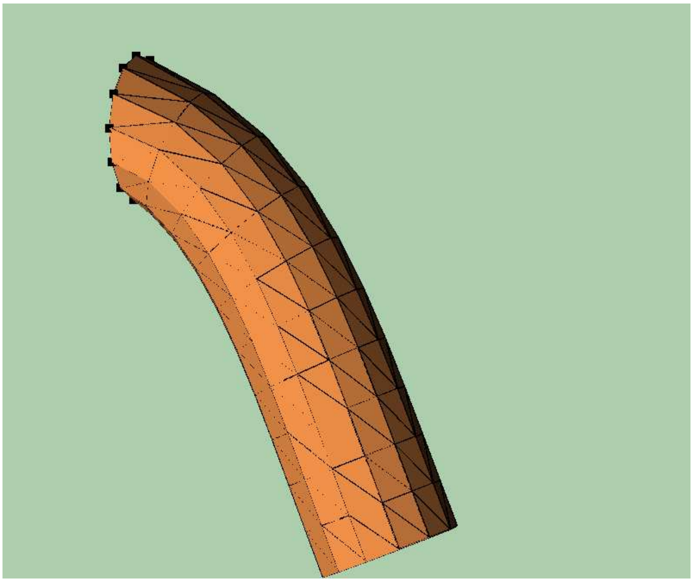


Figure 4.5: Compressible neo-Hookean material.

4.3.4 Veronda-Westmann elasticity

Veronda and Westmann [98] have proposed a three-dimensional constitutive description of soft tissue based on measurements of cat skin. Their work was also discussed in Section 2.6. This model has been used in offline simulations of soft tissue [51, 79]. Veronda and Westmann propose the following energy density:

$$W(\iota_1, \iota_2, \iota_3) = c_1(e^{\gamma(\iota_1 - 3)} - 1) + c_2(\iota_2 - 3) + g(\iota_3).$$

The function g was not specified further. In the compressible case, we should have $g(\iota_3) \to \infty$ if $\iota_3 \to 0$. For small deformations, we have $\iota_1 \approx 3$, so the exponential term can be linearized to $c_1\gamma(\iota_1 - 3)$. The effect of the exponential term is to resist stretching more when strains are large. This is consistent with the stress-strain relations for most types of soft tissue. The parameter γ measures the amount of nonlinearity.

We assume that the reference state is stress free ($\mathbf{S} = 0$ if $\mathbf{C} = \mathbf{I}$). To ensure consistency with the linear model, we require that for $\gamma \to 0$, the exponential term reduces to $2\mu(\iota_1 - 3)$. The following function fits this template:

$$W(\iota_1, \iota_2, \iota_3) = \frac{1}{2}\left(\frac{2\mu}{\gamma}\left(e^{\gamma(\iota_1 - 3)} - 1\right) - \mu(\iota_2 - 3) + \frac{\lambda}{2}\left(\iota_3 - 1 - \ln(\iota_3)\right)\right).$$

This energy density leads to the following stress tensor

$$\mathbf{S} = \left(2\mu e^{\gamma(\iota_1 - 3)} - \mu\iota_1\right)\mathbf{I} + \mu\mathbf{C} + \frac{\lambda}{2}(\iota_3 - 1)\mathbf{C}^{-1}. \tag{4.7}$$

The stress derivative is given by

$$\begin{aligned} \partial \mathbf{S}/\partial \mathbf{C} : \mathbf{H} = {} & \mu\left(2e^{\gamma(\iota_1 - 3)}\gamma - 1\right)\mathrm{trace}(\mathbf{H})\mathbf{I} + \mu\mathbf{H} \\ & + \frac{\lambda}{2}(\iota_3(\mathbf{C}^{-1} : \mathbf{H})\mathbf{I} - (\iota_3 - 1)\mathbf{C}^{-1} \cdot \mathbf{H}) \cdot \mathbf{C}^{-1}. \end{aligned} \tag{4.8}$$

When the nonlinearity γ tends to 0, and we assume small strains ($\mathbf{C} = \mathbf{I} + \mathcal{O}(\varepsilon)$), then (4.7) tends to

$$\begin{aligned} & \mu(2 - \mathrm{trace}(\mathbf{C} - \mathbf{I}) - \mathrm{trace}(\mathbf{I}) + \mathbf{C}) + \frac{\lambda}{2}\mathrm{trace}(\mathbf{C} - \mathbf{I})(\mathbf{I} - (\mathbf{C} - \mathbf{I})) + \mathcal{O}(\varepsilon^2) \\ & = \mu(\mathbf{C} - \mathbf{I}) + (\mu + \frac{\lambda}{2})(\mathrm{trace}(\mathbf{C} - \mathbf{I}))\mathbf{I} + \mathcal{O}(\varepsilon^2). \end{aligned}$$

The $\mathrm{trace}(\mathbf{C} - \mathbf{I})$ term, corresponding with volume preservation in the linear model, is not consistent with the linear case. The result of applying Veronda-Westmann to the test object is shown in Figure 4.6. Due to the exponential term, the object resists stretching more, and bends less. This results in a smaller tip deflection than the neo-Hookean material model.

4.4 Relaxation algorithms

The two relaxation algorithms tested are explicit SS22 time-integration with lumped masses and lumped damping, and the nonlinear CG algorithm. In this section we discuss how parameters for the dynamic algorithm were chosen, and how the line search for the CG algorithm was implemented.

4.4.1 Dynamic parameters

The implementation of a dynamic relaxation is straightforward, but running requires η and Δt to be set. In the linear case, we can compute the optimal choice for both parameters. In the nonlinear case, we must resort to a heuristic.

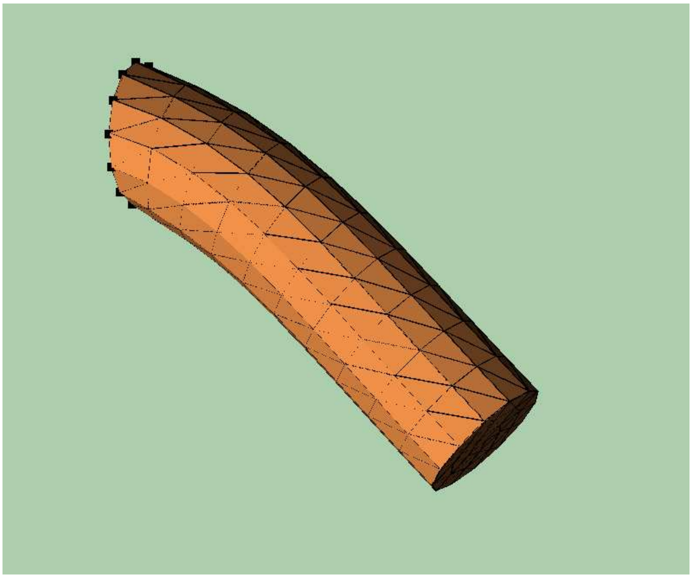


Figure 4.6: Veronda-Westmann material.

The critical time step can be computed exactly for linear elasticity. Nonlinear material can react more strongly to a change in deformation, and requires shorter time steps. Therefore, the critical time step is found by the following empirical procedure. A time step is considered stable if an undamped simulation does not blow up within 50 MFLOPs. A simulation is considered blown up if $\|u\|$ exceeds 10^{15}. An initial time step is estimated using the Courant-Friedrichs-Lewy criterion (2.54), and then it is repeatedly lowered by 15 % until a stable time step is found.

The critical damping was also determined by trial and error. Damping higher than η_{crit} yields smooth and slower convergence, while lower damping yields slower, oscillatory convergence. By manually trying out different η values and selecting the value yielding the fastest convergence, we can find the optimal η, listed in Table 4.4. To verify that η_{crit} is close to optimal, all convergence graphs also show results for damping of $2/3\eta_{\mathrm{crit}}$ and $3/2\eta_{\mathrm{crit}}$.

material	η_{crit}
linear	17
St. Venant-Kirchoff	20
neo-Hookean	21
Veronda Westmann	27

Table 4.4: Damping parameters for the experiments discussed

4.4.2 Line search

The nonlinear CG algorithm is a generalization of the linear CG method. It was discussed in Section 2.4.2. An implementation of the nonlinear CG algorithm requires a line search strategy. Such a strategy improves the energy $\Pi(x)$ of the current solution $x \in \mathbb{R}^n$ by taking a step $\alpha > 0$ in a given direction $d \in \mathbb{R}^n$. The optimal step is given by

$$\min_{\alpha \in \mathbb{R}^+} \Pi(x + \alpha d).$$

For a differentiable Π, the steplength follows from $g(\alpha) = 0$, where

$$g(\alpha) = \left(\frac{\partial \Pi}{\partial x}(x + \alpha d), d \right).$$

This is a one-dimensional equation, which may be solved with a Newton iteration. The Newton iteration was discussed in Section 2.4.3. In this case, the iteration can be defined as follows.

$$\begin{aligned} \alpha_0 &\leftarrow 0 \\ \alpha_{n+1} &\leftarrow \alpha_n - \left(\frac{dg}{d\alpha}(\alpha_n) \right)^{-1} g(\alpha_n), \qquad n \geq 0 \end{aligned}$$

We have

$$\frac{dg}{d\alpha}(\alpha) = (d, K(x + \alpha d))$$

The line search algorithm stops when the if needs more than $j_{\max}$ iterations, or if the update of α is small enough, as measured by a tolerance $\varepsilon_{\text{newton}}$. This leads to the following algorithm.

```
$j \leftarrow 0$
$\alpha_0 \leftarrow 0$
while $j < j_{\max}$:
  $r_j \leftarrow -(\partial\Pi/\partial x)(x + \alpha_j d)$
  $s_j \leftarrow -K(x + \alpha_j d)d$
  $\delta_j \leftarrow (r_j, d)/(s_j, d)$
  if $j > 0$ and $|\delta_j| < \varepsilon_{\text{newton}}|\delta_0|$:
    exit loop
  $\alpha_{j+1} \leftarrow \alpha_j - \delta_j$
  $j \leftarrow j + 1$
```

The update δ_j is not added to α_j if it is too small. Instead, the corresponding residual r_j is used to update the elastic force vectors in the main loop of the iteration.

In this Newton iteration, the result of the last calculation of $K(x + \alpha_j d)d$ is never used, which is wasteful. Therefore we propose a secant method [102]: the derivative g' in the Newton scheme is replaced by the finite difference approximation

$$\frac{dg(\alpha_n)}{d\alpha} \approx \frac{g(\alpha_n) - g(\alpha_{n-1})}{\alpha_n - \alpha_{n-1}}. \tag{4.9}$$

This leads to the following pseudo code:

```
$j \leftarrow 0$
$\alpha_0 \leftarrow 0$
while $j < j_{max}$:
  $r_j \leftarrow -(\partial\Pi/\partial x)(x + \alpha_j d)$
  $\gamma_j \leftarrow (s_j, d)$
  if $j = 0$:
    $s_j \leftarrow -K(x + \alpha_j d)d$
    $\delta_j \leftarrow \gamma_j/(d, s_j)$
else $\delta_j \leftarrow \gamma_j(\alpha_j - \alpha_{j-1})/(\gamma_j - \gamma_{j-1})$
  if $j > 0$ and $|\delta_j| < \varepsilon_{newton}|\delta_0|$:
    exit loop
  $\alpha_{j+1} \leftarrow \alpha_j - \delta_j$
  $j \leftarrow j + 1$
```

Since no evaluations of $\partial^2\Pi/\partial^2 u$ are left unused, we can expect that this method is more efficient. We may further speed up this algorithm by replacing the evaluation of $K(x)d$ in the first step by the finite difference approximation from (4.9). This introduces a scale-dependent parameter, since α_{-1} must be chosen for the problem at hand. We will refer to this algorithm as the scale-dependent secant algorithm.

4.5 Results

The hyperelastic models discussed were implemented, along with iteration methods for nonlinear CG and SS22 time-integration. This was done in the framework that we wrote for the work in Chapter 3. In addition, a truncated Newton algorithm, as discussed in Subsection 2.4.3, was implemented to compute reference solutions at stricter tolerances.

4.5.1 Tuning CG

The performance of the three line search algorithms (Newton, secant, scale-dependent secant) from the previous section is plotted in Figure 4.7. The scale-dependent secant algorithm is the fastest method. For our test cases, $\alpha_{-1} = -0.001$ was sufficient to obtain convergence. The line search algorithms all use a tolerance parameter ε. Figure 4.8 shows how different settings affect the convergence, and is representative of other models: The iteration converges within a few iterations, so the precise value of ε makes little difference in the convergence behavior.

There are different strategies for determining β in the nonlinear Conjugate Gradient algorithm. Both the Fletcher-Reeves strategy from Equations (2.48) and Polak-Ribière from (2.49) were implemented, but for our test problem there was no difference in performance. The rest of the experiments were conducted with normal secant line search, Polak-Ribière β selection and a large tolerance ($\varepsilon_{newton} = 0.1$) for the line search.

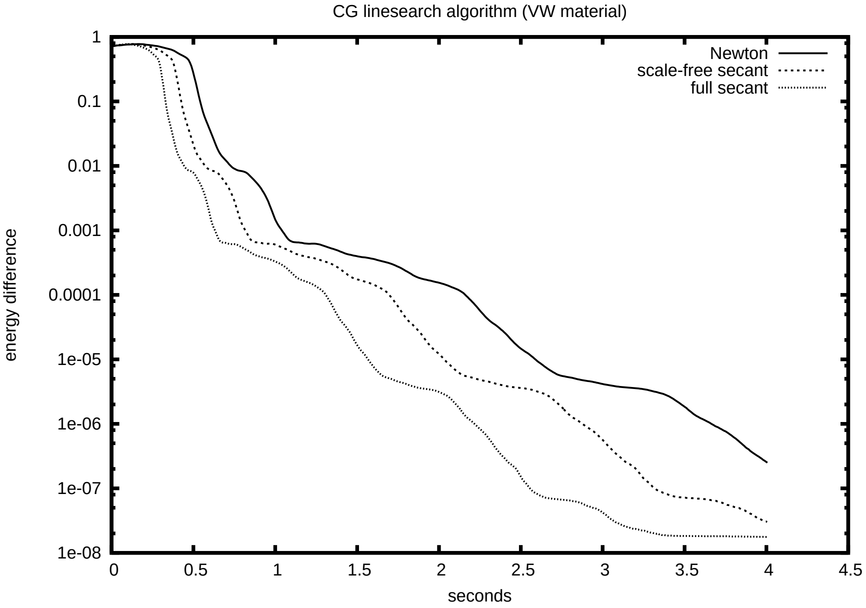


Figure 4.7: The performance of different line searches for the Veronda-Westmann problem.

4.5.2 Performance

By comparing FLOP count and processor time used, we can also estimate the MFLOP per second rate, which indicates how efficiently the CPU is used during computations. These numbers are given in Table 4.5. The baseline for the MFLOP/second rate was a repeated double vector add, coded in C++. For repeated adds of a 1024-double vector, the machine, a 1 Ghz Pentium 3, achieved 246 MFLOP/sec. The programs were compiled with GNU C++ version 3.2, with maximum optimization switched on, and visualization and convergence statistics turned off.

The dominating cost in computation were computations of the elastic forces, taking up 99 to 99.5 % of the operations. The MFLOP rates range between 50 and 90 % of the peak speed, indicating that the implementation performs in the order of magnitude of the machine peak speed.

4.5.3 Influence of mesh size

In Chapter 3, we have demonstrated that large meshes are needed for accurate results. Figure 4.9 compares the convergence of the large mesh and the small mesh from Table 4.2. The larger mesh leads to slower convergence, but static and dynamic are slowed down by the same amount, so all other experiments are performed on the small mesh.

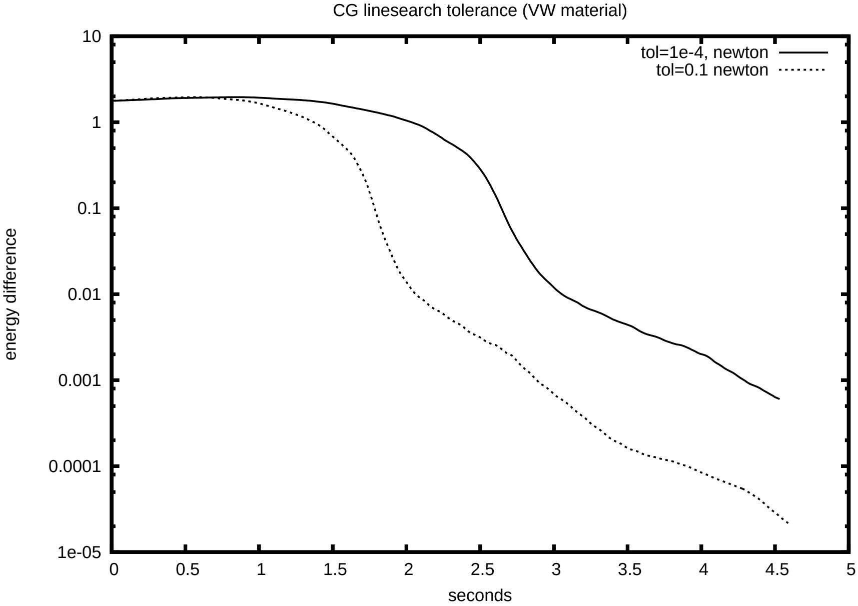
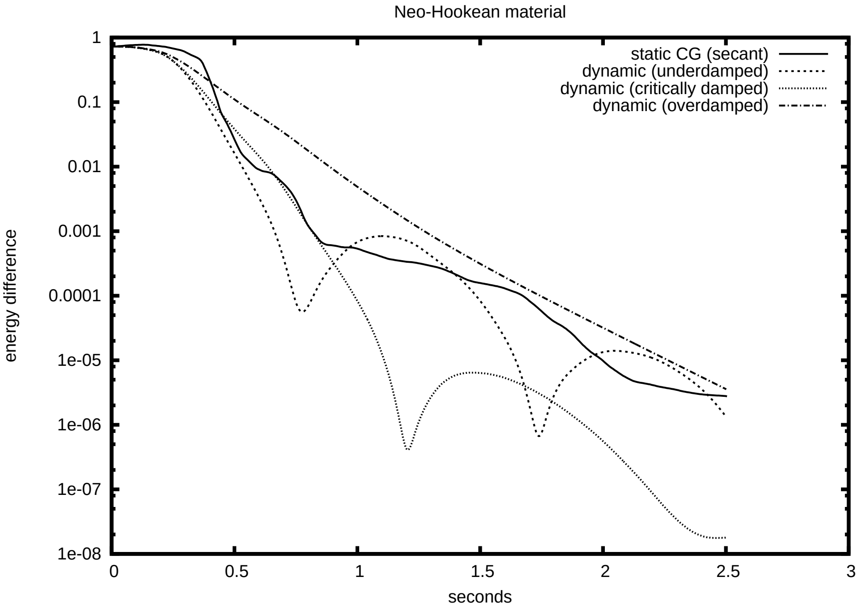


Figure 4.8: Impact of the Newton tolerance in the line search. (Veronda-Westmann problem, Newton line search). The performance is only slightly affected, but a looser tolerance is quicker.

material	relaxation	line search	% peak	iter/sec	MFLOP/iter
Linear elasticity	dynamic		49	628	0.2
	static		56	708	0.2
St Venant Kirchoff	static	Newton	48	71	1.7
	static	Secant	53	113	1.2
	dynamic		62	426	0.4
neo-Hookean	static	Newton	85	109	1.9
	static	Secant	77	144	1.3
	dynamic		54	356	0.4
Veronda-Westmann	static	Newton	90	91	2.4
	static	Secant	80	120	1.7
	dynamic		61	327	0.5
neo-Hookean	static	Newton	73	12	14.2
(large mesh)	static	Secant	66	16	9.9
	dynamic		49	41	2.9

Table 4.5: Machine dependent performance numbers for a PIII/1Ghz machine captured from the first 1.0 seconds that experiments ran. Timings per second are approximate numbers, and vary by a few percent across runs. The peak MFLOP rate is defined to be 246 MFLOP/sec: the performance for repeated double vector add of size 1024.



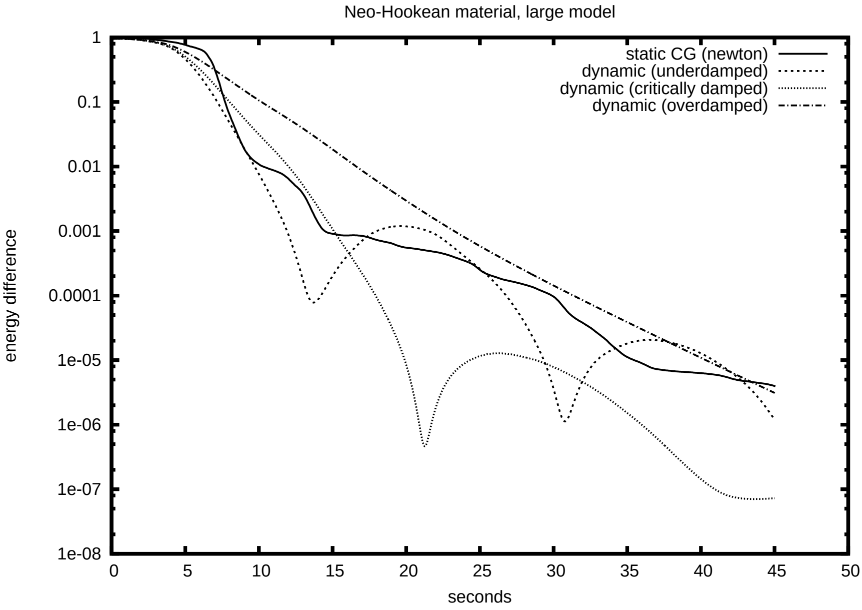


Figure 4.9: Cantilever experiment with neo-Hookean elasticity for different mesh sizes. The small model (top) and the large model (bottom) offer similar convergence.

4.5.4 Linear elasticity

Figure 4.10 shows the convergence of the linear case. In this case, the CG iteration outperforms dynamic relaxation, by approximately a factor 5 to 10. For example, an energy error of less than 10^{-4} takes 0.10 seconds with linear CG, and 0.89 seconds with dynamic relaxation.

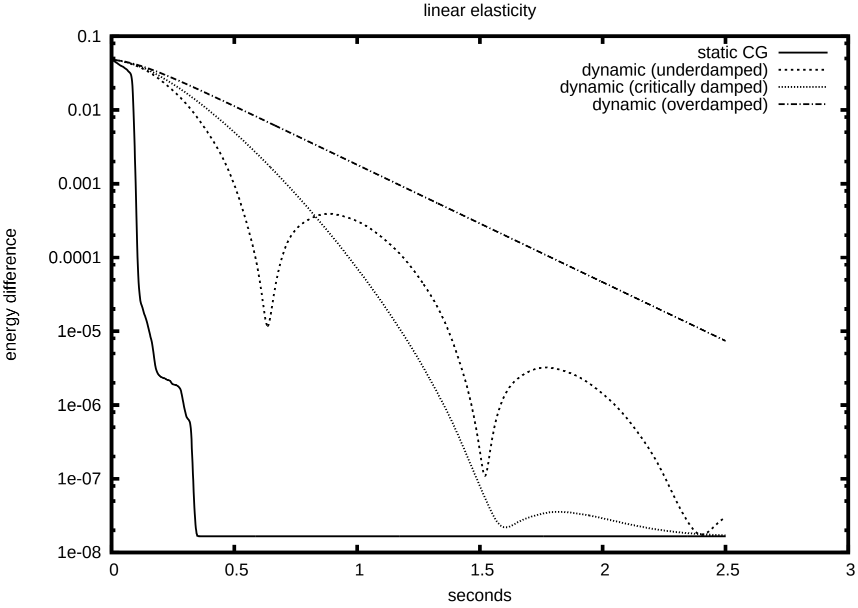


Figure 4.10: Convergence speed for the linear model, energy error

4.5.5 Nonlinear material models

For the nonlinear case, the CG iteration is as quick as a dynamic relaxation with optimal parameters; this is independent of mesh size and material characteristics. This can be seen in Figures 4.11 and 4.9. For St. Venant Kirchoff elasticity in Figure 4.12, dynamic relaxation is at a slight advantage. This seems to be caused by the element inversion. Stiffer material does not lead to element inversion. When the same experiment is repeated with $E = 2 \cdot 10^4$, both algorithms again have roughly the same speed, shown in Figure 4.13.

The lack of physical interpretation of the intermediate results of the CG iteration is evident in Figure 4.14. The static solution itself has a minimal residual force, but during the iteration the residual decreases erratically, and does not even descend monotonously. On the other hand, residual forces decrease smoothly during dynamic relaxation.

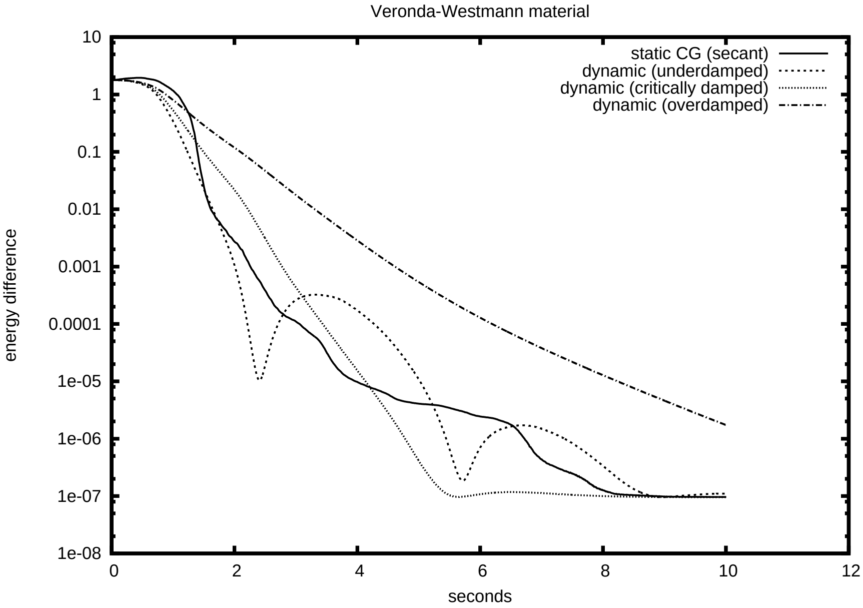


Figure 4.11: Convergence speed for exponential Veronda-Westmann

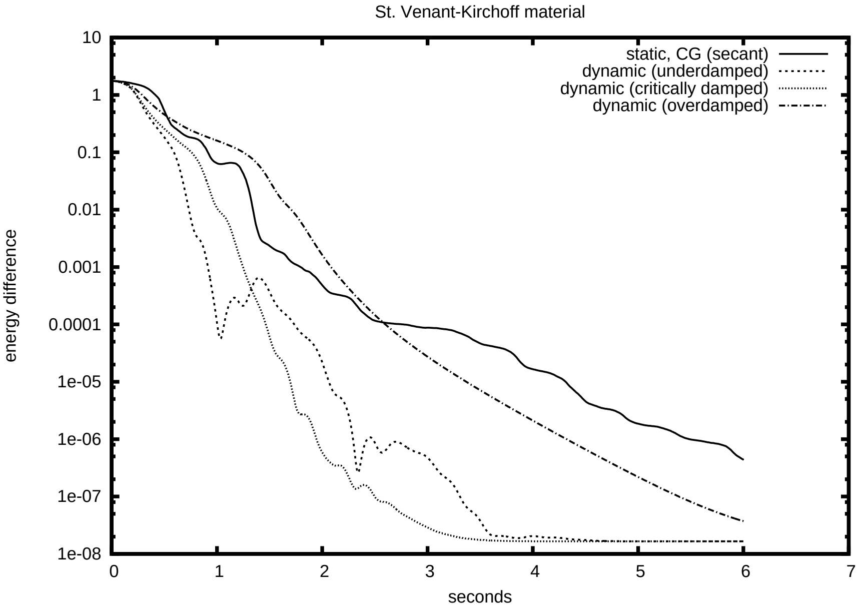


Figure 4.12: Convergence speed for St. Venant-Kirchoff material

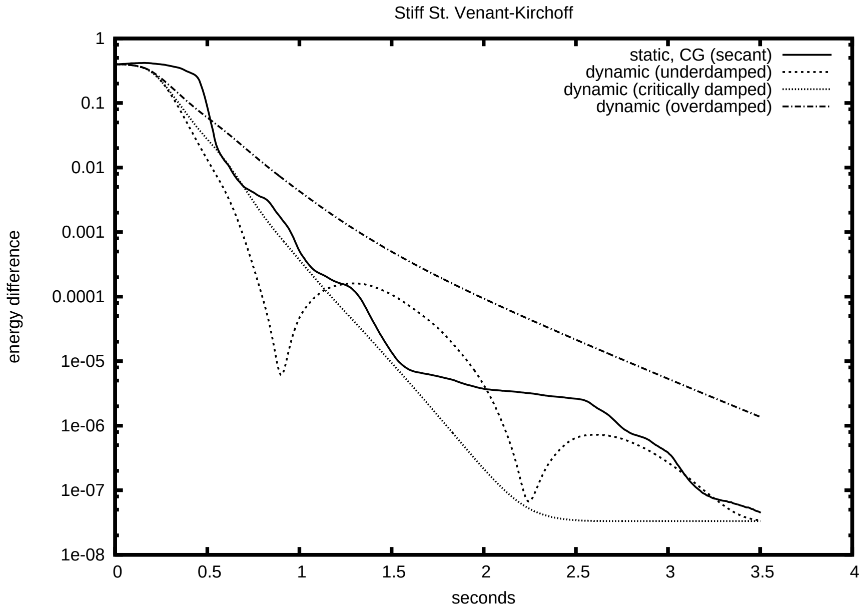


Figure 4.13: St. Venant-Kirchoff material with stiffer material ($E = 2 \cdot 10^4, \eta = 27s^{-1}$).

