Biomechanical Soft Tissue Modeling – Techniques, Implementation and Applications

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Dipl.-Phys. Markus A. Schill
aus Heilbronn

Oktober, 2001

Dekan:	Professor Dr. Herbert Popp, Universität Mannheim
Referent:	Professor Dr. Reinhard Männer, Universität Mannheim
Korreferent:	Professor Dr. Karl-Heinz Höhne, Universität Hamburg

Tag der mündlichen Prüfung: 19. November 2001

Preface

January 1996 to Today

While a team of talented people is particularly important in software development, I found myself all alone in January 1996, when Reiner Männer challenged me to build up a research group at his chair, the Department for Computer Science V at University of Mannheim. The group should deal with medical image processing and biomechanical simulations, where the latter was soon to become dominant due to the direction our work was heading for.

In early 1997 Joachim Meißner became the first diploma student of the research group to be. He developed a finite element model for the Dura mater, a strong skin surrounding the brain. The element he derived found its way to our current code.

The *EyeSi* project was born early in 1996 in a joint effort with Martin Schinkmann, Joachim Bender and Reiner Männer. After about a year of project planning, applications for funding and first design considerations, Marc Hennen started his diploma thesis in summer 1997, laying the fundaments for *EyeSi*. Clemens Wagner joined the team as a PhD student shortly there after, focusing on computer graphics and system architecture. In late 1997 I went to MERL in Cambridge, U.S.A. for a six months research stay and Clemens took over the responsibility for the group which we had named ViPA (Virtual Patient Analysis) in the meantime. Ever since then Clemens has been one of ViPA's major pillars. While I was studying biomechanical simulation algorithms with Sarah Gibson and decented into the depth of ChainMail, Clemens and Marc created the first computer graphical eye model. When Clemens mailed the first screenshots to MERL I pinned them to the walls of my cubical and felt thrilled.

As we were getting more and more engaged with the ideas of Virtual Reality, we decided to investigate its key technologies. Olaf Körner started a diploma thesis in July 1998 investigating haptic feed-back and implementing the first software for the newly acquired PHANToM device. His thesis was about the haptic exploration of volume data sets, however he also contributed greatly to architectural design questions related with other projects. As a matter of

fact Olaf convinced Clemens and myself that *vrmNode* and *vrmConnector* are basically the same.

The summer of 1999 brought a doubling of the group's staff. Johannes Grimm and Nickolaj Nock joined in July. Johannes carried on the brain simulations bringing them to new heights and Nick worked on color segmentation and the generation of world models from camera images. Johannes' work on finite elements and his implementation also became a part of the *vrmDesign*. Nick's work can be regarded as our first work towards optical tracking; another key technology for Virtual Reality. In August Olaf started his PhD. Based on the experiences so far collected with *EyeSi*, he began a second simulator project, which aims at developing a training simulator for endoscopy; internally the project was dubbed "GLdarm". Norbert Hinckers also lined up in August to carry on the haptic research in his diploma thesis. He developed our first, self-designed and very heavy, electro-magnetic force-feed-back device. Since he started Norbert was particularly interested in the *EyeSi* development and has greatly contributed to make the *EyeSi*-project what it is now. In Fall 1999 Thomas Ruf joined ViPA for his diploma. It was the time of the first redesign of the *EyeSi*-hardware. Thomas developed a new very fast optical, tracking system for *EyeSi*. Moreover, his computer knowledge in general, and his coding skills in particular made him a valuable source of information for us all.

The year 2000 brought ViPA its first international PhD member: Liu Bing from China. After developing an environment for the evaluation of tactile displays for blind people, she is now working on the stereo reconstruction- and calibration-processes for the optical tracking system. Johannes started his PhD in August focusing on physical modeling techniques.

Andreas (Mr. C++) Köpfle, also known as the kernel hacker, joined ViPA in May 2001 for a PhD. He will be coordinating the optical tracking developments. So far he has not only contributed to ViPA's work, but also to the linux USB kernel development. Only recently Ralf Panse started his diploma thesis, investigating self calibration methods for the tracking system.

Without having mentioned the many students, who also contributed greatly to the developments, these are the people who teamed up to make great and wonderful things ...

... but wait – this is not the acknowledgment section!

Comment

The structure of the thesis at hand does not strictly follow the way in which such a thesis is usually composed. Usually the first half describes what has been reached in research so far and the second half explains the findings and results

of the own work. Instead I decided to follow a concept that seems more natural to me, the sequence from *Techniques* over *Implementation* to *Applications*.

Here is a direct list of what I feel to be the major contributions to research from this work: Free-sampling (page 37), the Enhanced ChainMail algorithm (page 43), the implemented software architecture *vrmDesign* (page 73), which unites several modeling approaches and visualization techniques and of course the two simulation projects *EyeSi* (page 92), which I feel particularly associated with, and *Brain3D* (page 104).

Acknowledgment

I want to thank Prof. Dr. Reinhard Männer for giving me the chance to establish the ViPA research group at his chair, the responsibility he delegated to me and the freedom I had in my work. I also want to thank him for being my advisor for the thesis at hand. He always had an ear for my problems and sometimes fancy solutions to them.

Prof. Dr. Dr. Hans-Joachim Bender advised all medical projects conducted in the ViPA group. He supported me right from the beginning and provided valuable contacts that helped to accomplish my plans. He sometimes called me during the time at MERL; just to keep contact! Thank you.

I want to particularly thank Prof. Dr. Karl Heinz Höhne for readily taking the time to read and give his expertise on the thesis at hand.

Dr. Sarah Gibson invited me to MERL and provided me with a completely different research experience. Thank you for a time in which I could focus on my research and everything else was taken care of.

Many thanks to Clemens Wagner for endless discussions on senseless and also extremely meaningful issues (particularly the one in Dallas, I mean the meaningful!?).

Andrea Seeger and Christiane Glasbrenner always cheered me up and helped when things were going astray. In particular Andrea Seeger saved my day more than once.

Clemens Wagner, Norbert Hinckers, Johannes Grimm and Petra Joswig thank you for proof-reading and valuable comments.

A special thanks to all the people of the ViPA research group (they were already mentioned above) for their commitment and the perfect atmosphere they created. To all of you and all associated snipers and campers: “Go go go!!!”

There were also times when the kind of special support was necessary that can only be provided by the right people. Thank you Petra! Thank you Anna and Judith for just being there!

Last but not least, I want to thank my parents, Marianne and Dieter, for raising me the way they did and giving me the chance to do the things I have done.

Contents

1

Introduction

1.1 Motivation

The year is 2026. Dr. I.C. Foo, a promising young micro-surgeon is scheduled to perform a difficult ophthalmic surgery tomorrow morning. Exchanging the retina on a living human eye is far from being a standard procedure. The donor's retina has already been extracted and everything is prepared for the intervention tomorrow. Dr. Foo is well aware of all the risks involved with detaching the retina of his patient and fixing the "exchange part" in place. Disbelief in his own manual abilities can spoil the self-confidence needed to accomplish the task successfully. There are a lot of complications that can happen. When Dr. Foo enters his office at 5 p.m. this evening, he decides to train the operation once more and re-run the surgical procedure in his simulator. This time he will use the individual data scanned from his patient. He always feels better after preparing difficult surgeries with the VR device. After loading the patient's data, Dr. Foo enters the VR-OR.

Next day: Just to let you know, Dr. Foo's patient – Mr. C. Klope – is well up!

Micro surgical interventions are demanding both in performing and teaching. Great manual and psychological abilities are required to concentrate on the small-scale motion to be performed, often over a period of several hours, under microscopic vision as for example in intra occular surgery. Teaching these operative techniques is difficult as well. Animal cadavers fall short of human anatomy and certain pathologies are difficult or even impossible to be created for individual training. To enable surgeons to train and practise difficult interventions before actually performing them on the real patient makes the joint effort of several research fields necessary: medical visualization, general Virtual Reality technology, biomechanical modeling and medical image processing.

A major goal of computer graphics has been the realistic rendering of artifical scenes, objects, creatures and persons, trying to imitate reality; which has ever since been heavily used by the movie industries. In contrast to artists and animators, scientific visualization deals with real data. The area of **medical visualization** rapidly gained importance with the development of volume visualization as the data provided by most medical imaging devices is of volumetric nature. The three-dimensional rendering of human organs captured by e.g. MRI scanners provide a new view on the interior of the human body (see figure 1.1), which led to fascinating projects such as the visible human[1].

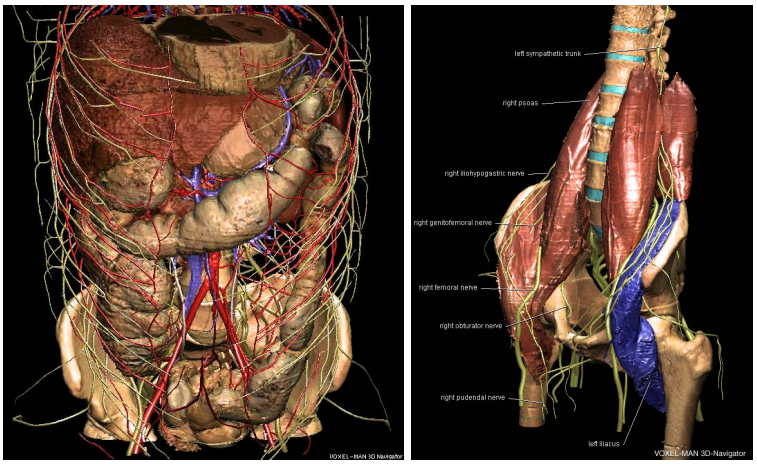

Figure 1.1: Volume rendered images of the interior of the human body. Images from Voxel-Man by Department of Computer Science in Medicine, University Hamburg [HPP+00a], [HPP+00b]

The capability to render images of the above quality and presenting them through stereoscopic vision systems[2] almost immediately generated the desire to touch and interact with the rendered objects. **Virtual Reality technology** constantly develops better displays and even general force-feed-back devices.

However, interacting with structures of the human body requires **biomechanical modeling** to determine the reaction of e.g. a squeezed organ. To generate a biomechanical model the anatomy and the biomechanical properties of the desired structures are needed.

Medical image processing provides segmentation algorithms which can be used to extract individual anatomy from a patient's medical data set. Also information about the state of the tissue at the relevant places can be extracted by medical image processing.

[1] www.nlm.nih.gov/research/visible/visible_human.html

[2] first red/green- then shutter glasses, later head-mounted-displays and today auto-stereo displays

Systems which combined contribution from several or all of the required research fields have already been used for planning and controlling surgeries. The potential of using Virtual Reality for the training of standard, as well as critical situations has been demonstrated by flight simulators in the past. Virtual Reality together with real time biomechanical simulation and visualization is also well suited to improve significantly the way physicians acquire knowledge, train manual abilities and learn surgical procedures.

1.2 Problem

There are at least three major problems in developing a simulation environment like the one Dr. I.C. Foo uses in the year 2026: (1) fast visualization and simulation algorithms, (2) a software architecture that allows the assembly of contributions from the different contributing research fields and (3) patient specific models. Two of those problems are addressed in this thesis. The third problem, the creation of patient specific models from scanner data, respectively the development of scanners that extract the required tissue information with sufficient resolution, falls into another research area.

To interact with an object like a human organ in a medical simulation requires to calculate the organ's reaction to the manipulation. There are two contrary approaches one can follow to determine how the tissue reacts. The organ's reaction can either be calculated with the physical theory of elasticity, which currently requires way too much computational time, or some "pseudo-physical" method has to be invented which produces similar results.

As scientific visualization and biomechanical simulation have so far evolved independently, they work with different data structures. In general medical simulations today use two or even more different representations of the object they simulate; one representation is generally required for visualization and one for simulation. Sometimes there is an extra representation for collision detection. The multiplicity of representations leaves the problem of synchronization and memory consumption.

1.3 Overview

First, this thesis addresses the need for suited simulation algorithms. Chapter 2 discusses briefly the modeling process in general and introduces two definitions which will be used to classify – and should help to assess – the modeling techniques presented later. Chapter 2 closes with a presentation of the physics of elasticity as the correct physical description still represents the gold standard for soft tissue simulations.

Chapter 3 gives an overview over the state of the art in biomechanical modeling. The basic principles of the presented techniques are explained, followed by examples of how they have been used to model soft tissue by the research community. Each technique is classified according to the definitions given in the previous chapter.

In chapter 4 the Enhanced ChainMail algorithm is presented together with several examples which illustrate its capabilities. The performance of the algorithm is analysed and a detailed discussion on its strength and weakness is given.

Then, in chapter 5, the thesis presents a software architecture which could overcome the multiple representations problem mentioned in the previous section. Several biomechanical simulation algorithms and OpenGL visualization are already implemented in the presented design.

Finally, two applications of biomechanical soft tissue modeling are presented, that were developed in the scope of this thesis. Both applications presented in chapter 6, the simulation of an intra-ocular surgery and the simulation of a decompressive craniotomy, were realized using the presented software architecture.

2

Biomechanical Modeling

Chapter 2 provides an overview over the modeling process in general and the physics (theory of elasticity) needed for biomechanical modeling in particular. The first section 2.1 introduces the idea of the *modeling circle*. The modeling circle describes the general modeling process. Single tasks found as parts of the circle occur in similar form in every modeling process.

The second section 2.2 suggests two definitions that can be used to characterize a particular modeling approach. Such a definition is desirable as it has become fashionable in the field of biomechanical modeling to call approaches "physical" or "physically based". Such a nomenclature is often chosen to suggest that the particular model is very close to physical reality, which is not always true. In chapter 3 some of the most common modeling techniques will be presented. The given definitions will be used to characterize each presented modeling technique.

A summary of the theory of elasticity will be given in section 2.3. The theory of elasticity represents the current gold standard in modeling deformable objects. Although elasticity theory deals with idealized bodies, which generally don't exist in real problems, approaches which follow the theory exactly often cannot be conducted. They lead to complex expressions that either cannot be solved at all due to numerical instability or require computational time which exceeds the time constraints for reasonable solving.

2.1 Modeling and the Modeling Cycle

Modeling is a very common task in computer science. Real-world processes are mapped into algorithmic representations that can be processed by a computer[1]. A model can be considered appropriate if the algorithmic representation is capable of reproducing or predicting the outcome of the real-world-process for a given set of start conditions. The more general the start conditions are allowed to be, the more general the model can be used. A model that maps exactly one real world process under one defined set of start conditions into a computer can be regarded as automation. Depending on the computational time required for finding the solution of the algorithmic representation, *on-* and *offline* simulations can be distinguished.

Soft tissue modeling deals with predicting the behavior of deformable tissue when forces or displacements are applied to one or several parts of the tissue. The tissue may be subject to certain constraints (starting/boundary conditions); it can be fixed or supported in one or many places. Soft tissue simulation covers research areas reaching from medical simulations – e.g. predicting the outcome of surgical procedures – over character animation – providing e.g. realistic facial expressions – to cloth/fabric simulation for the fashion industry.

In contrast to rigid body simulation, which can very well exploit the general laws of physics, the behavior of soft tissue can be much more complex and often requires different approaches. Using classical physical approaches mostly leads to formulations which cannot be solved analytically and whose numerical solutions are time-consuming and difficult or even impossible due to numerical instability. To overcome these problems, a variety of other (non-physics) approaches have been developed. Some of them are referred to as *physical* or *physically based* whereas others can clearly be associated with animation techniques.

No matter which category a particular model might fall into, the process of finding a model always includes similar tasks. In general, all models are designed by following the same sequence of steps:

1. Observe the domain of the problem and try to find regularities that can be used for a model.

2. Find an idealization for the problem which is capable to reproduce the observed regularities.

3. Construct a model, generally a mathematical formulation that reflects the regularity. In particular investigate the start and boundary conditions of the formulation.

[1] In this sense an accounting software models the real-world process of money flow in a company.

4. Use the model to make a prediction; solve the mathematical formulation.

5. Validate the prediction with experiments.

6. Correct model; start process all over.

Errors can occur in all steps of the modeling process. Most severely are the errors made in the idealization of the problem; wrong idealizations can never produce right simulation results. Numerical errors should always be considered as they always occur while solving the mathematical problem formulation numerically (step 4).

Figure 2.1 illustrates the modeling process described above.

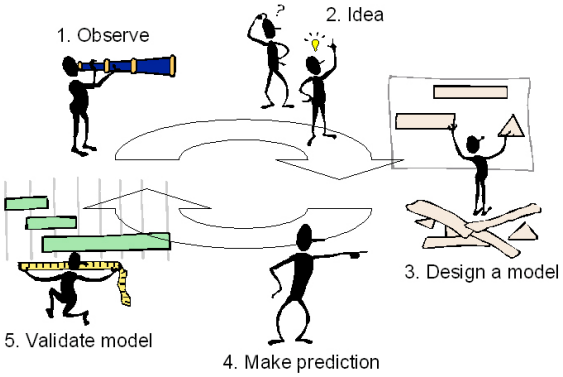

Figure 2.1: The modeling cycle.

Once a model is found, it has to be tested and validated. Testing includes the investigation of the valid scope of the model by varying the start and boundary conditions. There are various ways to validate a given model. Obviously the best way is to compare the real-world and the simulated system. Therefore it is necessary to determine exactly the status of the real-world system at the beginning and at the end of the process. As this is not always possible validation by an expert with knowledge about the behavior of the system has become common practice in these cases.

Online simulations have the great advantage that they produce their results within fractions of a second which makes the iterative modeling process faster. In addition, online simulations allow the interactive adaptation of model parameters, accelerating the testing and fitting of the model.

2.2 Physical and Descriptive Modeling

Whereas simulations use physical laws to predict a real-world process, animations do not necessarily have to follow physical laws. An animator transforms his/her own ideas of how an object should behave when exposed to external forces. Although animations don't have to follow causality, there is a strong desire to make animations follow some particular rules. This desire is mainly founded in the development of animation systems which are capable of evolving a given scene following such particular rules. Apart from using different rules the situation is very similar in a physical simulation. Given a set of start- and boundary condition and all effective forces, the system evolves according to the general laws of physics.

Research has produced a lot of different modeling approaches to calculate the shape changes of deformable objects. They reach from purely geometric techniques to the accurate physical description of a problem. In between these, several other forms have established, trying to overcome the lack of physics in geometric models on one side or the expensive modeling and time consuming calculations necessary in physical modeling on the other side. A lot of these models are called *physical* or *physically based*. At least with respect to the models capability of producing physically correct simulation results this denotation is misleading. Often the attribute is simply based on the use of physical terms like energy, mass or force in analogy to their physical meanings.

The following definitions will be used to classify the modeling techniques that will be introduced in the next section. There are two contrary approaches for finding an adequate model.

Physical modeling techniques: Physical models are constructed using physical laws. Usually the result is a system of differential equations, containing parameters with a physical meaning. The result of a physical model is only governed by these parameters. As the parameters reflect physical properties they can be measured in the real-world process. In case of biomechanical modeling such parameters are typically the Young's modulus E and the Poisson ratio ν. A good model makes correct predictions only if (math.: iff) the measured, real physical properties are used for the calculations. In physical modeling the **functionality** of the real-world process is modeled. Detailed physical knowledge about the simulated process is necessary.

Descriptive modeling techniques: Descriptive models do not exploit physical knowledge. Instead they parameterize observed behavior. Descriptive models mimic the desired properties of a real-world system. The free parameters of the descriptive model are used to fit modeled and observed behavior. Like in animations the quality of a descriptive model depends on the skill and experience of the designer. In general a descriptive model behaves similar to the real-world process, but not necessarily

exactly alike. A descriptive model models the **behavior** of the real-world process. No knowledge about how the process works is necessary.

Physical models are generally easier to validate as they use parameters that are measurable in the real-world process. Their drawback is, that they are expensive; in particular, the required numerical methods for solving the mathematical formulation are very time consuming.

Descriptive models can be difficult to validate - it is possible that once chosen model parameters produce correct results only for conditions that were observed during the parameter fitting. The quality of the model depends on the designer. The advantages of descriptive models are: (1) in order to construct the model no knowledge about the functionality of the real-world process is needed, (2) they can be very fast.

The *modeling pyramid* in figure 2.2 once more illustrates the necessary steps of the modeling process. A real-world process has to be idealized before it can be modeled with e.g. one of the techniques that form the base of the pyramid. Modeling approaches in the base are arranged regarding their physical respectively their descriptive character.

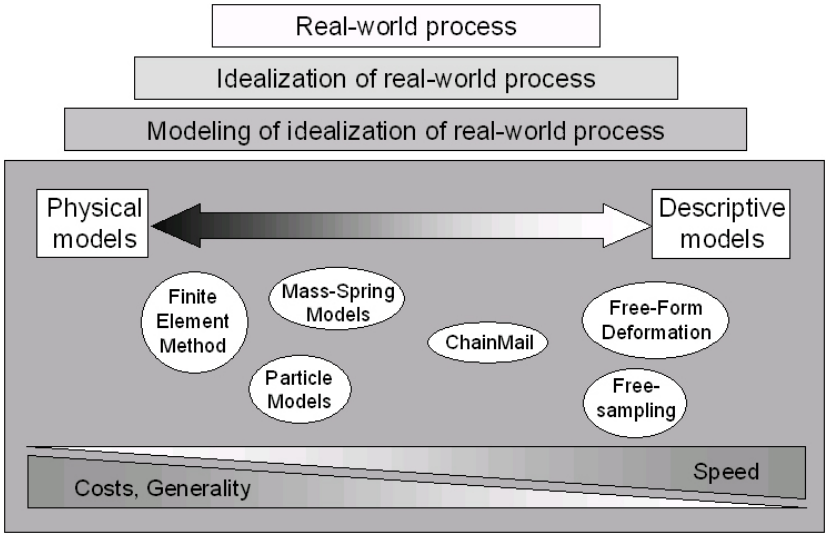

Figure 2.2: At first the real-world process is idealized, then the idealization is modeled. Some of the available modeling approaches are arranged in the base of the *modeling pyramid*. They are classified in the range from *physical* to *descriptive*. (The bubbles around the particular approach should have a soft boundary.)

Mostly the application one has in mind determines the modeling techniques

that can be used. If the goal is a real-time virtual reality simulator, like the EyeSi system presented in chapter 6, the time consuming physical approaches can not be used. The chosen modeling technique in turn determines the way in which the real-world problem should be idealized. A priori it is not possible to decide that one technique is better than the other. The modeling technique should be chosen according to the problem and its constraints. Choosing a modeling approach which is suited to solve the given problem is often a hard problem.

A physically correct set of rules, however, if it can be derived, will always be able to evolve a given start situation correctly. The theory of elasticity, which is presented in the next section, provides the framework for the physically correct problem formulation.

2.3 Theory of Elasticity

Elasticity problems are formulated using the laws of continuum mechanics. Figure 2.3 shows a deformable body in two configurations: the undeformed

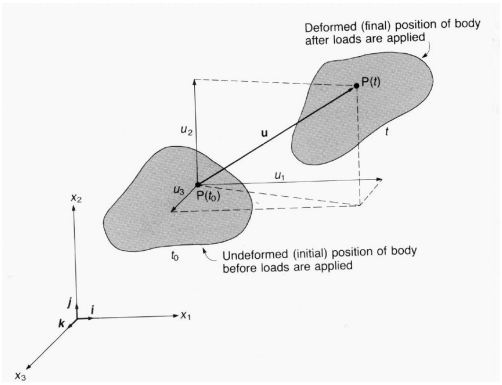

Figure 2.3: Displacement u from undeformed to deformed configuration.

initial configuration (at t_0) and after translation and deformation (at t). The displacement vector $\vec{u}$ describes the distance between the material point $P(t_0)$ and the same point $P(t)$ in the deformed configuration:

$$\vec{u}(x_1, x_2, x_3) = u_1(x_1, x_2, x_3)\vec{i} + u_2(x_1, x_2, x_3)\vec{j} + u_3(x_1, x_2, x_3)\vec{k} \qquad (2.1)$$

The difference between translation and deformation of a body is that during translation the distances between any pair of material points do not change. From this property a measure for deformation of the body can be defined: any measure that considers the change in length between any pair of neighboring material points can be used to measure the deformation.

The strain tensor

Using Eq. (2.1) leads to a deformation measure which is called *Lagrangian strain* tensor. In (2.1) the displacement was expressed as a function of the coordinates of the undeformed configuration. If $\vec{u}$ was expressed as a function of the deformed configuration this would result in a measure which is called *Eulerian strain* tensor. In the following only the Lagrangian strain tensor will be used. In Cartesian coordinates it has the form:

$$E_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \frac{\partial u_k}{\partial x_i}\frac{\partial u_k}{\partial x_j}\right) : i,j = 1,2,3 \tag{2.2}$$

The squared term in Eq. (2.2) is the reason for the nonlinearity of the general theory of elasticity. It makes the strain a nonlinear function of the displacements.

Most formulations of elasticity problems are limited to linearity which means they only use the first part of (2.2)

$$\epsilon_{ij} = E_{ij} \approx \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) : i,j = 1,2,3 \tag{2.3}$$

This approximation however, is valid only if the displacement gradients are small relative to unity, because only then the products of the gradients, last term in (2.2), are small relative to the gradients themselves and can therefore be neglected.

The classical linear strain tensor has the following form:

$$\begin{aligned} \epsilon_{11} &= \frac{\partial u}{\partial x} \\ \epsilon_{22} &= \frac{\partial v}{\partial y} \\ \epsilon_{33} &= \frac{\partial w}{\partial z} \end{aligned} \tag{2.4}$$

$$\begin{aligned} \epsilon_{12} &= \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \\ \epsilon_{23} &= \frac{1}{2}\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \\ \epsilon_{31} &= \frac{1}{2}\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right) \end{aligned} \tag{2.5}$$

The stress tensor

Stress is generated within a body that is exposed to external forces. Elastic forces try to bring the body back into its initial shape. Elastic forces are near effect forces, i.e. they operate on adjacent points only. Figure 2.4 illustrates the stresses acting on mutually perpendicular coordinate planes that form an infinitesimal volume element dV. The nine stresses can be arranged in a second order tensor[2]. The terms with identical subscripts are called *normal stresses* and the terms with mixed subscripts are called *shear stresses*. Considering

[2]The nine stresses σ_{ij} in figure 2.4 transform as a tensor. See [Bur87]

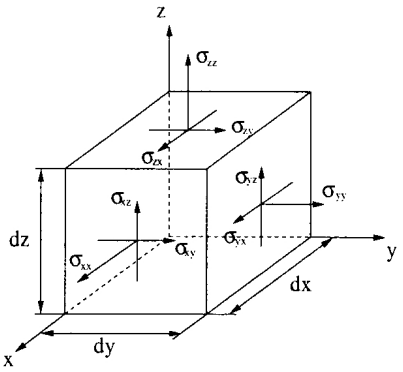

Figure 2.4: Stress on an infinitesimal volume element.

a balance of forces on the infinitesimal volume element of figure 2.4 with an external force f_x in x-direction results in

$$-f_x = \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} \tag{2.6}$$

or generally

$$\vec{F_i} = \frac{\partial \sigma_{ki}}{\partial x_k} \tag{2.7}$$

Also, looking at the balance of moments about each axis yields

$$\sigma_{ij} = \sigma_{ji} \tag{2.8}$$

Thus the stress tensor is symmetric.

The stress-strain relation: material law

The considerations so far are generally valid for all materials. However, to describe a particular material additional information is necessary: the relation between stress and strain which is material dependent. Well known is this relation from Hooke's linear law, where the relative length change of e.g. a wire with length l is directly proportional to the acting force F. A is the cross section of the wire and the force acts perpendicular to A. E is called Young's modulus and is a material dependent figure.

$$\frac{\Delta l}{l} = \frac{1}{E}\frac{F}{A} \tag{2.9}$$

which corresponds to the notation[3]

$$\epsilon = \frac{1}{E}\sigma \tag{2.10}$$

[3]Cauchy stress: $\sigma = \frac{F}{A}$ where F is normal to A. See also [MWMTT98].

One additional figure, besides E, is necessary to describe the properties of a particular material completely: the Poisson's ratio ν. Where E is responsible for length changes of the wire, ν expresses the contraction of the wire with respect to the two other coordinate axis. The Poisson's ratio is related to the degree of volume conservation that might be true for a certain material as it expresses the transverse contraction of a material that is stretched lengthwise.

The generalized form of Hooke's law has the following form:

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl} \tag{2.11}$$

where C_{ijkl} is a fourth-rank tensor with 81 (3^4) elastic moduli. Their number can be reduced to 21 independent elastic moduli using the symmetry of σ and ϵ.A material law considering all 21 independent entries represents the maximum possible degree of anisotropy.

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ & & C_{33} & C_{34} & C_{35} & C_{36} \\ & & & C_{44} & C_{45} & C_{46} \\ & & & & C_{55} & C_{56} \\ & & & & & C_{66} \end{bmatrix} \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \epsilon_{xy} \\ \epsilon_{yz} \\ \epsilon_{zx} \end{pmatrix} \tag{2.12}$$

Equation 2.12 shows only the independent entries. The matrix C is symmetric with $C_{ij} = C_{ji}$.

Most modelers limit their approach to the use of isotropic and homogeneous materials. Together with the assumption of linearity (eq. 2.3) this leads to the important and frequently used isotropic material law.

$$\vec{\sigma} = C\vec{\epsilon} \tag{2.13}$$

with

$$C = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ & 1-\nu & \nu & 0 & 0 & 0 \\ & & 1-\nu & 0 & 0 & 0 \\ & & & \frac{1}{2}-\nu & 0 & 0 \\ & & & & \frac{1}{2}-\nu & 0 \\ & & & & & \frac{1}{2}-\nu \end{bmatrix} \tag{2.14}$$

12 of the 21 elastic moduli from eq. 2.12 vanish for the isotropic case which makes handling much easier [4].

Finding an appropriate material law for a given object/material combination is one of the most difficult tasks in biomechanical modeling. In [Bur87] material laws for several configurations (including anisotropic) are derived. A very good

[4] A short comment on the implications involved with the solving of non-linear elasticity problems follows in section 3.1.1.

general discussion on anisotropy can also be found there. A brief compilation of linear and nonlinear elasticity theory can be found in [MWMTT98]. Material laws for other material properties like viscoelasticity and a section on how difficult it can be to find valid material laws is also presented there. Fundamental continuum mechanics can be studied in [Gol87].

3

Currently used Modeling Techniques

This section presents major modeling techniques used in biomechanical modeling. The techniques will be classified following the definitions introduced on page 8. The focus lies on presenting the basic idea of the particular modeling technique. References to literature are given where needed to allow further studies. Examples for each presented modeling technique will be used to illustrate its capabilities and at the same time provide an overview over the current status of research in biomechanical modeling.