4.5.6 Influence of mesh quality

The influence of mesh quality is demonstrated in Figure 4.15. A single short edge was introduced in the mesh, by inserting a node close to an existing node, using a Delaunay incremental flip algorithm [40]. The effect on linear CG is negligable. This can be attributed to the observation in Subsection 2.4.1 that the magnitude of isolated eigenvalues does not affect the performance of linear CG. In a dynamic setting, the critical time step is inversely proportional to the smallest edge length, hence element shape severely influences the convergence. The influence on the nonlinear CG iteration is also noticeable, but has a much smaller impact. In this experiment, the scale-dependent secant method failed to converge, showing its limited usefulness in practice.

4.6 Discussion

We have compared dynamic relaxation and iterative optimization as methods for finding the steady state of a solution. Dynamic relaxation has linear convergence for the linear problem: the number of iterations is proportional to $\sqrt{\text{cond}_2(M^{-1}K)}$, where M is the lumped mass matrix. For static CG the number of iterations depends on the set of eigenvalues, and in the worst case, it is bounded by $\sqrt{\text{cond}_2 K}$. For uniform meshes and constant mass density, M is almost a multiple of I which suggests that the performance of both is similar.

There are more similarities: dynamic relaxation can be speeded up by increasing

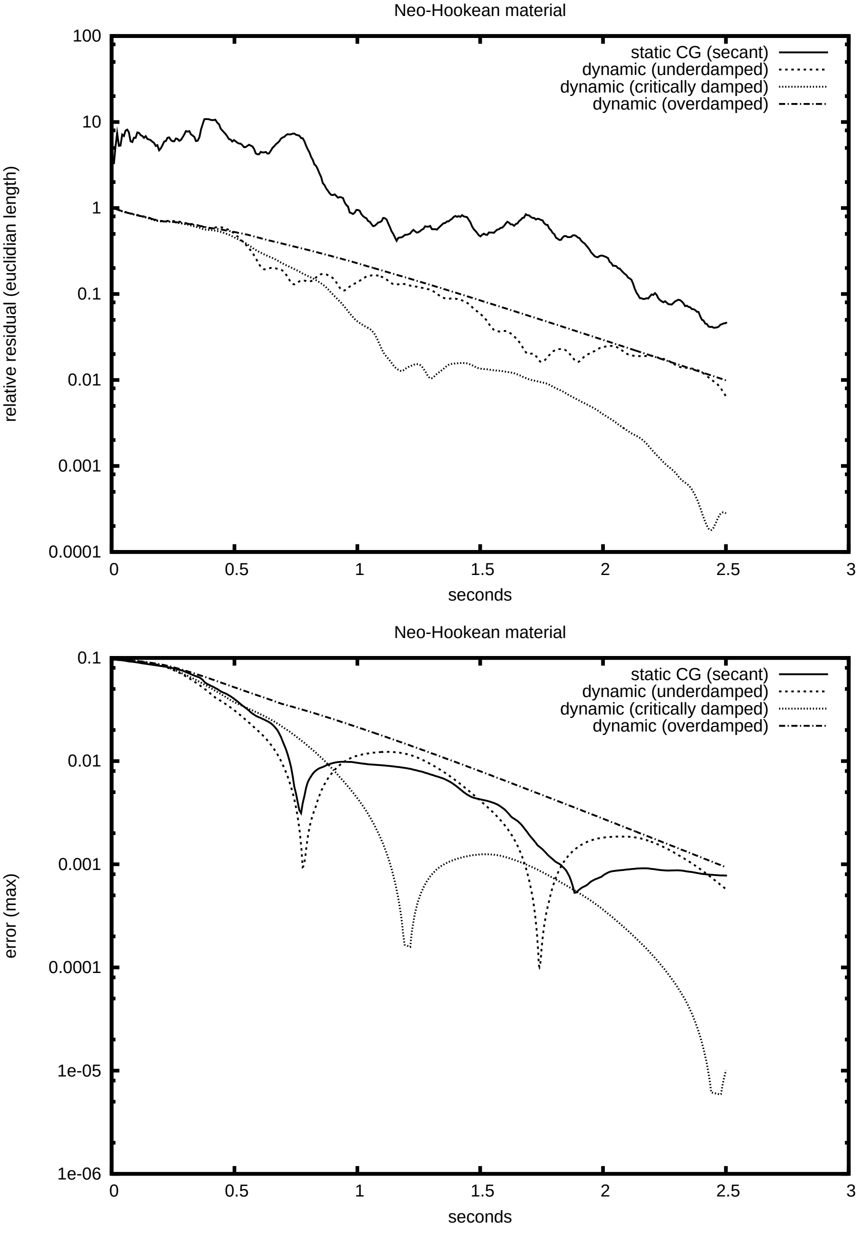


Figure 4.14: Evolution of the relative residual force $\|r\|_2/\|f^{ex}\|_2$ for neo-Hookean elasticity (top). Dynamic relaxation shows a smoother decrease than CG.

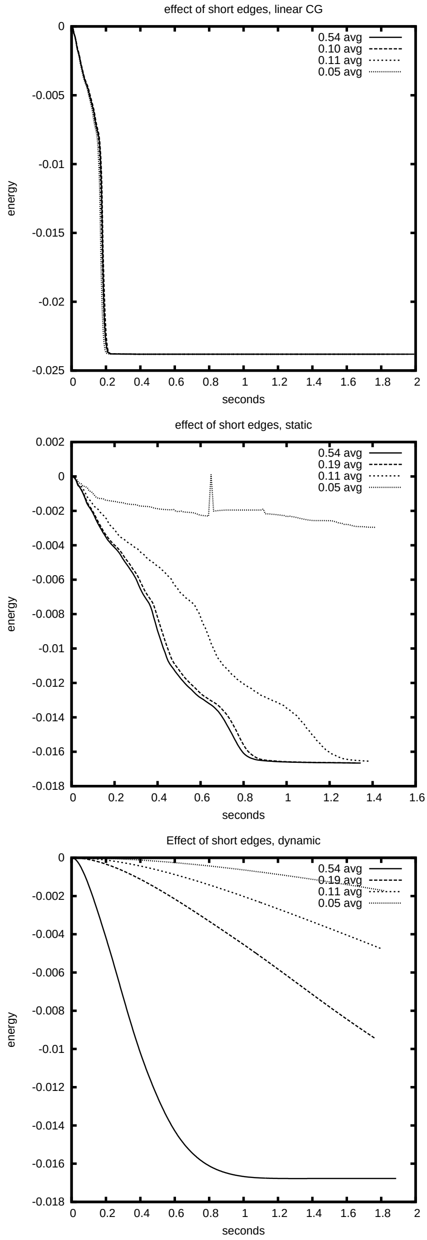


Figure 4.15: Energy decrease over time when a single short edge introduced in the mesh, for the Neo-Hookean material model. Top linear CG. In center the CG based static approach, and on the bottom the dynamic approach. The static approach is hampered less by short edges.

nodal masses of small elements. Analogously, a Conjugate Gradient iteration could be speeded up by using diagonal preconditioning. Parallels between time stepping algorithms for differential equations and optimization based solutions have been pointed out before [50]. In this case, considering the convergence analysis of CG, a direct parallel does not hold.

In the test-case that we have presented, the difference in performance between static CG and dynamic relaxation in the linear case is large: a factor 5 to 10. This could be caused by the symmetry of the test object: this symmetry implies that the stiffness matrix has duplicated eigenvalues. This favors static iteration, as clustered eigenvalues accelerate the convergence of linear CG.

For nonlinear models, the experiments indicate that the performance is comparable. There are qualitative differences between both methods: the physical underpinnings of relaxation ensure a smooth decrease in residual force, and non-conservative forces, such as friction to be added to the model. However, for fast convergence, both Δt and η must be selected experimentally for the situation at hand, and both parameters directly influence convergence speed. Moreover, they also depend on mesh characteristics, and in the nonlinear case the time step also depends on the magnitude of the forces applied. If a simulation includes online mesh changes or nonlinear elasticity, both parameters must be adjusted continuously.

Our test situation is inspired by a soft tissue simulation scenario. However, it has limited use in predicting the applicability of an algorithm in practice. The reason is that the experiment uses gravity, instantaneously switched on, as a test load. First, gravity is a load that is distributed over the entire body, while loads in interactive simulations are typically effected by simulated instruments, which act locally. Such localized loads have more high-frequency components, and this leads to more high-frequency components in the error. On the other hand, quick convergence requires choosing η low. In this case, the high-frequency components of the error will be underdamped and will persist for a long time. This will be noticeable as a "jelly like" vibrations. Secondly, the load is switched on instantaneously while the object is far from its resting position. Interactive simulations run at high update rates, and so loads change slowly between iteration steps. In practice, a deformation computed in the previous iteration step will be a good starting solution for the next step.

For the small mesh and the material parameters selected, the critical time-step is approximately 1 ms and requires an update rate of 1000 Hz. Our machine runs the dynamic simulation at 300 to 600 Hz, depending on the material model. This is close to real-time. Yet, it is not clear whether meaningful simulations can be constructed with meshes as small as these. Moreover, an accurate simulation should simulate mechanical properties of soft tissue, which are known to include viscoelastic effects and incompressibility. Both lead to larger FEM problems. For viscoelasticity, the history of deformation adds extra degrees of freedom. Incompressible problems introduce pressure as an additional variable to the problem, and require more degrees of freedom for the displacement functions to ensure existence of solutions.

In summary, the approach presented in this chapter already reaches the limits of interactive computation, while the material models used do not reflect real tissue behavior. We must address these limits for better simulations. We can distinguish three limits:

- the cost per iteration step,
- the condition number of the problem,
- the relaxation algorithms used.

A cost of a single iteration is determined by the number of elements and the per-element cost of the force computations. This implies that mesh change routines should keep the mesh size low, and only refine meshes where needed. Additional speedups can be gained by using parallel processing, at the cost of synchronization overhead and increased hardware costs.

Badly shaped elements increase the condition number of the problem slowing down the convergence of iterative methods. Explicit dynamic methods are especially susceptible to instabilities caused by small elements, so small elements require the use of small time-steps. A way to cope with such instabilities, is to use adaptive time steps for parts of the mesh [11]: this addresses the problem of instabilities, but introduces some overhead in keeping track of mesh parts that use different time steps. To a lesser degree, badly shaped elements also slow down static techniques. Therefore, it seems worthwhile to prevent such degenerate elements from occuring in the first place.

For FEM discretizations where element sizes are proportional to h, and volumes inversely proportional to the element count, the condition number of the stiffness matrix satisfies $\mathrm{cond}_2(K) \sim \frac{1}{h^2}$ [6]. A larger mesh, needed for a more accurate discretization, not only has more expensive iteration steps, but also requires a larger number of them. This is partly caused by the techniques that we have used: they are naive in the sense that they only use localized displacement/force calculations: applying a force locally to an object causes it to deform globally. For a global deformation, information must travel from the location where force is applied through the entire mesh. Both CG and dynamic relaxation update the location of a node using information from neighboring elements. The convergence of such an iterative method is therefore bound by the diameter of the mesh, since that determines the speed at which deformations propagate across the mesh.

This suggests that more advanced techniques should be used for improving the performance of relaxations. Preconditioning reduces the complexity of CG iterations, but is usually implemented with explicitly stored stiffness matrices. *Multigrid methods* [14] can solve FEM problems with n degrees of freedom in $\mathcal{O}(\log(n))$ iterations, by running iteration steps at multiple resolutions. The method requires that the problem is simulated on meshes with lower resolutions as well. For general unstructured meshes, computing such coarser grids is a complex task in itself [1], which suggests the use of structured meshes.

4.7 Conclusion

We have compared nonlinear Conjugate Gradients and dynamic relaxations for a scenario that is inspired by simulation of soft tissue, and found that nonlinear CG offers similar convergence as dynamic relaxation with optimal parameters. Therefore, both methods should be distinguished by their qualitative differences.

Dynamic relaxation offers a physical interpretation of the iteration results, but requires manual selection of simulation parameters, and the process is vulnerable to instabilities. Static iterative methods are more robust, but their intermediate results have no physical interpretation and are produced at lower frequencies.

Both approaches are affected by mesh quality and mesh size. In other words, better meshes promote faster convergence per iteration step, and smaller meshes offer cheaper iteration steps. Therefore, mesh modification algorithms, such as cuts, should keep mesh size down and mesh quality high. This observation is taken to its consequences in Chapter 5.

It can be argued that the experiment is limited in its scope, and not directly relevant to interactive surgery simulations. This suggests that more carefully setup experiments would give a better appraisals of both techniques. However, given the size of the problem analyzed and the performance numbers in Table 4.5, it seems more worthwhile to direct future research towards techniques to speed up either relaxation technique. Even when using high-quality meshes, both unpreconditioned CG and dynamic relaxation easily strain current computing hardware beyond its limits. It is therefore necessary to use more advanced algorithmic techniques to speed up deformation calculations. This observation will be taken to its consequences in Chapter 6.

Chapter 5

A Delaunay approach to interactive cutting in triangulated surfaces

5.1 Introduction

Our approach to deformation is based on the Finite Element Method (FEM). In this method, mesh size determines the computational requirements of a simulation. Larger meshes result in more degrees of freedom in the discretized problem, so solutions take more time to compute. This has motivated our work in Chapter 3, where we have tried a technique for simulating cuts that does not increase mesh size like subdivision techniques do. This technique creates flat elements in the mesh as artefacts, and we have found that they cause a considerable slowdown of the Linear CG algorithm. In Subsection 4.5.6 we have seen that element shape also affects the nonlinear CG and dynamic relaxation algorithms. Hence, mesh change operations, such as simulated cuts or cauterizations, should not only keep mesh size low; they should also keep elements of the mesh well-shaped. In this chapter[1] we address this problem for cutting in triangulations, by presenting a method that keeps mesh size low and keeps mesh quality high.

A FEM discretization is a form of interpolation: the continuous unknown in the problem is interpolated, so that the differential equation is transformed into a set of equations for a finite number of variables. The original problem is a differential equation, hence it is important that the derivative of the solution is approximated well by the interpolation. When we look at the influence of element shape on the derivative, we see that the large angles cause unbounded errors in the derivative. This is illustrated in 2D in Figure 5.1, and a similar argument also holds in 3D. Therefore, large angles should always be avoided. The convergence speed of iterative algorithms for linear problems

---

[1] This chapter was based on a paper published at the Fifth International Workshop on Algorithmic Foundations of Robotics (WAFR 2002) [74].

is related to the condition number of the stiffness matrix, as shown in Sections 4.1 and 2.4. The condition number is the ratio of largest and smallest eigenvalue of the stiffness matrix, and these eigenvalues can be bounded by eigenvalues of separate elements: each element can also be seen as a separate elastic object, whose deformations are also described by linear elasticity, and their elastic behavior can be condensed in a small *element stiffness matrix*. Therefore, mesh quality can be optimized by optimizing the shape of individual elements. In a second-order elliptic partial differential equation (like linear elasticity), both very large and very small angles, and small elements cause high condition numbers [87]. Therefore, a mesh with 'round' elements, i.e. no angles close to π and 0, and uniform element sizes is a good general purpose mesh.

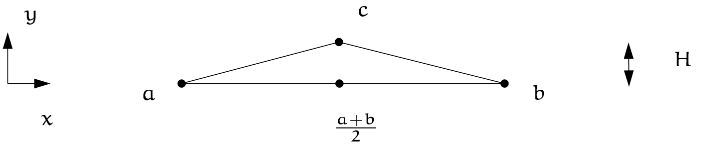


Figure 5.1: The derivative of a linear function with values $f(a)$, $f(b)$ and $f(c)$ in triangle abc in the plane has directional derivative $\partial f/\partial y = \frac{f(c)-(f(a)+f(b)/2)}{H}$. When $H \to 0$, the derivative tends to infinity.

Simulation of cuts in surgery simulation is related to simulation of other destructive surgical procedures. The first operation to have been simulated on volumetric meshes is cauterization. This was done by removing elements in contact with a virtual cauterization tool [26]. A disadvantage of element removal is that it produces a jagged surface on the virtual tissue.

For cutting, subdivision methods are the norm [11, 46, 66]: elements that are in contact with the scalpel are marked active, and subdivided to produce a cut conforming to the scalpel position. The subdivision moves along with the scalpel during its stay in an active tetrahedron. When the scalpel leaves an active element, the subdivision is entered in the mesh permanently. A 2D example of subdivision is in Figure 5.2. Subdivision methods always increase the size of the mesh. Moreover, these methods tend to produce degeneracies. This is caused by the use of an *active region*. Mesh modification is done only within a fixed region of the mesh, and if the scalpel moves close to the boundary of that region, poorly-shaped elements are inevitable. Ganovelli and O'Sullivan [45] try to counter the degeneracies caused by subdivision cutting. They deal with these degeneracies by collapsing short edges of the mesh. This approach does improve the quality of the mesh, but this solution does not repair all inconsistencies: not all edges may be contracted, and flat triangles and tetrahedrons, which do not contain short edges but are still degenerate, are not dealt with.

In Chapter 3 we have tried an approach where scalpel nodes are snapped to the trajectory swept by the scalpel. The advantage of this method is that the mesh size remains small, and few short edges are created. However, there are a number of disadvantages: since no new nodes are created, the resolution of the cut is bounded by the mesh resolution. The incision does not reach up to the position of the scalpel, but lags behind it. A more serious problem is that snapping can result in degenerate elements in the

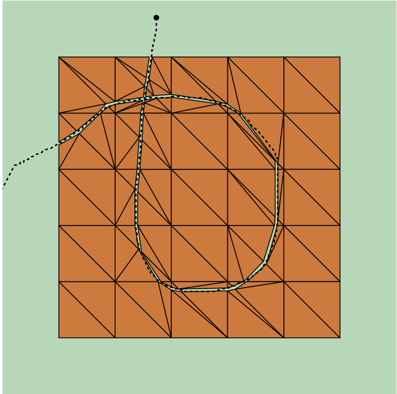


Figure 5.2: A cut produced by a subdivision method. The cut produces many small and flat triangles.

mesh. Such degeneracies are dealt with by subdividing flat elements, and collapsing the resulting short edges, effectively removing the flat element. Unfortunately, not all edges can be collapsed. Using existing mesh features as a basis for mesh modification is problematic: when the scalpel is not especially close to a mesh feature, it may not be possible to match the mesh topology to the scalpel path without introducing degeneracies.

In summary, producing high quality cuts in tetrahedral meshes is a difficult problem. For cutting in surfaces, there is an analogous problem, which has the same difficulties as volumetric cutting approaches. Serby et al. [84] propose a method which also relies on a form of node snapping: the scalpel is modeled as a line segment, and nodes from the mesh are projected onto that segment. In a post-processing step, the edge lengths and element volumes are optimized using a particle system. The result is a good looking cut, without a decrease in mesh quality and size. However, in reality, the path of a scalpel is not a large line segment, but a concatenation of several small ones, and their approach does not seem to address this issue. Bruyns et al. [21] use surface cutting with subdivision in large-scale simulations of various procedures.

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

5.2 Cutting in 2D

We state the general mesh cutting problem as follows: given a starting mesh, and positions of a user-controlled scalpel, modify the mesh at every moment to show an incision that represents the past trajectory of the scalpel, and ends exactly at the scalpel. The challenge in this problem is to produce a well-shaped mesh with few elements. The simplest case is the two-dimensional form. Here the mesh is a triangulation in the plane, and the virtual scalpel is a point. The challenge is to produce elements that have no large angles and no short edges.

5.2.1 Delaunay triangulation

Since we are putting emphasis on the quality of the mesh, we briefly review the Delaunay Triangulation (DT), a popular technique for generating a well-shaped triangulation of a given set of points. The DT of a set of points is defined as a triangulation where the circumcircle of every triangle does not contain any other points from the set. This property is also referred to as the empty-circle property. The empty-circle property can also be defined for edges of the mesh: an edge is called *legal* or *Delaunay* when the circumcircles of its incident triangles do not contain the opposite node of the other triangle. An example of an illegal edge is in Figure 5.3. Delaunay triangulations and Delaunay edges are intimately related: a Delaunay triangulation only has Delaunay edges.

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

There are various ways to measure element quality: for example, minimum angle, circumradius to shortest edge ratio, circumradius to inscribed radius ratio, etc. In 2D all these measures are equivalent up to constant factors [63], so the maxmin-angle property of the DT means that it is a reasonable meshing technique for virtually all element quality measures. When coupled with algorithms for point insertion it is a basis for many refinement meshing techniques [10, 25, 82].

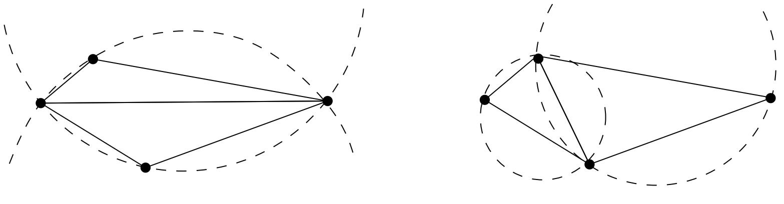


Figure 5.3: An illegal edge has a triangle whose circumcircle contains the opposite vertex of a neighboring triangle (left). By reconnecting ('flipping') the edge, the new circumcircles only contain the vertices it circumscribes (right).

5.2.2 Cutting and Delaunay flips

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

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

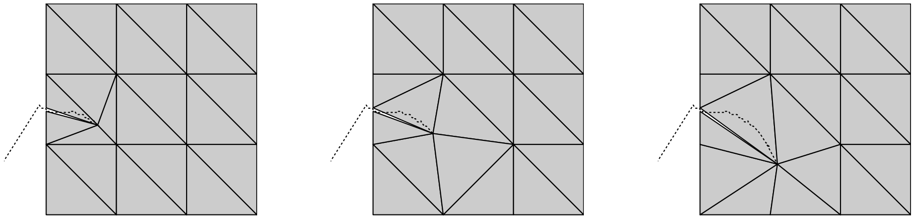
- Edges are flipped to improve the element angles.
- If an internal node is found close to the active node, it is removed.
- The incision is split, thus introducing new nodes to approximate the scalpel path.

The net effect of the last two actions is that nodes are removed in front of the scalpel, and inserted behind the scalpel. The technique is demonstrated in Figure 5.4; a more elaborate example is in Figure 5.8.

The Delaunay criterion tends to flip away triangles with angles approaching 180°, since these have large circumcircles. Nodes that are close together get connected. Such triangles have a small angle opposite a short edge. These triangles are also undesirable, so they are removed by by the second action.

When the scalpel enters the mesh, it is close to the boundary of the mesh, so a realistic incision would contain very short edges. To prevent these short edges, creation of incisions is postponed. When the scalpel enters the mesh close to an existing node, the node is moved and labeled active. If it enters in somewhere else, a new node marked active is inserted at the entry point. The active node moves along with the scalpel, creating a temporary dent. When the active node is sufficiently far from the entry point, it is “fixed up”: nodes are added at the entry point, creating an incision. The procedure is shown in Figure 5.6. If the scalpel is *retracting* (moving away from the object) before this fix-up happens, the dent is left in the mesh permanently.

After the entry fix-up, the cut edges are in almost in the same location. From a geometric point of view, we can identify both cut edges into a single edge, and check if this edge satisfies the empty-circle criterion. If it does not, new nodes are inserted, where the line connecting nodes opposite the cut edges intersect the scalpel path. We call this procedure a *incision split*, and it is demonstrated in Figure 5.5. When the scalpel passed that point previously, close nodes were removed, so the split will not lead to short edges. The newly inserted nodes are dilated slightly, to prevent numerical problems when a path self-intersects. The new nodes always lie on a line that connects two existing mesh nodes, hence the accuracy of the represented trajectory is bounded by the resolution of the starting mesh.



remove: flip: incision split:

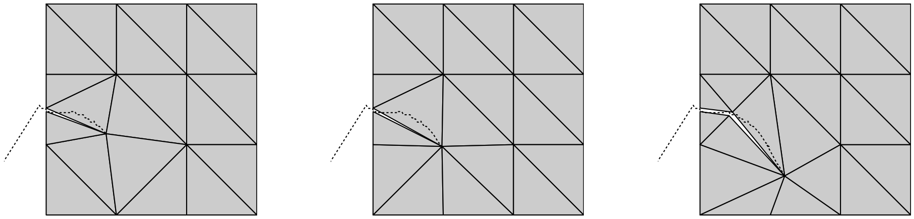


Figure 5.4: The evolution of a simple 2D cut is depicted from left to right. The key steps are to flip edges (center), to remove close nodes (left), and to insert nodes behind the scalpel (right).

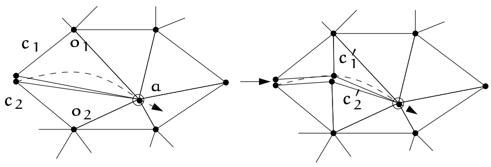


Figure 5.5: During an incision split, nodes are inserted in the cut edges. This happens when ac, with $c = (c_1 + c_2)/2$, fails the empty-circle criterion as diagonal of aco_1o_2. The newly inserted nodes c'_1 and c'_2 are inserted where o_1o_2 intersects the scalpel path.

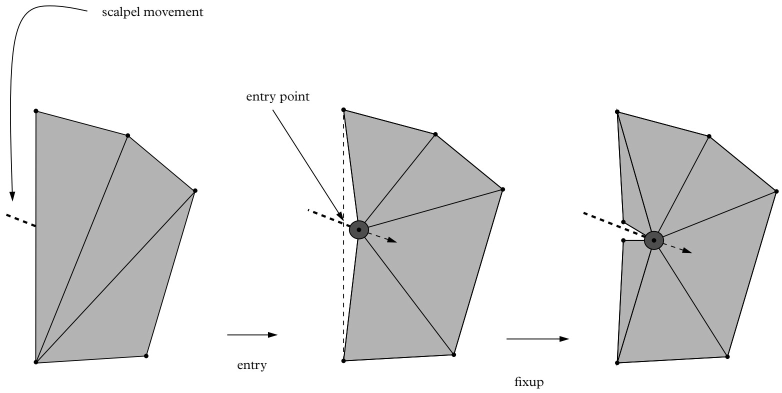


Figure 5.6: Incisions are postponed when the scalpel enters the mesh.

A similar situation arises when the scalpel exits. In this case, the scalpel comes arbitrarily close to the boundary of the mesh. To prevent arbitrarily flat elements from occuring, the exit point is predicted, and the cut is finished before the scalpel actually hits the boundary. At every step, the movement of the scalpel is extrapolated. When this extrapolation hits the boundary, and is close to the active node, the cut is finished: a node is inserted at the extrapolation, and the cut is dissected. This exit procedure still leaves relatively short edges permanently in the mesh. To rectify this, these edges are contracted. The procedure is shown in Figure 5.7 The boundary edges that are created with this exit operation are added to a list of forbidden edges. These edges are not tested for collisions during ensuing movements. This prevents artifacts when the actual scalpel movement differs from the predicted movement. If the scalpel is sufficiently far from the exit point, they are eligible for collision checking again.

5.3 Surface cuts in 3D

Triangulated surfaces in 3D are a common tool in computer graphics. They have also been used for surgery simulations [19, 22]. In these cases elastic behavior was simulated with damped mass-spring systems instead of the FEM. Nevertheless, the concerns for mesh quality continue to hold: flat elements are inverted more easily, and small elements correspond to short springs with small masses. Their high vibration frequencies translate into small time steps, which makes time integration expensive.

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

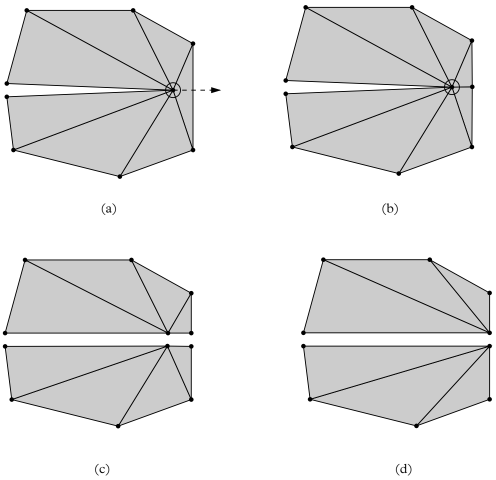


Figure 5.7: Exits are predicted: when the extrapolated scalpel movement hits a boundary closeby (a), then a new node is inserted at that location (b). The new edge is dissected (c) and contracted (d).

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

5.3.1 Single incision

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

Flipping on 3D surfaces is a delicate operation: some flips are topologically impossible (as demonstrated in Figure 5.9). This means that the all operations must be checked for failure cases.

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

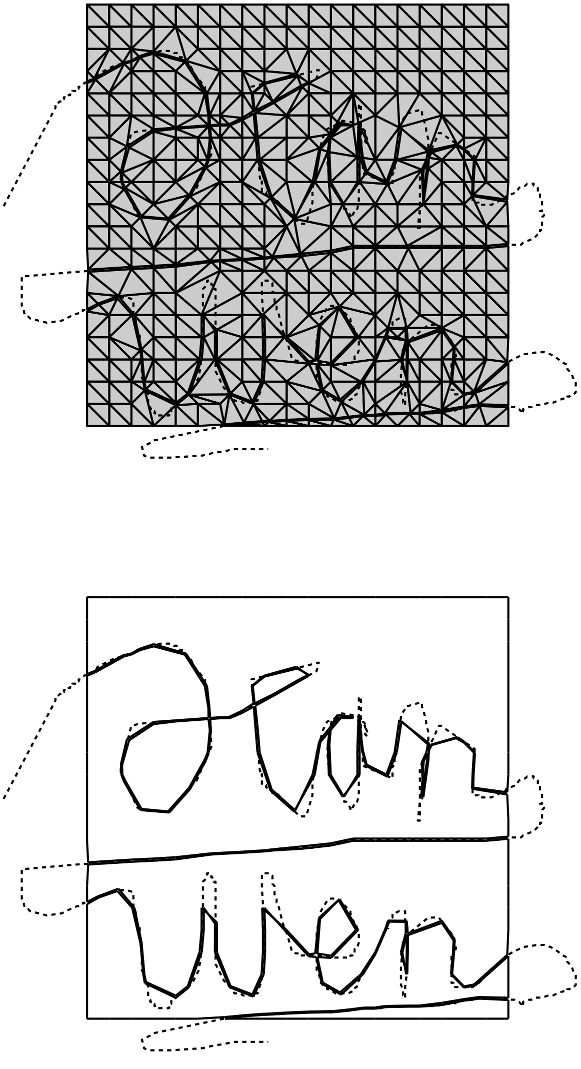


Figure 5.8: Delaunay cutting on a triangle mesh. Both the mesh itself and the boundary are shown. The scalpel trajectory is indicated with a dotted curve. Notice how strongly curved path segments are cut short in the realized cut. The starting mesh was regular and consisted of 722 triangles. The cut increased mesh size by only 57 triangles.

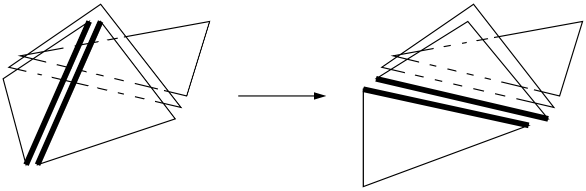


Figure 5.9: Not all flips in 3D are topologically valid: when flipping the bold edge on the left, the new edge pair (bold) occurs twice in the resulting complex. The triangles are displayed in an exploded view for clarity.

Node removal is done in a heuristic manner. Suppose that we want to remove a node v that is incident with n triangles. We say that two adjacent triangles incident with v form an *ear*, if the sum of the angles opposite v is less than π. When removing v, all ears are flipped until v is incident with only 3 triangles. Then v is removed, and the involved edges are flipped back if they violate the empty-circle criterion. Ears always exist: suppose that the triangles incident with v are numbered $i = 1, \ldots, n$, and have angles $\alpha_i, \beta_i, \gamma_i$, where γ_i is the angle at v (See Figure 5.10). Then there must be an $\alpha_{(i+1) \bmod n} + \beta_i < \pi$, since $n\pi = \sum_i (\alpha_i + \beta_i + \gamma_i) = \sum_i (\alpha_{(i+1) \bmod n} + \beta_i) + \sum_i \gamma_i$, and $\sum_i \gamma_i > 0$.

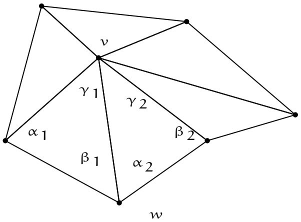


Figure 5.10: When removing a node, ears are flipped until v is in only 3 triangles. An ear has $\alpha_{(i+1) \bmod n} + \beta_i < \pi$, so the triangles incident with vw form an ear.

When the scalpel is almost parallel to the surface, small movements of the scalpel can result in large movements of an active node. For this reason it is necessary to control large movements. Large movements are subdivided using a maximum distance. This distance is computed as the minimum for all distances between the line spanned by the scalpel and lines spanned by the edge opposite the active node, as indicated in Figure 5.11. If the end-points of the scalpel segment cannot move further than $d_{\max}$ as indicated in Figure 5.11, then the scalpel will not hit that edge. This can be shown as follows: suppose that the scalpel position is given by the line segment $\mathbf{ht}$ for some $\mathbf{h}, \mathbf{t} \in \mathbb{R}^3$, and the points corresponding to $\mathbf{h}$ and $\mathbf{t}$ move by amounts $\mathbf{p}$ and $\mathbf{q} \in \mathbb{R}^3$ respectively, with $\|\mathbf{p}\|, \|\mathbf{q}\| < d_{\max}$. If the movement puts the scalpel on the line spanned by the edge, then $\lambda(\mathbf{p} + \mathbf{h}) + (1 - \lambda)(\mathbf{q} + \mathbf{t})$ is on the edge indicated for some $0 \leq \lambda \leq 1$. By the definition of $d_{\max}$ we have

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

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

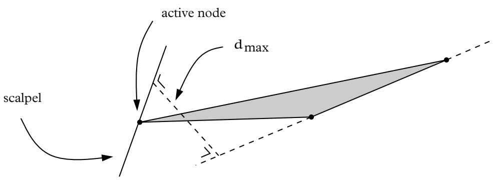


Figure 5.11: The maximum movement $d_{\max}$ for a scalpel and a single triangle. The total maximum distance is given by the minimum over all triangles incident with the active node.

5.3.2 Multiple incisions

In 3D, the scalpel can interact with the entire surface, and the scalpel may enter the mesh in any place. We can distinguish three cases, as demonstrated in Figure 5.12: the scalpel hits a boundary edge of the mesh, the scalpel tip enters through the surface, or the scalpel hits an internal edge of the mesh. The first case is handled completely analogous to the 2D case. The rest of the cases do not occur in 2D.

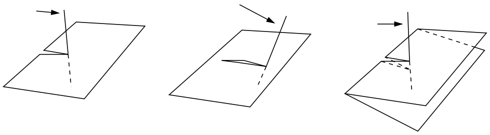


Figure 5.12: There are three ways for the scalpel to enter: by hitting the boundary, entering with the tip or hitting an exposed internal edge.

In the second case, a new active node is inserted in the incised triangle. When the active node is far enough from the entry point, a node is inserted at the original entry point. The result is an edge that connects the entry node with the active node. This edge is changed into a real incision, when the split action (shown in Figure 5.5) is executed. Until that time, the edge is constrained, so it cannot be flipped away. The third case is when the scalpel hits an exposed edge of the mesh. Then a single active node is inserted in the edge. During the next movement, the cut will branch into two incisions.

The scalpel is represented by a line segment, and line segments can interact with curved surfaces in many places: the scalpel may incise the surface in multiple locations, and during a cut a single incision may branch into multiple incisions, as is shown in Figure 5.14. We could apply the 3D remeshing process from Subsection 5.3.1 if we could rule out any interactions between different incisions. Fortunately, this seems possible: we can forbid interactions by ensuring that every triangle is incident with at most one active node. This is achieved by the following restrictions on the meshing process.

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

When an active node moves towards a restricted node, an annihilation is performed: all edges from the restricted node to an active node are dissected. This is demonstrated in Figure 5.13. When an active node comes close to an edge separating active nodes, then its movement is extrapolated, and a new node is inserted at the extrapolated point. The annihilation now proceeds with the newly inserted node. A normal boundary exit is a special case of an annihilation.

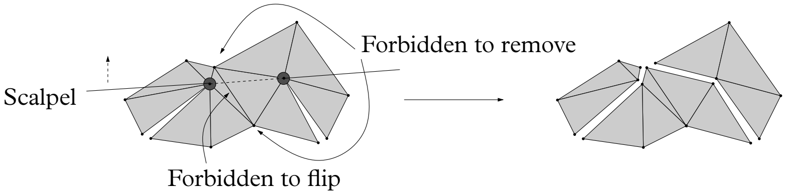


Figure 5.13: Edges that separate incisions may not be flipped, nodes separating incisions may not be removed. Instead, when an active node comes too close to such a forbidden node or edge, an annihilation is performed (right).

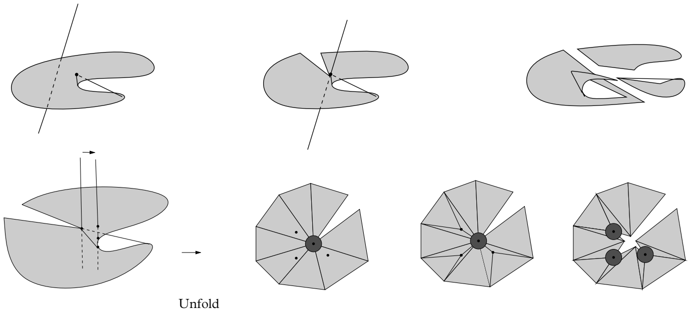


Figure 5.14: Folded surfaces may lead to branching cuts (top). In such cases, a movement will cause the scalpel to intersect multiple triangles (bottom; intersection points are indicated by dots). When this happens, multiple incisions replace the old active node.

When a scalpel movement is processed, the next position of the active node is determined by intersecting all triangles incident with the new scalpel position. If multiple active triangles are intersected, as demonstrated in Figure 5.14, then the cut will branch. New incisions are inserted, the old incision is marked as no longer active, and the edges connecting old and new active nodes are dissected. During a branch, the new nodes are close to the original incision. This leads to short edges and degenerate triangles. When the scalpel progresses further, these short edges disappear.

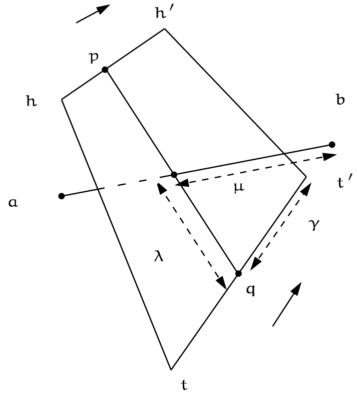


Figure 5.15: Intersecting a line sweep with an edge.

Collisions are computed as scalpel/edge intersections. The procedure for computing such intersections is as follows. We assume that the scalpel moves from $\mathbf{ht}$ to $\mathbf{h'}\mathbf{t'}$ for some $\mathbf{h}, \mathbf{t}, \mathbf{h'}, \mathbf{t'} \in \mathbb{R}^3$, and that we want to compute intersections with the line segment $\mathbf{ab}$, for $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ (See Figure 5.15). In other words, we want to solve

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

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

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

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

$$(\mathbf{c}|\mathbf{A})x = \mathbf{b}-\mathbf{t'}, \tag{5.1}$$

where $x \in \mathbb{R}^4$ is the unknown. When viewed as a linear system, a solution maybe given as $y + \alpha k$, where $y \in \mathbb{R}^4$ is a particular solution of the system, and $k \in \mathbb{R}^4 \backslash \{0\}$ is in the kernel of $(\mathbf{c}|\mathbf{A})$. We set $k = (1, -\mathbf{A}^{-1}\mathbf{c})$ and $y = (0, \mathbf{A}^{-1}(\mathbf{b}-\mathbf{t'}))$. Using these values, we may derive a quadratic equation for α from Equation (5.1). We thus obtain triples (λ, γ, μ) that define intersections.

5.4 Results

The 2D version of this algorithm has been implemented in a small application written in Python. The mouse controls a virtual scalpel that can perform cuts in uniform meshes on a square grid. A sample is shown in Figures 5.8 and 5.16.

Fix-up, node removal and exits depend on points being close to or far from the active node. These notions are expressed in global constant thresholds for the distance. These thresholds are denoted by $\varepsilon_{\mathrm{edge}}$ for node removal, $\varepsilon_{\mathrm{entry}}$ for entry fixup, and $\varepsilon_{\mathrm{exit}}$ for predicting exits. They were set to a fraction of the average global edge length $\bar{h}$. For Figure 5.8 we used $\varepsilon_{\mathrm{edge}} = \bar{h}/3$, and for Figure 5.16 we used $\bar{h}/3$ and $\bar{h}/8$. The entry and exit tolerances were set to $\bar{h}/6$. This choice is somewhat arbitrary, but if $\varepsilon_{\mathrm{edge}}$ is set larger than $\bar{h}/2$, mesh complexity will be *reduced* during a cut. Lower values set a threshold for edge length. This does not directly control the accuracy of the cut trajectory represented in the mesh: the example in Figure 5.16 shows that lowering the threshold can decrease accuracy.

In Figure 5.16 the new cutting scheme for 2D is compared with a subdivision approach, where each sliced triangle is replaced by three triangles. Element shapes are much worse in the subdivision mesh, as evidenced by the histograms of minimum and maximum triangle angle in Figure 5.17. On the other hand, the subdivision cut follows the scalpel path more closely.

The 3D version of this algorithm has been implemented in a small application written in C++. It also uses the mouse to obtain scalpel movements. Samples are shown in Figures 5.18 and 5.19. The 3D version assumes that all events are strictly ordered in time, and does not take special precautions for degenerate cases. When the scalpel enters or exit in a movement parallel to the surface, multiple events happen simultaneously, which leads to various failures.

Flips can be implemented in constant time and node removals can be accomplished in $\mathcal{O}(d \log d)$, where d is the degree of the node [34]. The expected value of d in surface meshes is 6, so the total remeshing process for a single movement can be implemented in constant time, which guarantees that the remeshing process itself is scalable to larger meshes. However, in the 3D version the scalpel can interact with all parts of the mesh at every step. In practice, efficient collision detection will be needed for a scalable implementation. Also, the number of events caused by a single scalpel movement depends on the angle between the surface and the scalpel. However, for small meshes (say, 1000 triangles) response of the 3D version is instantaneous when scalpel is more or less perpendicular to the surface (on a P3/1Ghz).

5.5 Discussion

We have presented an approach to cutting in triangulations that produces measurably smaller and better-shaped meshes than subdivision methods. The method is based on a model of a point-shaped scalpel that moves an active node through a static mesh. During movements, the area around the active node is remeshed. The approach generalizes curved surfaces in 3D, where the scalpel is line-shaped. The technique bears some resemblance to interactive mesh dragging, a technique proposed by Suzuki et al. [90] to

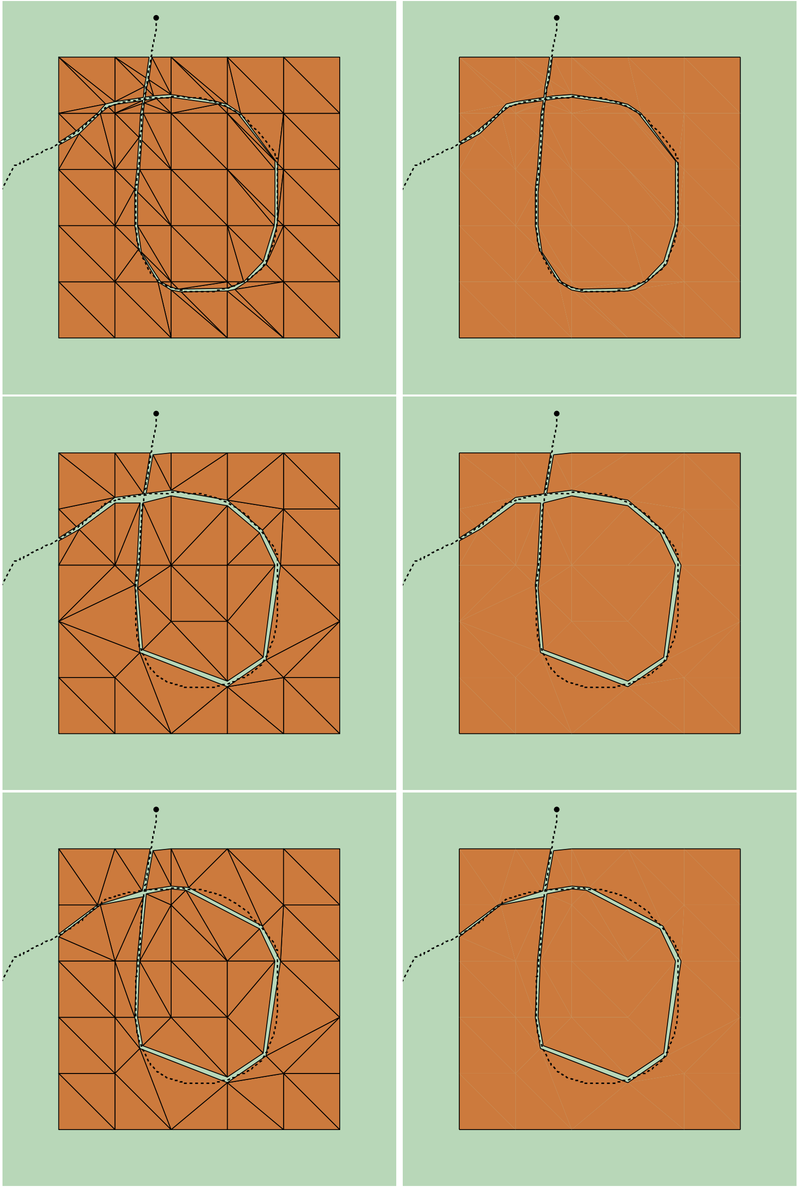


Figure 5.16: The same cut performed with both subdivision (top) and Delaunay cutting (center, $\varepsilon_{\text{edge}} = \bar{h}/3$ and bottom, $\varepsilon_{\text{edge}} = \bar{h}/8$). The starting mesh contains 50 triangles. The subdivision cut increases size by 62 triangles, the Delaunay cut in the center by 10, and at the bottom by 12 triangles. The scalpel trajectory is indicated with a dotted line. Notice that the mesh in the center picture matches the trajectory better than the mesh at the bottom, although $\varepsilon_{\text{edge}}$ was smaller.

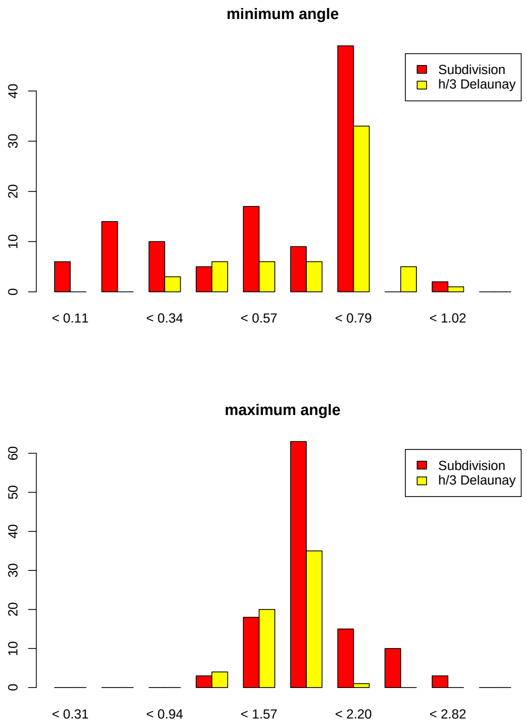


Figure 5.17: Histogram of minimum (top) and maximum (bottom) element angle for the subdivision and $\bar{h}/3$ Delaunay cut from Figure 5.16. The peaks correspond to the unaltered triangles (minimum angle $\pi/4$, maximum angle $\pi/2$). The Delaunay cut (right bars) yield less extreme element angles than the subdivision cut.

allow dragging operations on triangle surfaces in interactive geometric modelers. It is possible to generalize the single-incision 3D technique to multiple incisions consistently.

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

We have used a static model for mesh cutting. In the context of surgery simulation and deformable models, this is not realistic. Surgical instruments always interact with the deformed mesh, and exert force on the tissue that is cut. A full-fledged simulation would be able to respond to movements with reaction forces, which could be relayed to a force-feedback device. Deformation could be handled as follows: both the 2D and 3D algorithm include a step where the new position of the scalpel is intersected with parts of

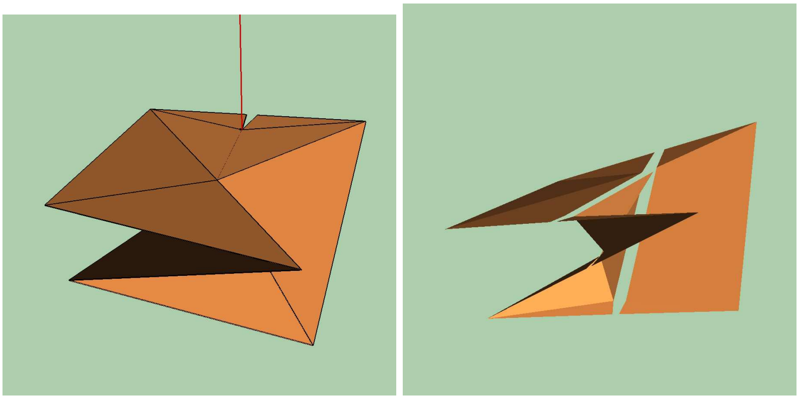


Figure 5.18: In principle, our approach also works for a cut branching into three cuts. The mesh has very sharp angles and a limited resolution, so edge flips produce an odd end result.

the mesh. When deformation is present, these intersections are done with the deformed mesh. The deformation of the triangles involved can be used to translate the point back to the reference situation. Our technique can then proceed as described using reference locations. However, this does not address a more fundamental limitation: our model assumes that a more or less uniform mesh is desirable. It can be expected that more accuracy for the deformations are required close to the scalpel, since this is where forces are exerted on the material. When accurate deformations are needed, the mesh should be adaptively refined close to the scalpel incision, and coarsened further away.

Entry and exit are also changed by the presence of deformation: surgical instruments must overcome a threshold in force to puncture membranes covering organs [15, 36]. Cutting requires less force than puncture, so after the instrument enters the tissue, it will make an incision immediately, ensuring that entry does not lead to arbitrarily small incision depths. This is a physical variant of our entry-fixup, and it would make our own entry-control superfluous. A similar observation might also hold for a scalpel finishing a cut. Removing entry control is attractive, since our approach depends on a heuristically chosen quantitity $\varepsilon_{\text{entry}}$, and it is possible to create a deadlock situation: if a scalpel entry has to be fixed up, the active node is part of the boundary. It is not always possible to preclude element inversion when moving boundary nodes, so the step-size control for large movements may halt further movements in some cases.

During a cut, nodes are added and removed; the nodes are added on line segments connecting existing nodes, which implies that the overall resolution of the mesh does not increase. Some interactive simulations of deformable objects refine meshes on demand to provide more accurate results in the region of interest [36, 76, 100]. Delaunay refinement algorithms [25, 82] seem to fit our framework of using Delaunay Triangulations, however more research is needed before this can be used in practice. Refinement algorithms need input geometries without small angles. Moreover, in 3D a surface triangulation is an approximation of a smooth surface. It is necessary to know the original

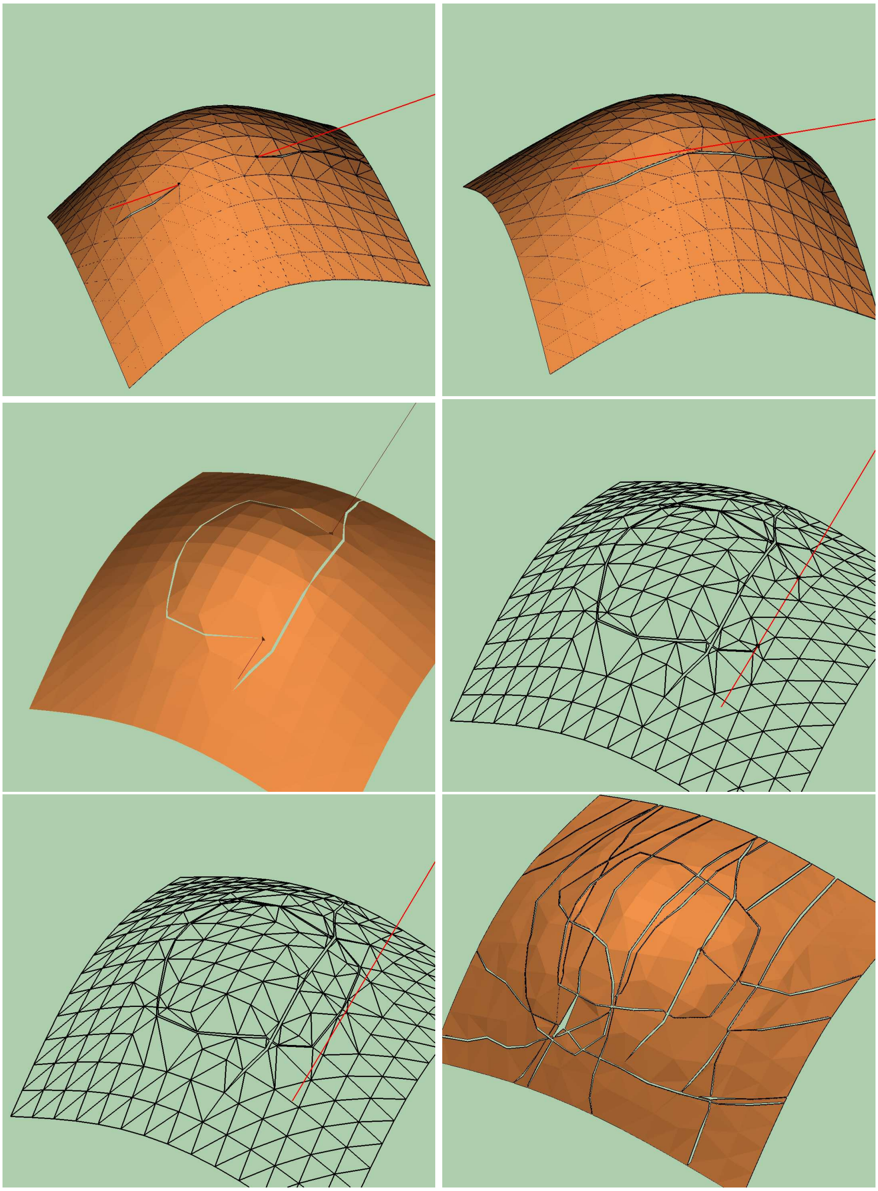


Figure 5.19: Screenshots from the 3D prototype, showing the cuts effected with mouse movements. The mesh is a 392 triangle surface mesh of a Gaussian bell-curve. The first 5 images show the evolution of two cut movements. The final image demonstrates more self-intersecting paths.

surface for determining where to insert new vertices; existing 3D surface meshing algorithms, such as Chew's guaranteed quality surface Delaunay refinement [25], also assume that a such smooth description is available. Hence, a cutting algorithm using refinement should be redesigned with the assumption that a surface shape itself is known.

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

Unfortunately, both the rationale for using Delaunay triangulation and our heuristics do not readily generalize to tetrahedral meshes. Delaunay tetrahedralizations of well-spaced points admit *slivers*, tetrahedrons that have four nearly planar and nearly cocircular vertices (see Figure 5.20). Slivers are flat tetrahedrons and therefore undesirable. Moreover, the higher-dimensional equivalent of edge flipping (face flipping), does not always work: there exist configurations of non-Delaunay faces which can not be flipped. This implies that flipping as a local improvement strategy does not always work. If the tetrahedralization is already Delaunay, then flipping non-Delaunay faces during insertion or removal of points in a mesh is always possible. Since cuts are non-convex, this suggests that a tetrahedral generalization must build a constrained or conforming Delaunay tetrahedralization with a moving boundary. A first step towards enabling such cuts would be to extend the current 3D surface cutting approach to cuts of closed surfaces: when cutting a closed surface, surface triangles are added to the inside of the incision, so the surface remains closed during cuts. This surface could then be used as a basis for making a tetrahedralization. An additional complication is that a constrained Delaunay tetrahedralization only exists if edges have no non-incident nodes close by (the surface should be *ridge-protected*) [86].

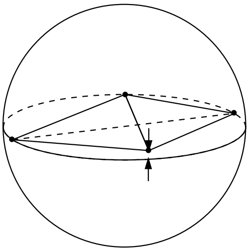


Figure 5.20: Slivers have an excellent circumcircle to shortest-edge ratio, so they can be present in Delaunay tetrahedralizations of well-spaced point sets. They are degenerate nevertheless.

Chapter 6

2D needle insertion

Inserting needles into soft tissue is one of the most often performed medical procedures. In some procedures, needles have to be inserted deeply into soft tissue to reach a target. For example, certain types of biopsies (taking tissue samples from organs) are performed with needles with a specialized needle tip, designed to extract a tissue specimen. Another application is *brachytherapy*, treating cancer by inserting radioactive seeds directly in the tumor. The primary application of brachytherapy is prostate cancer. In this application the seeds are delivered with a needle [3]. In general, when a needle is inserted into soft material, such as tissue, the material and the needle interact through friction forces, resulting in deformation of the material. This deformation makes it harder to reach the desired target and avoid vital organs. Simulations of needle insertions that take into account this deformation may help train and plan for such difficult cases.

6.1 Related work

Simulations of needle insertion for training purposes have been implemented earlier using ad-hoc models, e.g. [15, 60], which rely solely on force measurements performed at the inserting end. The work by DiMaio and Salcudean [36–38] presents a breakthrough. They present a planar virtual environment for needle insertion, where the mechanics of the system are derived from complete 2D measurements of deformation. These measurements are performed by recording a standardized needle insertion in a flat square slab of soft material with a camera. The video sequence is used to measure the deformation of the slab. By comparing the measured deformations with a FEM simulation of the same material, the friction forces along the needle shaft are estimated. An interactive simulation is then built, where the same friction profile is used to model the interaction between tissue and needle. This simulation uses a static linear elasticity model on a fine 2D grid. During the simulation, only a small portion of nodes is 'visible', i.e. relevant to the system response: the nodes on the boundary are needed for visualization, and those that interact with the needle are necessary for computing interactions with the tissue. The responses of this subset can be can be condensed into a small stiffness matrix, which is inverted and updated on the fly. As the needle advances into the tissue, more nodes are added to the condensed system. Updates of the inverted matrix take $\mathcal{O}(s^2)$ operations, where s is the number of visible nodes. Similarly, the effect of friction is to change the boundary conditions for nodes on the needle; these updates can also be done in $\mathcal{O}(s^2)$. The condensation process requires that the stiffness matrix corresponding to all the nodes of the mesh is inverted off-line.