In general, processes in nature are continuous with respect to space and time.[1] Modeling continuous processes with a computer necessarily leads to problems as computers can represent data only with a limited accuracy. Therefore all problems have to be parameterized in a discrete state vector. The problem is then generally solved with a numerical integration technique which approximates the solution at discrete time steps.

A discrete problem representation as necessary for the computer can be achieved in two ways: (1) A continuous problem formulation can be discretizised or (2) the problem itself can be formulated in a discrete way. Both approaches are common in biomechanical modeling. An example for the first are so called finite element models (see section 3.1), the latter way is often used in e.g. mass-spring models (see section 3.2).

[1] If scale is shifted to quantum mechanical dimensions this is no longer true. In our scales however this is a valid assumption

3.1 Finite Element Method

Strictly speaking the finite element method (FEM) is not a modeling technique, but a method to solve a particular formulation of a mathematical problem[2]. Elasticity problems as presented in section 2.3 can be formulated in this particular way. The term *FEM model* is often used for biomechanical simulations where finite elements are used to discretizise the underlying elasticity problem.

3.1.1 Basic Concept

Discretization means that the continuous domain of the problem is divided into smaller elements with a simple regular shape. Adjacent elements are joint at discrete node points. Any point P of the problem domain can be expressed within an element by the use of interpolation functions – so called shape functions – that interpolate P from the surrounding node points N_i. Also any unknown function $g(P)$ applied to the problem domain can be interpolated by the shape functions; any continuous problem $g(P)$ may be approximated by a finite set of equations $g(N_i)$ which hold the node coordinates as variables [Zie75].

FEM treats an object as a continuum. A continuous problem is approximated with the interpolation functions for each element in a way, that the values of the node variables $g(N_i)$ represent the continuous solution of the problem over the element. The element solutions are then joint to the global problem solution. Continuity between two elements is reached by arbitrary nodes on the border of adjacent elements[3].

The basic idea of *finite elements* is that the solution of the problem is easier over the simple and regular shape of the finite element than over the whole problem domain. Therefore the object is subdivided into smaller, but still continuous elements with a more simple shape. The shape and size of the finite elements (often triangles, tetrahedrons or cubes) lead to an easier mathematical description of the problem. A good approximation of the solution can be reached with simple interpolation functions within each element. The interpolation functions describe how the solution varies continously over the element. A valid finite element is characterized by the property that, given a particular element, the higher the discretization of the problem is chosen, the better the approximation of the solution will be. Usually, the interpolation functions are polynomials. The elements are connected at discrete node points. Each node point can have one or more node variables which represent e.g. the node's position and higher derivatives. As the node points correspond to the key points of the interpolation functions the continuity of the solution at element boundaries

[2] boundary value problems (ger. Randwertprobleme)

[3] Derivatives of $g(N_i)$ can also be used as node variables to guarantee higher order continuity.

can easily be achieved[4]. Since the number of key points of a polynome is given by the number of the polynome's coefficients, their number must match the number of the element's node variables. In general the order of the polynomial is chosen to be the lowest possible while still providing enough degrees of freedom (equals the number of coefficients). Using a polynomial of a higher degree than necessary, provides the advantage that the approximation of the solution is better already for a coarser discretization, while the handling of the approach becomes more difficult with respect to computation time and required memory size.

For the analysis of deformable bodies displacement based finite element methods are used. An equilibrium equation is derived using the principle of minimal potential energy. The material displacements over the object are varied until its potential energy is minimal. Such an approach is valid only for static or quasi-static processes[5] as in dynamic processes (see page 19) mass effects contribute to the total energy and minimizing the potential energy of the system does not lead to a stable state.

The Problem Formulation

A complete model of a deformable body is derived from the principle that a general body is in the equilibrium when its potential energy is at a minimum, i.e. a deformable body under external forces will take the shape that represents the minimum of the potential energy. To determine the new shape the total potential energy Π of a body under stress is calculated as

$$\Pi = \Lambda - W \tag{3.1}$$

with Λ denoting the total strain energy caused by the work W that is done by external forces acting on the body. Such external forces can be volumetric forces (e.g gravitation), acting all over the body, surface forces being applied only to the body's surface (e.g. pressure), or local forces acting on discrete points of the body.

Λ and W have to be expressed in terms of object deformation, which can be achieved by using the material displacement function equation (2.1). By varying the material displacements, the minimum of the potential energy can be found, which corresponds to the new shape of the object. The expression for the total potential energy is minimized by derivating it with respect to the material displacements and setting it equal to zero. The result is a continuous differential equation. Solving the differential equation leads to the material displacements that represent the new shape of the body.

[4]Polynomials may be created by specifying a set of (x,y) pairs (key points) and constructing the Lagrange interpolation polynomial that passes through them

[5]quasi-static processes change very slowly, so that they can be regarded as static within the analyzed time step.

Expressing the external work W in terms of material displacements is easy:

$$W = \int_V \vec{p}^T \vec{u}\ dV + \int_S \vec{q}^T \vec{u}\ dS + \sum_{i=1}^{m} \vec{F_i}^T \vec{u_i} \tag{3.2}$$

where $\vec{p}$ is a volumetric force, $\vec{q}$ a surface force and the $\vec{F_i}$ represent forces acting on discrete point of the body.

The strain energy Λ is given by

$$\Lambda = \frac{1}{2} \int_V \vec{\sigma}^T \vec{\epsilon}\ dV \tag{3.3}$$

with $\vec{\sigma}$ the stress vector and $\vec{\epsilon}$ the strain vector from equation (2.13). To find a corresponding formulation, equation (2.13) is used to replace $\vec{\sigma}$ in (3.3).

$$\Lambda = \frac{1}{2} \int_V \vec{\epsilon}^T\ C\ \vec{\epsilon}\ dV \tag{3.4}$$

Thus Λ is a function of the material law which gives the stress-strain relation and thereby governs the behavior of a deformable body under applied forces.

The material displacements can be brought into (3.4) by introducing a differential operator matrix B. (2.4) and (2.5) illustrate that the relation between $\vec{\epsilon}$ and $\vec{u}$ can formally be given by

$$\vec{\epsilon} = B\ \vec{u} \tag{3.5}$$

where B is a 6×3 matrix of differential operators.

$$\Lambda = \frac{1}{2} \int_V \vec{u}^T B^T\ C\ \vec{u}B\ dV \tag{3.6}$$

$$= \frac{1}{2} \vec{u}^T \left(\int_V B^T\ C\ B\ dV \right) \vec{u} \tag{3.7}$$

Substituting (3.2) and (3.7) into (3.1) gives the desired form for the total potential energy as a quadratic function of the node displacement vector $\vec{u}$.

The Solution with FEM

Section 3.1.1 illustrated how an elasto-mechanical problem could be formulated mathematically. However, the differential operator matrix B, used in (3.5), can not be given in a generally valid analytical form and is often difficult to determine for a given problem. Instead it is often easier to derive the relation (3.5) explicitly once the interpolation functions for the finite elements have

been chosen. Detailed presentations of such explicit approaches can be found in [Sch91].

Once the equilibrium condition is derived for each element, the global system is assembled. Contributions to node values at arbitrary node points for adjacent elements are added. In general, this results in a system of linear equations of the form

$$K\vec{U} = \vec{F} \tag{3.8}$$

where K is the stiffness matrix, that reflects the geometry of the problem domain and contains the appropriate material parameters. K is a very large, but sparsely occupied matrix. $\vec{U}$ is the vector of the node variables. It contains the nodal displacements which are the result of the accumulated forces $\vec{F}$ that act on the corresponding nodes. Before actually solving the equation system (3.8) for $\vec{U}$ the boundary conditions of the problem have to be considered. This is done by manipulating the rows and columns of K and $\vec{F}$ accordingly so that the solution will provide the desired values in $\vec{U}$. For a homogeneous boundary condition at node n the nth row and column of K are set equal zero and their diagonal element to one. Also the nth position of $\vec{F}$ is set to equal zero.

The equation system (3.8) can then be solved with either direct methods, like Gauss- or Cholesky-decomposition or iterative methods like conjugated gradients.

Dynamic and Non-linear FEM Analysis

The above considerations were based on the idea of equilibrium. This is only valid for static and pseudo-static (slowly developing) processes. FEM can also be used to solve dynamic problems, i.e. oscillations or systems that are in the process of reaching their equilibrium. In the first case a trigonometric sum is used as interpolation functions and the resulting problem is an eigenvalue equation. In the dynamic case two additional terms are needed in equation (3.8) to express the mass effects and damping of such a system. This results in a second order differential equation completely analogous to the Newtonian equation of motion.

$$M\ddot{\vec{U}} + D\dot{\vec{U}} + K\vec{U} = \vec{F} \tag{3.9}$$

M is the global mass matrix which holds the contributions of all the element mass matrices. The element's mass matrix is calculated by using its interpolation functions to express the object's density and integrate over the element's volume. D is the damping matrix which is sometimes also calculated from the contributions of each element. But since the required damping parameters are often difficult to determine D is generally constructed as a linear combination of M and K. The dynamic system can finally be solved with numerical integration techniques. Presentations can be found in [Bat90].

Static (pseudo-static) and dynamic problem formulations using the assumption of linear material response can be solved completely and correctly. However, although the linear assumption is useful in many cases, it represents an approximation of the true physical behavior of an object. For a physically realistic simulation all the non-linearities of an object should be taken into consideration. Non-linearity applied to soft tissue modeling is extremely complex. It is very difficult to find appropriate material laws to formulate the problem. Also, once formulated, it can be extremely hard to solve the problem mathematically. A common approach for solving non-linear problems is to combine a finite element discretization of the geometry with finite difference discretization of time and an iterative scheme that is called *Updated Lagrangian* which allows a step-by-step solution of the problem. In the Updated Lagrangian approach the non-linear equations are replaced by iterative relations which refer to the corresponding recently computed configuration. The iterative form takes account for changing the tensorial terms in each step as non-constant elasticity parameters are the character of non-linear materials. A chapter on solving non-linear elasticity problems with incremental methods can be found in [MWMTT98].

3.1.2 *FEM* in Soft Tissue Modeling

Various approaches have been suggested for soft tissue modeling with FEM. Most of them use the assumption, that the material reacts linear to stress. Larrabee [Lar86] calculate skin deformation to simulate skin flap design with FEM. The skin was modeled as a linear membrane that is attached to subcutaneous layers with linear springs (the mass-spring approach is presented in section 3.2).

Gourret et. al. [GMTT89] computed the interactions between the soft tissue of the fingers and a deformable object during a grasping task. In a dynamic formulation they used 3D elements with linear interpolation functions.

Celniker and Gossard proposed an approach for the design of free-form surfaces with FEM [CG91]. The user controls the shape of the object by applying external forces on its surface. They used triangular 2D-elements and Hermite polynomials as interpolation functions[6].

Koch et. al. [KGC+96] used a combination of FEM and mass-spring models to predict the outcome of cranio-facial surgery. The skin surface was modeled following the approach from [CG91] for finite element curve and surface free-form shape design. The skin surface was then fixed with springs to the bone. The facial surface was determined with laser scanners whereas the bone surface was extracted from ct scans. The spring stiffnesses where also derived from the ct data. The system is capable of predicting deformations of the facial shape after surgical procedures.

[6] Third order cubic polynomials

Keeve compares the results of cranio facial surgery simulation performed on the one hand with a mass-spring approach and on the other hand with an FEM model (see 27).

A team at INRIA[7] developed a simulator for liver surgery. The system allows the interaction with a deformable model of the liver using a force feedback system. The liver model used in the simulation did undergo several changes since its first presentation. In [CDA96] a pseudo static FEM approach was used. They state in [Del98] that the model was well suited to compute accurate and complex deformation of soft tissue. However, it was too slow to compute deformations in real time on powerful workstations. Therefore they investigated methods to speed up the calculations [BNC96]. The suggested condensation is a standard technique which basically removes nodes while increasing the degree of the interpolation functions (polynomial) used to approximate the solution in each element (compare page 17). In [CDA99] they proposed a method based on a set of pre-computed equilibrium solutions which allowed real time interaction with the model. Both [BNC96] and [CDA99] had the main drawback of not allowing topology changes as they occur when cutting. Changes connected with cutting the model discard any method based on pre-computation of the inverse of the stiffness matrix K (compare (3.8)) as cutting induces a modification of this matrix. Mass-spring models allow cutting without additional time requirements. Consequentially in [dCL99] mass-spring models were investigated. In [DCA99] tensor/mass models, an enhanced mass-spring model allowing anisotropic linear material properties, were introduced and combined with the pre-computing method described above. [PLDA00] summarizes the current development of the hepatic surgery simulator.

Besides skin simulations – mostly for cranio facial surgery (see above) – and the project at INRIA, the simulation efforts focus on simulating the biomechanical behavior of the brain.

In [HLL$^+$00] a finite element analysis was carried out with the goal to investigate brain contusion during an indirect impact. A head injury model was used to simulate forward and backward rotation around the upper cervical vertebra. Intra-cranial pressure and shear stress response were calculated. A relatively coarse 3D finite element model was generated which includes the skull and facial bone – to roughly simulate the inertia of the head – and internal structures like the Dura mater, Falx cerebri and tentorium of the cerebellum. The whole model comprises 1,455 nodes and 1,328 solid elements, mainly eight-node hexahedral element. The tissue was assumed to be homogeneous, isotropic and linearly viscoelastic. The biomechanical parameters used for the brain tissue are presented in table 3.1. The analysis was carried out with the commercial FEM package ANSYS.

In [TSBM94] the distortion and stress distribution in the brain caused by putaminal hemorrhage were calculated using a two dimensional FEM model

[7] `http://www-sop.inria.fr/epidaure/`

of a single cerebral hemisphere. The model was divided into 326 triangular elements. The bleeding within the slice was modeled by six vectors of force radiating from a small area. The material properties were considered to be linear and isotrop. Different properties were used for gray and white matter and the Falx cerebri. The parameters are listed in table 3.1.

In [PBWP99] the goal of the simulation was to study the biomechanics of acute obstructive hydrocephalus and to investigate where peri-ventricular edema will most likely appear. The analysis was carried out on one typical axial two-dimensional slice through the brain. The model consists of 679 quadrilateral elements and 2,208 nodes. The brain tissue was modeled as a two-phase material composed of a porous elastic matrix saturated by interstitial fluid. The biomechanical parameters used in this simulation are listed in table 3.1.

Hartmann presents a high resolution three-dimensional brain model in [Har99]. He investigates dynamic effects as well as the growth of tumors in the brain. The model is discretizised with approximately 150,000 cubical eight-node-elements. As the model is generated from MRI data in a pre-computing step all relevant internal structures are considered. In particular Hartmann investigated the effect of Falx cerebri and tentorium which were segmented manually and integrated into the model. According to his analysis the two structures show only little influence on pressure waves propagating through the brain in a dynamic simulation. In contrast they seem to have a large effect on material transport as for example in tumor growth. The parameters used in the simulation can also be found in table 3.1.

A three-dimensional brain model which describes the brain as a biphasic medium with a solid matrix saturated with interstitial fluid is introduced in [MPK$^+$99]. Their model consists of 25,340 nodes and 139,351 tetrahedral elements. The model also incorporates the Falx cerebri. The study in particular investigated the effect of the Falx cerebri on brain tissue deformation under gravity. The purpose is to improve the prediction of subsurface deformation for intra-operative registration. They conclude that the deformation across the middle line seems to be significantly damped by the Falx cerebri.

The time consuming work to do in finite element analysis is the decomposition of the stiffness matrix K. The dimension of the stiffness matrix is in generally very high.

Essa et. al. [ESP92], [ESP93] suggested to use a mode superposition method based on the modal analysis presented by Pentland [PW89]. The basic idea of the modal analysis is changing the basis of the problem from nodal displacements p to modal displacements q. With $q \ll p$ this brings a significant simplification for the solution of the linear algebraic system as it reduces the dimension of the stiffness matrix. The principle of modal analysis is explained in appendix A. Arbitrary deformations can be expressed by superposing deformation modes. Essa et. al. applied FEM to image processing tasks like shape fitting and motion tracking. They simulated the dynamics of deformable bod-

Tissue	Young's modulus [kPa]	Poisson's ratio	Density $[g/cm^3]$	K[a] $[m^4/(Ns)]$	used in
Brain	250	0.49	1		[HLL+00]
Brain	10	0.30		10^{-11}	[PBWP99]
Brain (white)	3.9	0.47			[TSBM94]
Brain (white)	120	0.499	1.04		[Har99]
Brain (gray)	7,8	0.47			[TSBM94]
Brain (gray)	75	0.499	1.04		[Har99]
Brain	2.1	not given			[MPK+99]
Membrane[b]	31500	0.45	1.133		[Har99]
Falx	900	0.47			[TSBM94]
Falx	12	not given			[MPK+99]
Dura	50000	0.45	1.13		[HLL+00]

[a] hydraulic permeability
[b] probably Falx and tentorium

Table 3.1: Biomechanical parameters of brain tissue used in various simulations.

ies by considering them as linear viscoelastic isotropic incompressible objects. They used a 27-node finite element cube for their simulation.

Chen and Zeltzer employed the modal decomposition approach to speed up their model of skeletal muscles [CZ92]. The goal of their simulation was natural character motion animation. The model also includes the muscle contraction based on Zajac's muscle force model (e.g. in [LB95]). For simplicity reasons the analysis was carried out on a cubic shaped bounding box of the muscle. 20-node iso-parametric brick elements with parabolic interpolation functions were used. The resulting deformations were mapped onto the real anatomy with free-form deformation (see section 3.3.1). Only very few elements were used per muscle which represents the tradeoff of quality versus calculation time.

Other approaches to speed up the simulation have already been discussed and are generally based on pre-computing major parts of the solution. The most trivial case of pre-computed speed-up is a beforehand invertation of the stiffness matrix K.

$$K\vec{U} = \vec{F} \tag{3.10}$$

$$\vec{U} = K^{-1}\vec{F} \tag{3.11}$$

The right side of (3.11) can be carried out very fast. Therefore deformations of an object with a fixed topology can be calculated very rapidly for different force vectors. However, all deformations are calculated from the initial rest state of the object.

Non-linear FEM approaches were developed mainly for lungs and ventricles. Their full discussion is beyond the scope of this work; details can be found in [LL78], [PCRH78], [Vaw], [KBS80], [LTY83], [JG73], [JBM73], [CM82], [NRBM83], [HBHP90], [HvCAH91], [Tab91a] and [Tab91b].

3.1.3 Classification

A carefully conducted finite element analysis in soft tissue modeling minimizes the energy function given in (3.1). If the two energy terms W (3.2) and Λ (3.3) reflect the situation correctly, the simulation will produce physically correct results. The major problem of the above assumption is that the strain energy (3.3) generally can not be given correctly at all as it depends on the material law (2.11), which is extremely hard to determine for living soft tissues. In addition the generally made assumption of linear material response represents an idealization which might not always be true.

Nevertheless, keeping these limitations in mind, finite element analysis can be regarded as the most physical of all available techniques provided that the idealization of the real world system has been done physically correct. To be more precise, actually an FEM approach itself can neither be called *physical* nor *non-physical*, the terms must be referred to the formulation of the underlying problem. If the underlying problem was formulated physically correct, the model can be called physical in the sense of the definition given in section 2.2.

The problem of correctly formulating a valid material law is best illustrated by the varying material parameters presented in table 3.1. These values were all used in FEM models of the brain. According to the definition of a physical modeling technique (see 8) the parameters should correspond to physical properties of the brain tissue and it could be expected that they lie in the same order of magnitude; which is not the case. In chapter 6 a sensitivity analysis regarding biomechanical parameters in a brain simulation using finite elements is presented. While research mostly has to work with estimated or uncertain biomechanical parameters we should be aware of the fact that they significantly determine the outcome of the simulation.

Currently it seems reasonable to used FEM models for the simulation of problems where enough time is available to perform and solve a careful analysis. “High-speed” FEM approaches generally limit the physical realism already in the idealization step.

3.2 Mass Spring

Mass-spring models have been used extensively for the modeling of deformable bodies. The reason for the wide acceptance of the mass-spring approach is its

easiely understandable, physically based concept, its simplicity of implementation, its relatively low computational demands and its straight applicability to various problems.

3.2.1 Basic Concept

Mass-spring models are based on an idea originally introduced in classical mechanics. A body of a mass m can be represented by a single mass point that has no extension and holds the complete mass of the body. The concept can also be applied to systems of mass points as for example relevant for astronomical problems. The center of mass (CoM) of a system of n masses moves exactly as if the total mass $m = \sum_{i=1}^{n} m_i$ of the system was present in the CoM and all external forces were applied to it.

A continuous body can also be regarded as a system of distributed mass points. A body of a volume V and a mass m can be divided into n smaller sub-volumes ΔV_i with masses Δm_i.

$$V = \sum_{i}^{n} \Delta V_i \tag{3.12}$$

and

$$m = \sum_{i}^{n} \Delta m_i = \sum_{i}^{n} \frac{\Delta m_i}{\Delta V_i} \Delta V_i \tag{3.13}$$

For $\Delta V \to 0$, $\Delta m \to 0$ and $n \to \infty$ equations 3.12 and 3.13 lead to the definition of a continuous body. The ratio $\frac{\Delta m}{\Delta V}$ is then called *density* ρ of the body.

As long as the distances between all points stay constant the system of mass points reassembles a rigid configuration. If distances between points change, the Lagrangian strain tensor, introduced on page 11, can be used to measure the degree of displacement. As presented in section 2.3 any change of distances between points leads to stress within the body.

In mass-spring models a deformable body is approximated by a system of mass points that are distributed over the volume of the original body. The stress that originates from distance changes between points is modeled with springs that are affixed between the mass points. In principle there is no limitation to how the mass points can be connected. However, the most common way is to simply connect nearest neighbors. Also, often linear (Hookean) springs are used to connect mass points; the use of nonlinear springs, however, is also possible. When the mass point n is moved by an external force the springs attached to n are strained and forces are exerted on adjacent mass points that start moving accordingly. In such a dynamic system the movement of a mass point is governed by Newton's Second Law:

$$m_i \ddot{\vec{x}}_i = \vec{F}_i \tag{3.14}$$

with m_i the mass of the ith mass point, $\vec{x}_i \in \mathbf{R^3}$ its position and

$$\vec{F_i} = \sum_j^n \vec{g}_{ij} - d_i\dot{\vec{x}}_i + \vec{f_{ext_i}} \tag{3.15}$$

the sum of the effective forces. The first term in equ. (3.15) is the force on m_i exerted by the spring between mass i and mass j, the second term is the velocity dependant damping that affects m_i and the last term contains external forces acting on the mass point.

For Hookean springs the force $\vec{g}_{ij}$ is directly proportional to the deflection from rest position of the spring between i and j.

$$\sum_j^n \vec{g}_{ij} = \sum_j^n \underbrace{\frac{\vec{x}_j - \vec{x}_i}{||\vec{x}_j - \vec{x}_i||}}_{1} \Big(\underbrace{k_{ij}}_{2} \underbrace{\big(||\vec{x}_j - \vec{x}_i|| - R_{ij}\big)}_{3} \Big) \tag{3.16}$$

with :

1 : direction of the force

2 : spring constant

3 : deflection from restposition

As (3.15) represents the equation of motion for a single mass point, a system of n mass points is assembled from n such equations. The positions of the n mass points are arranged in one position vector, which is used to characterize the system. The system's position vector has $3n$ entries[8]. The matrices M, D and K with each $3n \times 3n$ entries can be used to represent mass, damping and stiffness. For a system of mass points equ. (3.15) becomes:

$$M\ddot{\vec{x}} + D\dot{\vec{x}} + K\vec{x} = \vec{F}_{ext} \tag{3.17}$$

The system is evolved forward in time by solving the following system of first order differential equations:

$$\dot{\vec{v}} = M^{-1}(-D\vec{v} - K\vec{x} + \vec{F}_{ext}) \tag{3.18}$$

$$\dot{\vec{x}} = \vec{v} \tag{3.19}$$

When evolving such a system in time the size of the time step used for integration on the one hand governs the system's stability and on the other determines the time require for solving; if large time steps are possible the solution is reached faster. However, it is obvious that the time steps should be chosen according to the expected accelerations and velocities of the system. Stiff spring constants generally result in high spring forces and large accelerations of the participating masses. In conjunction with large time steps such mass-spring systems tend to accumulate energy and become very unstable.

There are various integration techniques available in literature (see e.g. [Deu96] for a detailed study on integration techniques with mass-spring models).

[8] x, y, z for each mass point

3.2.2 *Mass-spring* in Soft Tissue Modeling

Mass-spring models were intensely used in computer graphics to animate facial expressions. In an early approach by Platt and Badler [PB81] the skin was represented as a tension net; nodes were connected with linear springs, skin nodes were also spring-connected to underlying bones. A tension net is a static version of a mass-spring model, basically it solves the non-time dependent system

$$K\vec{x} = \vec{F} \tag{3.20}$$

The face was modeled with tension nets warped around an ovoid and forces applied to certain nodes to generate the desired facial expressions.

While still using a static approach Waters [Wat87] improved the model by introducing vector springs following the major directions of real facial muscles. Besides the directed spring concept they also modeled zones of influence, areas of nodes, that contracted e.g. elliptically, parameterized by radius and fall-off coefficients. In [TW90], [TW91] Terzopoulos and Waters used a dynamic mass-spring approach for facial modeling. The facial skin was modeled with three different layers: skin, subcutaneous tissue and a muscle layer that is fixed to the bone. The different layers were assigned different stiffness parameters corresponding to the real tissue parameters. Incompressibility of particular skin layers or regions was modeled by introducing additional forces that were calculated and directed to preserve the volume of the relevant regions.

In [WT91] they used a more simple two layer skin model, but instead of a generic face the system could generate individual face models using data from a laser scanner. The scan provides a texture for the top layer of the model as well as the surface of the individual face. In a preprocessing step the node mesh was refined were the texture provides a high feature density. Waters used computer tomography data in [Wat92] to generate the 3D face models. In [LTW93] and [LTW95] a generic model was adapted to individual data captured with a laser scanner. Features were identified in the scan and matched to the appropriate features of the generic model. In addition constraints were introduced to avoid the three layers from penetrating the bone.

Wu et. al. [WBMT99] simulated skin aging and wrinkles in facial animation with a two layer skin model. A plastic-visco-elastic skin surface is connected via springs to the underlying layer, simulating the connective fat tissue between skin and muscles.

Koch et. al. [KGC+96] used the combination of FEM and mass-spring models for the prediction of cranio-facial surgery outcome as described above (see 20).

A summary on facial simulation models with a special focus on the needs of cranio facial surgery simulation can be found in [KGKG99]. In his presentation Keeve compares the results of a mass-spring and an FEM based approach on the same real surgical case. The study proves that facial soft tissue can be

simulated with a higher precision using FEM instead of mass-spring. However, the results received from the mass-spring simulaton may be accurate enough for most cases of pre-operative planning and provide the advantage of interactive simulation rates.

In [KKH+97] a mass-spring approach was employed to model different kinds of inner organs. In their endoscopic simulator Kuehnapfel et. al. modeled organs with nodal meshes of (virtual) mass nodes interconnected with springs. A subset of nodes on the organs' surfaces correspond to control points which are used to manipulate the graphical representation of the object. Later Kuehnapfel et. al. also incorporated FEM analysis in their system [KCM99] using the method of inverting the stiffness matrix K beforehand to speed up calculation time (see above page 23).

In [RNP00] and [ARW+99] a mass-spring approach is suggested for virtual tissue deformation which uses a neuro-fuzzy system to adapt the behavior of virtual tissue to that of real tissue by trimming the elasto mechanical properties of the mass-spring system. The fuzzy set and fuzzy rules that determine the tissue behavior can be obtained by interviewing experts. As fuzzy rules use linguistic terms to define specific characteristics this might be advantageous when working with medical professionals. In contrast to above aproaches Radetzky et. al. do not solve the system of differential equations that can be derived from the mass-spring model. Instead the simulation process is performed by the propagation function of a recurrent neuronal network. The deformation is limited to a local area by setting the recursive depth of the algorithm to n springs.

3.2.3 Classification

Equations (3.8) and (3.9) represent an FEM model and equations (3.20) and (3.17) a mass-spring model. The two sets of equations appear to be almost similar. However, they represent two very different matters.

An FEM model treats a deformable object as a continuum; a continueous solution is derived for each part of the problem's domain, while guaranteeing continuity over the whole object. In contrast, the mass-spring approach doesn't treat the object as a continuum. Instead of the geometric discretization of the problem used in FEM, the mass-spring approach uses a discretization of the problem itself. It idealizes an possibly continuous body as a discrete array of mass points which are connected via springs. With respect to figure 2.2 the major difference between the two approaches lies in the very first step of the *modeling pyramid*. FEM and mass-spring use different idealizations for the initially same problem.

At first glance it might seem easier to solve a given elasticity problem with a mass-spring approach than with a finite element analysis. The mass-spring approach itself specifies how the problem has to be idealized: "Divide the mass

of the body into discrete mass points and connect these via springs"[9]. However, once the object is represented by discrete masses and a suitable spring topology is found, the most difficult part in mass-spring modeling starts: the elasticity parameters for the springs have to be adjusted. In general these parameters will have no meassureable correspondent in the real problem as the idealization was not performed sufficiently correct.

A given mass-spring model's behavior is governed by the spring constants and the assigned masses. Tuning these parameters to meet the demands of the simulation, has become an art of its own. In section 3.2.2 an approach was described that used neuro-fuzzy logic to adapt a mass-spring model to realistic behavior [RNP00]. The cited example shows which efforts are taken to find eligible parameters. In addition, the model governing parameters heavily depend on the chosen topology. They generally have to be readapted if the object is represented by a different mass-spring mesh[10].

Where incompressibility is a "natural" property of biological tissue it is difficult to model volume conservation with a mass-spring mesh. Additional springs have to be inserted to take care of cross-contraction when the body is stressed in one direction. This results in complicated mesh topologies that are even more difficult to tune to the desired behavior. In addition computational costs rise because additional springs have to be calculated.

Since in general the spring constants don't correspond to the measurable biomechanical parameters of the problem, the mass-spring approach is not a physical modeling technique in the sense of the definition given in section 2.2. However, the basic idea of the mass-spring approach is definitely physical and if the real-world process was a mesh of masses connected with springs it would of course model the situation physically correct. Problems emerge from the desire of mapping any given elasticity problem onto the easy understandable and manageable mass-spring approach while ignoring that the approach was initialy designed for a different physical problem class. Depending on the similarity of *what should be modeled* and *what is modeled* with a mass-spring system, the resultant model is more or less physically realistic. Representing a muscle fiber with a spring might be a valid simplification and result in a model with spring constants similar or close to the measurable biomechanical properties. However, when dealing with inhomogeneous volumetric objects, like e.g. the brain, it is very questionable whether the model governing parameters are still in accordance with what is measurable in reality after creating a mass-point-spring topology and tuning the spring constants to match the behavior of the real world process.