Alterovitz et al. [2,3] also simulate needle insertion in FEM meshes. They use a linear elasticity model on a regular 2D grid, but use an explicit GN22 scheme to dynamically solve these equations. GN22 is similar to the SS22 scheme from Equation (2.60). The model runs at visual update rates (25 Hz) for a 1250 triangle mesh. They specifically target brachytherapy for prostate cancer treatment. By varying tissue and friction parameters in the model, the effect of tissue parameters and needle characteristics on seed placement accuracy can be predicted.

6.2 Needle insertion in 3D

The needle applies force to the material, and can be considered a part of the boundary for the boundary value problem. In a two-dimensional setting, the boundary is one-dimensional, and can be represented accurately by a small set of edges. In 3D, forces should be distributed over surfaces. In other words, for a physically valid discretization in 3D, the needle cannot be represented as a set of edges, but must be represented as a surface. To represent this surface, the mesh must include elements with a size comparable to the needle diameter.

If we assume that a needle has a diameter of 1 mm, then the mesh should have elements of that size in the vicinity of the inserted needle. It is clear that there will be huge disparity between element size and object size. Let us assume that the object is cube shaped, is 10 cm in length in every axis, and is discretized using a uniformly refined mesh. The mesh would have approximately $10\,\text{cm}/1\,\text{mm} = 100$ nodes for each side, leading to approximately 10^6 nodes and $3 \cdot 10^6$ degrees of freedom. The dimensions of the stiffness matrix are $3 \cdot 10^6 \times 3 \cdot 10^6$. Simulations of this size cannot be run interactively in the forseeable future. For example, condensation of internal nodes requires storing the inverse of the stiffness matrix. The inverse is dense, and storing a dense $3 \cdot 10^6$ DOF matrix requires approximately 67 terabytes of memory. Moreover, for nonlinear elasticity, the stiffness matrix depends on the deformations. Since the stiffness matrix is not constant, precomputing matrix inverses does not make sense.

For large or nonlinear problems, iterative methods have to be used, and these do not require precomputed structures. If we do not rely on precomputed structures, then there is no need to start with a fine mesh, but we can refine the mesh where necessary: this is where the needle is inserted. In this chapter, we will use these observations to produce a 2D simulation that is functionally equivalent to the one shown by DiMaio and Salcudean, but uses an iterative solution method and adaptive mesh resolution. With this method, it is possible to exchange computation time and accuracy. To asses whether this approach can be called interactive, we assess how quickly our implementation runs for reasonable accuracy requirements. Before we give these results, we show how the problem is modeled, and how the simulation is implemented.

6.3 Physical model

The physical behavior of the system has two major aspects. The mechanical behavior of the tissue itself is governed by equations for two-dimensional elasticity, so-called *plane stress*. The interaction between the needle and the tissue is a form of *stick-slip* friction. When the needle moves slowly, material sticks to the needle. At higher speeds the material slips across the needle. Hence, the needle forms a part of the boundary where either displacements (stick friction) or tractions (slip friction) are prescribed.

6.3.1 Elasticity

For problems where loads and deformations occur in a plane, one coordinate can be removed from the 3D elasticity formulation. We will analyze the situation shown in Figure 6.1. We describe $\mathbb{R}^3$ in matrices with respect to the unit basis $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$. The coordinates are also written as $\mathbf{z} = (x, y, z)$. The object that we will analyze is a slab of material, lying in the xy-plane, symmetrically around $z = 0$. We assume that deformations in the z-coordinate are uniform, so we describe the z-coordinate of a motion as $p_3(x, y, z) = zq(x, y)$, for some function q. The motion $\mathbf{p}$, mapping reference points to deformed locations, is given by

$$\mathbf{p}(x, y, z) = \begin{pmatrix} p_1(x, y) \\ p_2(x, y) \\ zq(x, y) \end{pmatrix},$$

for some functions $p_1, p_2 : \mathbb{R}^2 \to \mathbb{R}$.

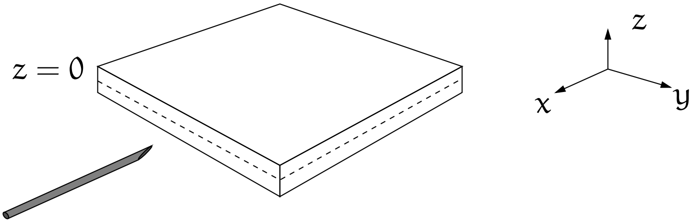


Figure 6.1: Plane stress. When loading thin slabs of material, the z sides are unloaded. Without loss of generality we assume that the slab is positioned around the $z = 0$ plane.

For ease of notation, we leave out the (x, y) dependencies, and denote $\partial f/\partial x$ by f_x. The deformation gradient is given by

$$\mathsf{F} = \begin{pmatrix} p_{1,x} & p_{1,y} & 0 \\ p_{2,x} & p_{2,y} & 0 \\ zq_x & zq_y & q \end{pmatrix}.$$

The strain tensor $\mathbf{C}$ can be written as

$$\begin{pmatrix} p_{1,x}^2 & p_{2,x}p_{1,y} & 0 \\ p_{2,x}p_{1,y} & p_{2,y}^2 & 0 \\ 0 & 0 & q^2 \end{pmatrix} + z \begin{pmatrix} zp_{1,x}^2 & zp_{2,y}p_{2,x} & qq_x \\ zp_{2,y}p_{2,x} & zp_{2,y}^2 & qq_y \\ qq_x & qq_y & 0 \end{pmatrix}.$$

In the $z = 0$ plane, the right term disappears. Except for the q^2 term, the left term is the exact 2D analogon of $\mathbf{C}$.

The q^2 term is eliminated by using the constitutive equations: when we suppose that no stresses act in the third dimension, then by the definition of the first Piola-Kirchoff stress tensor, we should have

$$\mathsf{T} \cdot \boldsymbol{e}_3 = 0.$$

The second Piola-Kirchoff stress tensor $\mathbf{S}$ is defined by $\mathsf{F}^{-1} \cdot \mathbf{T}$, so we have $\mathbf{S} \cdot \boldsymbol{e}_3 = 0$ as well. Since $\mathbf{S}$ is symmetric, we conclude that

$$\mathsf{S} = \begin{pmatrix} \tilde{S} & 0 \\ 0 & 0 \end{pmatrix}, \tag{6.1}$$

where $\tilde{\mathbf{S}}$ is the 2D restriction of the second Piola-Kirchoff stress tensor.

We will first analyze the results for the linear material from Equation (2.16),

$$\mathbf{S} = \mu(\mathbf{C} - \mathbf{I}) + \frac{\lambda}{2}\operatorname{trace}(\mathbf{C} - \mathbf{I})\mathbf{I}. \tag{6.2}$$

It follows from Equation (6.1) and Equation (6.2) that

$$0 = \mu(q^2 - 1) + \frac{\lambda}{2}(\operatorname{trace}(\tilde{\mathbf{C}} - \tilde{\mathbf{I}}) + (q^2 - 1)),$$

where $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{I}}$ are the 2D analogons of $\mathbf{C}$ and $\mathbf{I}$. Therefore,

$$q^2 - 1 = \frac{2\mu\lambda}{\lambda + 2\mu}\operatorname{trace}(\tilde{\mathbf{C}} - \tilde{\mathbf{I}}).$$

It follows that

$$\tilde{\mathbf{S}} = \mu(\mathbf{C} - \mathbf{I}) + \frac{2\mu\lambda}{\lambda + 2\mu}\operatorname{trace}(\tilde{\mathbf{C}} - \tilde{\mathbf{I}})\tilde{\mathbf{I}}, \tag{6.3}$$

We see that the reduction from 3D to 2D only impacts the volume-preservation part of the equations, and can be derived by setting $\boldsymbol{e}_3 \cdot \mathbf{S} \cdot \boldsymbol{e}_3 = 0$, and substituting the result for q^2 back into the constitutive equation. More complex constitutive equations give more complex expressions for q^2. However, if we set $\lambda = 0$ (or equivalently, $\nu = 0$), then the material has no volume preservation at all, and the 3D formulas can be translated to 2D directly. For example, Equation (4.6) specifies that neo-Hookean material satisfies

$$\mathbf{S} = \mu\mathbf{I} + (-\mu + \lambda(\sqrt{\iota_3} - 1)\sqrt{\iota_3})\mathbf{C}^{-1}.$$

When $\lambda = 0$, then setting $\boldsymbol{e}_3 \cdot \mathbf{S} \cdot \boldsymbol{e}_3 = 0$ implies $q^2 = 1$, and we can set

$$\tilde{\mathbf{S}} = \mu(\tilde{\mathbf{I}} - \tilde{\mathbf{C}}^{-1}). \tag{6.4}$$

We see that Equations (6.4) and (6.3) are the direct 2D analogons of 3D equations in (2.22) and (4.6). The tensors $\mathsf{T}, \mathbf{S}$, and $\mathcal{E}$ have their familiar meanings, but now they are located in $\mathbb{R}^{2\times 2}$. We use linear triangle elements, so the gradient $\mathbf{G}$ of the deformation is given by $\mathfrak{U} \cdot \mathsf{Z}^{-1}$. Nodal elastic forces are given by $\mathsf{T} \cdot \mathsf{Z}^{-*}$.

A difference between 2D and 3D is formed by the traction. If the 2D traction (unit N/m), is denoted by $\tilde{t}$, then the 3D traction t (unit N/m^2) is given by

$$t = \tilde{t}/d,$$

where d is the thickness of the slab of material.

6.3.2 Stick-slip friction

During a needle insertion, the needle and the material interact, exchanging forces. These forces are caused by friction. The magnitude of the friction depends on the relative velocity of the needle, i.e. the difference between the velocity of the needle and the material.

Stick-slip friction for an elastic system with one degree of freedom, like the one in Figure 6.2, leads to an ordinary differential equation. In this case, a mass m is attached to a spring with spring constant k, and the mass is subject to stick-slip friction f^{fr} applied by a needle moving at velocity v_{needle}. The friction depends on the relative velocity $v_{needle} - v$. The differential equation describing the movement u and velocity v of the mass are given by

$$\begin{pmatrix} \dot{v} \\ \dot{u} \end{pmatrix} = \begin{pmatrix} \frac{1}{m}(ku + f^{fr}(v_{needle} - v)) \\ v \end{pmatrix}. \tag{6.5}$$

If the relative velocity is nonzero, then the friction force is nonzero and has the opposite direction. An example of a velocity/friction function is in Figure 6.3. In general, the right-hand side of differential equation (6.5) is discontinuous. Integrating such systems requires special precautions, and small time steps [59].

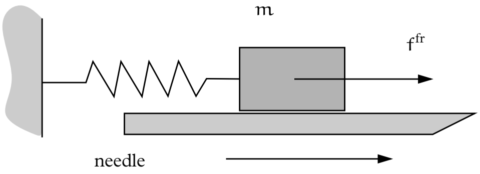


Figure 6.2: A single-degree of freedom stick-slip system

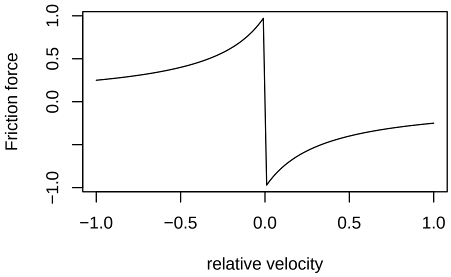


Figure 6.3: A typical stick-slip friction curve, after [59]. The force has a discontuity at $v_{needle} - v = 0$, causing problems during time integration.

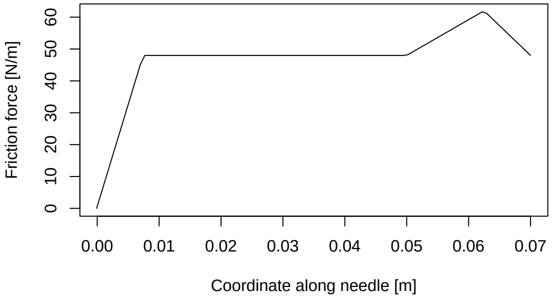


Figure 6.4: The force distribution used for experiments, for the insertion. Friction force varies along the shaft of the needle, and this graph shows how. For stick/slip friction, this distribution is used with deformed coordinates along the x-axis.

Due to these numerical problems, we use a quasi-static formulation. Nodes that lie on the needle, *needle nodes*, can have two states: either a node is fixed (sticking), which means that its location is attached to the user controlled needle and $v_{\mathrm{needle}} = v$, or its movement is constrained to be parallel to the needle shaft (slipping). A configuration of slipping and sticking nodes leads to a single static deformation problem. When the solution of this problem is found, the elastic reaction forces can be used to rearrange the boundary conditions. For every needle node, a static and a dynamic friction threshold is defined. Fixed nodes whose reaction forces exceed the static friction threshold are loosened, and slipping nodes whose elastic forces are less than the dynamic friction threshold are fixed again. If any of the conditions on the nodes are changed, then this rearrangement defines a new static problem, and the procedure is repeated. Once no further rearrangements are necessary, the final solution has been reached. In effect, this procedure is an iterative approach. To distinguish this outer-loop iteration from the relaxation procedure (an inner loop), we shall refer to these iterations as *rearrangements*.

The magnitude of the friction forces on points along the needle shaft is variable. DiMaio and Salcudean demonstrated [36] that the force distribution is similar to the one plotted in Figure 6.4, which is what we will use throughout this chapter. The force bulge near the tip accounts for the force required to make the needle tip cut into the material.

6.4 Needle representation

We will discretize the elasticity problem using the Finite Element Method on a triangle mesh. As we explained above, the needle surface is part of the boundary of the domain. In a *conforming* finite element method, the space of shape functions should be a subspace of the total solution set of the original equation. In the case of needle insertion, this implies that the needle should be represented in the mesh by a set of connected edges. In previous work, this was achieved by either relocating existing mesh nodes to be on the needle surface [2, 36], by inserting new nodes [2], or by deforming material locally to force nodes to be on the needle [38].

In Chapter 3 and Chapter 5 changes to the mesh were driven by visual reasons. Without subdivision or relocation, cuts will have a jagged appearance which looks unrealistic. For needle insertion, the needle itself is not visible. When we look at the solution accuracy, we observe that in general, the accuracy of a FEM solution is bounded by h, the size of the element, and p, the smoothness of the basis functions. For a given problem with solution $\hat{u}$, a linear FEM discretization will satisfy the following error bound for the approximation u_h [14]

$$\begin{aligned} \|\hat{u} - u_h\|_\infty &= \mathcal{O}(h^2|\log(h)|^{3/2}), \qquad h \to 0 \\ \|\hat{u} - u_h\|_2 &= \mathcal{O}(h^2), \qquad h \to 0. \end{aligned} \tag{6.6}$$

Neither relocating nodes in the mesh, nor unjudicious use of subdivision decreases h, so in general, neither will improve the accuracy of the solution.

In Chapter 3 we have seen that node relocation can easily lead to flattened elements, and in Chapter 5, we have seen that accomodating surface in the mesh by subdividing intersected elements can easily lead to short edges and a significant increase in mesh size. These problems motivate us to experiment with a scheme where nodes are not moved within the mesh.

The needle is assumed to be an inflexible line segment. When the needle tip enters an element through an edge, the closest node of the edge is marked as a *needle node*. The location of the node is decomposed in two components, as shown in Figure 6.5. The distance along the needle shaft between nodes, d_{needle}, is used as a basis for friction computations. The component perpendicular to the needle, d_{perp}, is used to compute new node positions when the needle is displaced.

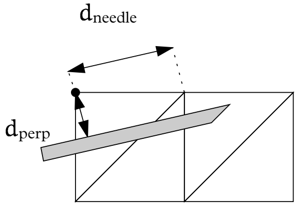


Figure 6.5: When needle nodes are intercepted, their position is decomposed in a component d_{perp} perpendicular to the needle, and a component d_{needle} along the needle. The component d_{perp} is held fixed during needle movements and d_{needle} is used for friction calculations.

6.5 Edge bisection

We will consider a square object that can be meshed with regularly arranged right-angled triangles, as shown in Figure 6.8. This is not an essential restriction, but it does ease the discussion and the implementation. To accomodate the needle, the resolution of the mesh is locally increased during a simulation. This refinement is done using *edge bisection*. In this case, the basis of edge bisection is a *simple bisect*, splitting an edge which is the hypotenuse of its two incident triangles. The following pseudo-code demonstrates the bisection of an edge (p, q), and is illustrated in Figure 6.6.

```
procedure simple-bisect ($p, q$):
  $n \leftarrow$new node halfway between $p$ and $q$.
  let $f_1$, $f_2$ be the faces incident with $(p, q)$
  if $f_1 \neq$ null:
    $f_1' \leftarrow f_1$ with $p := n$ substituted
    $f_1'' \leftarrow f_1$ with $q := n$ substituted
  if $f_2 \neq$ null:
    $f_2' \leftarrow f_2$ with $p := n$ substituted
    $f_2'' \leftarrow f_2$ with $q := n$ substituted
  replace $\{f_1, f_2\}$ with $\{f_1', f_1'', f_2', f_2''\}$.
```

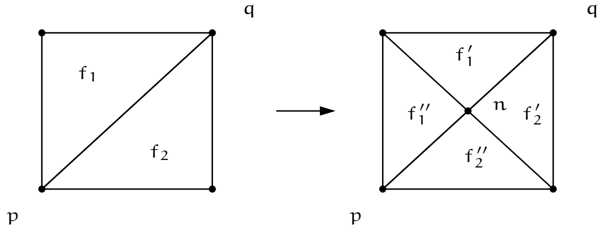


Figure 6.6: In a simple bisection, the faces $\{f_1, f_2\}$ are replaced by $\{f_1', f_1'', f_2', f_2''\}$.

The bisected edge must be the hypotenuse of its incident triangles. If this is not the case, the following recursive procedure first bisects neighbors before it bisects the specified edge e. It is illustrated in Figure 6.7.

```
procedure bisect($e$):
  let $f_1$, $f_2$ be incident with $e$
  if $f_1 \neq$ null $\wedge$ $e \neq$ hypotenuse$(f_1)$:
    bisect (hypotenuse ($f_1$))
  if $f_2 \neq$ null $\wedge$ $e \neq$ hypotenuse$(f_2)$:
    bisect (hypotenuse ($f_2$))
  simple-bisect ($e$)
```

For a mesh consisting of right-angled triangles, this refinement process is equivalent to two other techniques. The hypotenuse is always the longest edge in a triangle, and hence this procedure is equivalent to the so-called longest edge bisection [81]. When

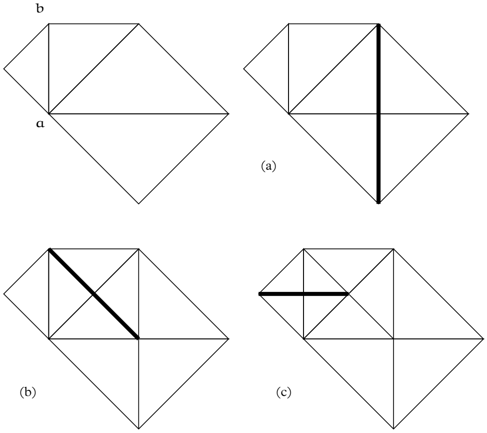


Figure 6.7: When an edge is bisected that is not the hypotenuse of its incident triangles, then its neighbors are first bisected: in pictures (a), and (b) neighbors are bisected, in (c), finally ab is bisected.

bisecting a hypotenuse, the angle opposite the hypotenuse is bisected. Since that angle is always the newest vertex in a triangle, we have a form of newest node bisection [85].

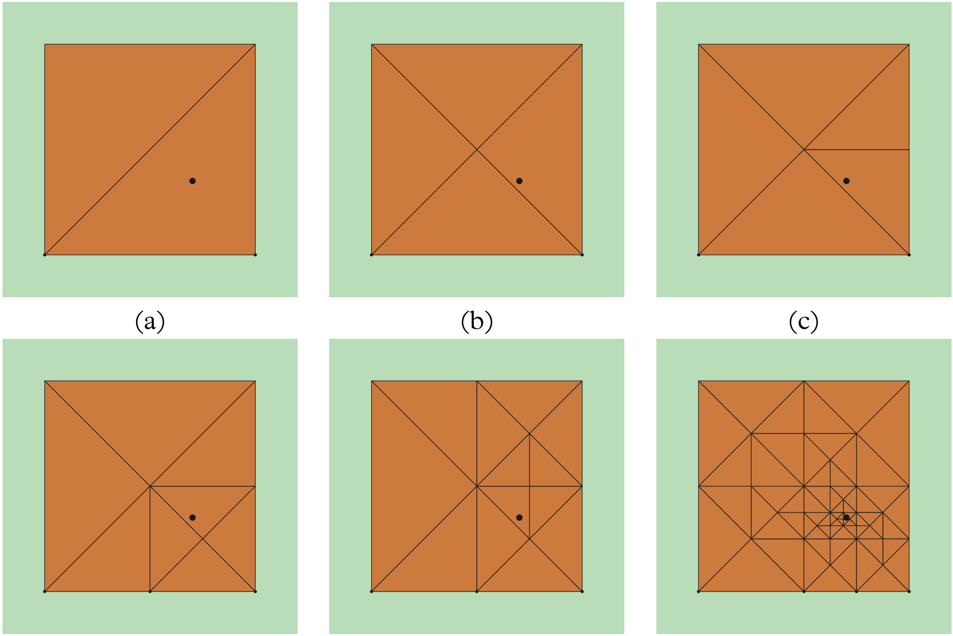
The bisection technique we described has the following properties.

- It is easy to implement.
- Triangles are naturally ordered in a hierarchy. This hierarchy can be stored in a binary tree. This tree is in effect a BSP tree, suitable for efficient point-location.
- All triangles are congruent to triangles of the starting mesh, and therefore well shaped.
- It can be extended to simplexes in arbitrary dimensions, and this extension maintains the above properties [62]. The only requirement is that the nodes are ordered in some way.

The starting mesh can be refined uniformly by subdividing all hypotenuses in the mesh repeatedly. The mesh can also be arbitrarily refined in a region, by repeated bisection. The following procedure refines around the point x until the edge lengths around x are less than h.

```
procedure refine-around ($x$, $h$)
  $t \leftarrow$ triangle containing $x$
  if $|$hypotenuse $(t)| > h$:
    bisect(hypotenuse (t))
    refine-around ($x$, $h$)
```

Figure 6.8 illustrates this refinement process.



(a) (b) (c)

(d) (e) (f)

Figure 6.8: Recursive edge bisection, with refinement around the point marked with a dot. Pictures (a) shows the starting mesh. Picture (b) and (c) show simple bisections. In Picture (d) neighbors are bisected as well, and in Picture (e), the bisections propagate even further. Finally, Picture (f) shows the result after 10 refinements.

6.6 Solution method

We will use the CG method, both in its linear and nonlinear form, to compute solutions to the FEM problem. Two modifications with respect to the method of Chapter 3 are necessary. There, we have only considered fixed nodes. In the case of needle insertion, nodes can also be constrained to move along a line. In effect, both types of constraint restrict the minimization problem on $\mathbb{R}^n$ to a subspace $W \subset \mathbb{R}^n$. In other words if u is a given starting configuration, and W the space of allowed node movements, then the solution is given by

$$\min_{w \in W} \Pi(u + w).$$

If P_W is the orthogonal projection on W, then we can rewrite this as

$$\min_{v \in \mathbb{R}^n} \Pi(u + P_W v). \tag{6.7}$$

The latter is a minimization problem in $\mathbb{R}^n$, and hence we can solve it with the techniques discussed in Section 2.4, using the function in (6.7) as objective In other words, we take

$$P_W \left(\frac{\partial \Pi}{\partial u}(u + P_W v) \right) \quad \text{and} \quad P_W \left(\frac{\partial^2 \Pi}{\partial u^2}(u + P_W v) \right) P_W$$

for the derivative and second derivative. In effect, search directions and second derivatives are first projected onto the constraints before they are used in the algorithms.

A second complication is the stopping criterion. The stopping in Chapters 3 and 4 uses the 2-norm of the residual force vector r, i.e.,

$$\|r\|_2 \le \varepsilon \|f^{\mathrm{ex}}\|_2, \tag{6.8}$$

for some *relaxation tolerance* $\varepsilon > 0$. This criterion does not work when nonzero displacements are prescribed without applying external forces: then the right handside is 0, which is impossible to satisfy due to rounding errors. We do have access to f^{react}, the reaction forces necessary to keep nodes in their constrained positions; these can also be regarded as a form of external force. They can be found as

$$(I - P_W) \frac{\partial \Pi}{\partial u}(u).$$

Therefore, we propose the following stop criterion for the relaxation loop.

$$\|r\|_2 \le \varepsilon \sqrt{\|f^{\mathrm{ex}}\|_2^2 + \|f^{\mathrm{react}}\|_2^2} + \frac{EdH}{\sqrt{n}} \varepsilon_{\mathrm{round}}. \tag{6.9}$$

In this criterion, $\varepsilon_{\mathrm{round}} \ll 1$ is a separate tolerance, H is the diameter of the object, and E is the Young modulus. In this equation, the first term is the analogon of Equation (6.8) that also uses reaction forces. The second term of the right hand side becomes significant when the first term is almost 0 due to rounding errors. In that case, the first term may be hard to satisfy due to rounding errors in the evaluation of $\|r\|$. The second term is a rough estimate of the rounding errors in both quantities. The quantity dH is a measure of the area of the object, so EdH is a scale-free measure of the force magnitude. The factor $\varepsilon_{\text{round}}/\sqrt{n}$, with , is a measure for the roundoff errors that accumulate during the evalution of $\|r\|$ and $\sqrt{\|f^{\text{ex}}\|_2^2 + \|f^{\text{react}}\|_2^2}$.

This stopping criterion does not use the linear elasticity assumption in any way, and may therefore be used for nonlinear iterations as well. For 3D, the term dH should be changed, for example to H^2.

6.7 Implementation

The previous sections provide enough technniques to implement a needle insertion simulation based on edge bisection and static elasticity. The implementation was based on the deformation and relaxation framework written in Chapter 4, and the mesh data structure that will be described in Chapter 7.

In the implementation, the starting mesh is uniformly refined until edge lengths have a specified length h_{start}. Then the simulation enters the following interaction loop where needle movement, friction and refinement are handled.

```
while true:
  move needle
  refine mesh down to $h_{ref}$ around the needle tip
  add intercepted nodes to needle
  compute solution using CG
  rearrange boundary conditions
  unconstrain nodes that slid off the needle
```

When the needle is moved, the locations of all needle nodes are expressed in d_{perp} and d_{needle}. Needle nodes are moved by computing d_{perp} and d_{needle} relative to the old needle position, and using them as coordinates relative to the new needle position.

6.8 Computational experiments

The question to be answered is whether the simulation is practically useful. As far as the physical model is concerned, both our scheme and the one of DiMaio and Salcudean are equivalent, since they use the same elasticity model, and the same friction parameters. Therefore, we can achieve the same result, provided that we may spend as much computation as necessary to attain the desired accuracy. Therefore, the real question is: how accurate is the system for a given amount of computation? We will measure accuracy in terms of average and maximum errors in the displacement. We set the desired accuracy at approximately 1 mm, which roughly corresponds to the resolution of CT scan and MRI scans.

The simulation runs a number of static linear problems in sequence. The question of accuracy versus speed will be answered by first considering the error in the simulation under static load. This gives us a combination of relaxation tolerance and mesh resolution. These settings are used to quantify the buildup of errors during a simulation. This error is considered by running an automated, standardized needle insertion at various insertion speeds. Timings of the same experiment show to what extent the simulation is interactive.

Finally, we will consider the error that is introduced by not relocating nodes, the *conformance error*. An advantage of our approach, which also allows nonlinear material, is demonstrated by comparing insertions for nonlinear and linear material.

6.8.1 Test situation

The test object is a slab with dimensions $0.10 \times 0.10 \times 0.01$ m, and material properties $E = 34\,\text{kPa}$, $\nu = 0.34$. We assume that the object is lying in in the xy-plane, with $0 \leq x \leq 0.1\,\text{m}$ and $0 \leq y \leq 0.1\,\text{m}$. Refinements of element size are done by factor 2: we set $h_k = 0.1\,\text{m} \cdot 2^{-k}$, and select both h_{ref} and h_{start} from $h_1, \ldots, h_7$. We have used $\varepsilon_{\text{round}} = 10^{-8}$ as the tolerance for rounding errors.

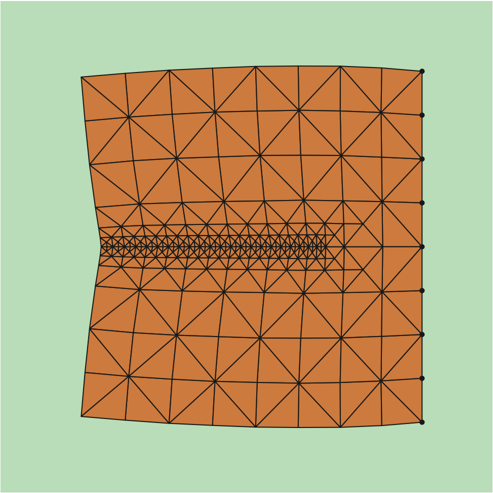


Figure 6.9: Experimental load for determining speed/accuracy. The object is fixed on the right, and a load is applied to the center, symmetrically. This picture shows $h_{\text{start}} = h_3$, $h_{\text{ref}} = h_6$.

The speed versus accuracy experiments are performed in a symmetric setting: in the static experiment, forces are applied along a line at $y = 0.05\,\text{m}$, while the boundary at $x = 0.1\,\text{m}$ is held fixed. This configuration is shown in Figure 6.9. It leads to a symmetric deformation. The “needle” is represented by edges in the mesh, and hence, the accuracy assessment is not affected by conformance errors. In the static problem, load is applied to the center of the object over a length of 70 mm, using the distribution in Figure 6.4. In the dynamic problem, the needle is inserted to a depth of 70 mm, with dynamic friction thresholds derived from the given force distribution. The static friction threshold is set at twice the dynamic threshold.

6.8.2 Residual tolerance

The stopping criterion in (6.9) is parameterized by a tolerance ε. In Table 6.1 shows how the the value of the tolerance affects displacement errors, by comparing solutions of the same problem calculated with different tolerances. We can see that very modest tolerances already are enough to make displacement errors small.

relaxation tolerance ε	average error [mm]	maximum [mm]
0.3	0.74	2.98
0.1	0.061	0.18
0.03	0.034	0.083
0.01	0.011	0.022
0.003	0.0038	0.0096
0.001	0.0008	0.0021

Table 6.1: Error dependency on tolerance for the residual force, relative to the solution computed at $\varepsilon = 10^{-8}$. This has been run on a mesh with $h_{\mathrm{ref}} = h_{\mathrm{start}} = h_7$.

6.8.3 Mesh density

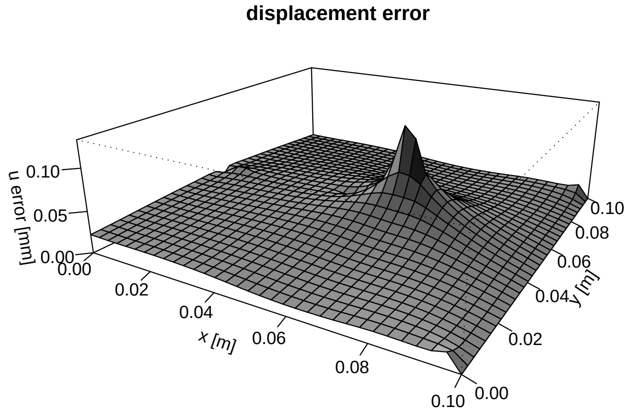
The impact of mesh resolution on the solution error was analyzed by comparing the results for $h_{\mathrm{ref}} = h_{\mathrm{start}} = h_7$ with results for coarser meshes. These results are given in Table 6.3. Here, both h values are varied from h_1 to h_7.

Plots of the displacement error for two combinations of h_{ref} and h_{start} are in Figure 6.10. It is evident that errors are especially present in the vicinity of "interesting" regions: regions where force varies, or near the corners of the fixed boundary. Hence, the mesh can be left more coarse outside these regions. This distributes the displacement error all over the object. Moreover, convergence is also quicker, as demonstrated in Table 6.2.

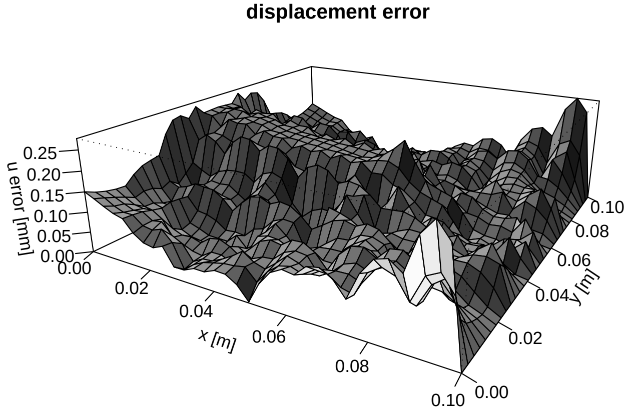
$\downarrow h_{\mathrm{start}}/h_{\mathrm{ref}} \rightarrow$	h_3	h_4	h_5	h_6	h_7
h_1	22	34	45	62	84
h_2	27	36	46	64	84
h_3	46	49	58	73	93
h_4		81	86	96	115
h_5			204	208	194
h_6				278	285

Table 6.2: CG Iteration counts for different combinations of h_{start} and h_{ref}. Reducing mesh resolution also decreases convergence requirements: information travels faster over coarser mesh parts, thus speeding up the relaxation.

The accuracy results for varying h_{start} and h_{ref} are in Table 6.3. In general, decreasing h_{start} and decreasing h_{ref} improves the accuracy. For $h_{\mathrm{start}} = h_3$ and $h_{\mathrm{ref}} = h_6$, we get an error of 0.44 mm, which is small enough by our standards.



h−refine = h_6, h−start = h_6



h−refine = h_6, h−start = h_3

Figure 6.10: Displacement errors, at the top $h_{\text{start}} = h_6$ and $h_{\text{ref}} = h_6$, on the bottom $h_{\text{start}} = h_3$, $h_{\text{ref}} = h_6$. Larger errors occur close to varying loads: the corners of the fixed sides, and close to the needle tip and entry point. Therefore, a finer mesh is only necessary in these regions. On the bottom, the mesh is coarser away from the needle, which leads to larger errors in those regions.

Average error [mm]					
$\downarrow h_{\text{start}}/h_{\text{ref}} \rightarrow$	h_3	h_4	h_5	h_6	h_7
h_1	0.34	0.33	0.33	0.30	0.28
h_2	0.26	0.24	0.24	0.21	0.20
h_3	0.15	0.13	0.13	0.10	0.08
h_4		0.09	0.08	0.04	0.03
h_5			0.07	0.03	0.00
h_6				0.02	0.00

Maximum error [mm]					
$\downarrow h_{\text{start}}/h_{\text{ref}} \rightarrow$	h_3	h_4	h_5	h_6	h_7
h_1	1.01	0.88	0.88	0.87	0.87
h_2	0.91	0.68	0.61	0.54	0.53
h_3	0.66	0.54	0.51	0.29	0.27
h_4		0.45	0.43	0.19	0.14
h_5			0.41	0.17	0.06
h_6				0.16	0.03

Table 6.3: Displacement errors for different mesh resolutions. These tables list mean and maximum displacement errors in mm (compared with a uniform mesh with element size h_7), with h_{start} vertically, and h_{ref} horizontally. Half of the table is empty, since $h_{\text{ref}} \leq h_{\text{start}}$. We see that smaller h size leads to smaller errors in general.

6.8.4 Insertion speed

If the needle moves very quickly during an insertion, it will pierce elements without requiring force. For example, if the needle moves from outside the object to its final position in one step, no forces will be exchanged between needle and tissue. For this reason, it can be expected that the insertion speed influences the end result. We assume that the better solutions are obtained when the speed of the needle is smaller. Hence, we measure that influence by performing automated insertions at different speeds, and comparing them to the results for a very slowly inserted needle.

The automated insertion is performed as follows: after each cycle of the interaction loop, the needle is advanced by a fixed amount, until it reaches the desired penetration depth. The loop continues until no more rearrangements of needle boundary conditions are needed. The system then has reached an equilibrium situation.

Results of this experiment are in Table 6.4. These results do not show a clear trend. For insertions speeds smaller than $1.0h_{\text{ref}}$ per update, we get errors that are in the same order of magnitude as the discretization error.

6.8.5 Timings

When the experiment of the previous subsection is timed, we can estimate how interactive the simulation is. Table 6.5 lists for each insertion speed the amount of CPU time used, and how many times the interaction loop from Section 6.7 is called. Di-

speed $[\frac{h_{\text{ref}}}{\text{rearrangement}}]$	avg error [mm]	max error [mm]
3.00	0.62	1.60
1.00	0.68	1.96
0.30	0.15	0.34
0.10	0.18	0.68
0.03	0.36	1.08
0.01	0.07	0.27

Table 6.4: Error introduced by insertion speed, for $h_{\text{start}} = h_3$ and $h_{\text{ref}} = h_6$, compared to insertion at the speed $0.005h_{\text{ref}}$. For this resolution, the discretization error was 0.11 mm average/0.29 mm maximum.

viding both gives an average update rate. We see that low insertion speeds give better update rates. This is confirmed by the average number of CG iterations necessary to find a solution. Smaller insertion speeds lead to a higher amount of coherence between subsequent solutions, thus reducing the average number of CG iterations required. For the lowest insertion speeds (0.10 h_{ref}/rearrangement and lower, we obtain update rates suitable for haptic interaction).

The experiment was done with two different refinement resolutions, $h_{\text{ref}} = h_6$ and $h_{\text{ref}} = h_7$. The number of rearrangements doubles when going from h_6 to h_7 (a decrease of a factor 2), this is consistent with the fact that insertion speed is proportional to h_{ref} in this experiment.

The average update rate is better than the worst update rate, which is indicated by the maximum number of CG iterations required. The maximum is larger than average. These larger counts are necessary during the beginning of the insertion, when only a small portion of the needle is in the tissue. Although the mesh is still relatively small at this stage, it might be necessary to set a fixed bound on the number of CG iterations when driving a force-feedback device.

6.8.6 Relocation errors

In the following experiment, we assess the magnitude of conformance errors. To this end, a node repositioning scheme similar to the one in Chapter 3 was introduced. Nodes selected as needle nodes are projected orthogonally onto the needle. Again, we quantify errors by applying a static load to the mesh, according to the friction distribution from Figure 6.4, but now forces are applied over a tilted line, so that the "needle" does not coincide with mesh edges. The resulting deformation is shown in Figure 6.11. By measuring the difference between a solution with relocation and without relocation to the solution a finer mesh, we can estimate the impact of relocation. Table 6.6 shows that these errors are smaller than the discretization errors, thus validating the approach.

6.8.7 Nonlinearity effects

Our solution method does not exploit the linearity of the problem. Hence we can also use nonlinear material models. In Chapter 4 we saw that this is necessary when large

speed $\left[\frac{h_{ref}}{update}\right]$	CPU time [s]	rearrangements	updates [Hz]	CG iters (avg)	CG (max)
		$h_{start} = h_3, h_{ref} = h_6$			
3.00	0.80	157	196.25	11.81	41
1.00	0.90	232	257.78	11.52	62
0.30	0.94	308	327.66	10.87	70
0.10	1.07	504	471.03	8.14	61
0.03	2.24	1478	659.82	5.31	57
0.01	5.77	4187	725.65	3.87	51
		$h_{start} = h_3, h_{ref} = h_7$			
3.00	2.46	243	98.78	12.67	57
1.00	2.80	401	143.21	10.88	82
0.30	3.15	561	178.10	10.03	66
0.10	3.48	1000	287.36	6.63	67
0.03	6.34	2966	467.82	4.18	53
0.01	15.68	8505	542.41	3.15	48

Table 6.5: Timings for an insertion procedure at different speeds, done with two resolutions. Smaller movements decrease the number of CG iterations necessary, and hence increase update frequencies. We can see that decreasing h_{ref} by a factor 2 doubles the number of boundary condition rearrangements required. Timings were done on a Pentium III/1 Ghz, with visualization switched off. The program was compiled with GCC 3.2 with maximum optimization options switched on.

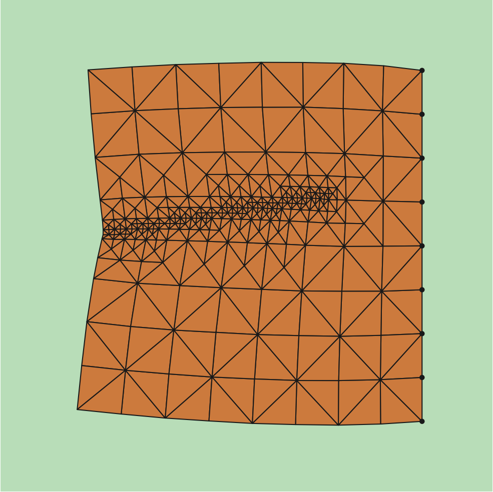


Figure 6.11: Applying forces along an angled trajectory.

	average error [mm]	max error [mm]
with relocation vs. without	0.019	0.23
discretization error with relocation	0.094	0.43
discretization error without relocation	0.095	0.37

Table 6.6: Magnitude of conformance errors for $h_{\text{start}} = h_3, h_{\text{ref}} = h_6$. We see that the difference introduced by node repositioning is smaller than the discretization error, computed relative to h_7 mesh resolution.

deformations are simulated. In Figure 6.12 a scenario is shown that involves large deformations. The needle is inserted sideways into a slab of material fixed at the bottom. In this case, the needle describes a trajectory that is curved, when viewed in the reference configuration. The material used is neo-Hookean material[1] versus linear material, with $E = 34\,\text{kPa}$ and $\nu = 0$. The difference between the final location of needle tip in both experiments is approximately 8 mm.

6.9 Discussion

In this chapter we have presented a novel method for computing simulated needle insertions into 2D elastic material. The method builds on previous work by its use of a quasi-static model of stick/slip friction with plane-stress elasticity. It is different in that it uses an iterative (and optionally nonlinear) relaxation algorithm, and adaptive mesh resolution. The mesh used has a high degree of regularity, and is refined near the inserted needle to improve the local accuracy. Nodes are not moved within the mesh, so the refinement technique, edge bisection, does not cause element shape deterioration. To assess the cost/accuracy ratio of this method, it was implemented in a prototype and subjected to a number of computational experiments. On the basis of these experiments, we conclude that accuracies around 1 mm for insertion in linearly elastic objects of size $10 \times 10\,\text{cm}$ can be achieved at haptic update rates on a 1 Ghz PC.

The performance of the method is related to coarseness of the mesh, and material model used. Both factors are related to the accuracy of the end result, so it is possible to improve response times by sacrificing accuracy. This compromise is controlled with the parameters ε, the tolerance for the relaxation, h_{ref}, the amount refinement around the needle and h_{start}, the global mesh resolution. For the scenarios analyzed, this proved to give adequate control. This manual step could be automated by estimating errors of the FEM discretization automatically, and refining the mesh adaptively during the simulation [32, 100]. Such estimations would also allow a more rigorous error analysis than those provided in Section 6.8.

In the current simulation, iteration counts for the CG algorithm are very low both due to the high spatial coherence between solutions, and due to the limited mesh resolution outside the region of interest. When finer meshes are necessary, additional techniques must be used to decrease iteration counts. Multigrid techniques can solve

---

[1] St. Venant-Kirchoff material was also tried, but caused element inversion, which lead to stalling convergence in the nonlinear CG relaxation

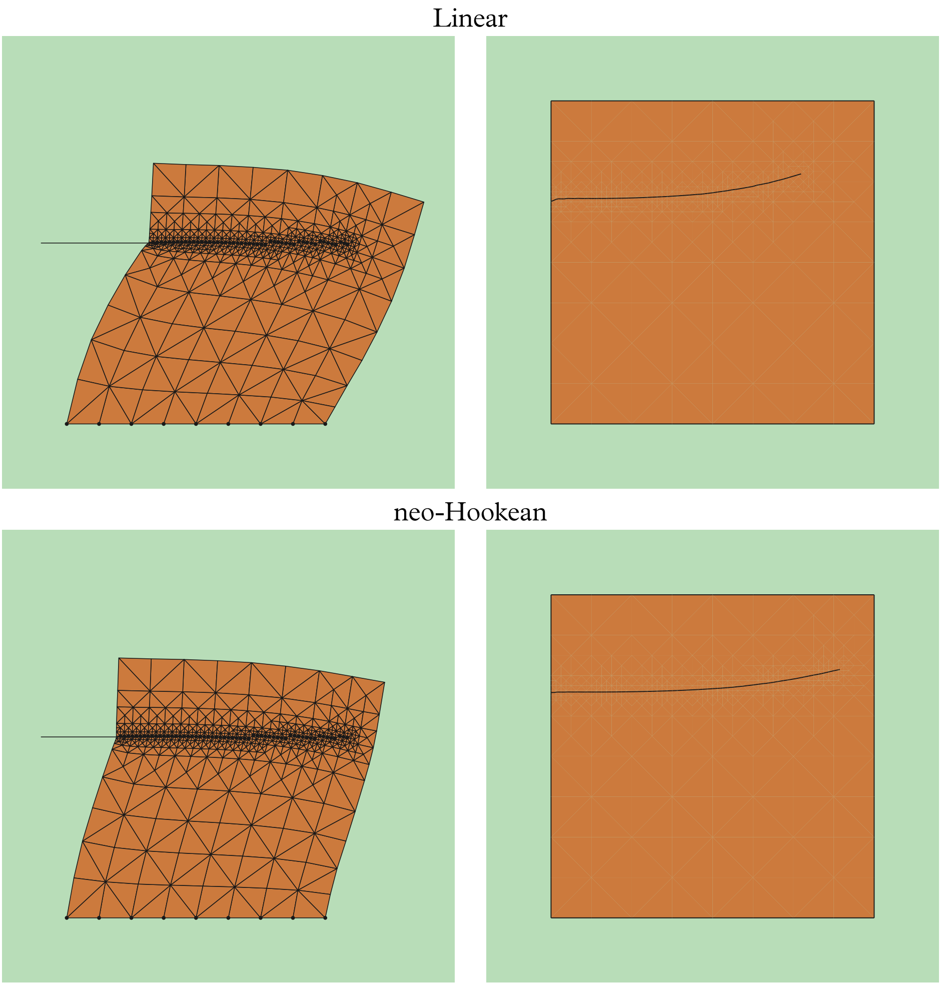


Figure 6.12: Linear elasticity (top) and neo-Hookean elasticity (bottom) compared, both in deformed configuration (left), and undeformed configuration (right). This was done with Poisson ratio $\nu = 0$, and insertion speed $0.01h_{\text{ref}}$/update. The distance between the tip reference locations in both experiments is approximately 8 mm. The linear experiment took 16 seconds of CPU time; the neo-Hookean experiment took 38 seconds, and ended with a 625 triangle mesh.

elliptic FEM problems in $\mathcal{O}(\log n)$ iterations, where n is the number of degrees of freedom. Multigrid methods run *smoothing* relaxations at different resolutions of the same problem. Refinement by edge bisection naturally produces such hierarchical meshes. Smoothing techniques, like the Gauss-Seidel iteration, work by traversing the stiffness matrix row-by-row or column-by-column. This implies that all edges of the mesh should be tracked across mesh changes. Since these smoothing techniques are also used as preconditioners for the CG iteration, a first step in this direction is to implement them as preconditioners.

The extensions mentioned in the previous paragraphs are aimed at improving response times for larger and more accurate simulations. In this light, it is instructive to compare the magnitude of different types of solution errors. From the computational experiments in Section 6.8 we can deduce that these errors have the following causes (in order of decreasing impact).

- The tissue model.
- The error buildup during insertion, an aggregrate of the error sources below.
- The discretization error, caused by using a coarse mesh.
- The conformance error, caused by not moving nodes onto the needle.
- The relaxation error, caused by using an iterative algorithm.

We see that a good tissue model is crucial to making accurate predictions of the effect of tissue deformations. Therefore, research into improvements in computational techniques should be accompanied by a more in-depth analysis of the mechanical properties of the organs being modeled. A complete analysis should also include sensitivity to needle flexion (we assume the needle to be rigid) and variation in material parameters.

The deformable object was a square slab of material. This simplifies implementation, but it is not an essential restriction. The refinement technique is easy to understand for regular simplicial grids of the square, but its only requirement is that vertices are ordered [62]. Moreover, for triangulations in 2D, there is a variety of techniques based on Delaunay refinement [25, 82] that can naturally accomodate more complex object geometries. However, this approach has been chosen with the intent of generalizing to 3D insertions easily. In 3D, both generating the Delaunay tetrahedralization for objects with complex geometrical shapes is much harder [86], and can generate slivers, a type of badly shaped elements. In contrast, edge bisection generalizes naturally to higher dimensions [62].

The 3D generalization of this method is work in progress. Figure 6.13 shows edge bisection in 3D, and Figure 6.14 shows needle insertions in a cube-shaped object. When we look at the computational aspects of the extension to 3D of our method, then we can note that the performance of CG is proportional to the root of the condition number of the stiffness matrix. This quantity is proportional to $1/h$, where h is the element size [6]. Since element size does not change when lifting the simulation to 3D, we can expect that the same number of CG iterations are required. The cost of a single iteration does change. For representing the needle shape accurately in 3D, more refinement is needed, and tetrahedralization require more elements per vertex. The fine-grained simulation in

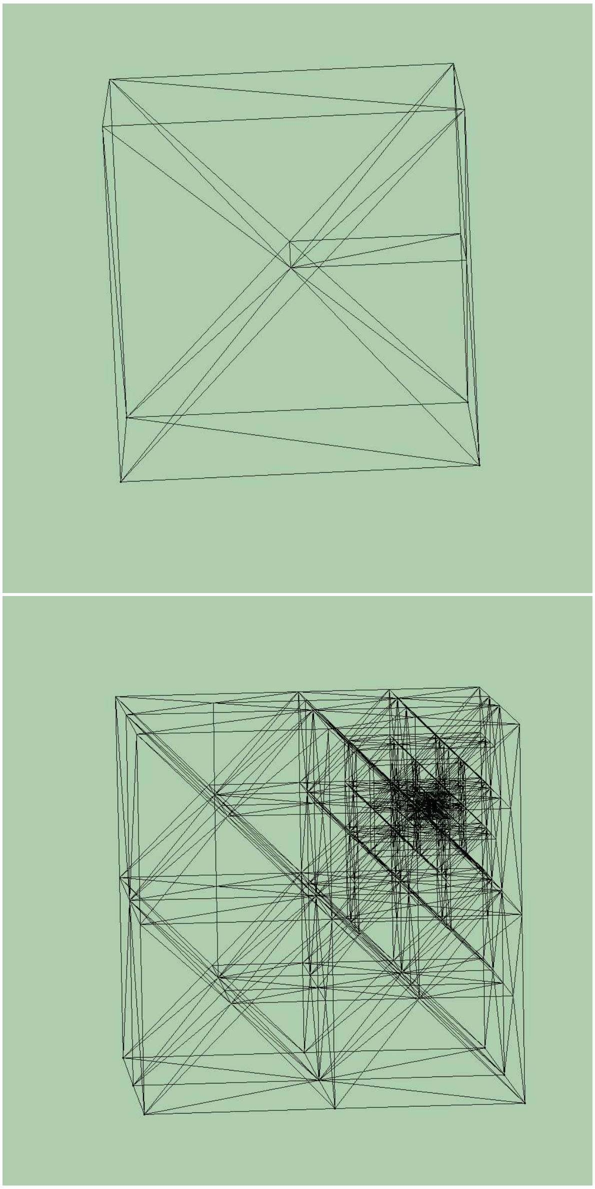
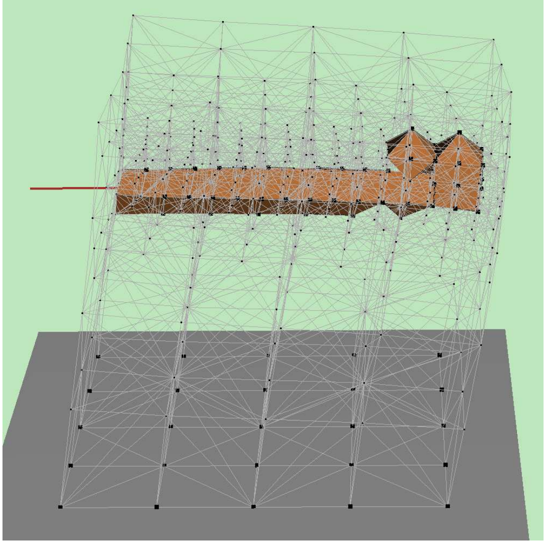
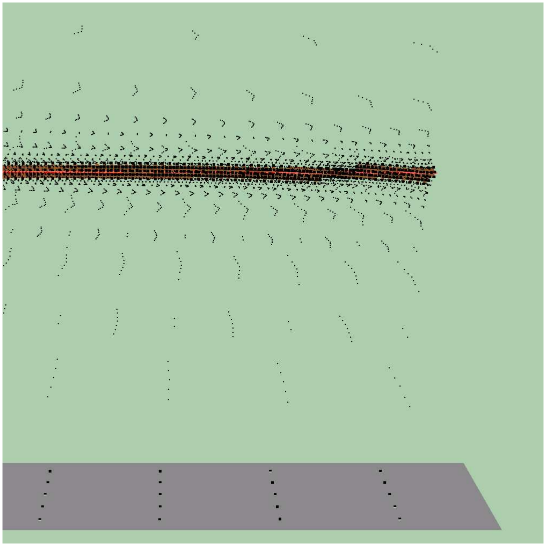


Figure 6.13: Recursive edge bisection in 3D, with refinement around a single point. On the top 3 levels of refinement, on the bottom 20 levels.



linear elasticity, 5 seconds, 2420 elements



neo-Hookean material, 1364 seconds, 29562 elements.

Figure 6.14: Insertions of a needle (radius 1 mm) from the left into a elastic cube (10 cm × 10 cm × 10 cm) fixed on the bottom. The needle is represented in the mesh by a jagged surface, which is shown in solid color. The geometry of the needle surface is accounted for in the friction forces. On the top: linear material, with 12 fold refinement around the needle. On the bottom a detail from a similar insertion into neo-Hookean material, with 20 fold refinement around the needle (Mesh edges are not shown). Timings were done on a P3/1Ghz.

Figure 6.14 corresponds to refinement to h_7. The simulation has approximately 30,000 elements, a factor 50 more than the example in Figure 6.12. Matrix operations are also more expensive in 3D. For example, a matrix/matrix multiply costs 12 flops in 2D and 45 flops in 3D, almost a factor 4. Combining these, we can estimate that each iteration in 3D is roughly 200 times more expensive. To reach the same update rates as the 2D simulation, techniques should be used that increase raw computation power: dedicated hardware and parallel processing.

Chapter 7

Mesh representation

All implementations in the previous chapters also manipulate meshes of triangles and tetrahedra. Since the focus of our work has been on *changing* meshes, a data structure has been developed where change operations are easy to specify and implement. In this chapter we will discuss the data structure, and explain how other parts of the program are grouped around it. It only applies to simplicial meshes of any dimension, for example, triangle and tetrahedron meshes. The data structure makes a distinction between how the connectivity of the mesh—objects connected with pointers—is stored, and its abstract definition—ordered sequences of vertices, so-called *simplexes*. Such simplexes are also used to describe triangulations abstractly in the field of algebraic topology [29].

Subdivisions, be them triangulations, tetrahedralizations or more general complexes of polyhedral cells, are usually represented by objects connected with pointers. Many such data structures exist for storing subdivisions of the plane, for example the doubly connected edge list [31], and the quad edge structure [48]. These structures all store the connectivity in slightly different ways at slightly different memory costs. Memory usage is an important issue when manipulating large meshes, so Campagna et al. [23] propose a triangle mesh representation where the choice between computation costs and memory costs can be made at compile time.

For 3D subdivisions, Dobkin and Laszlo [39] describe a data structure that can represent general complexes of cells. The cells can have any shape and may be infinite; the only restriction is that they must meet properly. The central notion of their data structure is the facet-edge: it represents the combination of a facet (a 2-dimensional cell) and an edge (a 1-dimensional cell). A cell complex is stored as a set of objects, each representing a single facet-edge. Every object contains references to the four adjacent facet-edge objects: the next and previous edge of the same face, and the next and previous facet that is incident with the same edge. Since neighboring facet-edges are stored explicitly, it is very easy and efficient to traverse all the facets incident to an edge, and all edges contained in a facet. Mücke [68] uses a simplified version of the facet-edge structure for maintaining the connectivity of tetrahedral meshes.

Brisson [16] proposes a generalization of the concept of facet-edge to d dimensions: mesh features are represented by so-called *cell tuples*. A cell tuple is a tuple $(c_0, \ldots, c_d)$, where each c_j represents a j-dimensional mesh feature, and $c_i \subset c_{i+1}$. In 3D, a cell-tuple represents a vertex as part of a specific edge and a specific face. For $k = 0, \dots, d$ the switch operator is defined: $\mathsf{switch}_k(t)$ is the unique cell tuple that agrees with t except in its kth component. For example, in 3D, the switch_3 operator moves from a vertex of a volumetric cell to the same vertex as part of the same edge and face, but from a neighboring volumetric cell. Data structures such as the facet-edge structure discussed above, the edge algebra discussed by Guibas and Stolfi [48] and various other mesh data structures [61, 99] can be expressed in terms of cell tuples.

In summary, meshes are typically represented by objects connected by pointers. Relations between objects, such as incidence, inclusion and neighborhood, are maintained by storing pointers between these objects. This structure allows for efficient traversal of the mesh: jumping between mesh features is a matter of following pointers. In this sense, our data structure resembles much of the previous work. However, we have chosen to make explicit what the mesh connectivity objects represent. This makes it possible to specify correctness of the data structure, prove algorithms dealing with meshes correct, and formally specify what change operations should do. Moreover, implementing such operations is easy. Two operations are provided to change the mesh connectivity, change-elements and replace-elements. These are generic operations, and can be used to implement high level mesh operations. All code for maintaining mesh connectivity is concentrated in these two routines, enhancing the modularity of the total system.

In this chapter we first discuss the underlying abstract mesh representation. This representation uses abstract oriented simplexes, a basic concept in algebraic topology [29]. Then we discuss how connectivity is stored in the program, and how it can be modified, in other words, how change-elements and replace-elements are implemented. Finally, we show how the rest of the system interfaces with mesh changes.