[9] In contrast, the idealization step in FEM can be very hard as it includes finding an appropriate material law.

[10] In [Deu96] the effect of different mass-point distributions and mesh topologies is investigated.

3.3 Other Modeling Techniques

3.3.1 Free Form Deformation (FFD)

In contrast to the modeling approaches described above, free form deformation has its roots in computer graphics. It was initially developed to intuitively deform graphical objects e.g. for animations. Free form deformation can efficiently be applied to several, very common, graphical representations such as points, polygons, splines, parametric patches and implicit surfaces.

Basic Concept

Early work was presented by Barr [Bar84]. He used an analytical description for a geometric mapping from $R^3 \rightarrow R^3$ which results in the deformation of graphical objects with parametrical representations. For example twisting an object around the z-axis can be described analytically by:

$$f(\vec{p}) = \begin{bmatrix} \cos(p_z) & -\sin(p_z) & 0 \\ \sin(p_z) & \cos(p_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{3.21}$$

Barr's technique remained restricted to a couple of such deformations. The application of these deformations is not very intuitive. A desired shape change of an object can only by achieved by trial and error, since no mapping of desired change to required transformation has been found.

Picking up the idea of using a parametrical graphical object representation and deforming the object by mapping the control points of the representation to a different location in space led Sederberg and Parry to what they named *free form deformation* [SP86]. Instead of deforming the object directly, their new idea was to embed the object in a space that is to be deformed. The shape of the embedded object changes in accordance with the space deformation. Sederberg suggested to think of space as a block of transparent plastic in which the object is fused-in. A deformation of the plastic results in an analogous shape change of the embedded object. Sederberg claimed that object deformations achieved in that way is "intuitively correct" [WW92].

The space in which an object is embedded, can mathematically be described with hyperpatches. The control points of the hyperpatches are used to control the space deformation. The embedded object is represented in a control point or vertex based description. If the positions of the object's vertices are known in the lattice coordinates of the hypertpatch before the deformation, the points can be set to their corresponding positions after the lattice deformation. A typical FFD (see figure 3.1[11]) is performed in the following manner [WW92]:

[11] `http://www.cs.unc.edu/~hoff/projects/comp239/finalproj/snapshots2d/index.html`

1. *Determine the position of the vertices in lattice space:* A lattice space triple (s, t, u) is assigned to each vertex which stays with it unchanged throughout the deformation.

2. *Deform the FFD block:* The FFD block is e.g. represented by a hyperpatch. Deformation is achieved by moving the control points of the hyperpatch from their undisplaced lattice positions.

3. *Determine the deformed positions of the object's vertices:* After deformation lattice space and object space are different. A transition can be given by using the (s, t, u) triple assigned to each vertex to the local coordinate system of the appropriate hyperpatch and use these local hyperpatch coordinates in the hyperpatch's defining equation to calculate its new position; of the deformed object.

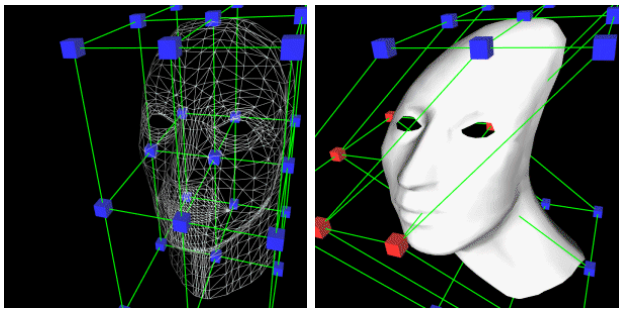

Figure 3.1: Example of a free form deformation. The red nodes of the space lattice were moved. (Free FormDeformer Snapshots by Kenny Hoff 4/28/97)

The original algorithm has been extended and modified by several others. Coquillart and Jancene [CJ91] introduced an animation technique based on two FFD blocks; one corresponding to the desired initial and one corresponding to the final state of the animation. The parameter sets for the states between beginning and end state were interpolated with a key-frame technique. As exact shape changes are difficult to achieve by moving control points Hsu et. al. [HHK92] suggested a direct manipulation technique (DFFD). Their method allows the direct manipulation of object surface points by converting the desired movement of object points to equivalent lattice control point movements. MacCracken and Joy introduced space lattices with arbitrary topology. The shape of the lattice governs the number of possible deformations. Therefore it is desirable to match the lattice space to the object's shape. In [MJ96] they presented a technique that uses a subdivision methodology to successively refine a 3-dimensional lattice into a sequence of lattices that converge to a specific region of 3d-space.

FFD in Soft Tissue Modeling

Yamashita et. al. used an adaption of Hsu's DFFD in their virtual reality environment called "ViSurf" to model the interaction between a user manipulated tool and a deformable surface [YYFS94]. They also included force-feedback into their simulation.

Cover et. al. implemented a combination of FFD and active surface approach (section 3.3.2) for their gall-bladder surgery simulator. They claim that Hsu's approach met their demand in allowing a direct object manipulation, but was too slow to provide real-time interaction [CEO+93].

Moccozet and Thalmann employed a generalized FFD model based on the Dirichlet FFD introduced by Farin [Far90] for the animation of a human hand. They modeled morphological variations of different hands with their approach and interactively simulated different types of hand activities [MT97].

Classification

FFD is a classical technique for soft object animation. It was initially developed to simplify and improve the process of creating computer animations. In contrast to the two methods presented previously (FEM and mass-spring), FFD is a *displacement based* method. Whereas in FEM and mass-spring simulations forces are applied to certain nodes which then results in a displacement (*force-based* method), in FFD the desired displacement is directly applied to the object. There is no physical analogy to the FFD method. The objects behavior is parameterized in a consistent way with weighted mathematical functions. The coefficients of the mathematical functions govern the model's behavior and have no real-world-counterpart. FFD is a descriptive modeling technique.

3.3.2 Deformable Models

The term deformable model summarizes several techniques of which the concept suggested by Kass et. al. called "snakes"[KWT88] is certainly best known. Others "Active Contour" models are "Active Surfaces", "Active Cubes" (e.g. [BN95]) or "Active Blobs" (e.g. [Whi94a]). All these techniques have a physical interpretation in common: a given contour is regarded as an elastic body that responds to applied forces. Deformable models are widely used in image processing for the identification and segmentation of image features.

Basic Concept

Typically, an energy function is defined in terms of the geometric degrees of freedom that are associated with the deformable model. The energy grows

as the model deforms from its "rest shape". Often additional "energy"-terms are included that constrain the smoothness of the model. Adding an external potential energy function deforms the model from its initial state. The external potential is generally defined in a way so that the model fits itself to the desired shape when the potential energy of the model is minimized[MT96]. In image processing such external potentials are assigned to the interesting image features.

Deformable Models in Soft Tissue Modeling

Where "snakes" and its derivates are widely used in image processing, only few people applied the technique to biomechanical simulations. The reason for this is probably that arbitrary shape changes always require to find a corresponding external potential. These potentials can not be defined in an intuitive way.

Cover et. al. [CEO+93] suggested a combination of elements of both free-form deformation and energy-minimizing surfaces for their deformable model. They simulated user interaction with the gall-bladder in their laparoscopic surgery simulator including cutting of tissue.

Bro-Nielsen [BN95] suggested "active cubes", which is a straight forward three-dimensional extension of the snake idea. In a CT scan the jaw is moved and the adjacent soft tissue is deformed accordingly. In the three-dimensional CT data set labeled bone voxels are moved to their desired new position and fixed. The system is then relaxed into the energy minimum.

Classification

Where the terminology used in deformable models is clearly physical, its use in biomechanical modeling is to be regarded as descriptive. There is no mapping of the model governing parameters to the real physical properties.

3.3.3 Implicit Surfaces

An implicit surface is defined as a set of points P satisfying an equation e.g. $f(P) = 0$. Such an equation results in what is also known as an "iso-surface". If two fairly continuous implicit surfaces $f(x, y, z) = 0$ and $g(x, y, z) = 0$ are given and the two functions have a common sign convention (e.g. positive on the inside, negative on the outside) then the implicit surface defined by f+g=0 is a blend of the shapes.

Basic Concept

The van der Waals potentials of atomic bond forces known from physics and chemistry led Blinn to his so called "blobby models" [Bli82]. He used a superposition of Gaussian potentials to define a surface for an animation of DNA. The same technique is called "metaballs" mainly in the pacific region where it is still very popular and has been further developed extensively. A metaball [NHK+85] is a 3-D modeling primitive that is expressed as a distribution function of a point charge in 3-D space. The surface of the metaball is defined as an isosurface where voltage equals a user defined threshold. The area where voltage is greater than the given threshold is defined to be inside of the metaball. With CSG (Constructive Solid Geometry)[12] the metaball primitives can be combined to complex shapes. An example for constructive and destructive superposition of two metaballs is given in figure 3.2.

Figure 3.2: Left to right show constructive superposition and destructive superposition of two metaballs and a sequence of two metaballs approaching eachother.

Bloomenthal discussed techniques for modeling organic forms (trees, leaves, arms) using "implicit modeling" [BW90], [BS91]. He also suggested algorithms for the polygonization of any implicit model, which is very important for the fast rendering of the model [Blo88][13].

Brian and Geoff Wyvill called their implicit surface models "soft objects" and contributed e.g. approximations of field potentials and how to accomplish animations with implicit surfaces [WMW86], [WW89].

Implicit Surfaces in Soft Tissue Modeling

Metaball simulations are typically used in body modeling and deformation[BPW93]. For example D. Thalmann et. al. used Metaballs for constructing and animating realistic human bodies. The Metaballs are used to simulate the gross behavior of bone, muscle, and fat tissue [JT95]. Metaballs are attached to the proximal joints of the skeleton, arranged in an anatomically-based approximation. In their "Bodybuilder", a software module which is designed to allow the user to perform simulation of the human body envelope, N. Thalmann et. al. also use a hierarchy of metaballs which are attached or connected to joint articulations of a skeleton [TK95].

[12] A modeling technique which organizes primitive solids as nodes in a tree of which the root is the desired object and the edges are boolean operators.

[13] The better known "marching cubes" algorithm, initially suggested in [LC87], is a less general variant of Bloomenthal's algorithm.

Classification

Metaballs or Blobs were first suggested for the simulation and visualization of molecules. In this context the model can be regarded as physical, even though using a Gaussian instead of Yukawa potentials is an approximation.

Later, Metaballs were used for character animation and object deformation. The use of Metaballs in object deformation has nothing to do with the initially physical concept of electrical potentials.

Implicit surfaces are a descriptive modeling technique.

3.3.4 Particle Models

Particle models deal with systems of mass points as presented in section 3.2 above. A set of physical rules is applied on each particle which governs the particle's behavior. A whole system of particles represents the deformable object.

Basic Concept

Particle models typically deal with lightweight objects, which have attributes like position, velocity, mass, charge, etc. The aggregation of particles defines the shape of the desired object. Forces can act upon the particles and change their distribution, i.e. the shape of the object. Generally there are global forces which only depend on the particle itself, like gravitation[14] or wind and inter-particle forces like real bi-directional gravitation and repulsion.

Particle systems were used in computer graphics to model natural phenomena such as fire and waterfalls [Ree83], [Sim90]. These first models moved the particles while forces and constraints acted on them, but did not consider particle interaction. Later models used spherically symmetric potential fields to take inter-particle influence into account [TPF89], [Ton91].

Szeliski and Tonnesen [ST92] suggested the use of oriented particles to model the surface of the object while previous particle models formed volumetric objects. They achieved the forming of surfaces by designing special interaction potentials which favor locally planar or locally spherical particle arrangements. To control the average inter-particle spacing they introduced long-range attraction forces and short-range repulsion forces as typically caused by a Lennard-Jones potential.

Desburn and Gascuel developed a method in [DG94] that combined particle systems and implicit surfaces for the animation of highly deformable material.

[14] Gravitation acts between earth and particle. Due to the mass difference $m_{earth} \gg m_{particle}$ the force acting from particle on earth is generally neglected.

In [DG95] they presented enhancements to the method. The basic model consists of a set of mass points P_i subject to attraction/repulsion forces F_{int} and fluid friction forces F_{fr} which depend on the local particle density. The forces applied by particle P_1 on P_2 are:

$$F_{int}(P_1 \rightarrow P_2) = \lambda \left(\left(\frac{r_0}{r} \right)^8 - \left(\frac{r_0}{r} \right)^4 \right) \frac{P_2 - P_1}{r^2} \tag{3.22}$$

$$F_{fr}((P_1 \rightarrow P_2) = \mu(r) \| \dot{P_1} - \dot{P_2} \| (\dot{P_1} - \dot{P_2}) \tag{3.23}$$

where $r = \|P_2 - P_1\|$ is the distance between the two particles, λ is a parameter for regulating the stiffness of the material and μ is a decreasing continuous function with a restricted scope of influence. Each particle has an associated field f_i which is a decreasing function of the distance. The surface of the object generated by the set of particles is defined as an implicit surface:

$$f(P) = \sum f_i(P) = s \tag{3.24}$$

with s being a given isovalue. In case of a collision, the contact surface is calculated based on the field potentials of the participating particles. In the next iteration forces based on the contact surface are generated and considered in the particle model. The method seems to be well suited for the animation of fluids or visco-elastic materials. It allows the collision and the fusion and separation of material pieces.

Particle Models in Soft Tissue Modeling

Particle models are typically used for the modeling of gases, fluids, fire, etc.. Wu et. al. implemented a deformable surface model using particles and demonstrated it with several simulation examples: the animation of wrinkles on the forehead-skin, the deformation of a head, a membrane shrinking around a given shape and an elastoplastic surface [WTT95].

Classification

A particle model simulating particle with the set of rules that corresponds to the true physical laws could certainly be regarded as physical. Again the main question is: What is the nature of the problem that was modeled with the approach. Particle models seem to be suited for the simulation of visco-elastic to fluid materials. That is because the nature of fluids is sometimes close to what is modeled with a particle model.

The use of particle models in soft tissue modeling, or other solid, elastic objects must be regarded as descriptive.

3.3.5 Free-sampling

Free-sampling, as suggested in [SRG+97], picks up the idea of deforming space to deform the objects embedded in the space just the way FFD (presented in section 3.3.1) does. The difference between FFD and free-sampling is that the latter is tailored to the needs of raytracing as e.g. used in volume rendering of medical data sets.

Basic Concept

In volume visualization, the path of a straight beam of light is followed through a data set. Transmission, absorption and reflection are calculated at each voxel and the cumulated reflected light is projected back to the viewing plane. In free-sampling the path of the sampling beam passing through the data volume is no longer straight, which results in a deformed visualization of the data set (see figure 3.3). There are various ways of deforming a data set via the

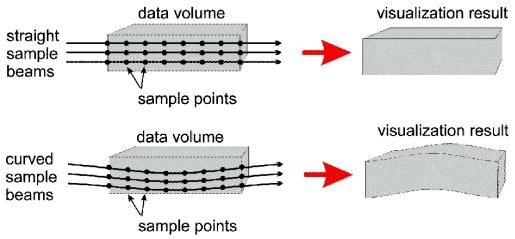

Figure 3.3: In the raytracing process the sampling beam does not necessarily have to be straight. If sampled along a bend path the result will be a deformed data set.

modification of the sample paths. Smooth bending of the sample path also bends the object smoothly. Discontinuous sample paths can result in cut open objects.

Free-sampling does not release the user from defining the desired deformation, as it requires a function or otherwise definition of the sampling paths. Figure 3.4 illustrates how the sampling paths can be included into the volume rendering process. An additional data volume contains the sample-offset information. Instead of the voxel the sampling ray would hit on a straight path through the volume, it uses the voxel that is found at the position given by the corresponding entry in the sample-offset information data set.

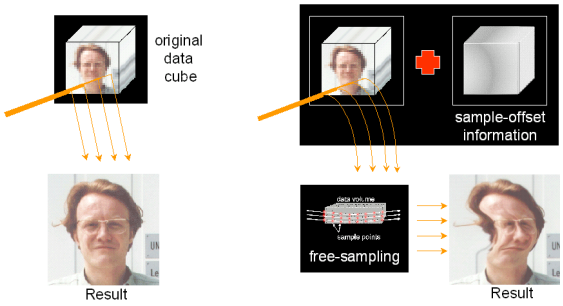

Figure 3.4: Illustration of the normal volume rendering process on the left and volume rendering with free-sampling on the right. Instead of the voxel the sampling ray would hit on a straight path through the volume, it uses the voxel that is found at the position given by the corresponding entry in the sample-offset information data set.

Free-sampling in Soft Tissue Modeling

Free-sampling was used in [SRG+97] to deform an MRI data set of a brain according to a pre-calculated finite element simulation. The deformation of the Falx cerebri, a strong septum in the brain dividing the brain's hemispheres, was calculated under various pressure gradients in [SSBM96]. The simulation results were then used for a manipulation of the original MRI data set to render realistic images of the simulation results (see figure 3.5).

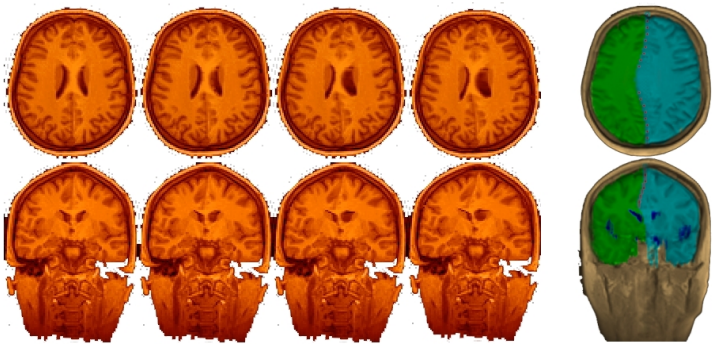

Figure 3.5: Sequence of free-sampled deformations of the brain based on finite element simulation of the Falx cerebri. The left pictures show the original data with a centered Falx cerebri; to the right the Falx and the surrounding brain tissue is deformed with free sampling according to finite element calculations.

Classification

As free-sampling is a method for the deformation of volume data sets that requires external information on how to deform the object, the classification depends on how the deformation (offset- sample-information) is aquired. Generally free-sampling is bare of any physical deformation calculation and therefore regarded as being a descriptive method.

3.3.6 Transmission Line Model

Transmission line models were devised in the 1970s to model problems in electromagnetics but have since then also been transferred to problems in acoustics and heat-transfer. The technique is only barely known in the biomechanical modeling community although its use for elastic deformation was suggested before [Bos92], [LWP94].

Basic Concept

The Transmission Line method calculates the path of information pulses through a grid of inter-connected nodes. At each node the information is re-emitted. The information pulse takes one time step to travel a link between nodes. The links are assigned a certain impedance which affects the path an information pulse takes. At each node the pulse is partially reflected. The remaining pulse is distributed along the other connections in a ratio according to the impedance of the link. There are also special link types called "stub lines" which are not connected to nodes and only reflect the pulse back to where it came from. "Stub lines" can be used to control the propagation speed of an information pulse. A known problem with TLM is that spreading out of the wavefront can generate dispersion errors. In [LWP94] two- and three-dimensional stress and deformation waves were propagated through very simple nodal structures. Very generally spoken, TLM deals with wave front propagation (see figure 3.6).

TLM in Soft Tissue Modeling

Unknown.

Classification

In [LWP94] a deformation and stress analysis was performed for a two- and three dimensional object using TLM. In [LWP94] the true physical equations describing the propagation of an elastic wave through an object is used. The

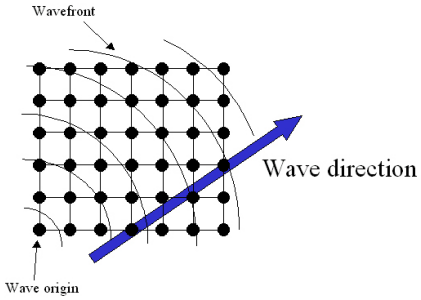

Figure 3.6: Wavefront propagation through a TLM mesh. A wave can "carry" different information, e.g. stress- or deformation information.

problem, however, is separated and solved on several transmission line networks which are superposed for the solution. The authors claim that they numerically compared their results to solutions achieved with a finite difference method. TLM, similar to FEM, is primarily a method to solve, or better approximate the solution of a particular problem class. Too few work has been done so far to decide whether biomechanical problems can be mapped to a mathematical formulation suited for TLM.

3.4 Summary

The most common techniques currently used in biomechanical modeling and computer graphics were introduced in chapter 3. Best known and most widely spread in biomechanical modeling are FEM and mass-spring approaches. All discussed techniques were presented in the context of biomechanical modeling together with a detailed overview over the state of the art of their application in biomechanical modeling.

The presented modeling approaches were also classified according to a scheme (introduced in section 2.2) opposing physical and descriptive approaches. The attempt to classify each approach showed, that in biomechanical modeling, techniques with initially strictly physical concept are often used in a descriptive way. This is mainly due to the fact that physical models are very costly. E.g. FEM yields computationally demanding systems of equations. To cut down computational costs simplifications of the proper approach have been proposed. Very common is limiting the approach to static or quasi-static problems and thus eliminating the time dependency (see section 3.1.1, [CDBN+96], [Del98]). Static simulations are generally faster than dynamic, since no integration over time is necessary. The other most commonly made assumption is that of linear elasticity. This significantly reduces the complexity of the material

laws and herewith simplifies the governing equations (see section 2.3). While most of the described simplifications allow a faster solution of the problem, they shift the approach to the descriptive side of the introduced scheme. The shift to the descriptive side of the scheme is even more serious with mass-spring than with FEM models. In most mass-spring approaches the "physical" parameters that govern the model's behavior are set highhanded or adjusted by very sophisticated proprietary algorithms [RNP00]. The used parameters can no longer be reproduced by measurement at the modeled real-world system, which is required by the definition of a physical model.

Similar developments can be observed with other techniques. Metaballs e.g. were first used to simulate and visualize molecules. Later the technique was applied to object deformation. The evolution of Metaballs, can be compared to what happened to mass-spring models. Both approaches were first utilized to simulate what they physically represent, namely atomic potentials (Metaballs) and beam-structures (mass-spring). Both models were capable and well suited to approximate the modeled problems. Due to their well understandable concept, their idea was then transferred to other problem classes. In these other contexts their use is still intuitive but the parameters used, shape of potentials (Metaballs) and masses and spring constants (mass-spring), no longer correspond to true physical properties.

Already the first step of the modeling process, the idealization of the real-world process, represents the first approximation. Other approximations follow, like e.g. discretization – whether it is the discretization of a continuous model or the discretization of the problem – or numerical approximations.

On the other side of the spectrum of available modeling techniques, initially descriptive models are being more and more enhanced. Descriptive models are up-valued by introducing more physics into their concepts. Such a development is triggered by the constantly increasing computational power available. The Enhanced ChainMail algorithm which is introduced in the next chapter e.g. combines a descriptive kinematic model with a physical dynamic model, an elastic relaxation process.

It is important to note that good models can be designed with either technique no matter whether it is physical or descriptive. In general the validation and the portability to other problems might be easier for physical models. The gap between physical and descriptive models will narrow in the future.

4

Enhanced ChainMail

Chapter 4 introduces a novel algorithm for soft tissue modeling, named ChainMail. ChainMail was first introduced by Gibson in [Gib97a]. The algorithm was significantly enhanced in the scope of this work. The initial concept lacked the capability to model inhomogeneous objects, which is a pre-requisite for its use in biomechanical modeling. This capability is provided by the new Enhanced ChainMail algorithm. Chapter 4 introduces the ChainMail algorithm as first introduced and then explains how the concept of the original approach had to be changed to allow the modeling of inhomogeneous materials.

The ChainMail algorithm very rapidly approximates the final shape of a deformable object by exploiting simple rules between neighboring elements. As it might not always result in a deformed object with a homogeneous distribution of elastic energy, it should be the first step in a two step process. The second step should be an elastic relaxation. The relaxation will provide fine adjustments of the object's shape and grant a valid energy distribution over the object.

4.1 ChainMail

ChainMail works with one-, two- and three-dimensional objects. Objects are represented as connected neighborhoods. In three dimensions each element is linked to its six nearest neighbors. ChainMail is based on geometric constraints. During the deformation process each element tries to satisfy given maximum and minimum distance constraints to its neighboring elements. The movement of an element depends only on the position of its nearest neighbors. Because of the similarity of the algorithm to a set of linked chains it has been dubbed ChainMail (see figure 4.1).

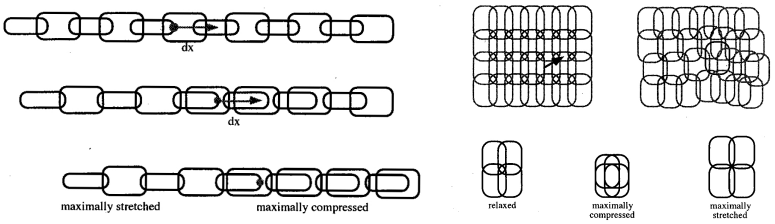

Figure 4.1: Illustrations from the original ChainMail publication [Gib97a]: Deformation of a 1D chain when the selected link is moved to the right by dx (left). Deformation of 2D ChainMail when the selected link is moved (right).

When one element (named *initial sponsor* in the following) is moved while the whole structure is in a relaxed state, the links between the initial sponsor and its neighbors can first compensate the motion in their links. Once the movement becomes too large so that a link is maximally stretched or compressed a displacement of the respective neighboring element takes place. In an iterative process the initial deformation is propagated through the structure. By changing the constraints of the links that connect the elements it is possible to model softer or more rigid objects.

4.1.1 Outline of the Original ChainMail Algorithm

The degree of deformation in which an object reacts to the displacement of one of its nodes is governed by the constraints that were a priori assigned to the connections between the nodes. Each horizontal link has a minimum and maximum allowed length; $minDx$ and $maxDx$. In addition two nodes connected by a horizontal link may only have a defined shear $\pm maxHorzDx$ against each other. Analogous constraints are valid for vertically connected node, with $minDy$ and $maxDy$ being the link length and $\pm maxVertDy$ the shear. Figure 4.2 illustrates the constraints. Once defined, these constraints are valid for all links within the object, which result in the model of an isotropic and homogeneous body. In three dimensions an additional triplet of constraints has to be considered. For simplicity the algorithm is illustrated in 2D in the following. The extension to three dimensions, however, is straightforward.

In two dimensions four lists are maintained in which the elements that are to be considered for a move (*candidates for a move*) are stored. The lists (top, bottom, left, right) classify the elements regarding which neighbor they have to be checked against.

When an element is moved by the user (initial sponsor), its position is updated accordingly and its four nearest neighbors (candidates for a move) are added

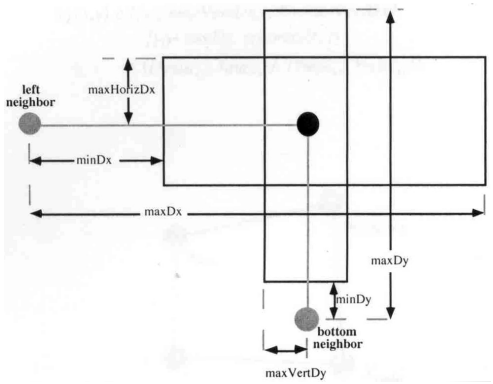

Figure 4.2: Illustrations from the original ChainMail publication [Gib97a]: The regions in which the element (black dot) can move relative to its left and bottom neighbor.

to the appropriate lists. Then a process is started which runs until all lists are exhausted:

1. The first element of the right candidates list is checked against its left neighbor and moved as far as necessary to satisfy the stretch and shear constraints illustrated in figure 4.2. The new position is given by:

   (a) Stretch constraint:
   $if\ (x - x_{left}) < minDx,\quad x = x_{left} + minDx;$
   $else\ if\ (x - x_{left}) > maxDx,\quad x = x_{left} + maxDx;$

   (b) Shear constraint:
   $if\ (y - y_{left}) < -maxHorizDy,\quad y = y_{left} - maxHorizDy;$
   $else\ if\ (y - y_{left}) > maxHorzDy,\quad y = y_{left} + maxHorzDy;$

2. If the previous step required a move of the element, its neighbors (top, bottom and right) are added to the respective candidate lists. The left neighbor is not added as the position was just adjusted to meet the constraints of this link.

3. Process right candidates list in the described way until the list is empty.

4. Process left candidates list in the corresponding way until it is empty. Bottom top and left neighbors have to be added to the appropriate candidates lists.

5. Process top candidates list. Only top neighbors have to be added to top list.

6. Process the bottom candidates list. Only bottom neighbors have to be added to bottom list.

Following the suggested list processing sequence grants that a minimum number of elements has to be processed until the algorithm reach its end. Changing the order, however, has no effect on the final shape of the object.

4.1.2 Discussion

The algorithm as presented is tailored to the deformation of initially convex bodies. Each element is processed only once. In case of non-convex objects it might be necessary to add and process elements which have a reduced number of neighbors, due to the concave shape, more than once. This might also imply that candidate lists have to be revisited to guarantee that they will be exhausted during the process[1]

Figure 4.3 shows the result of the deformation of a small setup (6×6 nodes) . The red arrow indicates the point and direction of the current interaction. The sequence shows single frames of an animation; each image represents a final ChainMail simulation step.

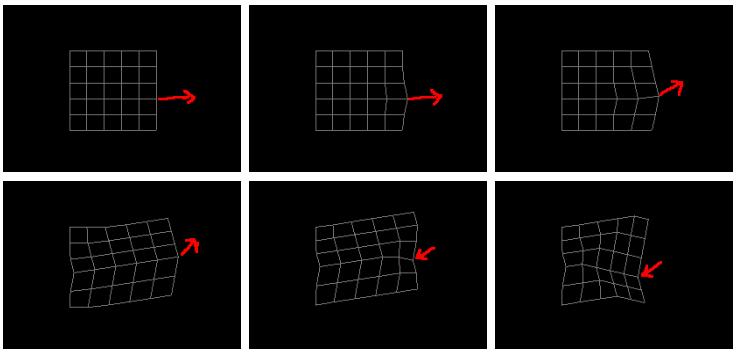

Figure 4.3: The sequence shows single frames of an animation; each image represents a final ChainMail simulation step. The red arrow indicates the current point and direction of interaction.

Figures 4.4 illustrates the iterations performed for a single simulation step of the original ChainMail algorithm. The iteration begins, when the initial sponsor is moved and it ends when all candidate lists are exhausted. In the example the initial sponsor is moved very far to the right in an initial step.

[1] A proof that each element is only processed once can be found in [Gib97b]. Since for the Enhanced ChainMail algorithm no limitation regarding the shape of the object exist the shape issue is not addressed here.

Mostly ChainMail will not leave the object in an optimal condition regarding the energy distribution within the body. Therefore an elastic relaxation process is started in a second step. It reduces an energy measure in the deformed object by locally adjusting relative element positions. The object's energy is given by a function that depends on the distances between the elements. This function can also be used to model different material properties like elasticity or plasticity.

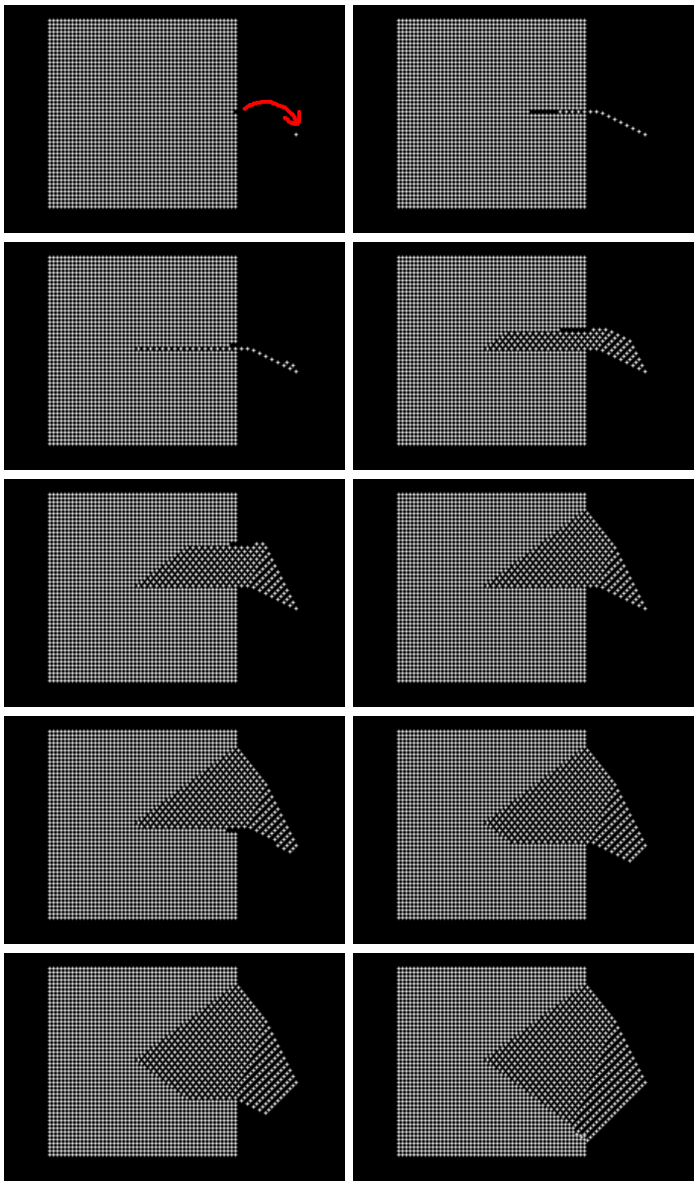

Figure 4.4: Original ChainMail on a 50x50 element matrix. The image shows a sequence of iteration steps of a single ChainMail simulation step; in a real simulation only the initial state (image before the first) and the last image would be visible. The element in the middle of the most right column is moved away to the right. According to the original ChainMail algorithm, the left/right neighbors lists are processed first and then the top and bottom neighbors. This is very well noticeable in the sequence. In the first image of the second row the left/right lists are already exhausted and the top list is processed. Top list is empty with the last image of row three and the bottom list is processed.

4.2 The Enhanced ChainMail Algorithm (ECM)

Enhanced ChainMail is a further development of the original ChainMail idea which lacked the capability to process inhomogeneous materials. As modeling inhomogeneous material is essential for biomechanical simulations, the term *Enhanced* was added to the algorithm's original name. However, ECM actually represents a generalization of the original ChainMail idea.

This section 4.2 provides a different, more general, view on the process described with ChainMail. A physical interpretation of the algorithm is given and then used to derive the principle that allows modeling of inhomogeneity.

4.2.1 Basic Idea

Applying ChainMail to an inhomogeneous object, i.e. link constraints that vary over the object, fails. Figure 4.5 shows how an object is torn apart if the original ChainMail algorithm is applied on an inhomogeneous object. The result is

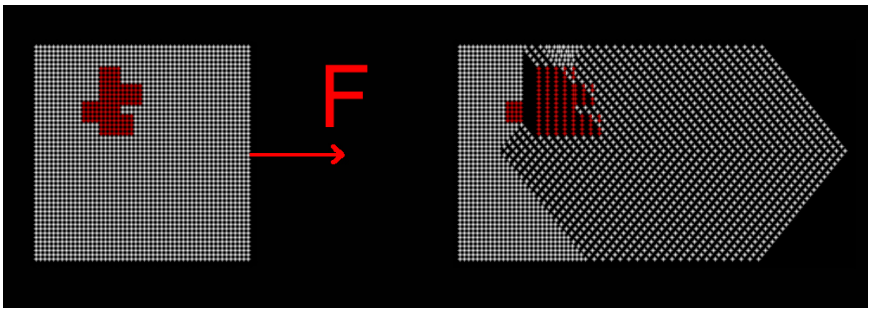

Figure 4.5: Applying the original ChainMail algorithm to an inhomogeneous object fails. The test object is the four connected neighborhood shown on the left. The material constraints are constant over the object except for the red t-shaped structure which is completely rigid. The right image shows the final state of the deformation with original ChainMail. The t-shape is torn apart by the algorithm. The link lengths between neighboring nodes within the red shape and at its right border are also false.

the image on the right of figure 4.5 showing a gap in the rigid red t-shape. The tearing apart of the structure which was supposed to be completely rigid, seems to suggest that the process simply finished too early. Therefore either always adding all neighbors of a moved element, changing the order in which the lists are processed or/and visiting the lists multiple times also after they were exhausted, should produce correct results. None of these straightforward

modifications nor a combination of them solves the problem. They either render unrealistic shapes or stick in an infinite loop where parts of the object are oscillated back and forth.

Seeking the basic principle behind ChainMail leads to understanding the algorithm as a method of transporting information through a mesh of interconnected nodes. However, the information needs to be transported to the appropriate nodes in the "correct" order; the nodes must be processed in the "correct sequence". The chronology in which information is spread through the mesh is important because each node[2] may alter the information it receives before passing a modified message on to its neighbors. Therefore the principle of causality must be respected. Considering physical behavior – the information propagated through the mesh could be the information of deformation – the correct order in which nodes have to be considered for a move is given by the order in which nodes condition their movement. Figure 4.6 shows the propagation of deformation information through the trivial case of a one dimensional chain. As the information that a deformation occurred, travels through the object, each element reacts by adjusting its own position. The distance each element moves is given by the material constraints. By adjusting its position, each element implicitly changes the information (size of the deformation) that is passed on to its neighbor. The process ends, when the deformation information is reduced to zero or no neighbors are left to pass the information to. The correct order which guarantees causality is very easy to comprehend in the one dimensional case.

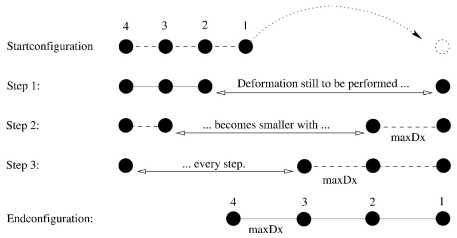

Figure 4.6: The deformation is initiated by moving element #1. In the next step, element #2 implicitly changes the information that is passed on to element #3 by adjusting its position according to the material constraint, maxDx, where maxDx is the maximum allowed distance between two elements. During the process the "Deformation still to be performed" becomes smaller with every step.

What is the "correct" way of propagating information through an arbitrary object? ChainMail can be regarded as a physical process in which deformation information travels through the material, just like a sound wave. In general, the

[2]respectively the link between two nodes

speed of sound is higher in material with a tight coupling between adjacent elements. In ChainMail the "stiffness" of a connection between two neighbors can be determined by the material properties of the two involved elements. In inhomogeneous materials, when neighboring elements can have different material constraints, the constraints for the link between two neighbors can be calculated using contributions from both elements. Figure 4.7 shows the constraints that are assigned to a single element in the Enhanced ChainMail algorithm and figure 4.8 illustrates how the two neighbors contribute to the link constraints that govern their relative behavior.

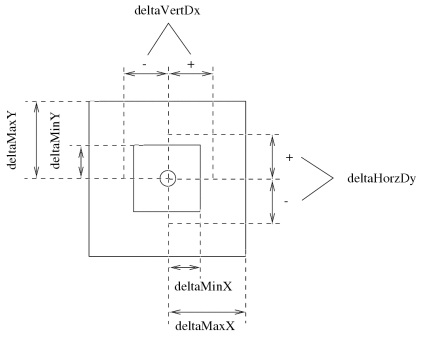

Figure 4.7: Constraints assigned to a single element in the Enhanced ChainMail Algorithm. $deltaMaxX$, $deltaMinX$, $deltaMaxY$ and $deltaMinY$ correspond to maximum allowed stretch and compression in the horizontal and vertical directions respectively. $\pm deltaVertX$ and $\pm deltaHorzY$ correspond to the maximum allowed shear to vertical and horizontal neighbors respectively.

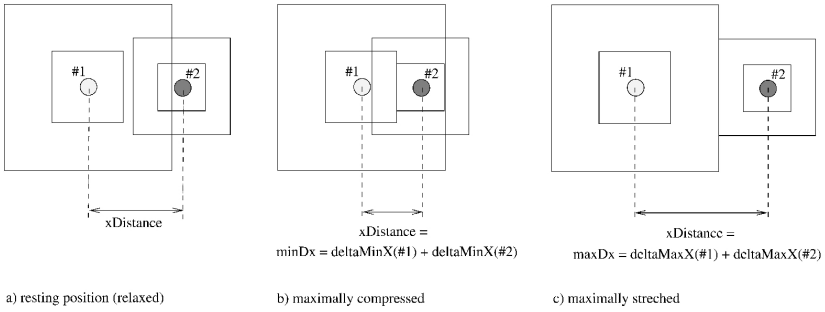

Figure 4.8: The contributions of both neighbors combine to give the constraints of the link between them.

The order in which neighbors of a moved element are processed determines to where the deformation information is propagated first. Hence, the process-

ing order can be used to model material and direction-dependent propagation speeds in the object. The ECM algorithm uses this observation to order the processing of elements so that the deformation travels most quickly through stiff materials. Such a processing order can be obtained by comparing the constraint violations between neighboring elements and always process the element with the largest constraint violation first. For example, the constraint violation for element #2 in step 1 of Fig. 4.6 is determined as follows:

$$amount\ of\ constraint\ violation = distance_{\#1,\#2} - maxDx$$

After moving an element, its neighbors are examined to investigate whether any constraints have been violated. If so, the neighbor with the largest constraint violation is processed next. This is equivalent to processing neighbors in the order in which their constraints were violated, guaranteeing that the deformation information is propagated fastest through stiff materials.

4.2.2 Outline of the ECM Algorithm

The Enhanced ChainMail Algorithm maintains one ordered list where all elements that must be considered for a movement are stored. The ordering criteria for the list is the size of the constraint violation.

The process starts when an element – the initial sponsor – is moved to a new position. The neighboring elements are inserted into the list, whereby the above ordering principle must be maintained, i.e. the element with the largest constraint violation is always at the top of the list. Then the first element is popped from the list and moved to satisfy the constraints against its sponsor (the element's sponsor is the element which had caused that the element was added to the list). Moving an element to satisfy constraints against its sponsor may again make an adding of elements to the list necessary. The list grows when an element is moved and its neighbors are added, it becomes smaller with each element that is popped from the top of the list and processed. The algorithm ends when the list is exhausted. The following is pseudo code for the Enhanced ChainMail Algorithm:

```
initialSponsor.position = newPosition ;
insertNeighborsToList(initialSponsor) ;
// the above function maintains the ordering principle
while (elements in list)
   popFirstElementFromList ;
   if(checkVsSponsor demands move)
      thisElement.updatePosition ;
      addNeighborsToList(thisElement) ;
```

It is intuitively understandable that the described process will terminate. The constraint violation corresponds to the amount of deformation that still has

to be performed by the object. As the algorithm works from large to small violations it always decreases the deformation that is still to be considered. Generally – however – one element is added more than once to the list of candidates.

The candidate list can be sorted easily if new elements are inserted into the correct position while they are added to the list. Currently the list is implemented as a binary tree, making it relatively fast and easy to insert and find elements. However element insertion remains the most time-consuming part of the algorithm.

4.2.3 Results

The Enhanced ChainMail Algorithm was implemented and tested in 2D as well as in 3D for both flat and volumetric objects. The algorithm was integrated into a general system which allows a 2D gray-scale image as well as a raw volume data set to be read in, material constraints to be set and the resultant model to be interactively manipulated. Constraints are set using a lookup table based on gray-scale values which can easily be modified.

Figure 4.9 shows the deformations achieved with ECM of the same dataset that was used in figure 4.2 to demonstrate the failure of the original ChainMail. The red arrow indicates the location and direction of the current manipulation. The object is grabbed at its right brim and pulled to the right (images one to five). Then the object is torn downwards (images six through nine). Images ten through fourteen show displacements to the left/top, which leads to a compression of the softer tissue and finally the object is stretched again in image fifteen.

With ECM the rigid t-shape is no longer torn apart and its deformation as well as the deformation of the softer tissue looks natural. Even material densifications can be observed around the t-shape when tissue is pushed together e.g. in front of the t-shape (third image in the first row).

Figure 4.10 illustrates the deformation of an object which contains a continuous transition from rigid to soft. A gray-scale gradient is embedded in a white surrounding. Gray values are directly mapped to material properties where black corresponds to completely rigid and white to soft material. The ChainMail structure has a resolution of 50×50. The initial distance between elements is 10 length units. The first picture in figure 4.10 shows the structure as it was generated from the gray-scale image. Starting with the second image, elements are used as vertices of an OpenGL triangle strip which renders a continuous object[3]. Link properties are set according to the concept introduced in section 4.2.1 and illustrated in figures 4.7 and 4.8 based on the gray values of the nodes.

[3]The continuous object is equivalent to the initial gray-scale image scaled by a factor of ten.

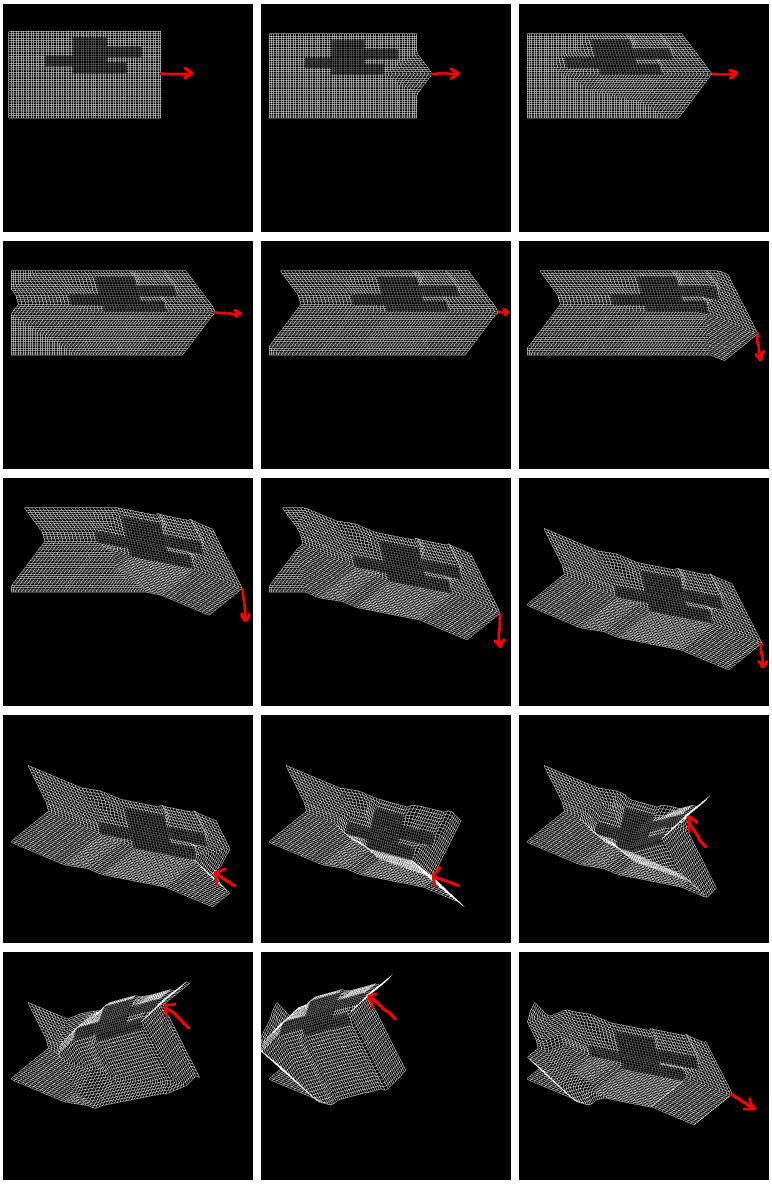

Figure 4.9: ECM algorithm, operating on the same t-shape data set that was used in figure 4.2 to illustrate the failure of the original ChainMail. The stiffer t-shape is embedded into soft tissue. The red arrows indicate the direction and location of the current manipulation. The frames are taken from an animation which can be performed interactively. Frames are rendered in *line-mode* (only the links are drawn).

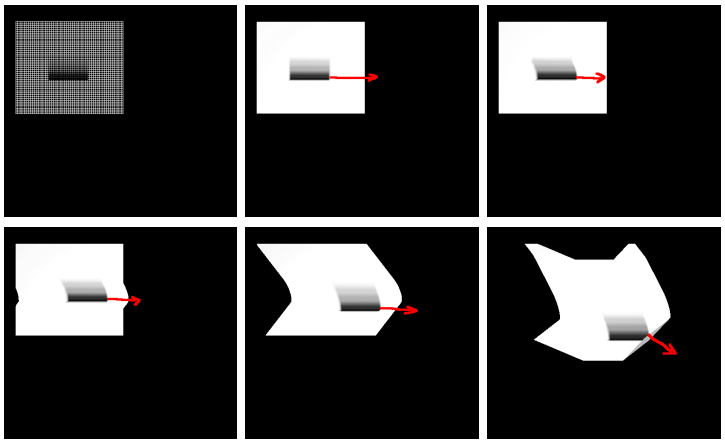

Figure 4.10: ECM algorithm: A gray-scale gradient, representing a material transition from rigid (black) to soft (white), is surrounded by soft tissue.

Gray values of two connected nodes are averaged and the result is used on a previously defined look up table to get the properties of the link between the two nodes. For the example shown in figure 4.10 the look up tables holding the link properties were generated in the following way:

1. *maxLength*: maximum allowed length of the link between two nodes. *maxLength* is equal to *maxDx* from figure 4.8. The interval $grayvalue = [0, 255]$ was linearly mapped to $maxLength = [10, 15]$ which means that for a link between two black nodes (gray value=0) the maximal length is set to 10.

2. *minLength*: minimum allowed length of the link between two nodes. *minLength* is equal to *minDx* from figure 4.8. The interval $grayvalue = [0, 255]$ was linearly mapped to $minLength = [10, 6]$.

3. *shear1*: maximum shear allowed in y-direction. *shear1* equals the sum of *deltaVertDx* (see figure 4.7) of the two participating nodes. The interval $grayvalue = [0, 255]$ was linearly mapped to $shear1 = [0, 7]$.

4. *shear2*: maximum shear allowed in z-direction. *shear2* equals the sum of *deltaHorzDy* (see figure 4.7) of the two participating nodes. The interval $grayvalue = [0, 255]$ was linearly mapped to $shear2 = [0, 7]$.

Link properties are illustrated in figure 4.11. The following pseudocode shows how the link properties are set. The directions x, y, z refer to the local coordinate system of each link[4].

[4]For the example shown in figure 4.10 the *shear2* constraint was not used since no deformation in z-direction was applied.

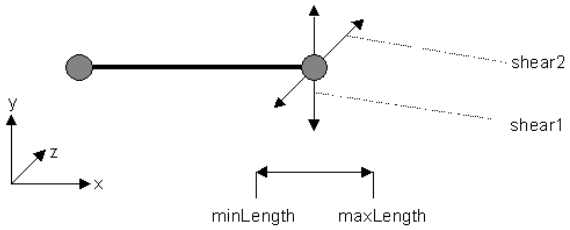

Figure 4.11: Link properties used in the implementation. The directions x, y, z refer to the local coordinate system of each link.

```
linkValue = link[i]->node[0]->getDensity() +
    link[i]->node[1]->getDensity() ;
linkValue /= 2 ;
link[i]->minConstraint.x = minLength[linkValue] ;
link[i]->minConstraint.y = - shear1[linkValue]  ;
link[i]->minConstraint.z = - shear2[linkValue]  ;
link[i]->maxConstraint.x = maxLength[linkValue] ;
link[i]->maxConstraint.y = shear1[linkValue]    ;
link[i]->maxConstraint.z = shear2[linkValue]    ;
```

The described mapping of gray values to material properties generates a fixed link between two black nodes – completely rigid material – no shear or length change is possible. A link between two white pixels (gray value=255) can be stretched by 50% (from 10 to 15 length units) and be compressed by 40%. The shear in y and z direction was set to be equal.

A deformation in z-direction of the local link coordinate system is shown in figure 4.12. A flat object is deformed in 3D bringing the z-constraint of the links into play. The out off plane deformation (pictures five and six) is governed by the *shear2* constraint. Rotating the object in space (third image of figure 4.12), grabbing an element within the large black structure in the middle of the setup and moving it in z-direction results in the configurations in the lower row.

The ECM algorithm can model any object shape. The next example (figure 4.13) shows how an irregular (non-square) shape is used with ECM. Again material properties are mapped to gray values with black being rigid and white being soft material. The example shows deformations of the initially flat object in all three spatial directions.

The modeling of volumetric objects is particularly important for medical simulations. Medical imaging devices like CT and MRI produce voxel data. These volume scans are often used as a data basis for bio-medical simulations. Since the simulation and visualization of volume models is disproportionately more

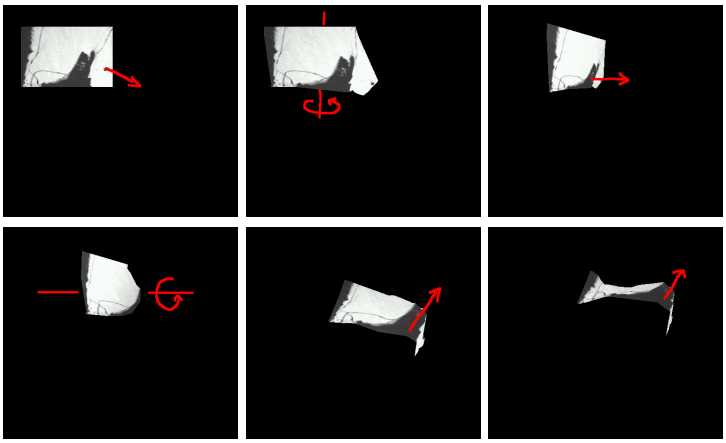

Figure 4.12: ECM algorithm for a 2D shape in 3D: Black corresponds to rigid white to soft material. After an initial in-plane deformation the flat object is turned in space (third image) and deformed into the third spatial direction.

sumptuary than the corresponding operations with surface models, it is very common to generate surface representations from the data delivered by medical imaging devices. This procedure, however, has several drawbacks:

- The complex physical behavior of a volumetric object has to be mapped onto a surface representation which is always a simplification.
- Biomechanical simulations are often conducted to learn more about the internal structures of an organ, e.g. the vascular system, which are not present in the surface representation.
- In surgery simulation cuts or tears in organs provide a view on inner structures.

The simulation of volumetric objects with ECM is straight forward. Instead of a gray-scale image a data volume $x \times y \times z$ is loaded, where $(x_i, y_i, z_i) \in [0, 255]$. In volumetric objects a typical node is connected to its six nearest neighbors instead of four in the 2D case. Again the gray-scale – or density[5] – is mapped to material properties. Figure 4.14 shows the deformation of a homogeneous cube.

An inhomogeneous volume example is shown in figure 4.15. For this example a 3D-dataset was generated by duplicating a gray-scale image 10 times with a z-offset. The cohesion of the dark rigid area is well observable in all 3 directions. The illustrated example comprises about 20.000 nodes.

[5]as provided by CT scanner

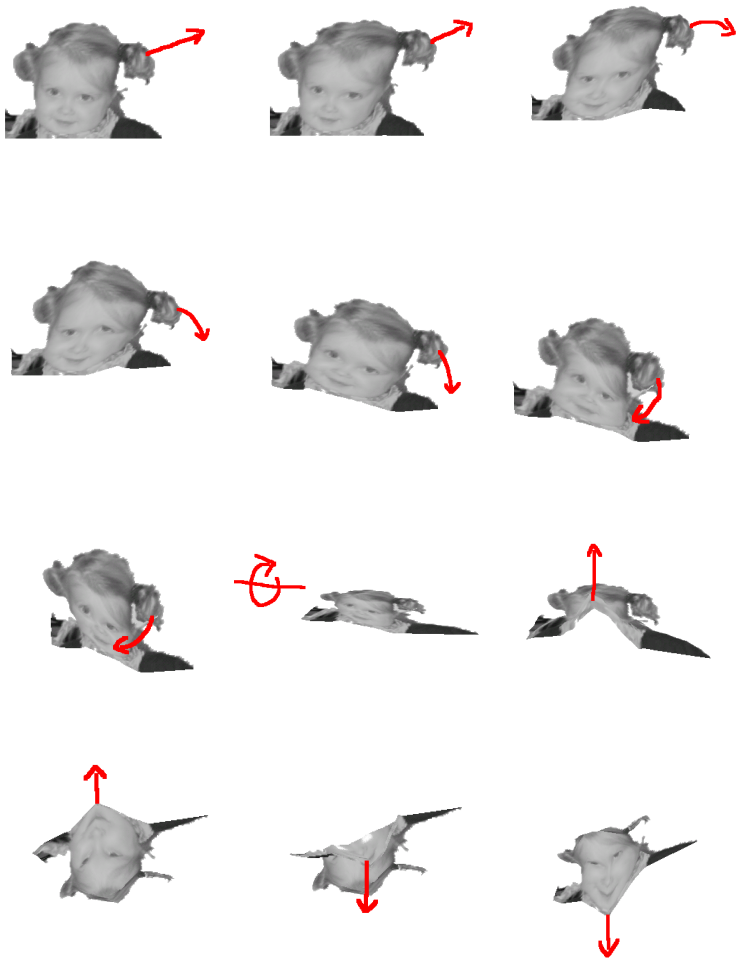

Figure 4.13: ECM algorithm for an arbitrarily shaped 2D object. The initially flat object is deformed in all three spatial directions.

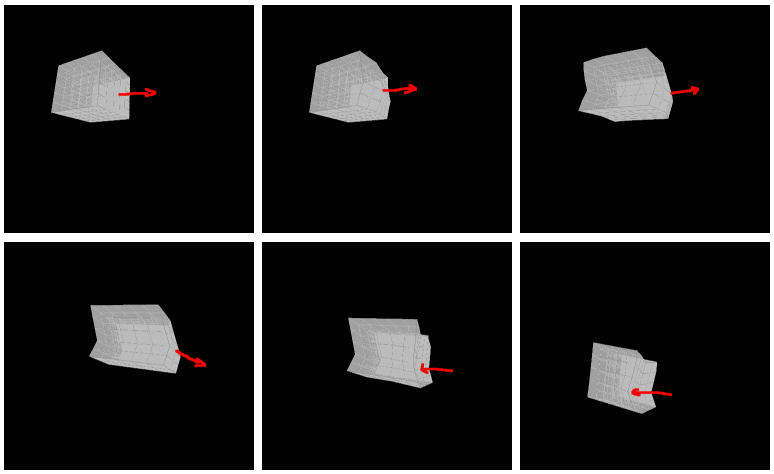

Figure 4.14: Deformation of a homogeneous cube. A typical node is connected to its six nearest neighbors.

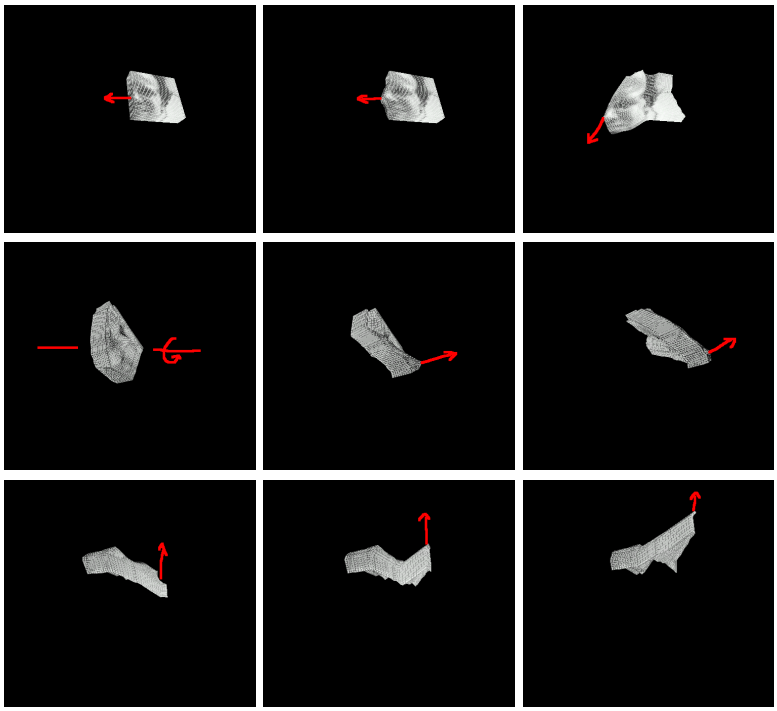

Figure 4.15: Deformation of an inhomogeneous volume.

Performance Analysis

The original ChainMail algorithm can be understood as a highly optimized variant of the ECM for the special case of homogeneous material parameters (see section 4.2.4). As no sorting is required when dealing with homogeneous material the processing order of elements can be optimized so that each element has to be processed only once. Because the ECM algorithm requires sorting it is slower than ChainMail. In addition it is often necessary to process the same node several times. Nevertheless interactive rates are reached on a standard PC hardware.