7.1 Abstract oriented simplexes

Domains with general shapes in the Finite Element Method are usually represented using *unstructured* meshes. These meshes consist of triangles (in 2D) or tetrahedra (in 3D). In *conforming* Finite Element Methods, shape functions should be admissible as solutions to the original, continuous problem. In the case of elastic problems, this implies that the functions should be piecewise continuously differentiable. This continuity condition (also known as *compatibility condition*), implies that the common interface of two elements should also be a mesh feature. In other words, mesh elements should be *properly joined.*

In a triangulated or tetrahedral mesh, mesh features are formed by convex hulls of vertices, so-called *geometric simplexes*. For example, let $\mathbf{a}_1, \dots, \mathbf{a}_4$ be an affinely independent set of points in $\mathbb{R}^d$, then $\operatorname{conv}\{\mathbf{a}_1, \mathbf{a}_2\}$, the convex hull of $\mathbf{a}_1$ and $\mathbf{a}_2$, is an edge, $\operatorname{conv}\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ is a triangle, and $\operatorname{conv}\{\mathbf{a}_1, \dots, \mathbf{a}_4\}$ is a tetrahedron. Proper joining of simplexes can be expressed as follows. Let $\mathbf{a}_1, \dots, \mathbf{a}_k \in \mathbb{R}^d$ and $\mathbf{b}_1, \dots, \mathbf{b}_l \in \mathbb{R}^d$, then $\operatorname{conv}\{\mathbf{a}_1, \dots, \mathbf{a}_k\}$ and $\operatorname{conv}\{\mathbf{b}_1, \dots, \mathbf{b}_l\}$ are properly joined if

$$\operatorname{conv}\{\mathbf{a}_1, \dots, \mathbf{a}_k\} \cap \operatorname{conv}\{\mathbf{b}_1, \dots, \mathbf{b}_l\} = \operatorname{conv} S,$$

where $S \subset \{\mathbf{a}_1, \ldots, \mathbf{a}_k, \mathbf{b}_1, \ldots, \mathbf{b}_l\}$. Properly joined simplexes are demonstrated in Figure 7.1.

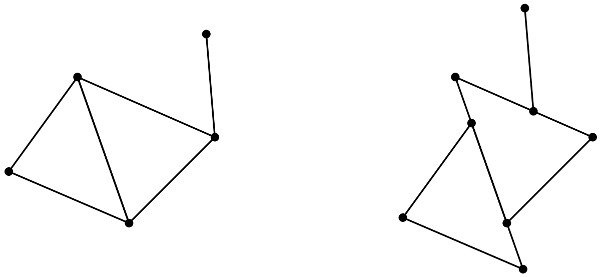


Figure 7.1: Properly joined simplexes (left), and improperly joined ones (right)

We can see that properly joined geometric simplexes are characterized by their sets of vertices. Hence, for reasoning with simplexes, it suffices to consider the discrete set of their vertices, and disregard the continous nature of convex subsets of $\mathbb{R}^d$. If we only consider sets of vertices, then the type of the vertices themselves is not relevant. Therefore, we will assume for the remainder of the chapter that vertices come from some set $\mathcal{V}$, which is left unspecified.

Simplexes can have orientations. For example, an edge can have two directions, and a triangle can have a normal pointing in two directions. This orientation is related to ordering of the vertices: if two vertices in a triangle are swapped, the direction of the normal is flipped. The orientation of a simplex can also be defined in terms of swaps. Let $a_0, \ldots, a_k \in \mathcal{V}$ be a sequence of $k+1$ vertices, for $k \geq 1$. A permutation π of these vertices may be decomposed into a number of swaps. If this number is even, then π is an *even* permutation, otherwise it is an *odd* permutation. The positive simplex $\langle a_0, \ldots, a_k\rangle$ is formed by the equivalence class of all even permutations of $a_0, \ldots, a_k$, i.e.,

$$\langle a_0, \ldots, a_k\rangle = \{\pi(a_0, \ldots, a_k) : \pi \text{ is an even permutation}\}.$$

Analogously, the equivalence class of uneven permutations forms the other orientation

$$-\langle a_0, \ldots, a_k\rangle = \{\pi(a_0, \ldots, a_k) : \pi \text{ is an odd permutation}\}.$$

The number k is also called the *dimension* of the simplex. 1-simplexes correspond to edges, 2-simplexes to triangles and 3-simplexes to tetrahedra. The above definition requires $k > 0$. For simplexes of one vertex, we simply assume that they exist in two orientations.

Containment of oriented abstract simplexes is defined with help of the subsimplex operation. This operation is defined as follows.

$$\mathsf{subsimplex}_{a_j}\langle a_0, \ldots, a_k\rangle = (-1)^j\langle a_0, \ldots, a_k \backslash a_j\rangle, \quad a_0, \ldots, a_k \in \mathcal{V}, k \geq 1.$$

The notation $a_0, \ldots, a_k \backslash v$ means the sequence $a_0, \ldots, a_k$ with v removed. This definition is independent of the representative chosen. This operation implies an inclusion relation. We have $\sigma \subset \tau$, if $\sigma = \tau$ or when there is some $v \in \tau$ such that $\sigma \subset \mathsf{subsimplex}_v \tau$.

We call the set of abstract simplexes K an oriented d-dimensional pseudo-manifold, or a *simplicial mesh* if the following conditions hold:

1. if τ in K and $\sigma \subset \tau$ then σ in K
2. Every σ in K is a subsimplex of some $\tau \in K$, where τ has dimension d.
3. if $\sigma \in K$ is a $(d-1)$-simplex, then it is subsimplex of only one d-simplex.

Two d-simplexes τ_1 and τ_2 are neighbors if there is a $(d-1)$-simplex $\rho \subset \tau_1$ such that $-\rho \subset \tau_2$. The boundary of a simplicial mesh K, denoted by ∂K is formed by the set of $d-1$ simplexes whose opposite orientation is not part of K. Since all simplexes in a pseudo-manifold are part of some d-simplex, we can characterize the structure by its set of d-simplexes.

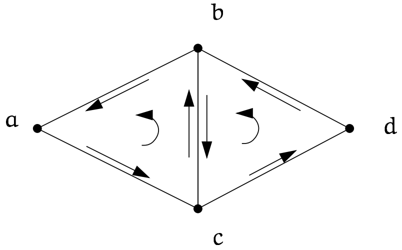


Figure 7.2: A simple 2-dimensional pseudo-manifold. We have $T = \{abc, bdc\}$ (leaving out the angled brackets in the notation of simplexes), and $K = T \cup \{ab, bc, ca, cb, bd, dc, a, b, c, d, -a, -b, -c, -d\}$. The boundary of K is formed by $\{ab, ca, bd, dc\}$; the other edges (bc and $-bc$) form a pair that connect abc and bcd. The orientation of triangles and edges are indicated with arrows.

7.2 Representing the mesh

The mesh representation discussed in the previous section can be directly implemented. In the next two sections, we show how this is done, both using class declarations (in C++ syntax) and pseudo-code. In this pseudo-code, we will refer to simplexes with the greek letters σ and τ. The variable t always refers to an Element object representing a d-simplex, and the variable f always refers to a Face object representing a $(d-1)$-simplex. In general variables are denoted by words printed in italic. In the pseudo-code we will equate maps, search structures that store a value v for some keys k, with a set of key/value tuples. This is done for the sake of notational convenience. In practice, such search structures will typically be implemented by balanced trees. We assume that sets of key/value tuples can be indexed, and that it supports the method keys that returns all keys in the map, and the method erase, that removes a single (key,value) tuple from the set. Examples of the use of these maps are given here.

```
$m \leftarrow \{(k_1, v_1), (k_2, v_2)\}$
m.keys()              (* returns $\{k_1, k_2\}$ *)
$m[k_1]$              (* returns $v_1$ *)
$m[k_1] \leftarrow w_1$          (* changes the value corresponding to $k_1$ *)
$m[k_3] \leftarrow v_3$          (* adds the tuple $(k_3, v_3)$ *)
m.erase($k_2$)        (* removes $(k_2, v_2)$ *)
```

If $\mathcal{V}$ is totally ordered, then we can define a canonical representation for each simplex. Let $a_0, \dots, a_k \in \mathcal{V}$. We can sort the vertices in a oriented simplex, while counting the number of swaps s, and take $(-1)^s \operatorname{sort}(a_0, \dots, a_k)$ as the representative of $\langle a_0, \dots, a_k \rangle$. In effect, this the canonical representation translates a k-simplex in a $k + 2$ tuple, consisting of the $k + 1$ vertices and the value of s. Since the elements of each tuple can be ordered, the tuples themselves can also be ordered, e.g. by the lexicographic order. This implies that the canonical representation can be used as a key in a lookup structure. In this way, we can create tables of objects with simplexes as keys.

The canonical representation of a simplex can be used to implement it. Assuming that there is some type Vertex representing vertices, the data of the Simplex type may expressed (in C++ syntax) as follows.

```
class Simplex {
  Vertex vertices[MAXDIMENSION+1];
  int dimension;
};
```

Let us assume for the remainder that a Simplex object can be created from a sign $q \in \{-1, 1\}$ and a sequence of vertices $b_0, \dots, b_k$, yielding the canonical representation $p\langle a_0, \dots, a_k \rangle$ with $a_0 < \dots < a_k$ and $p \in \{-1, 1\}$. Let σ and τ be Simplex objects, j an integer from $0, \dots, k$, and v and w Vertex objects. Then the following operations can be defined and implemented for the Simplex type.

- σ.count() returns $k + 1$, the number of vertices in σ.
- σ.dimension() returns k, the dimension of the simplex.
- σ.index (v) returns j such that $a_j = v$.
- σ.sign () returns p.
- σ.vertex(j) returns a_j.
- σ.subn (j) returns $(-1)^j q\langle a_0, \dots, a_k \backslash a_j \rangle$, the jth subset of σ.
- σ.subv (v) returns $\text{subsimplex}_v \sigma$.
- σ.mate() returns $-\sigma$.
- σ.substituted (v, w) returns σ with v replaced by w in the vertices of the simplex.
- compare (σ, τ) is a signed comparison of σ and τ. It can be implemented by lexicographic ordering.

- σ.sup (v) returns $p\langle v, a_0, , \ldots, a_k\rangle$, the unique $(k+1)$-simplex containing both the vertex v and the simplex σ.

Since a d-dimensional pseudo-manifold is characterized by a set of d-simplexes, a simplicial mesh can be succinctly specified as a set of Simplex objects of dimension d. However, traversing the elements of a mesh cannot be done efficiently with this representation. Therefore, d-simplexes and $(d-1)$-simplexes are also represented as objects, i.e. chunks of memory with a unique identity that can be referenced to by means of pointers. The base class for both objects is Mesh-feature. It contains the simplex that it is supposed to represent. One derived class represents d-simplexes, and is called Element, by analogy with naming of Finite Elements. Objects of the class Face represent $(d-1)$-simplexes.[1] The definition of Mesh-feature in C++ notation is as follows.

```
class Mesh_feature {
  Simplex simplex;
};
```

Each d-simplex contains $d+1$ faces, so the Element object has $d+1$ pointers to Face objects.

```
class Element : public Mesh_feature {
  Face * faces[MAXDIMENSION+1];
};
```

A face of an element is obtained by removing one vertex from its simplex. Faces and vertices in an element are related. This relation is used in opposite-vertex and opposite-face methods of Element objects. The method opposite-vertex for an Element object e takes a face f from e.faces, and returns a vertex from e.simplex such that

$$f.\mathsf{simplex} = e.\mathsf{simplex.subv}(e.\mathsf{opposite\text{-}vertex}(f))$$

Similarly, the function opposite-face, takes a vertex v from e.simplex, and returns a Face object from e.faces such that

$$e.\mathsf{opposite\text{-}face}(v).\mathsf{simplex} = e.\mathsf{simplex.subv}(v)$$

Both functions are also illustrated in Figure 7.3.

The faces variable must contain Face objects. The following invariant specifies in what order they are stored.

$$t.\mathsf{faces}[i].\mathsf{simplex} = t.\mathsf{simplex.subn}(i), \qquad i = 0, \ldots, d. \tag{7.1}$$

In a pseudo-manifold, each face is in exactly one d-simplex. Hence we may store pointers from Face objects to Element objects. The Face object also stores a pointer to its mate, the Face object with the opposite orientation

---

[1] This is in contrast with traditional terminology for simplicial complexes, where simplexes of all dimensions are called "faces."

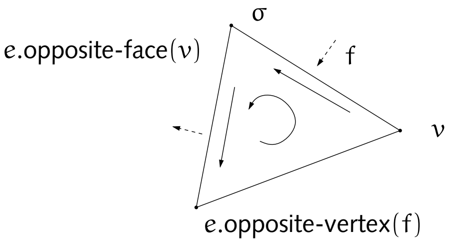


Figure 7.3: The body of the cycle-around algorithm illustrated: an element e is entered through f (dotted arrow), and left through e.opposite-face (v) (other dotted arrow). In this case, σ is a 0-simplex, i.e. a vertex. The orientation of the triangle and the edges are indicated with arrows.

```
class Face : public Mesh_feature {
  Element *element;
  Face *mate;
};
```

The element pointer in a Face object f satisfies the following invariant

$$f \in \mathsf{f.element.faces}, \tag{7.2}$$

where we treat the array faces as a set. The mate field of a Face object f satisfies

$$\mathsf{f.mate} = \mathsf{null} \vee \mathsf{f.mate.simplex} = -\mathsf{f.simplex}. \tag{7.3}$$

The connectivity of a simplicial mesh then is a collection of Element and Face objects such that invariants (7.1) to (7.3) are satisfied, and each d and $(d-1)$-simplex is represented by exactly one Element and Face object respectively, i.e., for all Mesh-feature objects t, u in the mesh we have

$$\mathsf{t.simplex} = \mathsf{u.simplex} \implies t = u. \tag{7.4}$$

This connectivity information is sufficient to traverse the mesh. We give the example of traversing Element objects incident with one particular $(d-2)$-simplex. In 2D, this routine traverses all triangles incident with a vertex, and in 3D all tetrahedra incident with an edge. It takes a Face object entry as argument, and a number $j \in \{0, \ldots, d-1\}$. It returns a set of d-simplexes that contain entry.simplex.subn(j). Termination and correctness of the algorithm can be proved using the integrity of the data structure, and properties of the simplicial mesh.

```
procedure cycle-around (entry: Face, j : $\{0, \ldots, d-1\}$)
  star ← $\emptyset$
  f ← entry
  v ← entry.simplex.vertex(j)
  while f $\neq$ null
```

```
    e ← f.element
    exit ← e.opposite-face(v)
    v ← e.opposite-vertex(f)
    star.add (e)
    f ← exit.mate
    if f = entry:
      f ← null
  return star
```

This code is also illustrated in Figure 7.3.

This routine builds a set of Element objects that contain $\sigma = \text{entry.simplex.subn}(j)$. This can be seen by considering the following loop invariant.

$$f = \text{null} \vee \text{subsimplex}_v(\text{f.simplex}) = \sigma.$$

If $f \neq \text{null}$, then $\text{f.simplex} = \sigma.\text{sup}(v)$, and $\text{e.simplex} = \sigma.\text{sup}(v).\text{sup}(w)$ for some vertex w. Hence $\text{exit.simplex} = -\sigma.\text{sup}(w)$. In the next step, either the loop exits because $\text{exit.mate} = \text{null}$, or f and v are changed such that the invariant holds again.

In addition, we see that the loop adds a sequence of Element objects with d-simplexes $(\tau_1, \tau_2, \ldots)$ to star. These d-simplexes are of the form

$$\sigma.\text{sup}(p_1).\text{sup}(p_2), \sigma.\text{sup}(p_2).\text{sup}(p_3), \sigma.\text{sup}(p_3).\text{sup}(p_4), \ldots$$

for a sequence of vertices $(p_1, p_2, \ldots)$. All simplexes of the sequence are unique. To see this, suppose that $\tau_j = \tau_i$ for some $j \leq i$. In other words

$$\sigma.\text{sup}(p_j).\text{sup}(p_{j+1}) = \sigma.\text{sup}(p_i).\text{sup}(p_{i+1}).$$

This implies $p_i = p_j$ and $p_{j+1} = p_{i+1}$. Their predecessors in the sequence are $\tau_{j-1} = \sigma.\text{sup}(p_{j-1}).\text{sup}(p_j)$ and $\tau_{i-1} = \sigma.\text{sup}(p_{i-1}).\text{sup}(p_j)$ respectively. Since τ_{j-1} and τ_{i-1} both contain the face $-\sigma.\text{sup}(p_j)$ they must be equal, implying that $p_{j-1} = p_{i-1}$. This argument can be continued inductively, until we have $\tau_1 = \tau_{i-j+1} = \sigma.\text{sup}(p_1).\text{sup}(p_2)$. The integrity of the data structure implies that entry is the only Face object whose simplex is $\sigma.\text{sup}(p_1)$. Therefore, if $i < j$ the if statement would have aborted the loop before τ_i is added in the i-th step. Therefore $i = j$. Since the mesh only contains finitely many d-simplexes, the loop must terminate.

7.3 Changing the mesh

In this section we show how mesh connectivity objects can changed in a generic fashion. This is done by two routines, replace-elements and change-elements. First we show how replace-elements can be implemented. In situations where the number of elements does not change, a different routine with additional desirable properties can be used. This is the change-elements routine. Finally, we show how cuts can be expressed with change-elements.

Every change in the mesh can be encoded as removing existing elements, exposing more of the boundary of the mesh, and attaching new elements to the boundary. The actual connectivity information is stored in Face objects, since their mate fields link neighboring elements. To update these fields properly, it is necessary to store the boundary of the pseudo-manifold. This is done with the following data structure for the mesh connectivity.

```
class Mesh_connectivity {
  set<Element*> elements;
  map<Simplex, Face*> boundary;
};
```

This definition uses the generic types `set` and `map`. The variable elements is a set of Element objects. The variable boundary maps simplexes to their Face objects for all boundary faces.

Let the $(d-1)$-simplexes of a set of d-simplexes T be given by

$$\mathsf{faces}(T) = \{\sigma : \sigma = \mathsf{subsimplex}_\nu(\tau), \nu \in \tau, \tau \in T\}.$$

Then, the boundary map of a simplicial mesh formed by T may be characterized as

$$\mathsf{boundary}(T) = \{(\sigma, f) : f.\mathsf{simplex} = \sigma, \wedge - \sigma \not\in \mathsf{faces}(T) \wedge \sigma \in \mathsf{faces}(T)\} \qquad (7.5)$$

The primary mesh change operation is replacing elements. The simplest way to implement it is by removing elements one-by-one, and adding new elements one-by-one. This is achieved by the following procedure.

```
procedure replace-elements (mesh: Mesh-topology,
    old-objects: set of Element, new-simplexes: set of Simplex):
  for e in old-objects:
    remove-element (mesh, e)
  for $\tau$ in new-simplexes:
    add-element (mesh,$\tau$)
```

When a single element is added, the connectivity can be maintained by removing boundary faces that attach to the new element, and adding other new faces of the element to the boundary.

```
procedure add-element (mesh: Mesh-topology, $\tau$: Simplex)
  e $\leftarrow$ new Element($\tau$)
  mesh.elements $\leftarrow$mesh.elements $\cup \{e\}$
  for j in $0, \ldots, d$:
    $\sigma \leftarrow \tau$.subn(j)
    f $\leftarrow$ new Face($\sigma$)
    f.element $\leftarrow$e
```

```
        e.faces[j] ← f
        if $-\sigma \in$ mesh.boundary.keys():
          f.mate ← mesh.boundary[$-\sigma$]
          f.mate.mate ← f
          mesh.boundary.erase ($\sigma$)
        else :
          mesh.boundary[f.simplex] ← f
          f.mate ← null
```

Similarly, the boundary can be updated during element removal.

```
procedure remove-element (mesh, e)
  for f in e.faces:
    if f.mate:
      mesh.boundary[−f.simplex] ← f.mate
      f.mate.mate ← null
      f.mate ← null
    else :
      mesh.boundary.erase(f.simplex)
  mesh.elements ← mesh.elements$\backslash\{e\}$
```

This code maintains mesh connectivity, but is not very efficient and replaces all Face objects, even the ones that were not changed. The following improvements solve these problems. First, in some cases, a mesh change modifies elements, but may leave certain faces in place. In the code shown below, these faces are maintained, so that pointers to these faces remain valid after the change. It achieves this by remembering old faces, and reusing those that also occur in the new configuration.

```
procedure replace-elements (mesh: Mesh-topology, old-objects: set of Element,
    new-simplexes: set of Simplex):
  oldfacemap ← { (f.simplex, f) : f $\in$ e.faces, e $\in$ old-objects }
  newfacemap ← {($\sigma$, f) : $\sigma = \tau$.subn(j), j = 1, . . . , d, $\tau \in$ new-simplexes,
    f = (if $\sigma \in$ oldfacemap.keys()) : oldfacemap[$\sigma$] else: null)}
  discard ← oldfacemap$\backslash$newfacemap
  (*)
  for ($\sigma$, f) in discard:
    (**)
    if f.mate:
      f.mate.mate ← null
      f.mate ← null
      mesh.boundary[f.simplex] ← f.mate
    else :
      boundary.erase (f)
  mesh.elements ← mesh.elements$\backslash$old-objects
```

```
for $\tau$ in new-simplexes:
  e←new Element ($\tau$)
  mesh.elements ← mesh.elements $\cup \{e\}$
  for j in $0, \dots, d$:
    $(\sigma, f)$ ← newfacemap[$\tau$.subn(j)]
    if f = null:
      f ← new Face($\sigma$)
      if $-\sigma \in$ mesh.boundary.keys():
        f.mate ← mesh.boundary[$-\sigma$]
        f.mate.mate ← f
        mesh.boundary.erase (-$\sigma$)
      else :
        mesh.boundary[$\sigma$] ← f
    f.element ← e
    e.faces[j] ← f
```

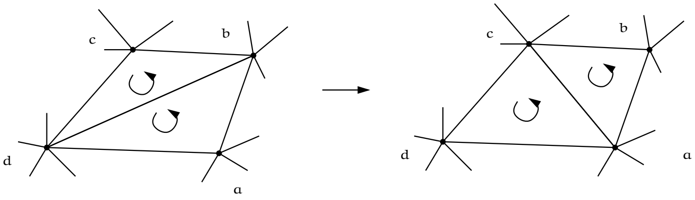


Figure 7.4: An edge flip changes the edge db to ad. This can be encoded as replacing the 2-simplexes $\{bcd, abd\}$ by $\{abc, acd\}$. The elements on the left contain edges ab, bc, cd, and da which are also present after the flip.

This code is still not optimal. For example, in the edge flip from Figure 7.4, the boundary does not change, while orientations of bd and ad are temporarily added to and removed from the boundary. When the boundary is large, these temporary changes may be expensive. They can be prevented by adding matched pairs to newfacemap before processing it. If the following code is added in the place marked with $(*)$ in the previous algorithm, then these unnecessary updates are prevented.

```
for $\sigma$ in newfacemap.keys ():
  if $-\sigma \in$ newfacemap.keys () $\wedge$ $\sigma$.sign() = 1 $\wedge$
      newfacemap[$\sigma$] = null $\wedge$ newfacemap[$-\sigma$] = null:
    $f_1$ ← new Face($\sigma$)
    $f_2$ ← new Face($-\sigma$)
    $f_2$.mate ← $f_1$
    $f_1$.mate ← $f_2$
    newfacemap[$\sigma$] ← $f_1$
    newfacemap[$-\sigma$] ← $f_2$
```

Similarly, the updates of the boundary, following $(**)$ in the pseudo-code, only have to be performed when $-\sigma \notin$ discard.keys().

Some types of mesh modifications do not change the number of mesh elements, only their connectivity. For example, in Figure 7.5, two faces are dissected and two other are glued together at the same time, leaving the number of faces and elements invariant. It is possible to implement this operation with replace-elements. However, there is a one-to-one correspondence for every face and element before and after the change, and this correspondence is lost when replace-elements is used. Therefore, we propose a second operation, change-elements that maintains this correspondence. Its argument is a substitution, that is applied to a number of elements. Abstractly speaking, a substitution s is a set of (τ, π)-tuples, where τ is a d-simplex, and $\pi : \mathcal{V} \to \mathcal{V}$ is a substitution on the vertices. If T is set of d-simplexes in the original mesh, then applying the substitution s entails forming the mesh

$$K' = \{\sigma : \sigma \subset \tau, \tau \in T'\} T' = \{\tau \in T : \tau \notin \text{s.keys}()\} \cup \{\pi(\tau) : (\tau, \pi) \in s\}.$$

When implementing this operation, the substitution takes the form of a set of tuples (t, π), where t is an Element object, and π a vertex substitution.

```
procedure change-elements (mesh: Mesh-connectivity,
    substitution: element/node-substitution map):
  oldfaces ←{ t.faces[j] : $(t, \pi) \in$ substitution, $j = 0, \ldots, d$ }
  for f in oldfaces:
    if f.mate:
      mesh.boundary[f.simplex] ← f
      f.mate ← 0
      f.mate.mate ← 0
    else :
      mesh.boundary.erase(f.simplex)
  for $(e, \pi)$ in substitution:
    $\tau$ ← e.simplex
    $\tau'$ ← $\pi(\tau)$
    newfaces ←{ $(\pi\sigma, f) : f =$ e.face(j), $\sigma = \tau$.subn(j), $j = 0, \ldots, d$ }
    e.simplex ←$\tau'$
    for j in $0, \ldots, d$:
      $\sigma'$ ← $\tau'$.subn(j)
      f ← newfaces[$\sigma'$]
      e.faces[j] ←f
      f.simplex ← $\sigma'$
      if $-\sigma \in$ mesh.boundary.keys():
        f.mate ← mesh.boundary[$-\sigma'$]
        f.mate.mate ← f
        mesh.boundary.erase[$-\sigma'$]
      else :
        mesh.boundary[$\sigma'$] ← f
```

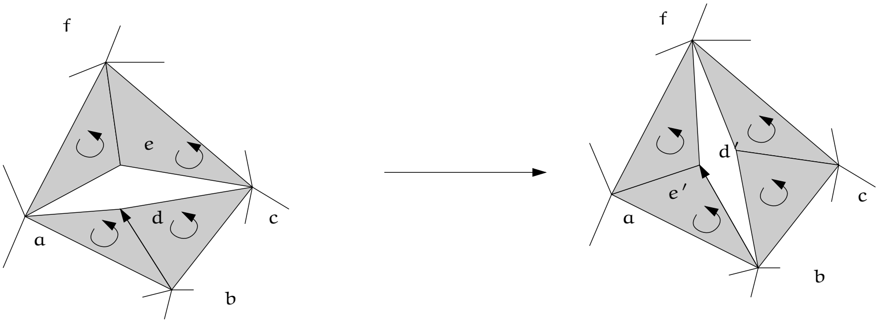


Figure 7.5: Cutting and stitching can be achieved using replace-triangles. The operation shown here can be effected as replacing $\{abd, bcd, cfe, aef\}$ with $\{abe', ae'f, d'cf, bcd'\}$. The operation can also be written as a vertex substitution, e.g. substitute $d := d'$ in abd. By specifying the operation like this, objects can be made persistent. Then bd (left) and be' (right) are represented by the same object, marked in bold.

The primary example of the change-elements operation is the dissect operation, also discussed in Chapter 3, which produces cuts along faces in a simplicial mesh. We show how a vertex substitution for a cut along a surface C can be defined. Let us assume that a simplicial mesh is given as T, a set of oriented d-simplexes, satisfying the conditions for a simplicial mesh, and K is the complex induced by T, i.e.

$$K = \{\sigma : \sigma \subset \tau, \tau \in T\}.$$

We assume that the cut is specified by a set of faces $C \subset \mathsf{faces}(T)$, such that

$$\sigma \in C \implies -\sigma \in C.$$

In other words, C is a set of face pairs from K. The star of a vertex v is the set of all elements incident with v, in other words,

$$\mathsf{star}(v) = \{\tau \in T : v \in \tau\}.$$

Let v be a vertex, and let τ and τ' be elements from $\mathsf{star}(v)$. We say that τ and τ' are (v, C)-connected if there are d-simplexes $\tau = \tau_1, \dots, \tau_k = \tau'$ in T such that all τ_i contain v, and all τ_i and τ_{i+1} are neighbors for $i = 1, \dots, k-1$ and

$$\mathsf{faces}(\tau_i) \cap (-\mathsf{faces}(\tau_{i+1})) \notin C, \qquad i = 1, \dots, k-1.$$

In other words, $\tau_1, \dots, \tau_k$ is a chain of elements containing v that does not cross C.

The notion of (v, C)-connectedness is an equivalence relation on $\mathsf{star}(v)$, so for each vertex v of T, we may partition $\mathsf{star}(v)$ into equivalence classes. Assume that these classes are given by $S_{v,1}, \dots, S_{v,l}$ for some $l \geq 1$. Let us assume that a unique vertex $w_{v,i}$ for each equivalence class $S_{v,i}$ is given. In practice this may be a ‘copy’ of v with a different number, or perhaps a vertex that is slightly displaced with respect to v. Let τ

be a d-simplex, then we define for $v \in \tau$ the node substitution

$$\varphi_\tau(v) = \begin{cases} v & \text{if } \mathrm{star}(v) \text{ is the single } (v, C)\text{-equivalence class,} \\ w_{v,i} & \text{if } \tau \in S_{v,i}, \text{ for some } 1 \le i \le k \text{ and } l > 1. \end{cases}$$

By construction, the mapping $v \mapsto \varphi_{\tau,S}(v)$ is injective. We can define a complex K' induced by a set of d-simplexes T' as follows.

$$\begin{aligned} K' &= \{ \sigma : \sigma \subset \tau, \tau \in T' \}, \\ T' &= \{ \langle \varphi_\tau(a_0), \ldots, \varphi_\tau(a_k) \rangle : \tau = \langle a_0, \ldots, a_k \rangle \in T \}. \end{aligned}$$

Since $v \mapsto \varphi_\tau(v)$ is injective, all elements of T' are d-simplexes, and all $(d-1)$-simplexes are unique within faces(T'). Hence K' is a d-dimensional pseudo-manifold.