An object of size $n \times m \times o$ was deformed with ECM by grabbing an arbitrary node and displacing it into:

1. direction $\vec{u} = (X, 0, 0)$
2. direction $\vec{u} = (-X, 0, 0)$
3. direction $\vec{u} = (X, Y, 0)$
4. direction $\vec{u} = (X, Y, Z)$

To average out variations the object was deformed 100 times into each direction in sequence before changing the direction. The displacement was chosen to be large against the size of object to guarantee that all nodes in the object must to be moved to satisfy the constraints. Different object sizes were tested. The test objects were inhomogeneous containing 255 different gray levels. The inhomogeneity was spread over the object. All measurements were carried out on a Intel Pentium III 450 Mhz running under Linux 2.4.4 using gcc version 2.95.2.

Object size: $12 \times 12 = 144$ **nodes.** (All nodes affected by deformation!):

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	0.804968	0.00559	123
(-X,0,0)	0.787949	0.00547	123
(X,Y,0)	1.58299	0.01099	245
(X,Y,Z)	2.35202	0.01633	371

Object size: $25 \times 25 = 625$ **nodes.** (All nodes affected by deformation!)

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	3.81201	0.006099	554
(-X,0,0)	4.22683	0.006762	552
(X,Y,0)	7.58899	0.012142	1108
(X,Y,Z)	11.385	0.018216	1664

Object size: $50 \times 50 = 2500$ **nodes.** (All nodes affected by deformation!):

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	19.73	0.007892	2250
(-X,0,0)	19.387	0.0077548	2250
(X,Y,0)	39.632	0.0158528	4500
(X,Y,Z)	59.685	0.023874	6752

Object size: $100 \times 100 = 10000$ **nodes.** (All nodes affected by deformation!):

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	101.472	0.0101472	9170
(-X,0,0)	101.969	0.0101969	9172
(X,Y,0)	202.138	0.0202138	18340
(X,Y,Z)	304.936	0.0304936	27514

Object size: $200 \times 200 = 40000$ **nodes.** (All nodes affected by deformation!):

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	508.573	0.012714325	39352
(-X,0,0)	502.53	0.01256325	39354
(X,Y,0)	996.893	0.024922325	78708
(X,Y,Z)	1496.99	0.03742475	118054

Object size: $316 \times 316 = 99856$ **nodes.** (All nodes affected by deformation!):

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	1445	0.014470838007	
(-X,0,0)	1417.43	0.014194740426	
(X,Y,0)	2863.53	0.028676594296	
(X,Y,Z)	4298.72	0.043048990546	

Object size: $22 \times 26 \times 20 = 11440$ **nodes.** (All nodes affected by deformation!):

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	221.175	0.019333479021	21091
(-X,0,0)	220.357	0.019261975524	21093
(X,Y,0)	445.261	0.038921416084	42184
(X,Y,Z)	656.348	0.057373076923	63275

Object size: $50 \times 50 \times 10 = 25000$ **nodes.** (All nodes affected by deformation!):

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	426.335	0.0170534	25137
(-X,0,0)	430.177	0.01720708	25129
(X,Y,0)	982.172	0.03928688	50241
(X,Y,Z)	1281.73	0.0512692	75371

Object size: $100 \times 100 \times 10 = 100000$ **nodes.** (All nodes affected by deformation!):

direction:	*time [ms]:*	*time/node [ms]:*	max. Cand.List
(X,0,0)	2298.69	0.0229869	180636
(-X,0,0)	2110.84	0.02110	180642
(X,Y,0)	3844.19	0.0384419	361296
(X,Y,Z)	5595.62	0.0559562	541765

As mentioned above it can happen during ECM that elements are added more than once to the list of candidates. For some measurements the size of the candidates list was protocoled. The largest size that was reached during the process is given in the tables as max. Cand.List. When more than one deformation direction has to be considered ((X,Y,0) and (X,Y,Z)) the list size generally exceeds the number of elements. This is also true for 3D objects. Figure 4.16 summarizes the measurements. The processing time of the ECM algorithm varies linear with the number of elements affected by the deformation[6]. The linear behavior is due to the local nature of the ECM algorithm. All interactions are local; moving an element only affects its direct neighbors. Adding n additional elements to the object increases calculation time by about n times the calculation time required for one element. The number of displacement directions, dispDir:(X,Y,Z), represents a factor in the algorithms complexity. For 2D objects the complexety of ECM is:

$$O(\#dispDir * \#nodes) \tag{4.1}$$

For a 3D oject another factor must be added, which takes account of the object's dimensions. In two dimensions each element has four neighbors, in three dimensional structurs a node has six neighbors. Figure 4.16 shows exactly this relation (six to four) when comparing 2D with the corresponding 3D displacement. The general complexety estimation for ECM result in:

$$O(\#objectDim * \#dispDir * \#nodes) \tag{4.2}$$

[6]precisely: the number of chains

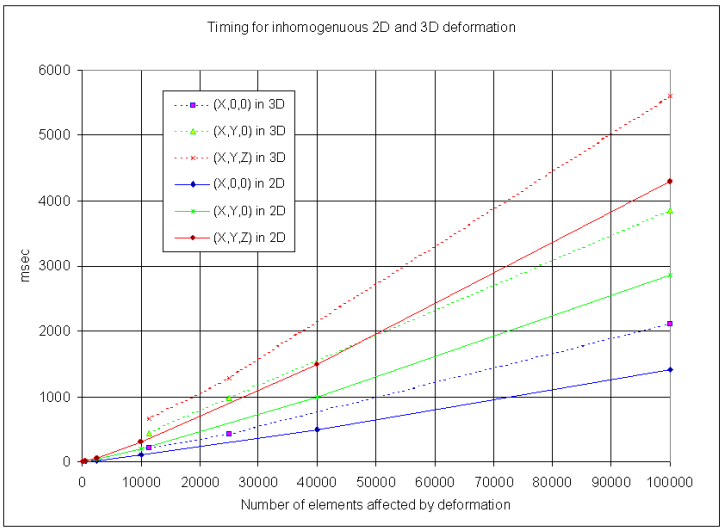

Figure 4.16: Timings for the deformation of inhomogeneous 2D- and 3D-objects of different size with the ECM algorithm. (X,Y,Z) refers to a displacement that was applied in the corresponding directions. Test system: Pentium III, 450 MHz, running Linux 2.4.4.

In ECM, as well as ChainMail the directions (x, y, z) are independent from each other. This issue will be discussed in the next section. However, due to the independence of the three directions, they can be processed independently from each other in separate threads. The solutions of the three processes are superposed in the end to give the final configuration of the object. Sofar the algorithm was not parallelized, but the functions *moveX(node, xDisplacement)*, *moveY(node, yDisplacement)* and *moveZ(node, zDisplacement)* are carried out one after the other. Parallel execution of the three functions on a $\geq$ three-processor computer should produce the same timings for (X,Y,Z) as for (X,0,0,).

Elastic Relaxation

The Enhanced ChainMail algorithm rapidly approximates the final shape of the object by following geometric constraints between neighboring elements. However, it is not ensured, that the final configuration reflects an optimal configuration with respect to energy distribution within the object. Therefore an ECM pass should be followed by an elastic relaxation pass, to equally spread the deformation energy over the object. If the object's energy is given by a function that depends on the distances between elements this function can be used to model different material properties. In the following an elastic relaxation algorithm is suggested that can be used to model different material

properties. The algorithm is inspired by a highly damped mass-spring model and was implemented and tested.

All operations on the vectors *dist*, *deflection* and *relaxDistance* are carried out on their components, e.g. $dist/2 = (dist_x/2, dist_y/2, dist_z/2)$.

```
do for all chains
  dist = chain->node1.getCurrentPosition() -
         chain->node0.getCurrentPosition();

   // mass-spring:
   // F = -D(x-x0) + dv ; x - x0 is the offset from the
   // restposition and D is the fraction of the offset
   // that we will move the node.

   deflection = dist - chain->getCurrentRestLength() ;

   relaxDistance = chain->getCurrentSpringConst( deflection ) *
                   deflection ;

   // each node should be moved half of relaxDistance
   relaxDistance /= 2 ;

   currentPosNode0 = node0.getCurrentPosition() ;
   currentPosNode1 = node1.getCurrentPosition() ;

   // node0 is always the right node,
   // node1 is always the left node
   node0.setCurrentPosition( currentPosNode0 + relaxDistance ) ;
   node1.setCurrentPosition( currentPosNode1 - relaxDistance ) ;
```

The above loop can be run repeatedly. By making the *currentSpringConst* a function of the deflection from the restlength non-linear and plastic relaxation can be modeled as well as elastic relaxation. Plastic behavior could be modeled e.g. with spring constants that become zero before the initial rest length of a link is reached.

A second relaxation method was also studied which uses a full featured mass-spring model on the final configuration of ECM to minimize the energy (see chapter 5). However, the above presented method has the advantage that it can be interrupted when user interaction, e.g. displacing the next node, requires a prompt system answer. After each relaxation pass over all chains the system is in a valid ECM configuration. The full featured mass-spring relaxation, if interrupted, can place nodes out of their geometric boundaries, which is fatal for the next ECM pass! If not interrupted, i.e. relaxed to the energy minimum, the mass-spring relaxation works fine.

4.2.4 Discussion

For inhomogeneous materials, the *first-moved-first-processed* element processing order that was used in the original ChainMail is no longer appropriate. As expected, a straightforward application of ChainMail using this processing order gave inconsistent deformations for inhomogeneous materials that resulted in holes and tears. The Enhanced ChainMail algorithm addressed this problem with a new element processing order. The *first-moved-first-processed* criteria was replaced by the more general principle *largest-constraint-violation-processed-first*.

How do the two principles relate?

The idea of regarding ChainMail as a process in which deformation information is being propagated through a connected neighborhood (see section 4.2.1) will be utilized to answer this question. In the original ChainMail implementation lists are maintained for left, right, top and bottom neighbors[7]; each list corresponds to a propagation direction. The candidate lists were processed until empty. This is feasible because in the current implementation propagation directions are independent and - don't interfere with each other. Causality is complied by processing the elements in the same order as they were added to the lists. While processing one list the relevant deformation information decreases (compare figure 4.6) which – for homogeneous material – conditions that the constraint violation in step $(i+1)$ of the list processing is always $\leq$ the violation that was processed in step i. Therefore in the homogeneous case the processing order *first-moved-first-processed* is equivalent to the more general *largest-constraint-violation-processed-first* principle.

In the current ChainMail implementation the different propagation directions are independent from each other. A deformation, e.g. in x-direction has no effect on the material in any other direction than x. This is an approximation of true physics; a real object would contract in the directions perpendicular to the main stress direction (compare section 2.3). The capabilities of the ChainMail algorithm are not limited to the current implementation. The current implementation was chosen for performance reasons. ChainMail is fast because it uses a very simple distance measure along the main coordinate axis. Calculating the constraint violation is the task which is performed most often during ChainMail. Therefore a considerable speed-up was achieved by using a distance norm along the main coordinate axis.

$$xDistance(n, n+1) = \|x_n - x_{(n+1)}\|; \tag{4.3}$$

$$yDistance(n, n+1) = \|y_n - y_{(n+1)}\|; \tag{4.4}$$

$$zDistance(n, n+1) = \|z_n - z_{(n+1)}\|; \tag{4.5}$$

The link constraints between two elements in a 2D environment can be violated with respect to the x-direction and independently with respect to the

[7] 2D case

y-direction or z-direction. In the first case either $maxLength$ or $minLength$ is violated, whereas in the latter $shear1$ respectively $shear2$. If link constraints were modeled physically correct the constraints would refer to the absolute length of the link between element n and $n+1$, e.g.

$$link\ constr. = \sqrt{xDist^2_{n,n+1} + yDist^2_{n,n+1} + zDist^2_{n,n+1}} \tag{4.6}$$

which is considerably more expensive to evaluate.

Not only the calculation of constraint violations but also their handling is accelerated considerably by using equations (4.3), (4.4) and (4.5) instead of (4.6). To satisfy the link constraints, the *candidate for a move* must to be moved the distance

$$\begin{aligned}
\Delta x &= amount\ of\ x\ constraint\ violation = xDistance - maxLength;\\
\Delta y &= amount\ of\ y\ constraint\ violation = yDistance - shear1;\\
\Delta z &= amount\ of\ z\ constraint\ violation = zDistance - shear2;
\end{aligned}$$

which again is a simple difference.

The calculation becomes more expensive, if constraints were set as illustrated in figure 4.17. Such a model would require the evaluation of equation (4.6) to determine the constraint violation. It would depend on the element's history,

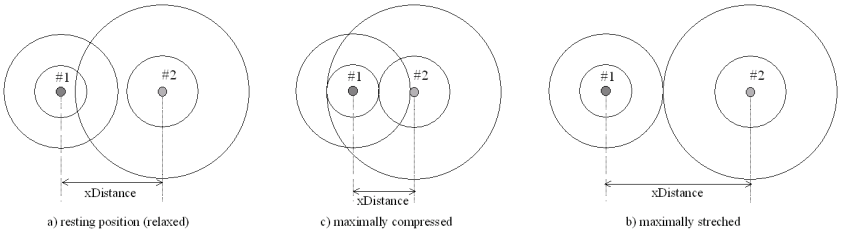

Figure 4.17: ChainMail uses square regions of interaction as shown in figure 4.7. A more appropriate model would consider these regions to be circular, which corresponds to using the distance norm given in eqn. (4.6) as link constraint.

how it had to be moved to comply the constraints (see figure 4.18). How would the link in I.2) react to the applied force F_2? it could either stay rigid and pass on shear to the next link or it could contract in x-direction to allow a similar result of the deformation as in II.). The propagation directions would no longer be independent and the individual constraints would have to be calculated for each link in each iteration step. Determining the new position for an element would also be more costly. Figure 4.19 compares the element positioning for circular and square interaction areas. In the circular case (blue) the*candidate for a move* should be placed where the straight line from candidate to sponsor intersects with the circular interaction area.

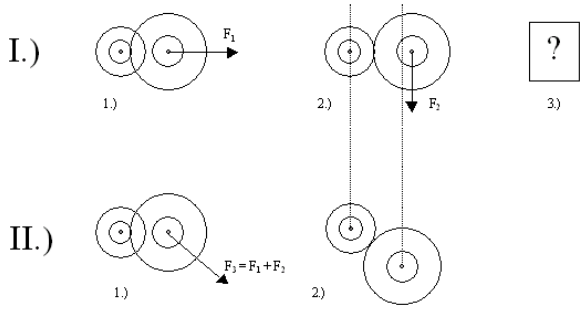

Figure 4.18: With circular interaction areas the result of a deformation process becomes dependent on the previous history of the link.

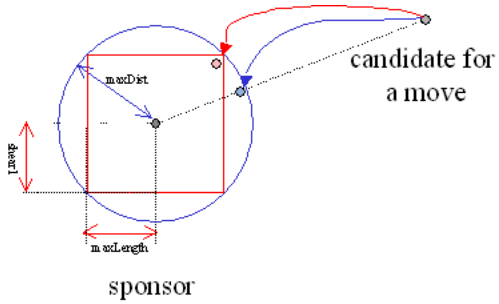

Figure 4.19: Difference between square (red) and circular (blue) interaction areas: The *candidate for a move* should be placed where the straight line from candidate to sponsor intersects the circular area. Determining this point is expensive. For the square interaction areas only two difference operations have to be performed.

The Enhanced ChainMail algorithm features the possibility of interactively deforming considerably large inhomogeneous data sets. Moreover ChainMail is based on a very well understandable and intuitive concept which puts it in line with mass-spring models.

As a matter of fact, comparing ChainMail and mass-spring suggests that a mass-spring system with a spring force of the following type

$$\begin{aligned} F_i &= 0 \quad \forall \quad i < i_{max} \\ F_i &= \infty \quad \forall \quad i \geq i_{max} \\ & \quad where\ i \in x, y, z \end{aligned}$$

and an infinitely high damping would result in a quite similar configuration as ChainMail. Assuming such a configuration, a spring that was elongated by

$x < x_{max}$ would exert no force at all. It would take up the deformation in its structure. Once $x >= x_{max}$ an insuperable force would move the neighbor along. Using a deformation-force-relation as shown above for the springs of a real mass-spring system leads to numerical instability as the masses are infinitely accelerated once x_{max} is exceeded. In our thought experiment we introduced an infinitely high damping to compensate for the acceleration. If very short time-steps could be calculated while moving the *initial sponsor*, the nodes would come to rest at the same places as with ChainMail[8].

ChainMail differs from mass-spring in the way objects are manipulated. A mass-spring model is deformed by acting forces whereas in ChainMail an initial deformation is applied; a node (initial sponsor) is set to its new location and the rest of the body adjusts its position accordingly.

The Transmission Line Method, introduced in section 3.3.6, follows a similar concept as ChainMail. In both models information waves are propagated through a network of interconnected nodes. However, the way nodes are processed in TLM is more similar to mass-spring than to ChainMail. TLM as well as mass-spring can be regarded as isochronous processes with an explicite time dependency, which means that for one time-step a loop running over all nodes is processed. There is no particular order in which the nodes have to be processed and the effects of adjustments, performed in the current time-step, are not considered until the next time-step. The system needs several time steps to reach its final state. E.g. in a mass-spring system a node A is moved because the springs connecting A to its neighbors are tense. Those springs are tense because in the previous time-step the neighbors of node A were moved. One time step follows the next until equilibrium is reached.

The situation is different with ECM. An ECM pass has a clearly defined chronology; the order in which nodes are processed matters! TLM and mass-spring have an explicit dependency on the time. Whereas in ChainMail t is an implicit parameter which is considered by keeping a certain processing order.

Since no explicit time dependency is present, ChainMail can not consider any dynamic effects and the system is always in a valid state at the end of a pass (one-step process). The algorithm results in a final configuration which states a position for each element within the constraints implied on this element by its neighbors. In contrast mass-spring models are generally time dependent. After each time step the model delivers a configuration that can be rendered, however the final object configuration is generally not reached until several time steps were calculated.

A limitation of the current Enhanced ChainMail implementation is, that it does not model volume conservation, an important characteristic of many biological tissues. There are at least two ways to impose volume conservation on the deforming volume: The first suggestion is to incorporate volume conservation

[8]The above view suggests that ChainMail represents a robust method to very rapidly solve a special type of linear equation, respectively unequation systems.

into the relaxation process. A certain size of the volume spanned by 8 adjacent nodes in 3D could be defined as energy minimum. The relaxation process would develop the system towards a configuration with the same volume as the start configuration. The second method would be to modify material constraints in dependency of the perpendicular link state. If e.g. a link is maximally stretched in x-direction, the constraints in y- and z-direction would be made smaller. This would require to check neighbors not only against constraint violations in the same direction as the displacement but also perpendicular to this direction. While this direct method seems to be easier at first glance it requires the administration of individual link constraints as link constraints now depend on the history of the deformation process.

4.2.5 *ECM* in Soft Tissue Modeling

The original ChainMail algorithm was developed for the manipulation of large volumetric data sets as produced by medical scanners like CT or MRI. It was first used to simulate the cartilage in the simulation of an arthroscopic knee surgery [Gib97a]. The knee model was created by hand segmentation of MRI data and is visualized with volume rendering. The system features a 5 DoF force feedback (3 translation, 2 leverage) which is generated by a self-designed enhancement to a PHANToM device. The cartilage, which is attached to rigid bone, can be manipulated with the arthoscope while realistic rendering and force sensation is provided.

The ChainMail algorithm was used in [BSV+98] for the deformation of soft-bodies while doing haptic volume rendering. When the stylus of a PHANToM device collides with a volume object in the scene, forces are generated on the stylus and the object is deformed accordingly.

In 1998 the Enhanced ChainMail algorithm was first published [SGBM98].

To illustrate the intuitive way in which objects can be manipulated with the Enhanced ChainMail algorithm a work-frame has been implemented for the manipulation of volume data sets. For the rendering of the original and deformed data sets the volume visualization software VG Studio[9] was used. The images in figure 4.20 show the deformation of a CT data set of a jaw. A voxel of the jaw bone at the front part of the chin was moved in the direction indicated by the red arrows on figures 4.20. The images on the top show only the boney parts of the data set to illustrate their deformation. The images on the lower part of figure 4.20 show also the soft tissue that was moved along with the bone by the ECM algorithm.

The Enhanced ChainMail algorithm was used in [YXR+00] to implement a semi-automatic 3D surface editing tool for generating models of the cortical surface. A model brain was superimposed on an arbitrary brain data set.

[9] by Volume Graphics, `http://www.volumegraphics.com`

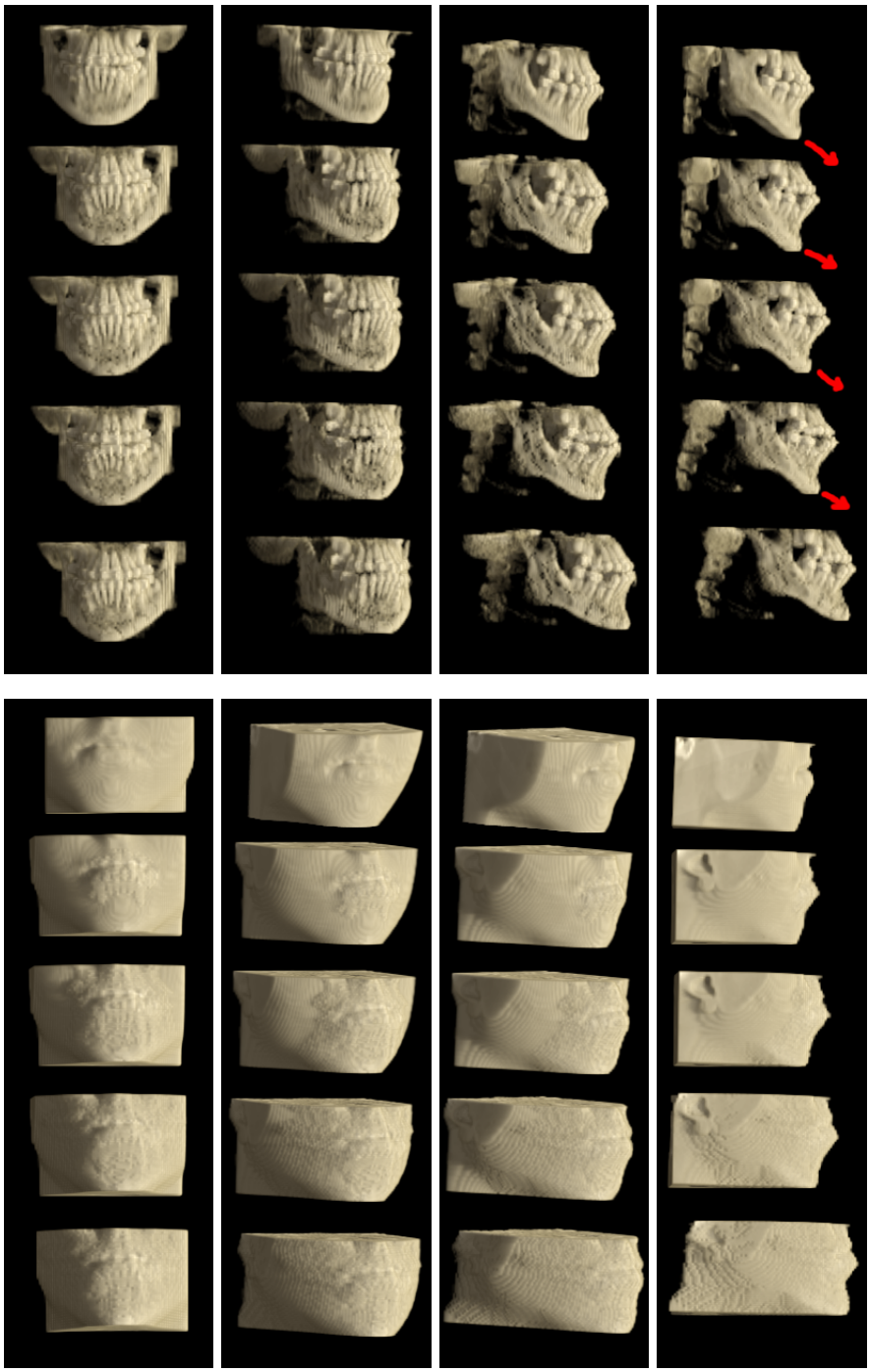

Figure 4.20:

Differing points on the model brain surface can be adjusted by mouse operation on the 2D image planes where the surface contours of the model data and the data set are superimposed. The model brain's surface is represented as an Enhanced ChainMail net. Therefore the adjustments made in a 2D image slice are propagated along the surface also perpendicular to the image plane while preserving topology and surface smoothness. Yu et. all. used ECM for the implementation of a suface editing method that can move local regions on a 3-D surface while preserving topology and surface smoothness.

4.2.6 Classification

ChainMail is a typical example for the process described at the end of section 3.4. A descriptive model is more and more enhanced to meet the demands of the real processes. At first ChainMail was only capable to model homogeneuous materials. With the development of the presented ECM algorithm inhomogeneuous material can also be modeled. Gibson has done considerable work to show that real material properties can be modeled in the relaxation step, which can also be regarded as an enhancement to ChainMail [Gib00].

Even though Enhanced ChainMail uses physical principles like sound wave propagation or elastic relaxation, ChainMail and Enhanced ChainMail are descriptive modeling techniques[10].

4.3 Summary

The Enhanced ChainMail algorithm was developed and presented utilizing the physical analogy of sound wave propagation in material. Enhanced ChainMail is well suited for the interactive deformation of various kinds of objects. The new ECM algorithm addresses the need for modeling complex, inhomogeneous materials. ECM can be regarded as a generalization of the original ChainMail – or more precisely the original ChainMail resembles a highly optimized special case of the ECM algorithm. For homogeneuous material (the original ChainMail case) the *first-moved-first-processed* principle becomes equivalent to the principle *largest-constraint-violation-processed-first* used in the Enhanced ChainMail algorithm (see section 4.2.4).

Several examples illustrating the capabilities of ECM in modeling inhomogeneuous two- and three-dimensional objects were presented. A frame work was developed which allows the generation of two- and three dimensional ECM objects from either gray scale images or raw volume data sets. These objects

[10]Figure 2.2 on page 9 presents a more differentiated view on the classification scheme. While in the respective sections the modeling techniques are classified only with respect to being descriptive or physical, figure 2.2 arranges the approaches in the whole range from physical to descriptive

can be deformed interactively in all spatial directions. Gray values are mapped to material constraints which can also be changed interactively.

The speed of the ECM algorithm is based on using a very simple distance norm between neighboring nodes, a difference along the main coordinate axis. Firstly this distance can be calculated much faster than the correct distance between the nodes, which would require the evaluation of a square root term. Secondly the used distance norm leads to rectangular regions of interest (see section 4.2.4) which makes the three spatial directions independent from each other. The independence of the spatial directions can be exploited for parallelizing the algorithm. Parallel execution of the three spatial directions reduces the complexity of the algorithm to:

$$O(\#objectDim * \#nodes) \tag{4.7}$$

An ECM pass leaves the object in a configuration which does not necessarily correspond to an optimal energy distribution. Therefore ChainMail will generally be followed by an elastic relaxation step. The energy stored in the object depends on the distance between nodes and can be reduced by locally adjusting relative element positions. Different energy-minimizing criteria can be set to the relative distance between two nodes. If e.g. the energy stored in a link is minimal only when the link has its initial length (the length it had when the deformation process started) the relaxation would run until the initial shape of the object was reached. Such a link criterium in the relaxation process models completely elastic material. Plastic behavior can be modeled by defining a link length range which corresponds to minimal energy.

5

Implementation

In [Boo01] Grady Booch states that the software industry faces the three harsh realities that developing complex software of quality is wickedly hard, that it is not getting any easier, and that there is a real shortage of skilled programmers.

> "In the face of such pressure, there are pragmatically only two things a development organization can do. First, the best way to reduce the risk of software development is to not develop any new software at all. That's why the reuse of assets – from code to models to pure intellectual property – is an essential practice of every successful development organization. Second, [...] there is an essential complexity to most interesting software, we can't expect to reduce complexity, we can only seek to manage it: that means raising the level of abstraction in the components we create [...]."

Finding a suited algorithm for one's simulation needs is often a time demanding process of experimenting with different approaches. Once a certain approach is implemented the simulation is fixed to the chosen algorithm. Changing or experimenting with other approaches generally requires to start all over again and reimplement the simulation with another algorithm.

Being capable to change the simulation algorithm of a running system with only little programming effort is highly desirable. While searching a suited algorithm various approaches could be tested regarding their suitability for the problem. If a similar problem had to be solved which required a different algorithm most parts of the existing system could still be used. This chapter describes the first steps towards such an architecture. The implementation currently called *vrmDesign* even allows to change the simulation algorithm during a running

simulation. This feature was used to relax an Enhanced ChainMail simulation with a mass-spring model.

In contrast to conventional modeling approaches which are generally hand tailored to the particular problem, the suggested architecture aims at high portability. The idea is to develop a tool box which offers various simulation algorithms all running on the same data structure and thus being exchangeable. After introducing the data structure developed for the *vrmDesign* the modeling approaches that were implemented so far will be presented.

5.1 On Masses, Elements, Chains and Vertices

When studying the various modeling approaches presented in chapter 3 one might discover similarities dispositional to the discrete nature of a computer (compare page 15). Most of the algorithms operate with spatial discretization where properties of an actually continuous object region are summarized in a single point. Such a point is called *mass* in case of a mass-spring approach, it is referred to as *node* in FEM or *element* in ChainMail. The discrete points are set in relation to each other by some sort of connection. The connection is called *spring* or *chain* in mass-spring respectively ChainMail or *element* in FEM.

Visualization has not been discussed so far, but it is obvious that visual feedback must be considered when designing a tool box for biomechanical simulations. Most commonly visualization is done with the computer graphics library OpenGL [Kil96], [WND97], [Rog92]. Alike the simulation algorithms, OpenGL stores properties of the object's visual appearance in discrete points called *vertices*. OpenGL represents the object's surface with triangle meshes. Vertices are connected with each other via triangles.

Figure 5.1 illustrates the similarities in the different approaches regarding their representation as meshes made up out of points and connections between them. The images in the top row of figure 5.1 show a mass-spring structure, a beam made up out of finite elements and a deformed Enhanced ChainMail cube. The lower row shows geometric objects as represented by OpenGL in wireframe mode on the left and shaded according to the properties stored in the vertices on the right.