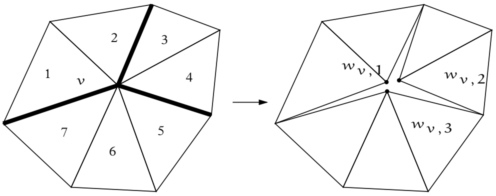


Figure 7.6: A dissection can be expressed as a node substitution. In the above example, the cut surface C (bold) partitions the 2-simplexes incident with v in 3 sets. Elements 5, 6 and 7 are (v, C) connected, but 1 and 7 are not. In the result v is substituted by three different nodes $w_{v,1}$, $w_{v,2}$ and $w_{v,3}$. Elements 5, 6 and 7 share the node $w_{v,3}$.

The dissect operation leaves faces that are not in C joined together. If $\sigma \in K$ and $-\sigma \in K$ but $\sigma \notin C$, then the d-simplexes τ_1 and τ_2 containing σ and $-\sigma$ respectively, obviously are (v, C)-connected for all vertices v of σ, hence both σ and $-\sigma$ are mapped to corresponding faces σ' and $-\sigma'$. However, not all faces in C also have to end up on the boundary of K'. An example is given in Figure 7.7.

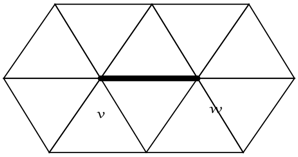


Figure 7.7: Face vw is not dissected by the cut surface marked in bold, since the surface does not split the elements around v and w into different components.

Finally, the dissect operation implies a connectedness condition. It is reasonable to assume that for $C = \emptyset$, the dissect operation does nothing. This implies that for every vertex v, star(v) forms a $(v, \emptyset)$-connected component. In other words all elements containing v should be connected to each other via $(d-1)$-faces. This is a desirable property, for star(v) can then be found by traversing the mesh starting from an arbitrary $\tau \in \mathrm{star}(v)$.

7.4 Interfacing with mesh connectivity

Both change-elements and replace-elements are characterized by changes to the set of elements and the boundary. Such changes are signaled to other parts of the simulation by the following mechanism. A Mesh-connectivity object maintains a list of Mesh-connectivity-watchers. These are objects that to take some special action upon changes to the connectivity. They can be characterized by a C++ class declaration as follows.

```
class Mesh_connectivity_watcher {
  virtual void process_changed_element (Element*);
  virtual void process_changed_boundary (Face*);
  virtual void init_elements (set<Element*> const*);
  virtual void init_boundary (map<Simplex,Face*> const*);
};
```

When a Mesh-connectivity-watcher is added to a Mesh-connectivity object, it is initialized with the current list of all Element and Face objects. After this initialization, the virtual functions process-changed-boundary and process-changed-element are called for every change to the mesh boundary and every element removed or added.

An example of a Mesh-connectivity-watcher is the following routine

```
procedure process-changed-boundary (f):
  $n \leftarrow$ normal of f
  if (f.mate = null) $\wedge$ ($n \cdot e_3 < 0$):
    for $v$ in f.simplex.vertices:
      deformation-constraints.fix-node ($v$)
```

This code assures that the deformable object is always fixed on one side. All boundary faces pointing in e_3 direction have their faces fixed.

In our implementation, the set of vertices simply is given by the positive integers, with natural ordering. For a linear FEM discretization, simplex vertices and nodes (interpolation conditions) coincide. This fact is exploited by taking vertex numbers as array indices for nodal quantities. For example, in a mesh with m vertices in 3D, nodal quantities like force and displacement are vectors from $\mathbb{R}^{3m}$. Such a vector is stored as an array of floating point numbers. The entries corresponding to a vertex $v \in \mathbb{N}$ are stored at locations $3v$ to $3v + 2$ in the array.

7.5 Discussion

We have presented a data structure for maintaining the connectivity of simplicial meshes in an arbitrary spatial dimension. The data structure can be specified in terms of *abstract simplexes*, ordered sequences of vertices. These simplexes can be represented directly in the computer, and are also used to specify and implement operations changing the mesh connectivity. Properties of the mesh and the integrity of the data structure, which are crucial in proving traversal algorithms correct, can be verified automatically.

We have discussed two operations to change mesh connectivity, replace-elements and change-elements, and have shown how to implement them. Unfortunately, neither change-elements nor replace-elements are guaranteed to deliver valid data structures, unless extra conditions are given on their arguments. An example is in Figure 7.8, where a mesh change is shown that violates Condition (7.4). Catching these mistakes requires storing more information of the mesh, making change operations more expensive. Fortunately, Conditions (7.1) to (7.5) can be checked automatically in a validation routine. Such a validation routine is expensive, but it can help program debugging. Less expensive checks can also help catching errors. For example, some errors can be caught by checking that no key occurs twice when forming newfaces in the replace-elements algorithm.

The algorithms presented do not have optimal performance. For example, the change-elements contains spurious updates of the boundary. Another source of overhead are updates of mesh.elements in replace-elements. During invocations of this routine old Element objects are removed from the mesh, and new ones introduced. The advantage is that it is easy to catch some programming errors: when an Element or Face object is discarded, it may be flagged as "invalid". Bugs caused by using invalid objects can thus be caught automatically. The disadvantage is that every replace-elements call—even if it does not change the number of elements—changes mesh.elements, and will cost $\mathcal{O}(\log(n))$ time, where n is the number of elements in the mesh. This overhead could be eliminated by reusing old Element objects.

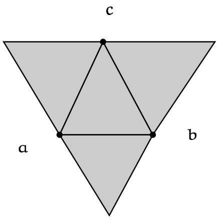


Figure 7.8: Not all invalid mesh changes can be caught. When peforming replace-elements($\emptyset, \{abc\}$) on the mesh shown above, the d-simplex abc and its faces will represented by two objects.

Only d- and $(d-1)$-dimensional mesh features are identified with objects. This limits its applicability; both for higher order FEM discretizations and for certain relaxation techniques it is necessary to also track edges, which are $(d-2)$-dimensional for $d = 3$, across mesh changes. In a higher order FEM discretization, nodes, i.e. interpolation conditions, are also located on edges of the elements. These nodes are shared by all elements containing that edge, so each edge must be uniquely identified. For a linear FEM interpolation, most off-diagonal entries of the stiffness matrix correspond to force/displacement relations of two nodes connected by an edge. Certain relaxation algorithms exploit matrix structure by traversing the matrix column by column or row by row, for example the Gauss-Seidel iteration [47]. A matrix-free implementation of this algorithm must maintain lists of edges incident to each node. It possible to tracking these edges using a Mesh-topology-watcher instance.

Chapter 8

Conclusions

In this thesis we have studied aspects of simulating interactively deformable objects, in the context of training systems for surgical procedures. It is said that such training systems should reproduce mechanical behavior accurately, which motivated our choice for the Finite Element Method (FEM) as a discretization and modeling technique. The linear elasticity approximation for mechanical FEM problems is attractive, since part of the solution can be computed in advance, yielding a guaranteed performance. Unfortunately, this technique cannot be used with more advanced, nonlinear material models and it is impractical when the mesh is large or is changed online. Motivated by this observation, we have proposed solution schemes based on static iterative relaxation methods.

In general, all computational techniques have a tradeoff between accuracy and computational cost. For example, the discretization error in a FEM approximation depends on the granularity of the mesh. More refined meshes lead to more accurate solutions, but also have increased costs due to their increased size. This tradeoff holds even more strongly for iterative solution techniques: their rate of convergence is also decreased by adverse mesh characteristics.

The influence of mesh characteristics and convergence speed is a common theme in this thesis. In Chapter 3 we have seen that flat elements significantly slow down the convergence speed of the CG algorithm, and in Chapter 4, we have seen that this slowdown is not limited to the linear CG algorithm. In Chapter 6 we have observed that information travels faster over coarser mesh parts, and that this can lead to improved convergence. Since size and quality of the mesh influence convergence speed, it is important to keep quality high, and size low while changing it. Hence, Chapter 5 shows a method for making cuts in meshes that produces better and smaller meshes than element subdivision, the most common technique for incorporating cuts in meshes. In Chapter 6 we showed a simple yet effective technique for locally refining meshes while maintaining element quality. That chapter also explored the connection between computational cost and accuracy more deeply. In the scenario of needle insertion, the accuracy of the solution, and the mesh resolution—and hence the update rates of the system—are closely coupled. The difference in the update rates can be large.

In literature on surgery simulations, systems are often described with qualifications such as “real-time”, or “runs at 25 Hz.” The performance of such systems is hard to evaluate with these numbers by themselves. In the first place, these numbers measure computational cost, and this measurement depends on the quality of the implementation, and the hardware and supporting software used. For this reason, Chapter 4 analyzes the speed of our static relaxation by comparing it to another algorithm coded within the same framework, and measures the speed of the implementation by comparing it to the speed of the machine/compiler combination. In the second place, even with an accurate indication of computational cost, an indication of accuracy is needed to evaluate the total cost of a solution. For this reason, Chapter 6 takes into account the accuracy of the solution when listing computation speeds.

We recall that our original choice for the FEM as a method for surgery simulation was motivated by its promise of higher realism. In other words, it was driven by the need for accuracy in the simulation. When we look at possible future extensions, we see that many other mathematical techniques are available that might improve accuracy or decrease computional costs. This is not surprising; tissue mechanics are complex, while an interactive simulation allows only small amounts of computation. There are virtually infinitely many areas of improvement. For determining which technical improvements really improve the overall solution, a broader view should be taken. Technical improvements should not only be mathematically justified, but also be tested against the problem being solved, i.e., simulation of surgical procedures for virtual training environments.

Bibliography

[1] Mark Francis Adams. *Multigrid Equation Solvers for Large Scale Nonlinear Finite Element Simulations*. PhD thesis, University of California, Berkeley California 94720, January 1999.

[2] Ron Alterovitz, Ken Goldberg, Jean Pouliot, Richard Taschereau, and I-Chow Joe Hsu. Surgical needle insertion and radioactive seed implantation: Simulation and sensitivity analysis. In *IEEE International Conference Robotics and Automation (ICRA)*, May 2003. Accepted for publication.

[3] Ron Alterovitz, Jean Pouliot, Richard Taschereau, I-Chow Joe Hsu, and Ken Goldberg. Simulating needle insertion and radioactive seed implantation for prostate brachytherapy. In J. D. Westwood et al., editor, *Medicine Meets Virtual Reality 11 (MMVR11)*, pages 19–25. IOS Press, January 2003.

[4] Stuart S. Antman. *Nonlinear Problems of Elasticity*, volume 107 of *Applied mathematical sciences*. Springer Verlag, 1995.

[5] Stuart S. Antman. *Nonlinear Continuum Physics*, chapter 1, pages 1–21. Springer-Verlag, 2000.

[6] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Value Problems Theory and Computation*. SIAM, 2001.

[7] Fred S. Azar, Dimitris Metaxas, and Mitchell D. Schnall. A deformable finite element model of the breast for predicting mechanical deformations under external perturbations. *Academic Radiology*, 8(10), October 2001.

[8] J. Baillie, H. Evangelou, P. Jowell, and P. B. Cotton. The future of endoscopy simulation: a Duke perspective. *Endoscopy*, pages 542–543, 1992.

[9] M. Beer-Gabel, S. Delmotte, and L. Muntlak. Computer assisted training in endoscopy (c.a.t.e.) from a simulator to a learning station. *Endoscopy*, 24:534–538, 1992.

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

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

[12] Daniel Bielser, Volker A. Maiwald, and Markus H. Gross. Interactive cuts through 3-dimensional soft tissue. In P. Brunet and R. Scopigno, editors, *Computer Graphics Forum (Eurographics '99)*, volume 18(3), pages 31–38, 1999.

[13] F. Boux de Casson and C. Laugier. Modelling the dynamics of a human liver for a minimally invasive surgery simulator. In Chris Taylor and Alan C. F. Colchester, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, volume 1679 of *LNCS*, pages 1156–1165, 1999.

[14] Dietrich Braess. *Finite Elements; theory, fast solvers and applications in solid mechanics*. Cambridge University Press, 2nd edition, 2001.

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

[16] Erik Brisson. Representing geometric structures in d dimensions: Topology and order. *Discrete and Computational Geometry*, 9:387–426, 1993.

[17] Morten Bro-Nielsen. *Medical Image Registration and Surgery Simulation*. PhD thesis, Dept. Mathematical Modelling, Technical University of Denmark, 1997.

[18] Morten Bro-Nielsen and Stephane Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 3(15):57–66, 1996.

[19] Morten Bro-Nielsen, David Helfrick, Bill Glass, Xiaolan Zeng, and Hugh Connacher. VR simulation of abdominal trauma surgery. In *Medicine Meets Virtual Reality* 6, pages 117–123. IOS Press, 1998.

[20] Joel Brown, Kevin Montgomery, Jean-Claude Latombe, and Michael Stephanides. Algorithmic tools for real-time micro-surgery simulation. *Medical Image Analysis*, 6(3):289–300, September 2001.

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

[22] Cynthia D. Bruyns, Steven Senger, Anil Menon, Kevin Montgomery, Simon Wildermuth, and Richard Boyle. A survey of interactive mesh-cutting techniques and a new method for implementing generalized mesh cutting using virtual tools. *The Journal of Visualization and Computer Animation*, 13:21–42, 2002.

[23] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel. Directed edges — A scalable representation for triangle meshes. *Journal of Graphics Tools*, 3(4):1–12, 1998.

[24] G. F. Carey. Parallelism in finite element modeling. *Communications in Applied Numerical Methods*, (2):143–153, 1986.

[25] L. Paul Chew. Guaranteed quality mesh generation for curved surfaces. In *Annual ACM Symposium on Computational Geometry*, pages 274–280. Association for Computing Machinery, 1993.

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

[27] Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):62–73, January-March 1999.

[28] Steven A. Cover, Norberto F. Ezquerra, James F. O'Brien, Richard Rowe, Thomas Gadacz, and Ellen Palm. Interactively deformable models for surgery simulation. *IEEE Computer Graphics and Applications*, pages 68–75, November 1993.

[29] Fred H. Croom. *Basic Concepts of Algebraic Topology*. Undergraduate Texts in Mathematics. Springer-Verlag, 1978.

[30] James W. Daniel. The conjugate gradient method for linear and nonlinear operator equations. *SIAM Journal on Numerical Analysis*, 4(1):10–26, March 1967.

[31] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*, chapter 9. Springer-Verlag, 2000.

[32] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM Press, 2001.

[33] J. E. Dennis and J. J. Moré. A characterization of superlinear convergence and its application to quasi-Newton methods. *Math. Comp.*, 28:549–560, 1974.

[34] Olivier Devillers. On deletion in Delaunay triangulations. In *Annual ACM Symposium on Computational Geometry*. Association for Computing Machinery, June 1999.

[35] Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. Topology preserving edge contraction. Technical Report RGI-Tech-98-018, Raindrop Geomagic Inc., Research Triangle Park, North Carolina, 1998.

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

[37] S. P. DiMaio and S. E. Salcudean. Simulation of percutaneous procedures. In T. Dohi and R. Kikinis, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, number 2489 in LNCS, pages 253–260. Springer-Verlag, 2002.

[38] Simon P. DiMaio and S. E. Salcudean. Simulated interactive needle insertion. In *Tenth Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 344–351. IEEE, 2002.

[39] David P. Dobkin and Michael J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[40] Herbert Edelsbrunner and Nimish R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.

[41] R. Fletcher. *Practical methods of optimization.* John Wiley & Sons, 1987.

[42] R. Fletcher. An overview of unconstrained optimization. Technical Report NA/149, Dept. of Mathematics and Computer Science, University Dundee, 1993.

[43] Lori A. Freitag and Carl Ollivier-Gooch. A comparison of tetrahedral mesh improvement techniques. In *5th International Meshing Roundtable*, pages 87–106. Sandia National Laboratories, October 1996.

[44] Y. C. Fung. *Mechanical properties of living tissues.* Springer-Verlag, 1993.

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

[46] Fabio Ganovelli, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. *Computer Graphics Forum*, 19(3):271–282, 2000.

[47] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* John Hopkins University Press, Baltimore, Maryland, 3rd edition, 1996.

[48] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[49] Kim Vang Hansen and Ole Vilhelm Larsen. Using region-of-interest based finite element modeling for brain-surgery simulation. In William M. Wells, Alan C. F. Colchester, and Scott Delp, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, volume 1496, pages 305–316. Springer-Verlag, 1998.

[50] Desmond J. Higham. Trust region algorithms and timestep selection. *SIAM J. Numerical Analysis*, 37(1):194–210, 1999.

[51] Gentaro Hirota, Susan Fisher, Andrei State, Chris Lee, and Henry Fuchs. An implicit finite element method for elastic solids in contact. In *Computer Animation*, 2001.

[52] D. Ives. Geometric grid generation, surface modeling, grid generation, and related isseus in computational fluid dynamic (cfd) solutions. In *Proc NASA-Conference*, 1995. NASA CP-3291.

[53] Doug L. James and Dinesh K. Pai. Artdefo—accurate real time deformable objects. In *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 65–72. ACM Press, 1999.

[54] Douglas Leonard James. *Multiresolution Green's Function Methods for Interactive Simulation of Large-scale Elastostatic Objects and Other Physical Systems in Equilibrium*. PhD thesis, University of British Columbia, 2001.

[55] Barry Joe. Construction of three-dimensional improved-quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, 16(6):1292–1307, 1995.

[56] Shmuel Kaniel. Estimates for some computational techniques in linear algebra. *Mathematics of Computation*, 20:369–378, 1966.

[57] Martin Kauer. *Inverse Finite Element Characterization of Soft Tissues wisth Aspiration Experiments*. PhD thesis, ETH Zürich, 2001.

[58] Prem K. Kythe. *An Introduction to Boundary Element Methods*. CRC Press, 1994.

[59] R. I. Leine, D. H. van Campen, A. De Kraker, and L. van den Steen. Stick-slip vibrations induced by alternate friction models. *Nonlinear Dynamics*, (16):41–54, 1998.

[60] Alan Liu, Christoph Kaufmann, and Daigo Tanaka. An architecture for simulating needle-based surgical procedures. In Wiro J. Niessen and Max A. Viergever, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, number 2208 in LNCS, pages 1137–1144. Springer-Verlag, October 2001.

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

[62] Joseph M. Maubach. Local bisection refinement for n-simplicial grids generated by reflection. *SIAM Journal for Scientific Computing*, 16(1):210–227, January 1995.

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

[64] A. R. Mitchell and D. Griffiths. *The Finite Difference Method in Partial DIfferential Equations*. John Wiley & Sons, 1980.

[65] M. Mooney. A theory of large elastic deformation. *J. Applied Physics*, 1:582–592, 1940.

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

[67] K. Mori, Y. Seki, J.-I. Hasegawa, J.-I. Toriwaki, H. Anno, and K. Katada. A method for shape deformation of organ and its application to virtualized endoscope system. In H. U. Lemke, M. W. Vannier, and K. Inamura, editors, *Computer Assisted Radiology and Surgery*, pages 189–194. Elsevier Science B.V., 1997.

[68] Ernst Peter Mücke. *Shapes and Implementations in Three-Dimensional Geometry*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1993.

[69] Ernst Peter Mücke. A robust implementation for three-dimensional Delaunay triangulations. *Internat. J. Comput. Geom. Appl.*, 8(2):255–276, 1998.

[70] Megumi Nakao, Tomohiro Kumroda, Hiroshi Oyama, Masaru Komori, Tetsuya Matsuda, and Takashi Takahashi. Combining volumetric soft tissue cuts for interventional surgery simulation. In T. Dohi and R. Kikinis, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, number 2489 in LNCS, pages 178–185. Springer-Verlag, 2002.

[71] Paul F. Neumann, Lewis L. Sadler, and Jon Gieser M.D. Virtual reality vitrectomy simulator. In William M. Wells, Alan C. F. Colchester, and Scott Delp, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, volume 1496, pages 910–917. Springer-Verlag, 1998.

[72] Han-Wen Nienhuys and A. Frank van der Stappen. Combining finite element deformation with cutting for surgery simulations. In A. de Sousa and J. C. Torres, editors, *EuroGraphics Short Presentations*, pages 43–52, 2000.

[73] Han-Wen Nienhuys and A. Frank van der Stappen. A surgery simulation supporting cuts and finite element deformation. In Wiro J. Niessen and Max A. Viergever, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, volume 2208 of *LNCS*, pages 153–160. Springer-Verlag, October 2001.

[74] Han-Wen Nienhuys and A. Frank van der Stappen. A Delaunay approach to interactive cutting in triangulated surfaces. In *Fifth International Workshop on Algorithmic Foundations of Robotics*, Advanced Robotics, page ? Springer Verlag, 2003.

[75] Jorge Nocedal. Theory of algorithms for unconstrained optimization. *Acta Numerica*, 1:199–242, 1992.

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

[77] G. Picinbono, H. Delingette, and N. Ayache. Non-Linear Anisotropic Elasticity for Real-Time Surgery Simulation. *Graphical Models*, 2002. In press.

[78] G. Picinbono, J.-C. Lombardo, H. Delingette, and N. Ayache. Anisotropy, interaction and extrapolation for surgery simulation. *Journal of Visualisation and Computer Animation*, 2001.

[79] Dominique P. Pioletti. *Viscoelastic properties of soft tissues.* PhD thesis, EPFL, Lausanne, 1997.

[80] AlliedSignal plastics. Snap-fit design manual. http://www.honeywell-plastics.com/ed/snapfit/intro.html, 1998.

[81] M. C. Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21:604–613, 1984.

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

[83] Markus A. Schill, Clemens Wagner, Marc Hennen, Hans-Joachim Bender, and Reinhard Männer. EyeSi — a simulator for intra-ocular surgery. In Chris Taylor and Alan C. F. Colchester, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, volume 1679 of *LNCS*, pages 1166–1174, 1999.

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

[85] E. G. Sewell. *Automatic generation of triangulations for piecewise polynomial approximation.* PhD thesis, Purdue University, West Lafayette, IN, 1972.

[86] Jonathan Richard Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Annual ACM Symposium on Computational Geometry*, pages 76–85. Association for Computing Machinery, 1998.

[87] Jonathan Richard Shewchuk. What is a good linear element? Interpolation, conditioning and quality measures. In *11th International Meshing Roundtable Conference*, pages 115–126. Sandia National Laboratories, September 2002.

[88] Dave Shreiner and OpenGL Architecture Review Board, editors. *OpenGL reference manual.* Addison-Wesley, 1999.

[89] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2000.

[90] Hiromasa Suzuki, Takashi Kanai, Yusuke Sakurai, and Fumihiko Kimura. Interactive mesh dragging with an adaptive remeshing technique. volume 16, pages 159–176. Springer-Verlag, 2000.

[91] Naoki Suzuki, Asaki Hattori, Akihiro Takatsu, Takahiro Kumano, Akio Ikemoto, Yoshitaka Adachi, and Akihiko Uchiyama. Virtual surgery system using deformable organ models and force feedback system with three fingers. In William M. Wells, Alan C. F. Colchester, and Scott Delp, editors, *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, volume 1496, pages 397–403. Springer-Verlag, 1998.

[92] G. Székely, Ch. Brechbühler, R. Hutter, A. Rhomberg, N. Ironmonger, and P. Schmid. Modelling of soft tissue deformation for laparoscopic surgery simulation. *Medical Image Analysis*, 4(1):57–66, March 2000.

[93] R. Taghavi. Automatic, parallel and fault tolerant mesh generation from cad. *Engineering with Computers*, 12:178–185, December 1996.

[94] Frank Tendick, Michael Downes, Tolga Goktekin, Murat Cenk Cavusoglu, David Feygin, Xunlei Wu, Roy Eyal, Mary Hegarty, and Lawrence W. Way. A virtual environment testbed for training laparoscopic surgical skills. *Presence*, 9(3):236–255, 2001.

[95] Demetri Terzopoulos and Keith Waters. Analysis and synthesis of facial image sequences using physical and anatomical models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):569–579, June 1993.

[96] Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors. *Handbook of grid generation*, chapter 21, pages 21–6–21–8. CRC Press, 1999.

[97] A. van der Sluis and H. A. van der Vorst. The rate of convergence of conjugate gradients. *Numerical Mathematics*, 48:543–560, 1986.

[98] D. R. Veronda and R. A. Westmann. Mechanical characterizations of skin—finite deformation. *J. Biomechanics*, 3:111–124, 1970.

[99] K. J. Weiler. *Topological Structures for Geometrical Modeling*. PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, 1986.

[100] Xunlei Wu, Michael S. Downes, Tolga Goktekin, and Frank Tendick. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. *Computer Graphics Forum*, 20(3), 2001.

[101] Yan Zhuang and John Canny. Real-time global deformations. In *Algorithmic and Computational Robotics*, pages 97–107. A. K. Peters, 2001.

[102] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method, Solid Mechanics*, volume 2. Butterworth-Heinemann, 2000.

[103] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method, The Basis*, volume 1. Butterworth-Heinemann, 2000.

Appendix A

Tensor calculus

A.1 Tensors

The basis for expressing elastic equations is Euclidian 3-dimensional space, i.e., $\mathbb{R}^3$ with the Euclidian inner product. Vectors from $\mathbb{R}^3$ are denoted by bold lower case letters, e.g. $\mathbf{a}$, $\mathbf{b}$, and are also known as first-order tensors, or 1-tensors. We assume that an inner product on $\mathbb{R}^3$ is given, and that it is denoted by $\mathbf{a} \cdot \mathbf{b}$ for $\mathbf{a}$ and $\mathbf{b} \in \mathbb{R}^3$.

If we have a basis $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ for $\mathbb{R}^3$, then we can determine the components of a vector with regard to that basis using the dual basis. The dual of $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ is denoted $\{\mathbf{a}^1, \mathbf{a}^2, \mathbf{a}^3\}$. It is determined uniquely by

$$\mathbf{x} = \sum_i (\mathbf{a}^i \cdot \mathbf{x})\mathbf{a}_i, \qquad \mathbf{x} \in \mathbb{R}^3$$

A basis is called *orthonormal* if it is equal to its own dual.

The set of linear mappings from $\mathbb{R}^3$ to $\mathbb{R}^3$ is a 9-dimensional space, denoted by $\mathrm{Lin}(\mathbb{R}^3, \mathbb{R}^3)$, or Lin for short. The elements of Lin are also known as second order tensors, and are printed in bold upper case, e.g. $\mathbf{T}$ and $\mathbf{E}$. The identity tensor is denoted by $\mathbf{I}$.

When we evaluate a linear mapping $\mathbf{A}$ in a point $\mathbf{x}$, we write $\mathbf{A} \cdot \mathbf{x}$. The product $\mathbf{A} \cdot \mathbf{B}$ of two 2-tensors $\mathbf{A}$ and $\mathbf{B}$ is the 2-tensor defined by

$$(\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{x} = \mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{x}).$$

The transpose (or adjoint) $\mathbf{A}^*$ of $\mathbf{A}$ is the unique tensor satisfying

$$\mathbf{u} \cdot (\mathbf{A} \cdot \mathbf{v}) = (\mathbf{A}^* \cdot \mathbf{u}) \cdot \mathbf{v}. \tag{A.1}$$

For notational convenience, we set $\mathbf{v} \cdot \mathbf{A} := \mathbf{A}^* \cdot \mathbf{v}$. The transpose is a linear operation on 2-tensors. A tensor is called symmetric if $\mathbf{A}^* = \mathbf{A}$.

A function taking $\mathbf{u}$ to $\mathbf{u} \cdot \mathbf{A} \cdot \mathbf{u}$ is called a quadratic form. If $\mathbf{u} \cdot \mathbf{A} \cdot \mathbf{u} > 0$ for all $\mathbf{u} \neq 0$, then the quadratic form is positive definite, and if $\mathbf{u} \cdot \mathbf{A} \cdot \mathbf{u} \geq 0$, then it is positive semidefinite. Negative definite and negative semidefinite are defined similarly.

If $\mathbf{a}$ and $\mathbf{b}$ are 1-tensors, then we can construct a linear mapping from $\mathbf{a}$ and $\mathbf{b}$ by setting

$$(\mathbf{a} \otimes \mathbf{b})(\mathbf{x}) = (\mathbf{b} \cdot \mathbf{x})\mathbf{a}, \qquad \mathbf{a}, \mathbf{b} \in \mathbb{R}^3. \tag{A.2}$$

This mapping is called a dyadic product, *dyad* or tensor-product. To illustrate the meaning, when $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ and $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$ are orthonormal bases, then the dyad $(\mathbf{a}_2 \otimes \mathbf{b}_1)$ applied to $\mathbf{x}$ takes the magnitude of the $\mathbf{b}_1$ component of $\mathbf{x}$, and maps that component to $\mathbf{a}_2$. If $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ and $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$ are bases of $\mathbb{R}^3$, then set of dyads given by $\{\mathbf{a}_i \otimes \mathbf{b}_j | i, j = 1, 2, 3\}$ has nine elements, and it forms a basis of the linear mappings of $\mathbb{R}^3$. Since these dyads form a basis, we may also use Equation (A.2) as a definition for function application. Higher order tensor products (3-tensors and 4-tensors) may also defined, to represent mappings between $\mathbb{R}^3$ and Lin and between Lin and Lin.