In mass-spring (see section 3.2.1) the points are assigned properties like mass, position, velocity, acceleration. The springs typically hold information about the nodes they are connected to, length, spring constant and damping. A certain finite element always has a given topology, e.g. a triangular finite element has triangular shape. Depending on the used interpolation functions the nodes of the chosen element type hold a number of node variables (compare section 3.1.1). Typically node variables in structure analysis are the position of a node

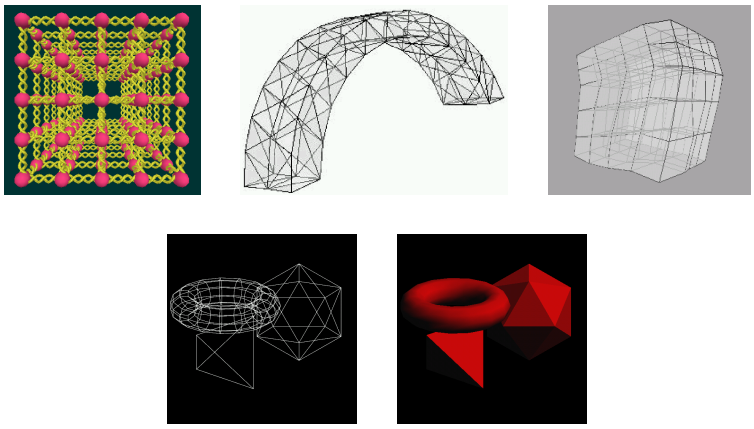

Figure 5.1: Similarities in object representation. Top row: mass-spring, finite element and ChainMail, lower row: OpenGL example vertex mesh representation on the left and shaded according to the properties stored in the vertices on the right.

and higher derivatives thereof. Structure information is stored in e.g. a triangle mesh were each triangle holds information about the nodes that form the triangle. In Enhanced ChainMail a chain-element is connected via chain-links to its neighbors. Besides its position, each element is also assigned a parameter which summarizes the material properties of the immediate surrounding of the point's location. As presented in section 4.2 chains have properties like rest length and min- and max length with respect to the main coordinate system axis. The chain parameters can easily be created from the material properties stored within the points. As mentioned above, in OpenGL vertices form primitives like triangles, where each vertex has a position and holds information which determines the optical appearance of the primitive in the visualization process. Such properties are e.g. color and normal vector.

5.1.1 Nodes and Connectors

The objects which are to be simulated have to be represented in a data structure. The data structure we are looking for should be suited for the different simulation algorithms as well as for visualization. A single data structure that could be used by all simulation algorithms is a pre-requisite for the planned tool box system, because if all the simulation algorithms run on the same data structure, they can easily be exchanged.

The similarities presented in the previous section, suggest to look for an abstraction of the various forms of nodes and connections between them. The

terms *node* and *connector* will be used for the wanted abstractions. Nodes represent the spatial discretization of the problem whereas the connectors can be thought of as an ordering principle on the set of nodes. A connection between two nodes will be called *2-connector*, between three nodes *3-connector* and so on. A connector will basically store pointers to the nodes it connects. Figure 5.2 shows how a set of nodes (bottom row) could be ordered with different connector types to represent a particular object. A 3-connector represents a triangle and a 2-connector could be a spring or a chain between two nodes. The example illustrates how a set of 2-connectors (a, b, c, d, e) can represent the cross shaped structure on the right by defining a topology for the nodes. The 3-connectors (A, B, C) create the triangulated shape on the top right of figure 5.2. The figure also shows a 1-connector which corresponds to a look-up-table. 1-connectors could be used e.g. to resort the set of nodes without actually having to rearrange the nodes in memory. The two examples given in figure 5.2 refer to different sets of nodes.

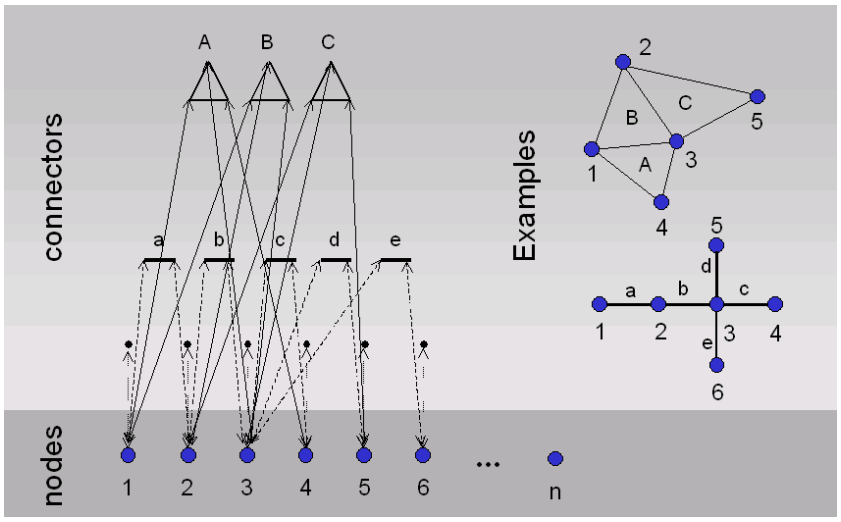

Figure 5.2: A set of connectors "forms" the object by arranging the nodes in a topology. E.g. the set of 2-connectors (a, b, c, d, e) create the cross shape in the examples on the right from a given set of nodes. The 3-connectors A, B, C generate the triangulated shape on the top right. The two examples refer to different sets of nodes (positions on the nodes differ).

In a real simulation several ordering principles will be used on the same set of nodes. E.g. 2-connectors will be used to represent the ChainMail mesh and at the same time 3-connectors will form the triangles needed for OpenGL rendering. Maybe certain nodes have to be stored in a list, which could very well be done with an array of 1-connectors. Figure 5.3 illustrates 1-, 2- and

3-connectors on a single set of nodes.

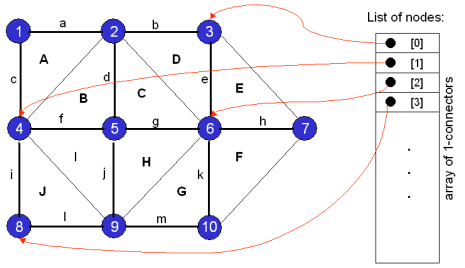

Figure 5.3: Different connector types operating on a single set of nodes. The 2-connectors ([a, m]) create a mesh that could either be used for mass-spring or ChainMail simulation. The 3-connectors ([A, J]) form triangles that could be used for OpenGL rendering. The list of 1-connectors on the right can be used to store and re-order nodes.

A connector is basically implemented as a list of pointers to nodes. As already indicated in figure 5.2 by the two pointed arrows connecting nodes and connectors, it is useful if not only the connectors "know" the nodes they connect but also vice versa; the nodes store pointers to the connectors that concern them. Most simulation algorithms require to access the connectors via the nodes: e.g. in ChainMail the *initial sponsor* is moved and then adjacent neighbors are checked whether they also require a move. The criteria if and how far the neighbors have to be moved is given by the connector between the nodes. In homogeneous material this criteria is identical over the whole object and can be stored globally, but for inhomogeneous material the constraints vary and are therefore stored in the connector. For regular topologies the neighborhood relationship of the nodes can be stored implicitly in the data structure[1], but irregular structures require to explicitly store the topology. This can also be achieved by the bi-directional pointers between nodes and connectors.

5.1.2 Object-oriented Analysis and Design

The key to implementing the desired data structure which could serve as a basis for simulation and rendering is an object-oriented analysis and design. At the beginning of this section (5.1) the similarities in object representation of different simulation algorithms and OpenGL rendering were presented. Following the guide-lines of object-oriented design the problem domain was analyzed in terms of principal abstractions. The abstractions *node* and *connector* were introduced and their usability was demonstrated. Next, we will map these ab-

[1] think of a matrix $n \times m$, where the neighbors of the entry x_{ij} can be determined easily.

stractions into classes, using the two major concepts of object-oriented analysis and design: polymorphism and inheritance.

Object-oriented programming languages allow the creation of object hierarchies with common method names for operations. The operations should be conceptually similar, but can be implemented differently for each class in the hierarchy. This functionality is referred to as polymorphism. Operations defined for arguments of a certain class will also accept arguments of a polymorphic class. If e.g. the classes node, massNode and chainNode were polymorphic where node was the superclass and an operation:

```
OPERATION simulate(node myNode)
{
  myNode.simulate() ;
}
```

was defined, *simulate()* would also accept nodes of the type massNode or chainNode as an argument.

Polymorphisms are often generated by inheritance. Inheritance is a mechanism for automatically sharing methods and data among classes: classes inherit the operations and data of their parent classes. The programmer only needs to implement the differences to the parent class. If certain methods (operations) of the parent class are inappropriate they have to be overwritten in the child class – which generates a polymorphism. Due to inheritance object-oriented programs consist of hierarchies of classes, where classes become more specific with each inheritance level.

5.2 System Architecture: *vrmDesign*

The *vrmDesign* was implemented and – so far – successfully adapted to mass-spring, ChainMail, FEM models and OpenGL rendering. The architecture can be extended to other modeling approaches that fit into the node-connector abstraction. The presentation given in this section focuses on the functionality of the design. Details are skipped and simplifications are made where possible to emphasis the basic design concept.

5.2.1 Basic Classes

The part of the *vrmDesign* that deals with nodes and connectors was dubbed *vrmGraph* , since nodes and connectors represent a mathematical graph. The *vrmGraph* class hierarchy is presented in figure 5.4. As mentioned above, connectors store pointers to the nodes they connect. Nodes should also hold pointers to the connectors that they are part of. This suggests that connectors and

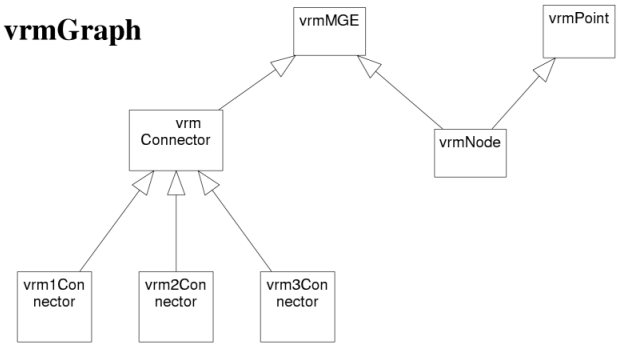

Figure 5.4: vrmGraph class hierarchy. *vrmMGE* is a top level class that provides the management of pointer lists to other *vrmMGEs*.

nodes are conceptually similar and can be generalized in one top level class. The class is called *vrmMGE* .

vrmMGE
The *vrmMGE* mainly provides functionality for the management of lists of pointers to other *vrmMGEs*. Several lists of such pointers can be created and maintained within one *vrmMGE* . This functionality is needed e.g. in a node that belongs to connectors of different types; different connector types are stored in separate lists.

vrmPoint
A *vrmPoint* represents a simple point in space. It holds three coordinates and the methods to set and get the values of (x, y, z).

vrmNode
The *vrmNode* inherits its properties from *vrmPoint* and *vrmMGE*. The part of the interface that is inherited from *vrmMGE* is supplemented with methods that refer to adding, removing and finding particular *vrmConnectors* in the *vrmNode's* lists.

vrmConnector
vrmConnector is also a child of *vrmMGE*. In addition to the inherited functionality the maximum number of allowed pointers to *vrmNodes* can be set. Pointers to nodes can be added separately or as an array. *vrmConnector* has methods that refer specially to the manipulation of pointers to nodes. These methods do a type check on the argument; they can only be used with a *vrmNode* as argument.

vrm1-, 2-, 3Connector
Essentially the same as *vrmConnector* with specification of one, two or three

vrmNodes as arguments.

5.2.2 Modeling MassSpring

The basic classes from *vrmGraph* will be specialized in this section for the use in mass-spring simulations. Following the toolbox concept of the *vrmDesign* the basic properties of masses and springs are stored in special property classes called *vrmSpringBase* and *vrmMassNodeBase*. A node for a mass-spring simulation e.g. is then created by inheritance from the *vrmNode* class and the appropriate properties class. Implementing the properties required for a par-

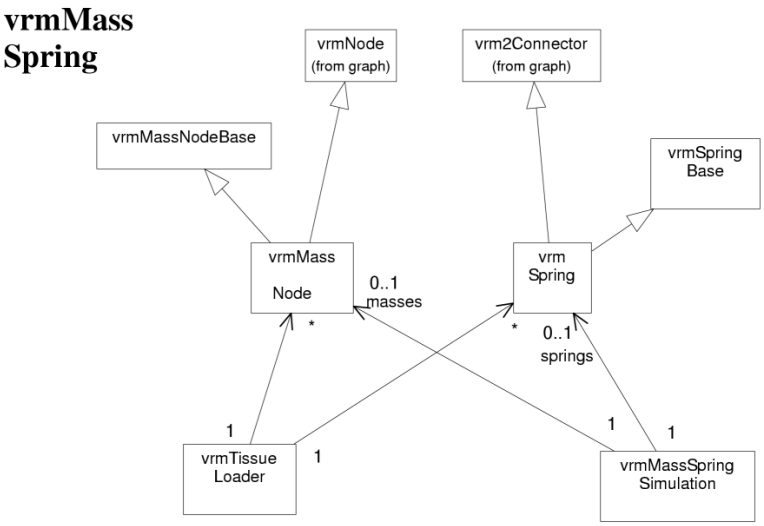

Figure 5.5: vrmMassSpring class hierarchy. Basic properties of masses and springs are stored in special property classes called vrm(...)Base.

ticular simulation in separate property classes makes the combination of two or even more simulation algorithms possible. The simulation node would then inherit the properties needed for algorithm A from the properties class aNodeBase and the properties for algorithm B from bNodeBase.

vrmMassNodeBase
vrmMassNodeBase stores and manages the access to the parameters a mass-spring simulation expects to find in a mass-node. Examples are mass, current velocity, current acceleration and as the case may be external forces acting on the node. Another important property is whether the node is constraint or free.

vrmSpringBase
In analogy to the mass node *vrmSpringBase* stores and manages access to the parameters needed in a spring. Such parameters are e.g. the spring constant, its zero length and the damping.

vrmMassNode
vrmMassNode unifies the properties from *vrmMassNodeBase* with *vrmNode*. *vrmMassNode* is the object that is used in a mass-spring simulation to represent the mass points.

vrmSpring
vrmSpring is derived from a *vrmConnector* and the *vrmSpringBase* class. In a mass-spring simulation it represents a spring.

vrmTissueLoader
The *vrmTissueLoader* represents a basic class present in all simulations. The tissue loader reads in the node information from e.g. a file and creates the topologies necessary to run the simulation. In most cases at least two different topologies are necessary: one for the simulation algorithm and one for the visualization (see figure 5.3).

vrmMassSpringSimulation
vrmMassSpringSimulation is the actual mass-spring simulation algorithm class. It operates on the structures filled by the *vrmTissueLoader*. The simulation holds lists of the springs and nodes the simulation operates upon.

5.2.3 Modeling ChainMail

Again the basic simulation classes *vrmChainNode* and *vrmChain* are derived from the *vrmNode* respectively from *vrmConnector* implemented in *vrmGraph*. Alike the mass-spring implementation the special properties required for a ChainMail model are stored in property classes; the actual chain node is derived from *vrmNode* and *vrmChainNodeBase*, the *vrmChain* from *vrm2Connector* and *vrmChainBase*. The *vrmTissueLoader* is supplemented by a *vrmChainMailImport* class which utilizes a scalar value stored in the connected *vrmChainNodes*[2] to determine the properties of each *vrmChain*.

vrmChainNodeBase
The basic properties of a chain node are stored in *vrmChainNodeBase*. Again a simulation node with the appropriate features will be created by inheritance from *vrmChainNodeBase*. The properties stored in a typical chain node are: the scalar value mentioned above (which we will call density from now on) and the information whether the node is fixed or free.

[2]The scalar value could, for example, be interpreted as the density around the node point if the data underlying the model stems from a CT scanner. In a mass-spring model it could be the mass assigned to the node.

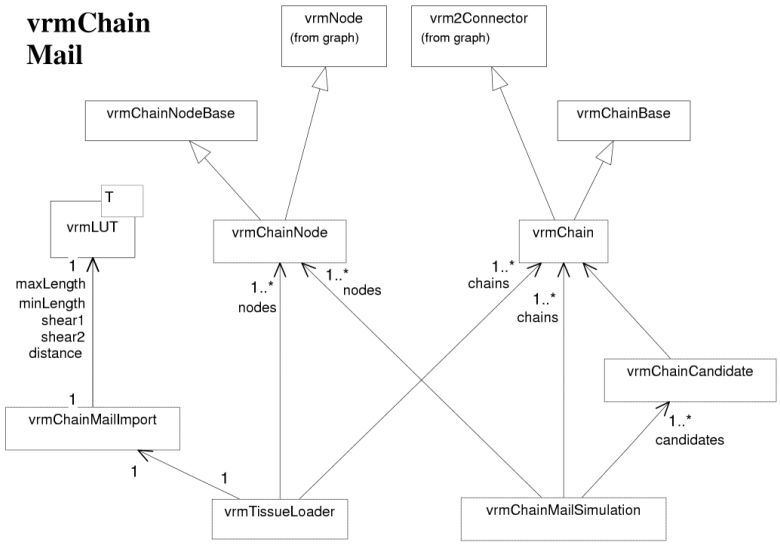

Figure 5.6: vrmChainMail class hierarchy..

vrmChainNode
A *vrmChainNode* is derived from *vrmNode* and *vrmChainNodeBase*. It has methods to set and change its position (inherited from *vrmPoint* ; through *vrmNode*) and the node can access all the connectors it is joint with (inherited from *vrmMGE* , also through *vrmNode*).

vrmChainBase
Every *vrmChain* locally stores its individual constraints. The variables and methods to manage and check the constraints are therefore stored in the *vrmChainBase* class. In the ECM algorithm each chain connects two *vrmChainNodes*. The two nodes have a minimal and maximal distance in all three spacial directions independently (see figure 4.11), i.e. the *vrmChain* has three restlengths and three min. and max. length constraints, one for each direction. Each constraint is related to a possible constraint violation in this direction and a sponsor for this violation. The variables restlength, minConstraint, maxConstraint, violation and sponsor are stored within a vector (x, y, z) and managed in *vrmChainBase*. In addition each *vrmChain* has methods to check whether any of its constraints are hurt.

vrmChain
The *vrmChain* represents the object that is used in a ChainMail simulation. It combines the properties from *vrmChainBase* and *vrm2Connector*. Access to the connected nodes is granted by the functionality inherited from *vrmMGE*.

vrmChainCandidate
vrmChainCandidate is the object type that is stored in the list of candidates for a move. The object holds a pointer to the actual *vrmChain* that is to be checked for a constraint violation. The candidate list itself is held in the *vrmChainMailSimulation* class and is implemented as a STL[3] multiset. As the entries of the multiset have to be ordered according to their constraint violation the greater operator (<) has to be defined for the objects stored in the multiset. In the *vrmChainCandidate* class the greater operator and all other comparators are implemented with respect to the constraint violation; one *vrmChainCandidate* is greater than the other if its constraint violation is greater.

vrmChainMailSimulation
The ChainMail simulation class has two major methods: *move* and *relax*. *move* takes as first argument the pointer to the node that is the initial sponsor of the process and as second argument a vector which holds the displacement applied to the initial sponsor. By applying the displacement to the initial sponsor the ChainMail process is started. *move* initializes a STL multimap that is ordered according to the constraint violations. The neighbors of moved nodes are added to this ordered list. The ChainMail (chainreaction) runs until all entries from the list are processed and the list is empty. Since all nodes and chains are bi-directionally connected ChainMail needs no global structure were all nodes or chains are stored. Instead the process starts with one node and by following the connections it travels through the object. The *relax* method can be implemented in various ways. It represents the process that optimizes the energy distribution in the object, by locally adjusting node positions.

vrmLUT
vrmLUT is an auxiliary class that creates a look-up table by linearly interpolating values between the given key points of a function. It is used to create look-up tables which take the density values of the two nodes connected with a *vrmChain* and return the length and shear constraints for the *vrmChain*. By that the material properties of the simulated object can be changed by manipulating the key points of the linear function given to the *vrmLUT* class.

vrmChainMailImport
Support class for the *vrmTissueLoader*. It reads in a gray scale image or volume data set while interpreting the gray values as material properties and assigning appropriate values to the *vrmChains*.

5.2.4 Modeling FEM

The adaption of the presented design to FEM approaches as suggested in the following was used to implement the simulation of a swelling brain (see section 6.2). The implementation aims at elasticity problems and was adapted

[3]Standard Template Library, a part of the C++ programming language

to static and dynamic FEM analysis. Again basic classes from *vrmGraph* are used to derive the major components of the approach: *vrmFemNode* and *vrmFemElement*. In contrast to the implementations of mass-spring and ChainMail presented above it is not a priori clear how many nodes will be connected by a particular element in an FEM approach; the implementation must allow the deduction of different finite element types, connecting different numbers of nodes. Therefore the *vrmFemElement* inherits its basic properties from *vrmConnector*, which is not limited in the number of nodes it can manage. The actual number of nodes connected by the finite element is specified later in the particular element type class that is derived from *vrmFemElement*.

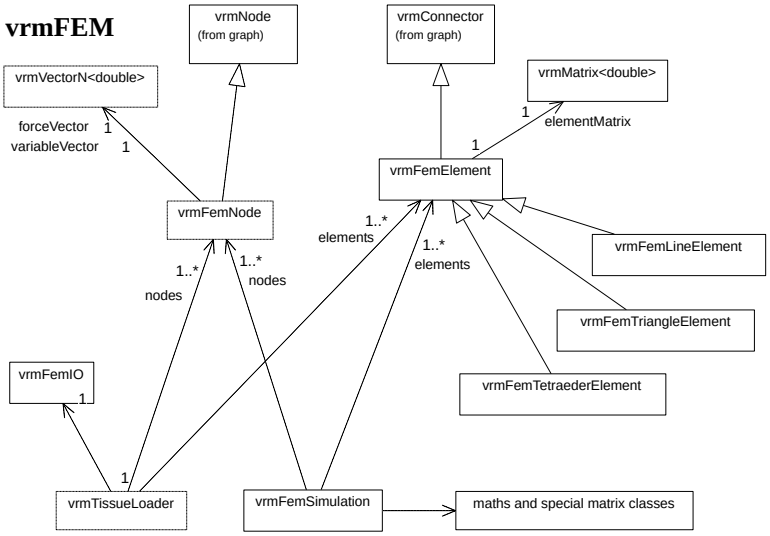

Figure 5.7: vrmFEM class hierarchy..

vrmFemNode

Finite element nodes store the socalled node variables (see section 3.1). These node variables depend on the interpolation functions that were chosen when deriving the element type. Within *vrmFemNode* the node variables are stored in a vector whose length can vary according to the number of node variables. In general the first three node variables are the node's position in space (x, y, z). In biomechanical analysis the possible other node variables are higher derivatives of the nodes position, describing smoothness and continuity of e.g. surfaces. The *vrmFemNode* inherits position related properties from *vrmNode* – if present, other node variables are calculated when assembling the element matrices. The force vector present in every *vrmFemNode* stores the possibly

acting forces on the particular node. The position in the force vector corresponds to the node variable it acts upon. Accordingly the first three entries of the force vector can be used in elasto-mechanical analysis to set acting displacement forces. In analogy to the previously described implementations of mass-spring and ChainMail each node also has variables that describe its state like fixed or free, its color or additional topological information like a global number or whether the node lies on the surface of the object.

vrmFemElement
In the class *vrmFemElement* the interface to the concrete finite element types is defined. Methods in *vrmFemElement* are mostly declared as virtual; the actual implementation is done in the classes derived from *vrmFemElement*. This approach is necessary since *vrmFemSimulation* needs to access the different types of *vrmFemElements* through an identical interface. The most relevant *virtual* function in *vrmFemElement* is *calcMatrix()* which calculates the element matrix and fills the appropriate variable *elementMatrix*.

vrmFemLineElement
Implementation of a simple two node element. It behaves basically like a spring.

vrmFemTriangleElement
Concrete implementation of a triangle element with a in-plane-strain and a bending term. For the deduction see [Gri00], [ner98]. The triangle element is used in section 6.2 to model the Dura mater, a strong skin covering the brain.

vrmFemTetraederElement
Concrete implementation of a tetrahedron element. This volumetric element is used in section 6.2 for the modeling of brain tissue.

vrmFemSimulation
The actual simulation class is responsible for the assembly of the global stiffness and mass matrices as well as the global force vector. *vrmFemSimulation* can access the necessary information for the assembly in the elements and nodes. After having set up all the necessary variables, an equation system of the form (3.8, page 19) or (3.9, page 19) has to be solved. The implemented vector and matrix classes hold all operations to solve such equation systems including memory saving representation of large matrices and sorting of nodes to reduce the bandwidth of the matrices[4].

vrmFemIO
vrmFemIO provides special methods for importing and storing data sets and forces to the *vrmTissueLoader*.

No rule without exception
The separate implementation of simulation specific functionality in *vrm(...)Base* classes was not followed when implementing *vrmFem*. FEM simulations run in a completely different time frame (hours or days) as e.g. ECM

[4]The mathematical classes of the *vrmDesign* are not further presented.

or mass-spring (fractions of seconds). Therefore the combination of FEM with any of the other approaches does not seem reasonable and had not been a primary goal. Besides that there is an implementational issue which led to this excursion from the concept: Whereas for the *vrmFemNode* the separation could have been realized easily, doing so for the *vrmFemElement* would have made the design unnecessarily complicated. The particular functionality of a *vrmFemElement* which could have been placed in a *vrmFemElementBase* class relates to e.g. assembling the element's stiffness matrix. Such tasks require access to the nodes connected by the element because node variables are needed for the assembly. The functionality to access to nodes is implemented in *vrmConnector*. It would not have been available in a *vrmFemElementBase* class. Therefore *vrmFemElement* was directly derived from *vrmConnector* without the use of a *vrmFemElementBase* class.

5.2.5 Visualization

OpenGL visualization is vertex-based, which means that the vertices store all properties that govern the looks of the rendered scene. Vertices are connected to triangles to define the topology of an object's surface in the scene. Properties assigned to vertices are e.g. color, alpha value[5] or a normal vector which can be used in the lighting model. With these properties OpenGL fits well into the concept of the *vrmDesign*. Vertices can be realized as *vrmNodes*, triangles and triangle strips as *vrmConnectors*.

vrmTriangleBase
The class *vrmTriangleBase* stores useful information about the triangle such as a unique id[6] or whether front, back or both sides of the triangle should be rendered. Furthermore *vrmTriangleBase* provides a method which takes two vectors as argument and returns the normal vector. Each triangle's normal vector is needed since the normal vectors of the vertices are interpolated from the normal vectors of the triangles a particular vertex is part of.

vrmTriangleStripBase
A triangle strip is defined as a list of vertices where the first three vertices form the first triangle, each succeeding vertex together with the two last defines a new triangle. Triangle strips are generally a little more efficient than a list of triangles describing the same object. *vrmTriangleStripBase* holds basically the same functionality as *vrmTriangleBase*.

vrmGraphicsTriangle
The actual triangle used by the *vrmVisualization*. *vrmGraphicsTriangle* inherits from *vrm3Connector* the access functions to the nodes, i.e. vertices.

vrmGraphicsMesh
A graphics mesh is represented as a triangle strip. As it is not a priori known

[5] determines the object's opacity
[6] can be used for select mode rendering.

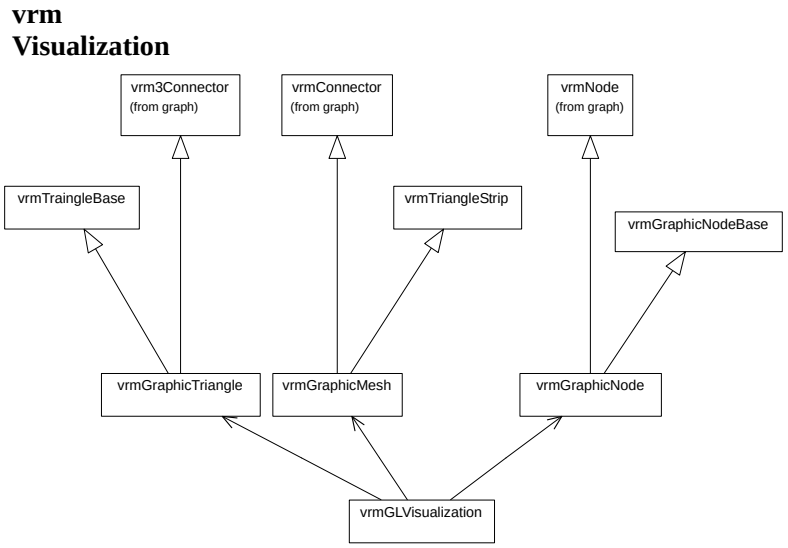

Figure 5.8: vrmVisualization class hierarchy.

how many vertices will be used in the mesh the *vrmGraphicsMesh* inherits the node access from *vrmConnector*.

vrmGraphicsNodeBase
Graphics nodes, or vertices, store all the information that is relevant for rendering such as color, opacity and texture coordinates. In addition *vrmGraphicsNodeBase* stores a normal vector which it can interpolate from a vector of normal vectors passed to the appropriate member function as an argument.

vrmGraphicsNode
In addition to the *vrmGraphicsNodeBase* functionality the *vrmGraphicsNode* inherits node properties from *vrmNode*.

vrmGLVisualization
The *vrmGLVisualization* currently works with lists of *vrmGraphicsTriangles*, *vrmGraphicMeshes* or directly with *vrmGraphicsNodes*. It uses OpenGL to either render triangles, triangle strips or points. The methods provided by *vrmGLVisualization* have identical interfaces no matter what is currently rendered. *vrmGLVisualization* also provides select mode rendering, an OpenGL method for identifying particular nodes or triangles in a scene[7].

The *vrmVisualization* module currently only supports OpenGL. A DirectX

[7]Select mode rendering is often used to find the node that is currently under the mouse cursor.

visualization class is under development. There are plans to also incorporate volume visualization.

5.3 Summary

As finding a suited simulation algorithm for a given problem can be a time demanding task which often requires to try and implement several test environments, a software architecture was suggested that enables the developers of biomechanical simulations to easily try out different simulation approaches. In an object oriented approach simulation and visualization algorithms were analyzed and mapped into a class hierarchy. The key to the implemented software architecture is the *node-connector* abstraction introduced on page 75. As demonstrated, some simulation and visualization approaches can be represented in a the *node-connector* abstraction. So far mass-spring, Enhanced ChainMail, finite element analysis and OpenGL rendering have been implemented using the presented concept. The architecture allows the rapid implementation and testing of simulations using one of these approaches. Due to their common derivation from the *node-connector* concept different modeling approaches can even be combined, using the same data structures. In this way the Enhanced ChainMail algorithm was combined with a mass-spring model. The ECM was used for the fast shape approximation of the simulated object, the mass-spring algorithm was then used for the elastic relaxation whenever no user interaction occurred.

The implemented architecture is open to extensions. Other simulation or visualization approaches that fit into the *node-connector* abstraction can easily be added. Current candidates for the extension of the architecture are Microsoft's Direct3D and Volume Graphics'[8] volume visualization.

Figure 5.9 shows the inheritance tree of the implemented data structure. Node and connector are basically of the same type and find a common abstraction as *vrmMGE*. It was attempted to detach the basic functionality of all the connector and node flavors in the inheritance tree into separate *vrm(XYZ)Base* classes[9]. By this the classes needed for e.g. a mass-spring simulation can easily be created by inheritance from the *vrmNode* respectively *vrmConnector* together with the appropriate *vrm(XYZ)Base* classes: *vrmMassNodeBase* and *vrmSpringBase*. On the application level it will certainly be necessary to visualize the simulation results. As visualization is also included in the *vrmDesign* the concrete simulation node, *mySimNode*, can also inherit the properties needed for visualization; from *vrmGraphicsNodeBase*. The aggregation of properties belonging to different tasks, e.g. mass-spring simulation and OpenGL visualization in one class is only possible because basic functionality was separated

[8] www.volumegraphics.com

[9] The separation was not realized for FEM. FEM simulations run on a different time scale (see page 85) and will not be combined with the other simulation approaches.

into *vrm(XYZ)Base* classes. An object like *mySimNode* could be derived from *vrmNode*, *vrmMassNodeBase* and *vrmGraphicsNodeBase*. Such a node would be suited for mass-spring simulation and brings all the properties needed for a visualization with OpenGL. The same mechanism was used for the combination of mass-spring and ECM, described above.

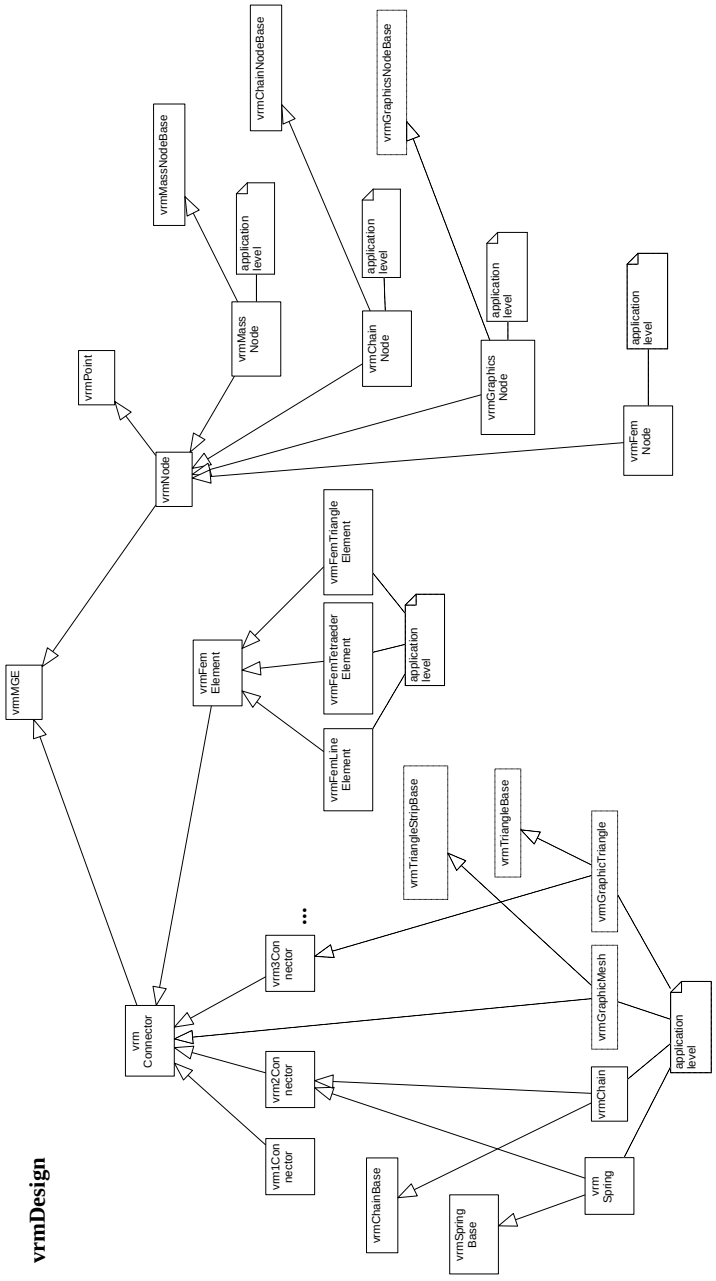

Figure 5.9: vrmDesign class hierarchy.

6

Application

The study of available modeling techniques (chapter 3) and the extension of the available modeling instrumentation (chapter 4, section 3.3) that was conducted in the scope of this thesis was motivated by the desire to built simulations for two concrete medical problems: the real-time simulation of intra ocular surgery for training purposes and the off-line simulation of a decompressive craniotomy for supporting the surgeon with operation recommendations. Both simulations are based on the architecture presented in chapter 5.

The first section of this chapter presents the simulator *EyeSi* that was projected, designed and implemented while accomplishing the work on hand. *EyeSi* is a computer-based medical workstation for the simulation of a vitrectomy that allows training and rehearsal of eye surgeons [SWH+99]. The surgeon manipulates two original instruments inside a cardanically suspended mechanical model of the eye. The instruments' positions are optically tracked and monitored by a PC which renders the scenery using a computer graphical model of the eye and the instruments. Stereoscopic images are presented to the user through two small LCD displays that are mounted to the system and emulate the stereo microscope used in real operations. The simulator offers the training of intra-ocular navigation as well as interaction with pathological tissues using mass-spring and 3D-ChainMail models. All operations (tracking, rendering, collision detection, tissue manipulation) are computed in real-time on a PC. A lot of issues had to be solved for the realization of the simulator EyeSi, which, although they are mainly novel and highly innovative, could not find their way into the description at hand. Due to the limited space references to more detailed literature are given.

The second section deals with the simulation of a diffusely swelling brain in the constraint space of the firm skull. An advanced volume increase of the brain tissue caused by the development of edema across the whole brain leads to a rise of

intra-cranial pressure (ICP) which – if a critical ICP level is reached – becomes life-threatening. In these cases the surgeon supplies additional space to the swelling brain by opening the skull. Depending on the location and size of the craniotomy more or fewer brain tissue can extrude from the skull and thereby lower the ICP. The goal of the simulations, that use finite element analysis, was an outcome estimation of different possible craniotomy techniques. The project was initiated and first, approximating simulations were carried out. The *Brain3D/4D* project will be continued in another dissertation[1].

6.1 EyeSi – A Simulator for Intra-ocular Surgery

Operating inside the eye is one of the most demanding tasks in microsurgery: the involved structures are extremely sensitive, the field of view is limited and the operation is performed under a microscope, making hand-eye coordination very difficult. In general, two instruments are inserted into the eye. One is a lamp that lights the operation area. The other one is an operative instrument, used to interact with the pathological tissue. There are several operative devices which mainly differ in the way they interact with the tissue. They range from simple picks and cutters to highly sophisticated instruments like the vitrector, a cutting and sucking instrument which is used to remove material from within the eye. Figure 6.1 illustrates a so called vitrectomy. Typical pathologies which make intra-ocular surgery necessary include opaque vitreous humor, diabetic retinopathy and detached retinas. In case of diabetic retinopathy the surgeon has to peel off pathological membranes covering the retina.

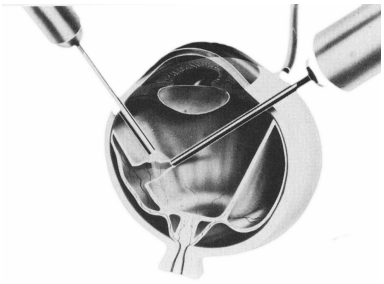

Figure 6.1: Illustration of a vitrectomy. The vitrector is inserted from the right, the lamp from the left side. Picture [Fre85]

Interacting with pathological tissue inside the eye is very difficult. Even the pure navigation of the operative instruments is a delicate task in the limited

[1] look for: Johannes Grimm, Institute for Computational Medicine, University Mannheim, Germany

space available. Collisions with the highly damageable retina can be fatal and have to be avoided in any case. The stereo microscope, used in eye surgery, provides three dimensional view, which is a first means of orientation inside the eye. To estimate the distance between a surgical instrument and the retina, the surgeon additionally uses the strong shadow of the instrument that is cast onto the retina with the lamp.

Currently surgeons acquire practical knowledge in intra-ocular surgery mainly by assisting an experienced surgeon. This takes usually two years and bears risks for the patients and the surgeons. The risky, long, and expensive training can be significantly improved by using a simulation system that provides the surgeon with a realistic operation environment.

6.1.1 Previous Work on Eye Surgery Simulation

There are several projects which have dealt with eye surgery simulation[2] in the past ([oGIaG], [MSBH94], [aUoI]). [MSBH94] and [oGIaG] focus on the simulation of cataract surgery. They either use Finite Element simulations to calculate tissue reaction or previously measured interaction forces. The first could not be accomplished with interactive rates whereas the latter is limited to pre-defined interaction points. In addition [NSM98] describes a current project at the University of Illinois which also aims at the development of a vitrectomy simulator. The project emerged from [aUoI], the development of an anatomical eye atlas. Based on this atlas, the simulation provides detailed graphics which also include the periphery of the eye like eye muscles and parts of the face. In contrast to [oGIaG] and [MSBH94], [NSM98] uses mass spring models to achieve interactive simulation rates. All mentioned projects use OpenGL surface graphics and expensive graphics workstations for rendering.

The Illinois simulator, as presented in [NSM98], lacks an adequate mechanical setup. An eye model is absolutely necessary to provide realistic instrument handling and a simulation of the eye's movement during surgery. Also, including a high precision tracking of instruments- and eye motion is a prerequisite for a correct simulation: the scale of real motion and simulated feed back must be the same.

6.1.2 Surgery in Cyberspace

Simulators using virtual reality, must be constructed in a way that the user feels as if he/she is actually undergoing the simulated situation. It is intended, that the user forgets his/her surrounding and operates completely naturally in

[2]The overview over the state of the art is given at this place instead of chapter 3, because the projects mentioned in this section rather deal with the design of complete simulators than with biomechanical modeling alone.

the virtual reality. All necessary senses of the user must be involved in order to achieve this feeling. Virtual reality aims at merging real sensory perception with virtually presented and by that create a complete involvement (immersion). In order to simulate an intra-ocular operation the stimulation of the visual, haptic and possibly acoustic senses are feasible.

In the case of EyeSi stereoscopic images of the virtual operation scenario are shown on two small LCD displays which are mounted in the shape of a microscope eyepiece. Through these displays the surgeon watches the computer generated operation scenario in the used three-dimensionality. The original surgical instruments are introduced into a mechanical model of the eye. The artificial eye is constructed so that its characteristics of movement correspond to those of the human eye. It is fixed under the mask of a human face. The artificial eye serves as interface to the virtual world. Instruments can be moved in the same way as they are moved when operating a real eye. Rotating the eye out of its rest position generates back-driving forces corresponding to those generated by the muscles of the real eye. An optical tracking system observes the eye model from below. It sends information about the current position and direction of the instruments as well as the orientation of the eye to a PC which updates the computer graphical model. Thus, the movements of the instruments and the mechanical eye cause their virtual counterparts to move correspondingly.

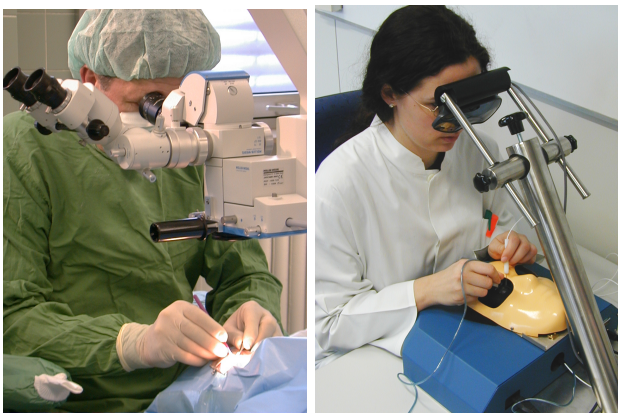

Figure 6.2: An operation in the eye is performed under the stereo-microscope (left side). The simulation (right side) takes all relevant aspects of the surgery into account. Two small LCD displays provide the same three-dimensional view as the microscope and the user works with original instruments in a mechanical model of the eye.

Mechanical Eye: The Interface to Virtual Reality

The mechanical eye is a metallic hemisphere with cardanic suspension which is mounted under a facial mask. It has the same rotational degrees of freedom as the human eye in the eye socket. The effect of the eye muscles, turning the eye back into its rest position, is modeled by a set of springs with appropriate strength which are fixed to each rotational axis. The mechanical eye has two puncture marks through which the surgical instruments are inserted.

The equator of the eye is marked from below with two small color spots. Together with the rotary invariant center of the spherical eye, these markers define the plane that gives the rotational state of the mechanical eye. The surgical instruments are also marked at their tips. Their orientation can be determined with only one marker per instrument, because the exact position of the puncture marks in the mechanical eye are known. The rotational state of non rotational symmetric instruments like, e.g. a vitrector or forceps, is measured with a magnetic field sensor. For this purpose two rare-earth magnets are attached under the mask to generate a suited magnetic field.

Figure 6.3 shows a sequence of pictures illustrating the transition from real to virtual world. The left picture shows the real eye with inserted instruments, the middle picture shows the metallic hemisphere of the mechanical eye which serves as an interface to the virtual world and the right picture shows the virtual eye[3].

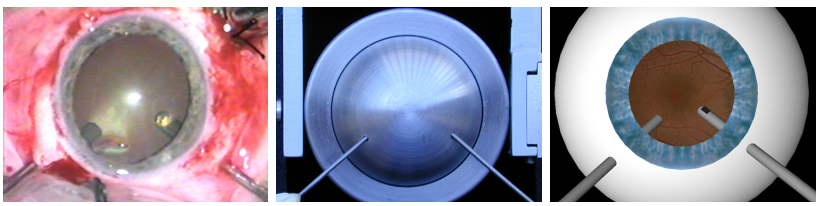

Figure 6.3: The real eye in its orbital cavity has three rotational degrees of freedom (left). In the mechanical eye (middle), this behavior is modeled by a cardanical suspension and springs which pull the eye back into its rest position. The virtual eye (right) with two surgical instruments inserted.

Optical Tracking: Connection between Worlds

During eye surgery two instruments are inserted into the eye. The tip of each instrument has three translational degrees of freedom while being constricted

[3]While operating the microscope is focused through the lense and almost nothing of the eye's periphery is visible.

at its insertion point. In addition the rotation of non-rotational symmetric instruments must be measured. The eye can be tilted in three axes, adding another 3 degrees of freedom to the system. All these movements must be identified in real-time.

Optical tracking systems provide high accuracy and work contact free. Optical tracking is well suited to determine the instruments' positions and the two eye markers (orientation of the eye)[4]. No commercial solution that meets our requirements of narrow setup space (the tracking system has to fit into the body of the simulator) and high accuracy in the small volume of interest, essentially the eye volume (approx. $3 \times 3 \times 3\ cm^3$), is available.

After several experiments with commercial frame grabbers and "intelligent cameras"[5] an FPGA-based image capturing and analyzing hardware was developed [Ruf00]. The latency connected with the conventional image capturing process – composing the image in the memory of the frame grabber, transfering the image into the computer's main memory (via DMA), copying the image from the designated DMA memory area to the application's memory and analyzing the image there – was to high for a comfortable virtual reality[6].

Figure 6.4 (middle) shows a photo of the developed hardware dubbed USB-FPGA. The hardware grabs a camera image (left), analyzes the image doing a color segmentation (right) and sends the sensor coordinates of each colors' center of mass via USB[7] to the PC. The camera sends each pixel of the image[8] with a pixel clock of ca. 12 MHz (for PAL) to the FPGA, which basically classifies the pixel as belonging to a relevant color or not. If the pixel belongs

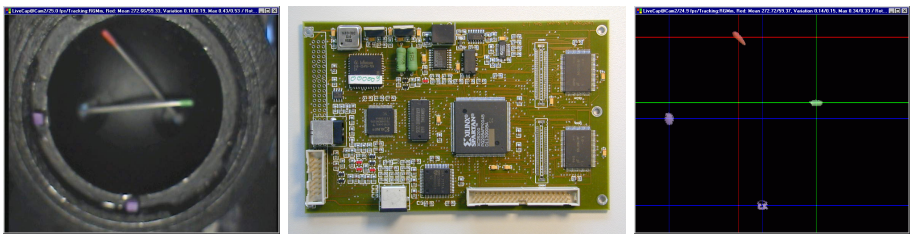

Figure 6.4: The images delivered by a PAL camera (left) are captured and analyzed by the USB-FPGA hardware (middle) and the colors' centers of mass (right) are transferred to the PC.

to a relevant color its sensor coordinates are considered in the color's center

[4] As mentioned above the rotational state of the instruments is measured magnetically.
[5] Camera with integrated rudimentary image processing.
[6] Researchers working in the area of human perception of computer generated images report that minimal update rate that allows a feeling of immersion is 10Hz, but higher update rates (up to 20 Hz) are preferable [Hel93].
[7] Universal Serial Bus
[8] through a video processor which converts analog PAL signal to digital pixels

of mass calculation. The transfer of the colors' centers of mass takes place as soon as the last pixel of a field has been classified. The position information is available to the application after a maximum latency of 2 msec after the end of the field. The latency on the movement of the object is mainly determined by the sampling rate of the used cameras. The EyeSi tracking uses PAL cameras running with a field frequency of 50 Hz. The average latency on object movement is therefore about 12 msec (10 msec average latency while a field is captured + 2 msec for the transfer of the positions to the PC).

Three cameras are mounted below the mechanical eye. Their field of view is adjusted to consider exactly the relevant volume (see figure 6.4; left). They observe the relevant volume from different perspectives. The equator of the eye and the tips of the instruments are marked with color tags (see figure 6.4; left). As the cameras see the scenery from different angles a stereoscopic back projection can be used with two of the camera images to calculate the positions of the markers in space. The third camera is automatically activated when an occlusion occurs on one camera. The cameras were calibrated using the technique introduced in [Tsa87] in a free implementation of the algorithm [Wil95]. Besides the internal camera parameters like the effective focal length and the first coefficient of the radial lens distortion, the algorithm also supplies the external camera parameters: position and rotation regarding a world coordinate system. The external parameters are required for the stereoscopic back projection.

Figure 6.5 illustrates the 3D-reconstruction. In a continuous measurement of a fixed instrument's tip the determined 3D-coordinates vary within 10 μm ($\pm 5\mu m$) in the xy-plane and within 30 μm ($\pm 15\mu m$) in the z-direction (repeat accuracy). Measurement of the absolute accuracy in the required scale is expensive and has not been done so far.

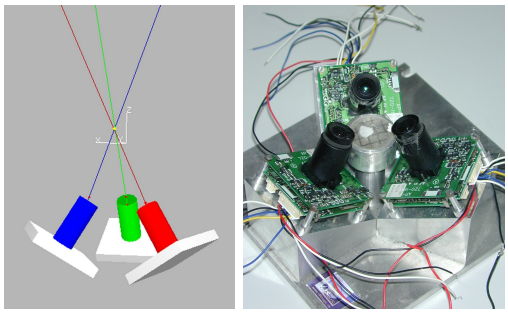

Figure 6.5: 3D-reconstruction with the tracking setup used in EyeSi.

The spatial resolution of the developed tracking system is sufficiently high for the simulated type of retinal surgery. The positioning accuracy of a micro

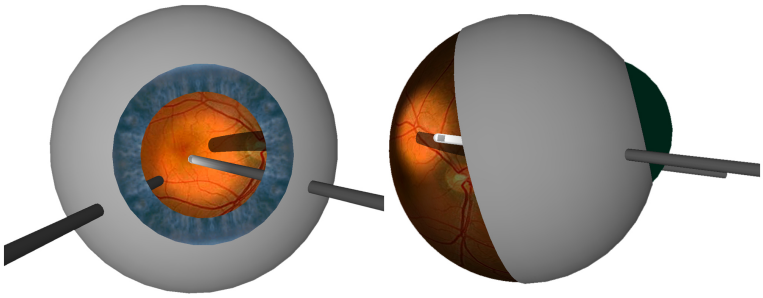

Figure 6.6: A computer graphical eye model was generated following anatomical information from literature. The rear part of the eye is semitransparent to allow a view on the retina. Instruments are inserted into the eye; left is a cold light source, right a vitrector.

surgeon holding the instrument is about 0.1 mm and is superposed by physiological tremor of about 8 – 10 Hz and lower frequency aperiodic erroneous motion [RK99].

Visualization: The Virtual Eye

The optical tracking system registers the positions of the instruments as well as the orientation of the mechanical eye and transmits the information to the computer. The computer modifies the virtual scene correspondingly to the new tracking data. Based on anatomical information[9] a computer graphical model of the eye and several instruments were developed using OpenGL. Texture mapping is used to project real photos of the eye's background and an iris on the appropriate positions in the computer graphical eye. All objects were implemented using the *vrmDesign* introduced in chapter 5. Figure 6.6 shows the eye model.

The graphical model includes light effects and shadows. The lamp produces a spot light and the instrument casts a shadow onto the retina. A realistic shadow generation is very important to the simulation as the shadow is one, of two, important means of navigation for the surgeon. The distance between the instrument's tip and the tip of the shadow provides depth cues. The other one is the 3D-view through the stereo microscope. The stereo buffer of OpenGL is used to produce stereo images. Stereo glasses with two high resolution mini LCDs present the calculated pictures to the user (compare figure 6.2). Each LCD has a resolution of 800×600 pixels. Figure 6.8 illustrates the importance

[9] There is extensive anatomical data available from literature. Gullstrand's norm eye (Allvar Gullstrand ∗1862, †1930; 1911 Nobel price for medicine) comprise detailed information on geometry and refraction indices of materials of the eye.

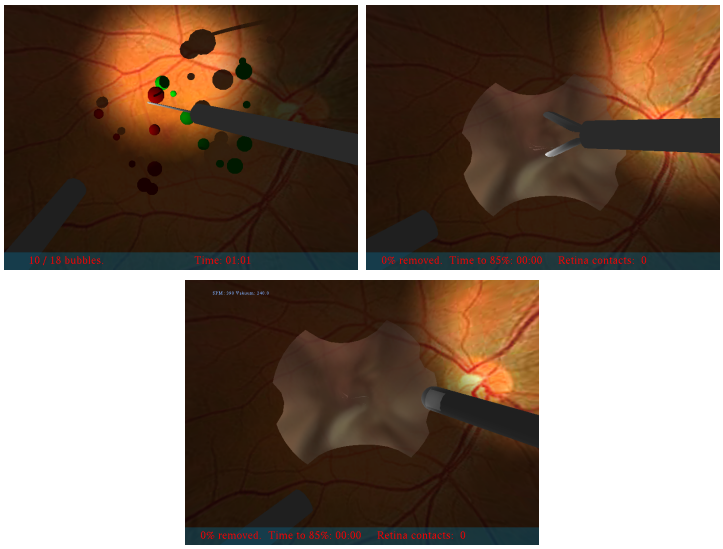

Figure 6.7: Several instruments were modeled. From left to right: a needle, forceps and vitrector. All images are screenshots from simulations.

of the shadow for estimating the distance to deeper structures. The implementation of the shadow follows the volume shadow concept as suggested in [Kil00].

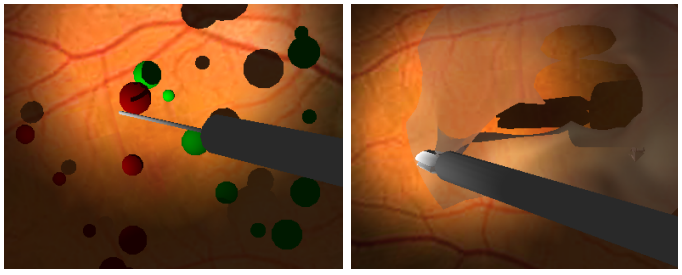

Figure 6.8: Illustration of the importance of shadows for the estimation of distance to deeper structures. All images are screenshots from simulations.

The PC used for the EyeSi simulator is equipped with a graphics adapter from nVidia Corporation. Several of nVidia's graphics extensions are used to speed up rendering. With the current hardware setup, a Pentium III 500 MHz with nVidia Geforce 2[10] the rendering of the scene requires about 40 msec in

[10]The GeForce 2 chip was converted to a Quadro 2 Pro by moving two resistors on the

full simulation mode. Full simulation mode requires four times rendering of the sceen; the shadow generation is a two-pass-process and stereo mode contributes another factor of two.

Biomechanical Simulation: Imitation of Reality

When instruments are moved inside the eye, collisions with structures of the eye may occur. If, e.g. an instrument touches a pathological membrane, the membrane has to change its shape according to the forces exerted by the instruments and the membrane itself. If forces superceed a material specific limit the membrane must tear in. Slight touching of the retina may lead to bleeding which the surgeon perceives as a red spot on the retina. Such events must be visualized immediately. For these reasons it is important that the computer can quickly determine whether and where collisions between virtual objects occurred. Once a collision is detected, displacements and forces acting between the object have to be determined. The forces are used to calculate the reaction of e.g. a pathological membrane which is described by a biomechanical model. In EyeSi, the interaction with a pathological membrane makes high demands on the computer respectively the biomechanical model. The calculation of the biomechanical reaction has to be fast since otherwise the simulation begins to flag and becomes unrealistic. Evidently, the speed of the calculation depends on the complexity of the membrane's mathematical description which in turn depends on the approach chosen for the biomechanical model of the membrane.

Where feasible, simple geometric intersection calculations are used to detect collisions between instruments and eyeball. Due to the spherical shape of the eye ball this can be accomplished very fast. As soon as collisions with more complex shapes have to be considered, e.g. pathological tissues in the eye, collision detection becomes more expensive. In order to accelerate collision detection in these cases a new, graphics-hardware supported approach for collision detection was developed[11]. The basic idea of the approach is to place an additional camera (collision camera) within e.g. an instrument which should always be oriented towards the tip of the instrument. The images that the collision camera sees changes as soon as a collision occurs at the tip of the instrument (the membrane suddenly becomes visible to the collision camera as it permeates the instrument). The charm of this approach lies in the fact that the collision calculations are carried out by the GPU[12] and not the host's CPU. The approach is particularly superior to existing collision detection algorithms (e.g. RAPID [GLM96], SOLID [vdB97]) when deformable objects are observed; the time required for the collision calculation depends only on the number of triangles in the scene. Thus, the described collision detection, which

board. Instructions from `http://www.guru3d.com`

[11] *Hidden-Surface Based Collision Detection with 3D-Graphics Accelerators:* C. Wagner, M. Schill, R. Männer, to be published.

[12] The GPU is highly optimized for these calculations.

would only be used in simulations with deformable objects, costs an additional rendering pass. With the current graphical model and hardware one rendering pass takes about 10 msec.

As for the need of a fast response to a collision, in EyeSi two descriptive biomechanical modeling approaches were applied: mass-spring and Enhanced ChainMail. The latter was primarily intended for the modeling of the vitreous humor in the eye. Simulation of the vitreous humor requires volume visualization which has not yet been incorporated into the *vrmDesign* and therefore is not part of the EyeSi simulator at its current state. Nevertheless, ECM has also been tested for its suitability to model membranes. The Enhanced ChainMail algorithm generally has advantages over mass-spring regarding numerical stability[13] (see page 67).

Currently pathological membranes in EyeSi are simulated with a mass-spring model. It produces good results within the known limitations of mass-spring models (see below). The model consists of a triangulated mesh of mass points and springs. The governing equation is (compare: equation 3.16)

$$\vec{F_i} = \left(\sum_{j}^{N} \frac{\vec{x_j} - \vec{x_i}}{\|\vec{x_j} - \vec{x_i}\|} (k_{i,j} \|\vec{x_j} - \vec{x_i}\| - R_{i,j}) \right) - D_i \dot{\vec{x}_i} \tag{6.1}$$

where $\vec{F_i}$ is the force on mass i, $k_{i,j}$ is the spring constant of the spring between i and j, $R_{i,j}$ is the length of this spring and D_i the damping connected with spring i.

The biomechanical properties of the model were derived phenomenologically in an iterative process together with an experienced surgeon. In the sense of the concept introduced in section 2.2 the implemented mass-spring model represents a descriptive approach. However, convincing behavior of soft membranes was achieved. Figure 6.9 shows an example of membrane peeling off the retina. A membrane tissue with approximately 1,500 mass nodes can be solved within < 10 ms.

The drawbacks of conventional mass-spring systems were experienced when simulating stiffer structures. The system then showed a tendency to numerical instability. Moreover, also in cases where soft membranes were simulated, well conditioned boundary constraints had to be chosen: to avoid a collapse of the triangulated mesh the tissue had to be fixed along its rim. In addition the stability was improved by setting the velocities and accelerations of all nodes equal zero after each timestep. By this the dynamic effects in the mass-spring system were lessened, which even had a positive effect on the membrane's appearance; its movements became more realistic.

Figure 6.10 gives a comparison between simulation and real operation. The left column shows video captures from real surgery, the right column screen shots

[13] This is particularly true for 3D modeling.

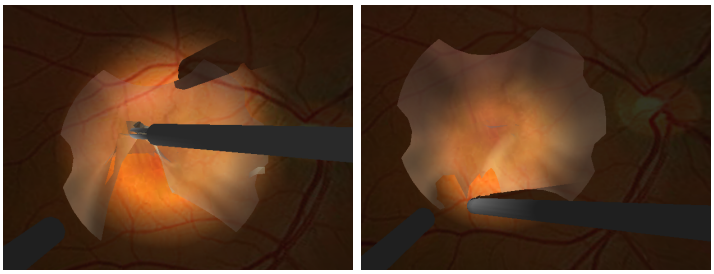

Figure 6.9: Simulation of membrane peeling with forceps (left) and with vitrector (right).

from the simulator. The images demonstrate the degree of realism reached with the simulator: light effects, shadow, eye's background and instruments were modeled in conformity with the real surgery.