We can also express matrix multiplication using dyads. Let $\mathbf{A} : \mathbf{a} \otimes \mathbf{b}$ and $\mathbf{B} := \mathbf{c} \otimes \mathbf{d}$, then we have

$$(\mathbf{A} \cdot \mathbf{B})(\mathbf{x}) = (\mathbf{a} \otimes \mathbf{b})\mathbf{c}(\mathbf{d} \cdot \mathbf{x}) = (\mathbf{b} \cdot \mathbf{c})(\mathbf{a} \otimes \mathbf{d})(\mathbf{x}).$$

Since the dyads form a base of Lin, we may also define linear operations in Lin in terms of dyads. For instance, the transpose or adjoint can be defined as

$$(\mathbf{a} \otimes \mathbf{b})^* = \mathbf{b} \otimes \mathbf{a}.$$

The trace is a linear functional on Lin: it takes a linear mapping, and returns a number. It can be defined in terms of dyads

$$\operatorname{trace}(\mathbf{a} \otimes \mathbf{b}) = \mathbf{a} \cdot \mathbf{b}. \tag{A.3}$$

With the help of the trace operator we can define an inner product on Lin. The inner product between $\mathbf{A}$ and $\mathbf{B}$ is denoted by $\mathbf{A} : \mathbf{B}$, and is given by

$$\mathbf{A} : \mathbf{B} = \operatorname{trace}(\mathbf{B}^* \cdot \mathbf{A}). \tag{A.4}$$

We have $\operatorname{trace}(\mathbf{A}) = \mathbf{A} : \mathbf{I}$. When 2-tensors are represented by matrices, then the trace corresponds to the sum of the diagonal elements. The inner product on Lin corresponds to the Euclidian inner product on $\mathbb{R}^{3 \times 3}$.

By use of tensor products, we may extend these inner products to tensor products between arbitrary dimensions. The inner product of a k and l-tensor product

$$(\mathbf{a}_1 \otimes \cdots \otimes \mathbf{a}_k) \cdot (\mathbf{b}_1 \otimes \cdots \otimes \mathbf{b}_l) = (\mathbf{a}_k \cdot \mathbf{b}_1)(\mathbf{a}_1 \otimes \cdots \otimes \mathbf{a}_{k-1}) \otimes (\mathbf{b}_2 \otimes \cdots \otimes \mathbf{b}_l).$$

Similarly, we may extend the inner product for two tensors to arbitrary dimensions.

$$(\mathbf{a}_1 \otimes \cdots \otimes \mathbf{a}_k) : (\mathbf{b}_1 \otimes \cdots \otimes \mathbf{b}_l) = (\mathbf{a}_{k-1} \cdot \mathbf{b}_1)(\mathbf{a}_k \cdot \mathbf{b}_2)(\mathbf{a}_1 \otimes \cdots \otimes \mathbf{a}_{k-2}) \otimes (\mathbf{b}_3 \otimes \cdots \otimes \mathbf{b}_l).$$

The expression

$$\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$$

measures the oriented volume of the parallelepiped spanned by $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$. It is called the *scalar triple-product* of vectors $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$. The determinant of a mapping $\mathbf{A}$ from

Lin measures how the volume of a parallelepiped changes when it transformed through $\mathbf{A}$. This definition is independent of the parallelepiped used. In other words, given an arbitrary set of independent vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$, then the determinant is uniquely defined by

$$\det \mathbf{A} = \frac{(\mathbf{A}\mathbf{a}) \cdot (\mathbf{A}\mathbf{b} \times (\mathbf{A}\mathbf{c}))}{\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})}. \tag{A.5}$$

The determinant satisfies $\det(\mathbf{A} \cdot \mathbf{B}) = \det \mathbf{A} \det \mathbf{B}$, and hence $\det(\mathbf{A}^{-1}) = 1/\det \mathbf{A}$, if $\mathbf{A}^{-1}$ exists. It follows that $\det(\mathbf{X}\mathbf{A}\mathbf{X}^{-1}) = \det(\mathbf{A})$: the determinant is invariant under a change of basis.

A vector $\mathbf{v}$ is called an eigenvector of $\mathbf{A}$ if there is a number λ, the eigenvalue, such that

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{c}.$$

Eigenvalues are given by the roots of the *characteristic polynomial.* The characteristic polynomial of a 2-tensor $\mathbf{A}$ is defined as $\det(\mathbf{A} - \lambda \mathbf{I})$. We can expand this expression as a polynomial, thus obtaining

$$\det(\mathbf{A} - \lambda \mathbf{I}) = -\lambda^3 + \iota_1 \lambda^2 - \iota_2 \lambda + \iota_3.$$

The coefficients ι_1, ι_2 and ι_3 in this expansion are called *invariants* of $\mathbf{A}$. Since the determinant is invariant under change of basis, the invariants also are. If λ_1, λ_2 and λ_3 are the eigenvalues of $\mathbf{A}$, then we have

$$\begin{aligned} \iota_1(\mathbf{A}) &= \lambda_1 + \lambda_2 + \lambda_3, \\ \iota_2(\mathbf{A}) &= \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1, \\ \iota_3(\mathbf{A}) &= \lambda_1\lambda_2\lambda_3. \end{aligned} \tag{A.6}$$

The invariants can also be determined directly from $\mathbf{A}$. We have

$$\begin{aligned} \iota_1(\mathbf{A}) &= \operatorname{trace}(\mathbf{A}), \\ \iota_2(\mathbf{A}) &= \frac{1}{2}((\operatorname{trace} \mathbf{A})^2 - \operatorname{trace}(\mathbf{A}^* \cdot \mathbf{A})), \\ \iota_3(\mathbf{A}) &= \det \mathbf{A}. \end{aligned}$$

A.2 Tensor calculus

A function $f : \mathbb{R} \to \mathbb{R}$ is differentiable in $x \in \mathbb{R}$, when there is a number $d(x)$ and a function $r(x, h)$, such that

$$f(x + h) = f(x) + d(x)h + r(x, h), \qquad r(x, h) = o(h) \text{ when } h \to 0.$$

In other words, f is differentiable in x if it may be linearly approximated in a neighborhood of x. The function $d(x)$ is the derivative of f in x, also denoted by $\frac{df}{dx}$.

This definition can be generalized to higher dimensions. Let $\mathcal{V}$ and $\mathcal{W}$ be finite-dimensional vector spaces. A function $F : \mathcal{V} \to \mathcal{W}$, is called Fréchet-differentiable in v if there is a linear mapping $L : \mathcal{V} \to \mathcal{W}$ such that

$$F(v + h) = F(v) + L(h) + o(h), \quad h \in \mathcal{V}.$$

Since L is a linear mapping, we may write it as some product of some D in the tensor product space of $\mathcal{V}$ and $\mathcal{W}$. This representant D of the mapping L is called the derivative.

For example, if a function f maps vectors from $\mathbb{R}^3$ to numbers, then it is differentiable in $\mathbf{x}$ if there exists functions $f_{\mathbf{x}}$ and r, such that

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + f_{\mathbf{x}}(\mathbf{x}) \cdot \mathbf{h} + r(\mathbf{x},\mathbf{h}),$$

and $r(\mathbf{x},\mathbf{h}) = o(\|\mathbf{h}\|)$. The mapping $\mathbf{h} \mapsto f_{\mathbf{x}}(\mathbf{x}) \cdot \mathbf{h}$ is a linear mapping. The function can be represented as an inner product of $\mathbf{f}_{\mathbf{x}}(\mathbf{x})$ and the argument. Hence, $\mathbf{f}_{\mathbf{x}}(\mathbf{x})$ is is called the derivative or gradient of f. It is denoted as $\frac{\partial f}{\partial \mathbf{x}}$. Other notations include grad f or ∇f.

If $f : \mathrm{Lin} \to \mathbb{R}$ is a differentiable function taking linear mappings to scalars, then there exists a function r and $\partial f/\partial \mathbf{A} : \mathrm{Lin} \to \mathbb{R}$, such that

$$f(\mathbf{A}+\mathbf{H}) = f(\mathbf{A}) + \frac{\partial f}{\partial \mathbf{A}} : \mathbf{H} + r(\mathbf{A},\mathbf{H}), \quad \mathbf{A} \in \mathrm{Lin},$$

where $r(\mathbf{A},\mathbf{H})/\|\mathbf{H}\| \to 0$ when $\mathbf{H} \to 0$. The derivative $\frac{\partial f(\mathbf{A})}{\partial \mathbf{A}}$ is also denoted by $f_{\mathbf{A}}(\mathbf{A})$.

Some derivatives of standard functions are given here.

$$\begin{aligned} \frac{\partial\, \mathrm{trace}(\mathbf{C})}{\partial \mathbf{C}} &= \mathbf{I}, \\ \frac{\partial \det(\mathbf{C})}{\partial \mathbf{C}} &= \det \mathbf{C}\mathbf{C}^{-*}, \\ \frac{\partial \iota_2(\mathbf{C})}{\partial \mathbf{C}} &= \mathrm{trace}(\mathbf{C})\mathbf{I} - \mathbf{C}, \end{aligned}$$

The inverse of a 2-tensor is another 2-tensor, so taking inverses is a function from Lin to Lin. Its derivative is a linear function from Lin to Lin, which may be represented as a 4-tensor. To avoid the hassle of representing 4-tensors, we simply give the derivative applied to some $\mathbf{H} \in \mathrm{Lin}$:

$$\frac{\partial \mathbf{C}^{-1}}{\partial \mathbf{C}} : \mathbf{H} = -\mathbf{C}^{-1} \cdot \mathbf{H} \cdot \mathbf{C}^{-1}.$$

We mention one differential operator that we shall encounter further, the divergence. The divergence of a vector field $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^3$ is defined by

$$\mathrm{div}\, \mathbf{f} = \mathrm{trace}(\partial \mathbf{f}/\partial \mathbf{x}).$$

The divergence of a tensor field $\mathbf{T}$ is defined by

$$\mathrm{div}\, \mathbf{T} = \partial \mathbf{T}/\partial \mathbf{x} : \mathbf{I}.$$

In one dimension, the value of an integral over an interval of a continuous function is given by the values of its primitive at the boundaries of that interval. A similar theorem holds in higher dimensions. If Ψ is a tensor valued function, and continuously differentiable on its domain Ω, and continuous on the closure of Ω, then

$$\int_{\Omega} \partial\Psi/\partial\mathbf{x}\, \mathrm{d}v(\mathbf{x}) = \int_{\partial\Omega} \Psi \otimes \mathbf{n}(\mathbf{x})\, \mathrm{d}a(\mathbf{x}),$$

where $\mathbf{n}$ is the outward pointing normal on $\partial\Omega$. This theorem can applied to tensors of different order, e.g. the divergence theorem. One form that we employ is

$$\int_{\mathcal{B}} \operatorname{div} \mathbf{T}\, \mathrm{d}v(\mathbf{x}) = \int_{\partial\mathcal{B}} \mathbf{T} \cdot \mathbf{n}\, \mathrm{d}v(\mathbf{x}). \tag{A.7}$$

Samenvatting

Wie kent niet het spelletje *Flight Simulator*? Gewapend met een joystick kunnen we tegenwoordig vanachter het buro een (virtueel) vliegtuig de hele wereld rondsturen. Achter dit ogenschijnlijk onschuldige tijdsverdrijf gaat echter een serieuzere toepassing schuil. Door vliegtuigen na te bootsen op computers is het mogelijk om piloten te laten oefenen met vliegen zonder dat dure vliegtuigen de hangar hoeven te verlaten. Ook rampscenario's, zoals het uitvallen van een motor, kunnen worden getraind zonder noemenswaardige risico's of kosten.

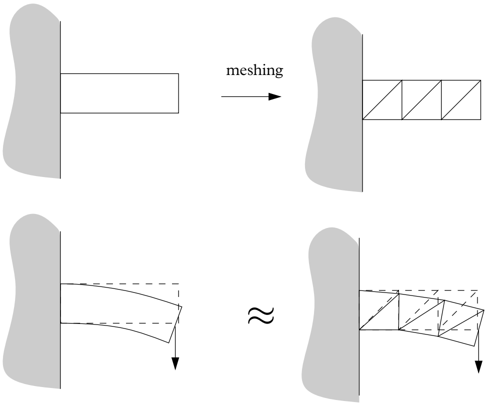
Net als piloten moeten chirurgen in hun werk complexe handelingen uitvoeren, waarbij het gevolg van een fout ernstig kan zijn. Het is lijkt daarom aantrekkelijk om ook chirurgen te trainen met behulp van computers. Een dergelijk simulatie systeem bootst de anatomie van een patient na, en stelt de te onderwijzen student in de gelegenheid om te experimenteren met snijden, schroeien, hechten, etc. in virtueel lichaamsweefsel. De student hanteert een speciale joystick, die de handelingen doorgeeft aan een computersimulatie. Deze simulatie berekent direct het resultaat van de manipulatie, bijvoorbeeld een vervorming of snede. Het resultaat wordt direct zichtbaar gemaakt, en via *force-feedback* ook voelbaar gemaakt in de joystick.

Een kernprobleem bij het construeren van zulke systemen is simuleren hoe menselijk weefsel vervormt. Enerzijds moet deze vervorming enigszins realistisch zijn, zodat de simulatie als geheel geloofwaardig is, anderzijds is er voor het berekenen van die vervormingen slechts zeer weinig rekentijd beschikbaar: om goede terugkoppeling voor de gebruiker te garanderen, moet het antwoord direct getoond moet worden. Tussen deze twee tegenstrijdige belangen moet dus een compromis gevonden worden.

Realistische simulatie van vervorming is op zich geen nieuw onderzoeksterrein. Botstests voor auto's worden bijvoorbeeld al jaren doorgerekend op de computer met een techniek die bekend staat als de Eindige Elementen Methode (*Finite Element Method*, FEM). De kern van deze techniek is dat het te vervormen object wordt opgedeeld in blokjes (in ons geval driehoeken of tetraeders). Uit de mechanische eigenschappen van dit collectief van blokjes worden vergelijkingen afgeleid, en daarmee wordt een oplossing berekend. De eindige elementen methode is geïllustreerd in figuur A.1.

Dit proefschrift onderzoekt aspecten van interactieve vervormings met de Eindige Elementen Methode, waarbij we de toepassing op chirurgiesimulatie in ons achterhoofd houden.

Bij het toepassen van eindige elementen op medische simulaties is er een extra complicatie. De opdeling in blokjes (het rooster), kan tijdens de simulatie veranderen. Bij-



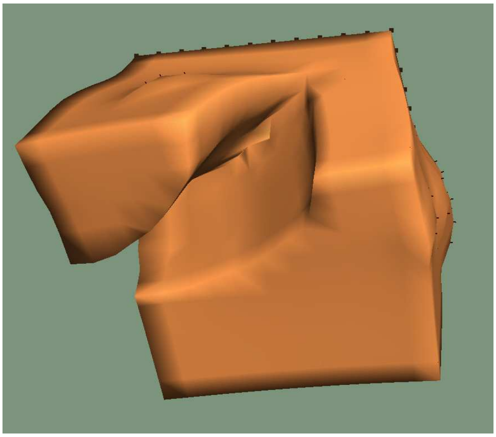
Figuur A.1: In de eindige elementen methode wordt het te simuleren object (boven links) in driehoekjes ("blokjes") opgedeeld (boven rechts). De exacte oplossing van een probleem (onder links) wordt dan benaderd door het geroosterde object (onder rechts).

voorbeeld, als er gesneden wordt in een object, dan verandert de vorm van het object, en dus ook het rooster. Deze veranderingen hebben invloed op het ontwerp van simulaties, immers alle informatie die van het rooster afhangt verandert ook, en kunnen we beter helemaal niet uitrekenen of opslaan. Daarnaast is er een numeriek effect dat minder voor de hand ligt: de eigenschappen van het rooster zijn van invloed op de snelheid en de precisie van de berekeningen: mooiere roosters leveren preciezere en snellere berekeningen. Als het rooster veranderd wordt (bijvoorbeeld door een gesimuleerde snede), mag dit de kwaliteit van het rooster dus niet nadelig beïnvloeden.

Hoofdstuk 3 beschrijft onze eerste pogingen om een eindige elementen simulatie te construeren. De simulatie gebruikt een eenvoudig elasticiteitsmodel (lineaire elasticiteit), met een statische iteratieve oplossingsmethode: een numeriek proces dat stapje voor stapje naar de uiteindelijke oplossing toe gaat. Het systeem heeft ook de mogelijk om te snijden in het gesimuleerde object. Hierbij hebben we een techniek geïntroduceerd die de grootte van het rooster (het aantal blokjes) constant houdt. Dit is aantrekkelijk, aangezien dit de rekentijd per stap laag houdt. Figuur A.2 laat een plaatje zien van het prototype in actie.

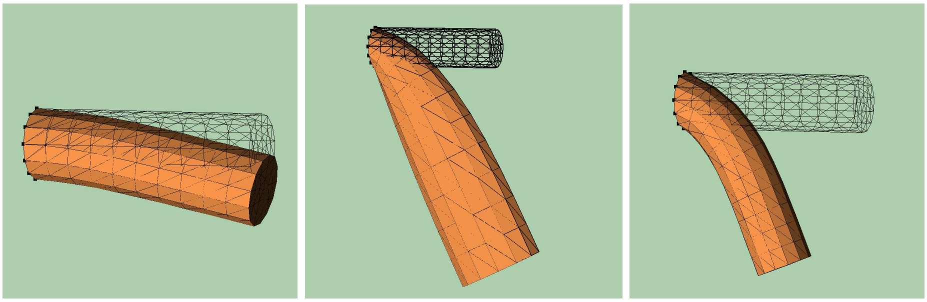
Het systeem voldeed aan onze eerste verwachtingen, maar tijdens de implementatie kwamen we onverwachte problemen tegen. Ten eerste, het lineaire elasticiteitsmodel is alleen realistisch voor beperkte gevallen, namelijk als de vervormingen relatief klein zijn. Ten tweede, onze snijtechniek verandert de vorm van de rooster-blokjes zodanig dat elke stap van het oplossingsproces maar traag naar de oplossing toe gaat: na het maken van snedes nemen de prestaties van het programma af.

Hoofdstuk 4 onderzoekt de statische iteratieve aanpak voor het doorrekenen van deformaties nauwkeuriger. De aanpak uit hoofdstuk 3 is hiervoor uitgebreid naar ingewikkelder materiaalmodellen. Het verschil tussen deze modellen is geïllustreerd in figuur A.3. Om na te gaan hoe snel onze algoritme werkt, is het vergeleken met de stan-



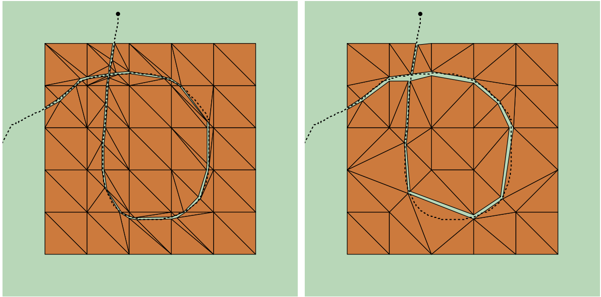
Figuur A.2: Een lineair elastisch object met een snede. Het object is aan de achterkant vastgezet, en wordt zijwaarts uitgerekt.

daard methode voor het doorrekenen van elasticiteitsvergelijkingen: dynamische relaxatie met demping. Uit experimenten kunnen we afleiden dat bij verstandige implementatie onze aanpak altijd minstens even snel is als de dynamische. Uit deze experimenten blijkt ook nogmaals dat de roosterkwaliteit van grote invloed is op de rekensnelheid, en dat drie-dimensionale simulaties te rekenintensief zijn voor serieuze interactieve toepassigen.



Figuur A.3: Lineaire elasticiteit is een benadering die geldig is voor kleine vervormingen (links). Bij grotere belastingen van zacht materiaal leidt het tot onrealistische resultaten (midden). Een niet-lineair materiaal, zoals neo-Hookeaans materiaal (rechts), geeft betere resultaten, maar vereist ingewikkeldere algoritmes. Door te testen hoe snel het object de ruststand bereikt vanaf de beginstand (als *wire frame* weergegeven), kunnen we een nagaan hoe snel een bepaalde algoritme is.

Het is moeilijk het inwendige van ruimtelijke objecten te visualiseren en te begrijpen. Dit maakte het lastig de problemen van de snijtechniek in hoofdstuk 3 te begrijpen. In hoofdstuk 5 werpen we daarom nogmaals een blik op snijden in roosters, maar doen dat voor platte, twee-dimensionale objecten. We presenteren een techniek om met een puntvormige scalpel snedes te maken in een rooster opgebouwd uit driehoeken. Om de kwaliteit van het rooster te waarborgen, gebruiken we daarbij een standaard methode om goede triangulaties te genereren, de Delaunay-triangulatie. Deze nieuwe snijtechniek produceert kleinere roosters met beter gevormde driehoeken, hetgeen ook te zien is in figuur A.4. We laten ook zien hoe deze aanpak te generaliseren is naar gebogen oppervlakken in de ruimte, waar één scalpel op meerdere punten tegelijk een incisie kan maken.

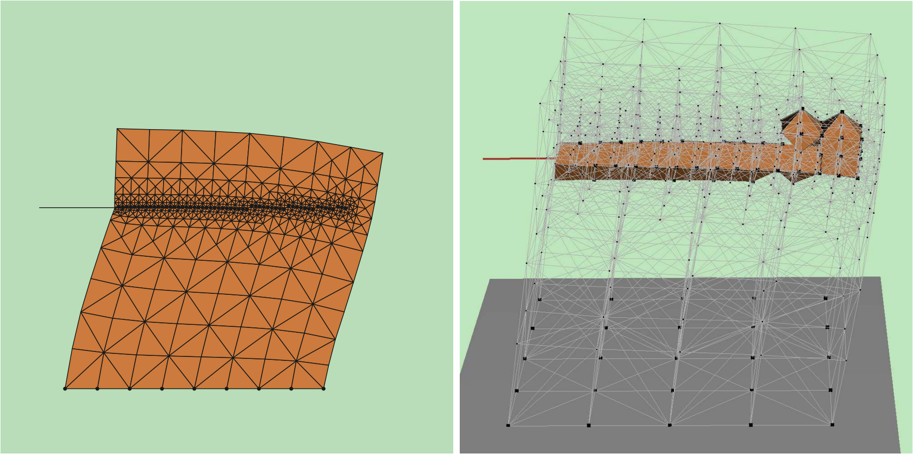


Figuur A.4: Het maken van snedes in roosters kan met behulp van *subdivisie*: driehoekjes worden onderverdeeld om een snede te representeren (links). Het resultaat bevat echter veel driehoekjes, die overwegend plat zijn, hetgeen ongewenst is uit oogpunt van de eindige elementen methode. Door Delaunay-triangulaties handig te gebruiken (rechts), kunnen we hetzelfde pad met minder driehoekjes weergeven, die ook nog minder plat zijn.

Zoals gezegd, voor serieuze toepassingen zijn drie-dimensionale modellen te duur om interactief door te rekenen. In hoofdstuk 6, bekijken we daarom een twee-dimensionaal probleem, namelijk simuleren hoe zacht weefsel vervormt als er naalden in worden gestoken. Dit probleem is al opgelost door anderen, maar daarbij werd een aanpak gebruikt die fundamenteel beperkt was tot platte objecten, en lineair materiaal. In dit hoofdstuk presenteren we een aanpak die ook andere modellen kan gebruiken, en bovendien generaliseert naar ruimtelijke problemen. In dit hoofdstuk wordt er niet gesneden, maar het rooster wordt wel gewijzigd door verfijningen. Door het rooster alleen daar te verfijnen waar nodig, kunnen we goedkoop een relatief precieze simulatie doen. Figuur A.2 geeft een voorbeeld van zo'n simulatie.

Al het werk in de voorgaande hoofdstukken is uitgeprogrammeerd. Bij al deze systemen, moet de simulatie bijhouden hoe de blokjes—driehoeken of tetraeders—met elkaar verbonden zijn. Bij het het implementeren van deze systemen is een datastructuur ontwikkeld, die de laag-bij-de-grondse administratie van al die verbindingen onderbrengt in een aparte programma module. Roosterwijzigingen kunnen daardoor in de rest van het programma op een abstract nivo gespecificeerd en geprogrammeerd kunnen worden. Hoofdstuk 7 beschrijft deze datastructuur, en geeft algoritmes om de administratie op orde te houden.

De motivatie van dit onderzoek was hoe vervormingen ten behoeve van medische



Figuur A.5: Een simulatie van een naald insertie in 2D (links). Door rondom de naald het rooster te verfijnen, kan tegen geringe kosten een hoge precisie worden behaald. Door een eenvoudige verfijningstechniek te gebruiken, kan hetzelfde procedé ook in 3D worden toegepast (rechts).

trainingssimulaties realistisch gesimuleerd konden worden. Deze toepassing bracht ons op een eindige elementen aanpak met veranderende roosters. In hoeverre dit voldoende is om bruikbare simulaties te bouwen, blijft een open vraag: realisme is niet alleen een kwalitatief maar ook een kwantitatief begrip. Meer kwantitatief realisme, met andere woorden, een hogere precisie, wordt betaald met intensiever berekeningen en dus tragere programma's. De haalbaarheid van zulke simulaties staat of valt dus met de gewenste precisie, en die kan enkel in de praktijk worden bepaald. Dit geldt ook voor eventueel verder onderzoek. Er zijn vrijwel oneindig veel technieken om de huidige aanpak te verbeteren. De mechanica van levend weefsel is immers complex, terwijl een interactieve simulatie zeer weinig rekentijd toestaat. Om een zinvolle keuze te maken uit al deze mogelijkheden, is het nodig technieken te toetsen aan praktijktoepassingen.

Dankwoord

Vanaf deze plek wil iedereen bedanken die hun steentje hebben bijgedragen aan de bevalling van dit boekje. Ten eerste ben ik natuurlijk veel verschuldigd aan mijn dagelijks begeleider Frank van der Stappen, die vrijwel altijd tijd voor me had. Ik denk met plezier terug aan vele interessante gesprekken, niet alleen over onderzoek maar ook over allerlei buitenschoolse activiteiten. Mijn promotor Mark Overmars verdient ook een bijzondere vermelding voor zijn bedrevenheid en ervaring als onderzoeker en begeleider. Het is wonderlijk hoe hij tegen mijn ras-cynisme in altijd de positieve kanten van mijn onderzoek naar boven wist te brengen.

I would like to thank the reading committee (Jan van Leeuwen, Max Viergever, Henk van der Vorst, Markus Gross, and Ken Goldberg) for reading my thesis and their comments. I also want to thank my proofreaders Arno Kamphuis and Sergio Cabello for reading and commenting on preliminary versions of this thesis.

Dank aan Peter Bozarov, die me begeleiden van de andere kant liet meemaken. Ondertussen zitten we in het zelfde schuitje, en mag mijn code ook de vergetelheid in.

Ik heb de afgelopen jaren genoten van de ongedwongen sfeer in het instituut. Ik wil alle collega's bedanken die mijn verblijf aangenaam hebben gemaakt: first and foremost thanks to my roommate Sergio, for discussing how to perform bodily functions with the computer, and much more. Dank aan Marleen, omdat rood de kleur is van geluk. Zeger, voor alle vuige gersigheid. Thanks to Guille—May the Force be with you. Dank aan Arthur voor de meeslepende videodiners, en René en Jasper voor wijn en stapels pankoeken daarbij.

Dank aan alle muziek-vrienden, klimmaatjes, en buren die me—gelukkig—dwongen het toetsenbord te verlaten, alsmede alle rare tiepens die me er nog enthousiaster weer lieten plaatsnemen.

Dank aan mijn ouders voor hun onvoorwaardelijke steun, en natuurlijk pappa voor het vinden de laatste taalfouten, en mamma voor de kalligrafie. Dankjewel Han-Kwang voor het laten gappen van je LaTeX stijl-files.

Dank aan Tommy, omdat ze me zo lief volstrekt niet begreep. Tot slot, voor iedereen die zijn naam hier mist: ik ben je misschien vergeten, maar toch bedankt voor alles!

Curriculum vitae

Han-Wen Nienhuys werd op 15 januari 1975 geboren te Eindhoven. In juli 1992 behaalde hij het VWO-diploma aan het Hertog Jan College (thans Scholengemeenschap Were Di) te Valkenswaard. In datzelfde jaar begon hij aan de opleiding Technische Wiskunde aan de Technische Universiteit Eindhoven. Zijn afstudeeronderzoek was getiteld “Convoluties van signalen uit meer-dimensionale domeinen” en werd verricht bij dr. ir. S. J. L. van Eijndhoven. In februari 1998 behaalde hij het ingenieursdiploma met lof. Van maart 1998 tot maart 2003 was hij in dienst van het Instituut voor Informatiekunde en Informatica van de Universiteit Utrecht als AIO, eerst bij prof. L. G. L. T. Meertens, en vanaf maart 1999 bij prof. dr. M. H. Overmars, bij wie het in dit proefschrift beschreven onderzoek werd verricht.

Colophon

This thesis was typeset in $\LaTeX 2^{\varepsilon}$. No electrons were harmed during the production of this thesis.

The characters on the cover (Pinyin transcription “yóu rèn yǒu yú”, pronunciated more or less like “yo run yo ew”) form a Chinese saying. Literally translated it means “the wiggling blade has room to spare.” This expression is used to express confidence that someone has more than ample capacities for the task at hand. It was calligraphed by the author’s mother. The proverb was derived from a parable in *The secret of caring for life*, a chapter from the Taoist text *The Inner Chapters* by the Chinese philosopher and poet Chuang Tzu (ca. 360 BC). In this parable, Chuang Tzu illustrates how our conscious self falls away when we are fully absorbed in a task. A translation of this parable is on the back cover.

Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school. ASCI dissertation series number 88