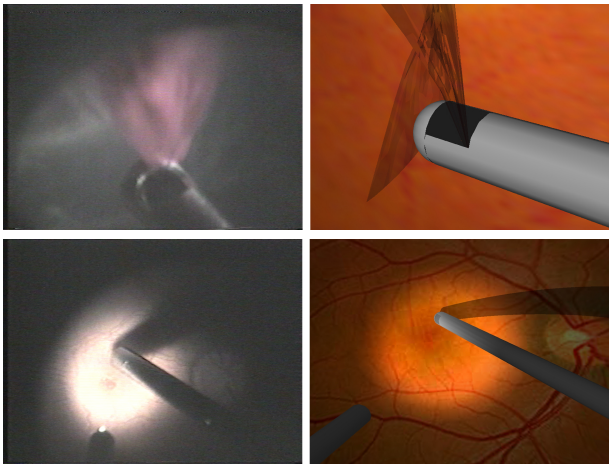

Figure 6.10: Video captures (left) and simulation snapshots (right) showing membrane peeling (top) and instrument navigation (bottom).

One advantage of virtual reality vs. reality is that a difficult surgery can easily be subdivided into less complex tasks which can be trained independently from each other. In virtual reality it is also possible to create abstract training tasks that do not have to exist in reality but which emphasize a particular training aspect. To exercise bi-manual instrument handling an abstract task was implemented where the user has to move the tip of a needle into a sphere and hold it steady within the sphere for a defined period. After this period the sphere changes its color and the user can proceed to the next sphere. Spheres of

varying size are distributed over the eye; the smaller and closer to the retina, the more difficult to treat. Figure 6.8 (right) shows a screenshot of the navigation training session.

6.1.3 Summary

EyeSi is a virtual reality simulator for the training of intra-ocular surgery which incorporates all essential details from the real operation scenario like stereo-microscope setup, original surgical instruments and mechanical eye which behaves like a real eye and serves as an interface to the simulator.

Currently it is possible to exercise instrument-navigation inside the narrow space of the eye. In addition the simulator allows the training of membrane peeling off the retina. Abstract tasks were implemented where helpful to emphasize particular training effects. Biomechanical models allow interaction with pathological tissues. Two descriptive approaches were included in the simulator: mass-spring and Enhanced ChainMail. ECM will be used as soon as volume visualization is available in the *vrmDesign* to model the vitreous humor. ECM has considerable advantages over mass-spring regarding stability, which particularly comes to play when three-dimensional objects are modeled.

Great importance is attached to the handling of instruments during the simulation. The mechanical eye (see Figure 6.3 (middle)) provides a passive tactile feedback similar to what a surgeon experiences when operating a real eye. The instruments are optically tracked when introduced into the mechanical eye and can be moved and rotated in a natural way.

The platform of EyeSi is off-the-shelf pc hardware, currently an Intel Pentium III CPU and a nVidia GeForce 2 GPU. In full simulation mode (interacting with membrane and stereo rendering turned on) the framerate is above 20Hz. It reaches up to over 50Hz in mono mode.

We have not yet finished clinical evaluation studies, the prototype is currently tested in clinical practice.

Future developments will include a training curriculum for surgeons in education. The program will provide a performance analysis informing the trainee about his training success in terms of accuracy and time needed to complete a certain task.

In addition to training the simulator can also be used for the development of new operative instruments and surgical techniques. New instruments can rapidly be prototyped in a simulation. New operation techniques could be developed, tested, demonstrated and taught to colleagues.

6.2 Simulation of Decompressive Craniotomy

Nature protected the highly sensitive brain by surrounding it with firm bone (figure 6.11 (left). In general the skull keeps the brain save from physical disturbance, but in cases of increasing intra-cranial pressure (ICP) the skull turns out to be a very confined space. Space demanding processes, like e.g. edemas or hematomas can rise ICP to a life threatening level at which the sufficient supply of the brain is no longer granted. In such cases neurosurgeons open the skull. The surgery, a decompressive craniotomy, reduces the pressure inside the skull by providing additional volume for the swelling brain; brain can swell out of the skull. While planning decompressive craniotomy the surgeons face questions like: Where to open the skull? How large should the opening be made? How can areas of high stress be kept away from functional centers of the brain? There are mainly two common techniques for carrying out a decompressive craniotomy. Depending on where the surgeon opens the skull they are called bi-frontal or lateral craniotomy. Figure 6.11 shows a bi-frontal (middle) and a lateral opening (right) of the skull.

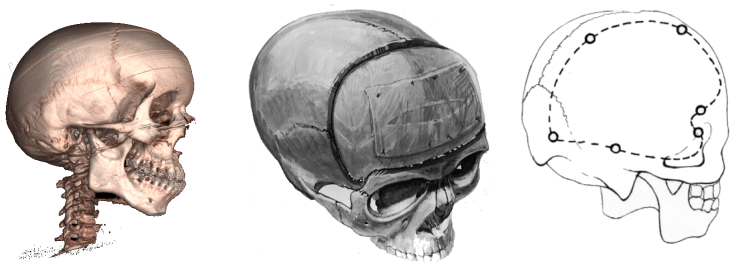

Figure 6.11: Left: The skull is a firm bone which protects the highly sensitive brain from physical disturbance (image courtesy Volume Graphics). Middle: Illustration of bi-frontal craniotomy. Right: Illustration of lateral craniotomy.

The goal of simulating a decompressive craniotomy is to support the surgeon in planning the surgery. As a first step towards simulating the complete problem it was investigated whether there is a significant difference between the two most common techniques regarding the ICP decrease they can generate[14].

6.2.1 Method

The software architecture presented in chapter 5 was used to develop a volumetric finite element model to simulate a decompressive craniotomy. The

[14]The whole problem is further investigated in a dissertation by Johannes Grimm at the Institute for Computational Medicine, University Mannheim.

brain was segmented from an MRI data set (figure 6.12 (left)) using VGStudio[15] and scaled to a standard volume of 1,700 ml. It was then discretizised with tetrahedron elements. In addition the Dura mater, the strong leather-like skin surrounding the brain was modeled with triangular elements. When the bone is removed the Dura mater plays an important role in holding the brain in place. Both element types use the assumption of linear material response. The tetrahedron element is a 4-node element which uses a linear interpolation function. The triangular element is a 6-node element which uses linear interpolation functions for in-plane strain and cubic interpolation functions for bending. The element matrices for both elements are given in appendix B. The two element types were combined in the simulation. Such a combination of elements can easily be realized with the concept of the *vrmDesign*. On the node-connector abstraction level the two element types can be regarded as *vrm3Connector* (triangle)[16] and *vrm4Connector* (tetrahedron). The two connector types form different topologies on the same set of nodes (compare section 5.1.1); the tetrahedrons discretizise the volume whereas the triangles discretizise the surface of the brain. The triangles connect a subset of all nodes, i.e. the nodes that lie on the surface of the brain and represent the Dura mater. Figure 6.12 (right) shows the brain model used in the simulation. It consists of 34,000 tetrahedrons and 8,000 triangular elements.

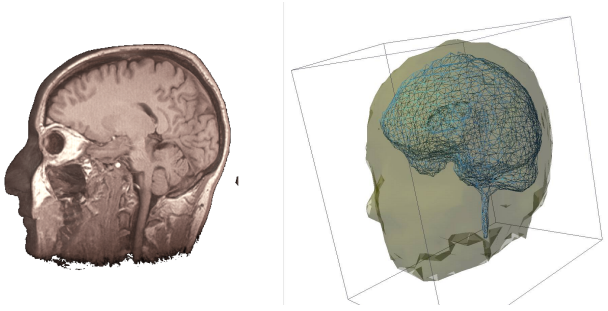

Figure 6.12: Original MRI data set (left) and brain model derived from the MRI scan (right)

Two material parameters are required for the simulation. The Poisson's ratio ν and the Young's modulus E completely describe the elastic behavior of an object. The specific material parameters of brain tissue are controversial. The values used in this study are based on measurements by Patin et. al. [PEHP93]. For brain tissue E was set to 6 kPa and for the Dura mater an E of 2.5 MPa was used. The Poison's ratio ν was set to 0.48 for both materials. As common in

[15] www.volumegraphics.com

[16] Since node variables are also present on the edges of the triangle a *vrm6Connector* could also be used. In the actual implementation the edge nodes are included into a special kind of 3Connector.

cases that lead to decompressive craniotomies, a homogeneous pressure inside the skull was assumed. In such cases the high ICP is caused by a global edema. If the skull is assumed to be a closed shell, a volume increase ΔV induces a pressure increase following the relation:

$$\begin{aligned} \frac{\Delta V}{V} &= -\frac{1}{K}\Delta p \\ &with: \\ K &= \frac{1}{3}\frac{E}{1-2\nu} \end{aligned}$$

The development of a global edema was simulated by applying a pressure inside each tetrahedron and calculating the force on the tetrahedron's surfaces. The surface force was then distributed on the elements' nodes. For the simulation of a homogeneous ICP the pressure in each element was set equally. The model, however, also allows the simulation of pressure gradients inside the skull, since the pressure can be set separately for each element.

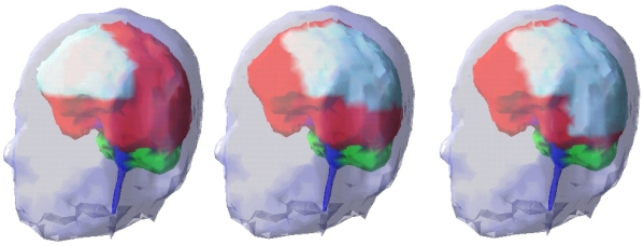

Figure 6.13: In simulations three different craniotomies were considered. The areas where the bone was removed in the simulations are marked in bright color. The figure shows a bi-frontal opening (left), a lateral opening (middle) and a lateral opening including the temporal lobe (right)

With the model the two steps of performing a decompressive craniotomy, (1) removal of the bone and (2) opening and expanding the Dura mater with synthetic patches were simulated at different ICP levels. The two most common ways of opening the skull, bi-frontal and lateral, were compared in simulations. In addition a third method, a lateral opening including the temporal lobe, was simulated. This third opening was suggested by neurosurgeons because they assume that an opening which would allow the temporal lobe to be shifted to the front, as a whole, could be beneficial to the total pressure decrease gainable by the surgery. Figure 6.13 shows the three investigated openings.

6.2.2 Sensitivity analysis

To gain more intuition on the way the physical parameters Poisson's ratio ν and Young's modulus E influence the simulation results a sensitivity analysis

on these parameters was carried out. The craniotomy simulation was carried out under fixed condition with either varying ν or E. Intra-cranial pressure was set to 20 mmHg. While ν was varied, E was set to 6kPa and while E was varied ν was fixed at 0.47. Table 6.1 and 6.2 present the results. Figure 6.14 shows them graphically. Incompressible material has a ν close to 0.5 (compare

ν	$\Delta V/V$	Δp [mmHg]	$p_0 - \Delta p$ [mmHg]
0.47	6.16%	15.39	4.61
0.40	16.07%	12.05	7.95
0.30	27.38%	10.27	9.73
0.20	37.57%	9.39	10.61

Table 6.1: While all other simulation parameters are kept constant, the Poisson's ratio ν was varied.

E [Pa]	$\Delta V/V$	Δp [mmHg]	$p_0 - \Delta p$ [mmHg]
10,000	3.64%	15.16	4.84
9,000	4.05%	15.20	4.80
8,000	4.57%	15.25	4.75
7,000	5.25%	15.31	4.69
6,000	6.16%	15.39	4.61
5,000	7.44%	15.51	4.49
4,000	9.41%	15.69	4.31
3,000	12.79%	15.99	4.01

Table 6.2: While all other simulation parameters are kept constant, the Young's modulus E was varied.

section 2.3). It is well understandable that a material that is less compressible (large ν) shows a large pressure decrease Δp[17]. It is very interesting that Δp the ICP decrease reached with the craniotomy does merely depend on the Young's modulus E. E only has an effect on the amount of tissue that protrudes from the skull. This observation is important for the validation, where the amount of tissue that protrudes is determined (see below).

6.2.3 Preliminary Results

The simulation was carried out for several levels of raised ICP. An ICP of 10 mmHg is regarded as normal, an ICP of 40 mmHg can be called high whereas 80 mmHg are considered critical. Exemplary simulation results are presented in

[17]The opposite case is maybe more intuitive: If you press your finger into a balloon filled with water (incompressible) the balloon will at once bulge out at another place.

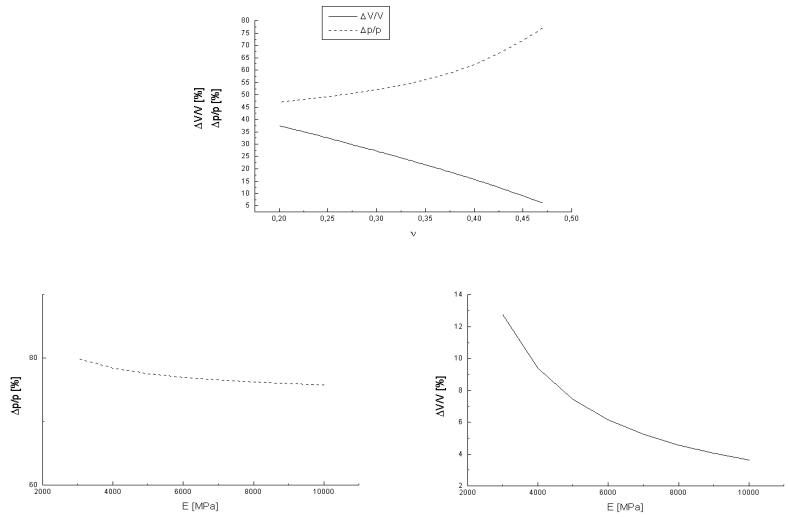

Figure 6.14: Graphical presentation of the results of the sensitivity analysis. The simulation relevant value, ICP decrease (Δp) depends almost exclusively on the size of the Poisson's ratio, whereas the Young's modulus has an effect on the size of the volume that swells out of the skull.

table 6.3. All results lie in a reasonable range and were considered as medically realistic.

Figure 6.15 shows a simulation result graphically. The three images on the left show the brain surface prior to surgery (blue) and after the complete opening – bone and Dura mater – (transparent) for a lateral opening of the skull. The initial ICP in the presented images was 40 mmHg. The larger image on the right shows the displacement vectors of the nodes.

6.2.4 Discussion

As observed in real surgery, the simulations reproduced that the bone removal leads only to a small decrease in ICP. The major effect is achieved by opening and extending the Dura mater. Comparing the two standard methods bi-frontal and lateral opening of the skull suggests that the lateral opening is considerably superior with respect to decreasing ICP. The areas of the two openings compared here were approximately of the same size. As expected a larger lateral opening results in a higher pressure decrease. The simulation of the larger lateral opening (including the temporal lope) showed that regarding ICP levels this approach brings only marginal improvement. The major benefit

	bone removal		Dura mater opening	
ICP(t=0) [mmHg]	ΔV/V [%]	ICP(br) [mmHg]	ΔV/V [%]	ICP(dmo) [mmHg]
	bi-frontal opening			
10	0.4	9.1	1.5	6.2
40	1.5	36.2	6.7	23.1
80	3.2	72.1	15.2	41.9
	lateral opening			
10	0.9	7.6	2.1	3.1
40	3.9	30.2	11.8	10.6
80	8.2	59.4	25.9	15.3
	lateral opening incl. temporal lobe			
10	1.1	7.2	3.0	2.5
40	4.6	28.6	12.8	8.0
80	9.5	56.2	27.5	11.2

Table 6.3: ICP(t=0) denotes the intra cranial pressure prior to the surgery, ICP(br) after bone removal, ICP(dmo) after Dura mater opening. ICP is measured against atmospheric pressure. ΔV/V gives the volume change in %.

of an opening that includes the temporal lope might be an advantageous stress distribution in the tissue since the lobe as a whole could be shifted.

The presented simulation results still have a number of limitations. Linear and homogeneous material properties were assumed to be valid for the whole brain and for all displacements. This obviously is a simplification. Internal structures like ventricles or the Falx cerebri are not yet considered in this simulation. However, finite element simulations of the Falx cerebri were carried out in previous work [SSBM96] and will be adapted to the *vrmDesign* and included into the simulation. Also, currently the brain is treated as being one continuous piece of tissue. However, the brain's temporal lobe e.g. is only connected to the rest of the brain in the ventral part. It lies on the rest of the brain without a connection in the frontal part. This is the cause why neurosurgeons expect that including the temporal lobe in the craniotomy could be beneficial to the stress distribution in the tissue. The correct anatomy will be considered in the next model.

Current work also includes the validation of the model. For this purpose a 3D-scanner was placed in the operating room to measure the volume protruded from the skull after bone removal and Dura mater opening. Figure 6.16 shows a 2D image of such a scan. 3D-scans taken after different steps during a real surgery will be used together with an ICP monitor to validate the model.

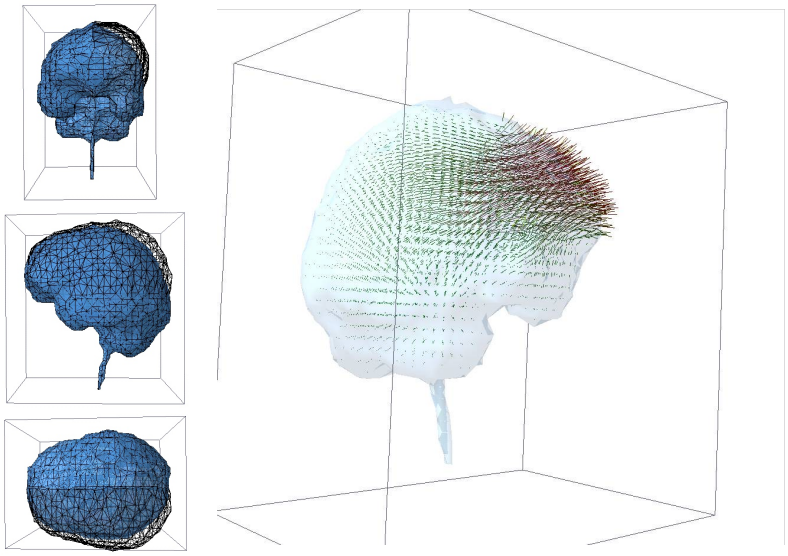

Figure 6.15: Graphical simulation result for a lateral opening of the skull with an initial ICP of 40 mmHg after Dura mater opening. The three images in the left column show the original brain surface (prior to opening) in blue and in transparent the surface of the brain after the swelling process came to an end. The image on the right shows the displacement vectors of the nodes which correspond to the shift of tissue that was induced by the opening.

Unfortunately so far no relevant craniotomies were performed while the scanner was setup in the OR.

In its current state the model allows a comparison between the two standard operation techniques for decompressive craniotomies. Estimations of the ICP decrease in dependence of the size and position of the opening are possible.

6.3 Summary

Two applications using biomechanical soft tissue models were presented in chapter 6: an intra-ocular surgery simulation and the simulation of decompressive craniotomy. The two applications make different demands on the biomechanical models used to simulate the soft tissues present in the particular case.

The presented simulator for eye surgery, *EyeSi*, was developed as a training tool for students and physicians. *EyeSi* makes heavy use of Virtual Reality to generate the highest possible degree of realism. One of the most important constraints for a simulator like *EyeSi* is real time performance. The real

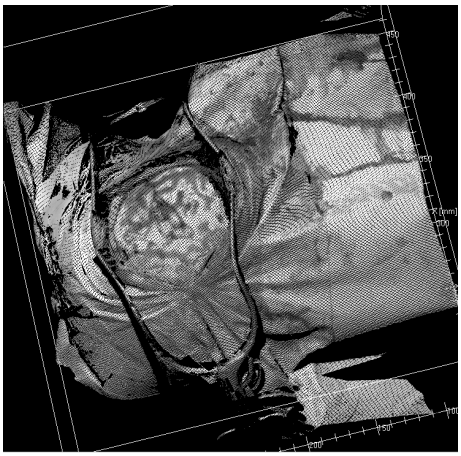

Figure 6.16: Three dimensional scans of the brain's surface are performed at different steps of the surgery. Such scans will be used to determine the volume protruded from the skull. In connection with an ICP monitor this will allow a validation of the model.

time constraint is particularly true for the biomechanical simulation. Therefore descriptive modeling approaches are used for tissue modeling. Enhanced ChainMail will be used for the modeling of the vitreous humor in the eye, as soon as volume visualization will be integrated into the underlying architecture, the *vrmDesign* (see chapter 5). A mass-spring model is used to simulate the interaction between surgical instrument and pathological membranes in the eye. The model governing parameters, spring constants and masses, were adjusted together with experienced ophthalmo-surgeons to match the behavior of real membranes. However, descriptive models can generally only act qualitatively like real tissue.

The simulation of craniotomy aims at a quantitative analysis. In contrast to the simulator *EyeSi*, time constraints are no issue and physical approaches can be followed when modeling the soft tissue. The two standard ways of performing decompressive craniotomies were compared in FEM simulations to decide which technique is superior. The brain tissue and the Dura mater, a strong skin surrounding the brain, were modeled with finite tetrahedron, respectively triangular elements. The biomechanical parameters used for the simulations are actually measured values, that were taken from literature. To cope with the great uncertainty that is assigned to published biomechanical properties (compare table 3.1 on page 23) a sensitivity analysis on these parameters was performed beforehand. The pressure decrease Δp that is achieved through the craniotomy is mainly sensitive to the Poisson's ration ν and almost independent from Young's modulus E, whereas the change in volume mainly depends

on E. This dependency becomes relevant when the model is validated. For this purpose a 3D-scanner is placed in the operating room to directly measure the volume protruding from the opening.

Both simulations were implemented using the software architecture presented in chapter 5.

7

Summary

The capability to visualize medical data sets in real time and the possibility to present these images three-dimensionally in the space right in front of the observer immediately creates the desire to touch, feel and interact with the presented organs. Interaction, however, comprises that the organ reacts to the forces applied to it; it must deform in a natural way when being touched. The reaction of soft tissue to applied forces can be calculated with biomechanical simulation algorithms. Several modeling approaches exist. A scheme is suggested which allows the classification of arbitrary modeling approaches with respect to the degree of physical realism contained in the model. Two definitions are given to characterize a *physical model* on the one hand and a *descriptive model* on the other hand. The parameters that govern the behavior of physical models are physical values that can be measured in the real world. An analysis of the available modeling techniques on the basis of the introduced definitions yields that models with initially physical concept are very often used to model processes they were not made for. However, the descriptive use of physical models, or the direct use and development of descriptive models is important for biomechanical modeling for one reason: calculation speed! The calculation time required to solve a physical model generally surmounts the time available in any real time simulation task.

Besides well known approaches like mass-spring, finite element, particle models and others the ChainMail algorithm is investigated. Where ChainMail in its original formulation lacked the capability to model inhomogeneous material, it is exceptionally stable and converges in one step to a valid configuration. However, this configuration does not necessarily represent an optimal energy distribution in the object. Therefore ChainMail is generally followed by an elastic relaxation. In this thesis ChainMail is generalized to the Enhanced ChainMail (ECM) algorithm which is capable to model inhomogeneous, volumetric objects and is fast enough for real time simulations. The ECM algorithm

is derived using the physical analogy of sound wave propagation in material. Two and three dimensional objects can interactively be deformed with ECM. The algorithm was applied to various soft tissue objects as for example a ct data set of a jaw. A performance analysis was conducted and revealed a linear dependency on the number of elements affected by the deformation. As the three spatial deformation directions can be calculated separately the algorithm is well suited for parallelization. The original ChainMail used a geometric consideration to relax the object to a homogeneous energy distribution. In the inhomogeneous case the varying material properties are to be considered in the relaxation process. Two approaches, which are both based on the mass-spring idea, were implemented for the inhomogeneous relaxation and provide natural object behavior.

While now in principle being able to simulate and visualize an object in real time, a software architecture is required to team up simulation and visualization. As visualization and simulation have so far evolved independently, they work with different data structures. Multiplicity of data representations leads to the problems of data consistency and high memory consumption. Using the principles of object oriented analysis and design, a software architecture is developed which provides a universal data structure for several simulation and visualization approaches. Biomechanical simulation algorithms including Enhanced ChainMail, mass-spring and finite elements can be run on the same data together with any visualization technique. Even the combination of different biomechanical algorithms in one simulation has been realized. An Enhanced ChainMail configuration was relaxed with a mass-spring model.

The versatility of the developed architecture is demonstrated by two medical simulations which represent the two opposite positions of the scheme introduced in the beginning. The first is the simulation of an intra-ocular surgery, which makes heavy use of Virtual Reality techniques. Designed as a training and educational tool the simulator *EyeSi* relies on descriptive real time tissue simulation and visualization. The second deals with the simulation of decompressive craniotomy. The medical problem requires a physical model as the project's goal is to provide exact predictions on tissue behavior to support surgeons in surgery planning.

Bibliography

[ARW+99] L.M. Auer, A. Radetzky, C. Wimmer, G. Kleinszig, F. Schroecker, D.P. Auer, H. Delingette, B. Davies, and D.P. Pretschner. Visualization for planing and simulation of minimal invasive neurosurgical procedures. In Chris Taylor and Alan Colchester, editors, *miccai1999*, volume 1679 of *Lecture Notes in Computer Science*, Cambridge; UK, September 1999. Springer.

[aUoI] Biomedical Visualization Laboratory at University of Illinois. Model of the Eye. `http://www.bvl.uic.edu/bvl/eye`.

[Bar84] A. H. Barr. Global and local deformations of solid primitives. In *Proc. of SIGGRAPH 84*, volume 18 of *Computer Graphics, Annual conference series*, pages 21–30, July 1984.

[Bat90] Klaus-Jürgen Bathe. *Finite-Elemente-Methoden*. Springer, 1990.

[Bli82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. on Graphics*, 1(3):235–256, July 1982.

[Blo88] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[BN95] Morten Bro-Nielsen. Modelling elasticity in solids using active cubes – application to simulated operations. In Nicholas Ayache, editor, *Proc. CVRMed '95*, pages 535–541, 1995.

[BNC96] M. Bro-Nielsen and S. Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum*, 25(3):57–66, 1996.

[Boo01] Grady Booch. Developing the future. *IEEE Comunications*, 44(3):119–121, March 2001.

[Bos92] I.E. Boston. *Transient Stress Analysis by the Transmission Line Matrix Metho*. PhD thesis, University of Hull, 1992. Reference found in: [LWA].

[BPW93] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, 1993.

[BS91] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251–256, July 1991. (SIGGRAPH'91 proceedings).

[BSV+98] Jon Burgin, Bryan Stephens, Farida Vahora, Bharti Temkin, and William Marcy. Haptic rendering of volumetric soft-bodies objects. In J.K. Salisbury and M.A. Srinivasan, editors, *Proc. of the third PHANTOM Users Group Workshop*, The Artificial Intelligence Laoratory and The Research Laoratory of Electronics at MIT, Cambridge, MA, USA, 1998.

[Bur87] D.S. Burnett. *Finite Element Analysis: From Conception to Applications*. Addison-Wesley, Reading, 1987.

[BW90] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(2):109–116, March 1990.

[CDA96] S. Cotin, H. Delingette, and N. Ayache. Real time volumetric deformable models for surgery simulation. In *Vizualisation in Biomedical Computing*, pages 535–540. Springer, 1996.

[CDA99] S. Cotin, H. Delingette, and N. Ayache. Real-time elastic deformations of soft tissuees for surgical simulation. *IEEE Transactions On Visualization and Computer Graphics*, 5(1):62–73, Jan–Mar 1999.

[CDBN+96] S. Cotin, H. Delingette, M. Bro-Nielsen, N. Ayache, J.M. Clement, V.Tassetti, and J. Marescaux. Geometrical and physical representations for a simulator of hepatic surgery. In *Medicine Meets Virtual Reality: Health care in the information age*, number 29 in Technology and Informatics, pages 139–151, Amsterdam, Oxford, Tokyo, Washington, 1996. IOS Press.

[CEO+93] S. Cover, N. Ezquerra, J. O'Brien, R. Rowe, T. Gadacz, and E. Palm. Interactively deformable models for surgery simulation. *IEEE Computer Graphics & Applications*, pages 68–75, November 1993.

[CG91] G. Celniker and D. Gossard. Deformable curve and surface finite-elements for free-form shape design. In *Proc. of SIGGRAPH 91*, Computer Graphics Proceedings, Annual conference series, pages 257–266. ACM, 1991.

[CJ91] S. Coquillart and P. Jancene. Animated free-form deformation: An interactive animation technique. *Computer Graphics*, 25(4):23–26, 1991.

[CM82] G. W. Christie and I. C. Medland. A non-linear finite element stress analysis of bioprosthetic heart valves. In R. H .Gallagher, B. R. Simon, P. C. Johnson, and J. F. Gross, editors, *Finite Elements in Biomechanics*. John Wiley & Sons, Chichester, UK, 1982.

[CZ92] David T. Chen and David Zeltzer. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. In *SIGGRAPH '92*, number 26, in 2, pages 89–98, July 1992.

[DCA99] H. Delingette, S. Cotin, and N. Ayache. A hybrid elastic model allowing real time cutting, deformations and force-feedback for surgery training and simulation. In *Computer Animation*, Geneva, Switzerland, Mai 26–28 1999.

[dCL99] F. Boux de Casson and C. Laugier. Modeling the dynamics of a human liver for a minimal invasive surgery simulator. In Chris Taylor and Alan Colchester, editors, *miccai1999*, volume 1679 of *Lecture Notes in Computer Science*, Cambridge; UK, September 1999. Springer.

[Del98] Herve Delingette. Towards realistic soft tissue modeling in medical simulation. Technical Report 3506, INRIA, Sophia-Antipolis, September 1998.

[Deu96] Oliver Deussen. *Untersuchung effizienter Verfahren zur Bewegungssimulation deformierbarer Körper*. Number 215 in Reihe 20. VDI Verlag GmbH, Düsseldorf, 1996.

[DG94] Mathieu Desbrun and Marie-Paule Gascuel. Highly deformable material for animation and collision processing. In *5th Eurographics Workshop on Animation and Simulation*, Oslo, Norway, September 1994.

[DG95] Mathieu Desbrun and Marie-Paule Gascuel. Animating soft substances with implicit surfaces. *Computer Graphics*, 29:287–290, 1995.

[ESP92] I. Essa, S. Sclaroff, and A. Pentland. A unified approach for physical and geometric modeling for graphics and animation. In *Proc. of Eurographics*, volume 11, pages C129–C138, 1992.

[ESP93] I. Essa, S. Sclaroff, and A. Pentland. *Directions in Geometric Computing*, chapter Physically-based Modeling for Graphics and Vision. Information Geometers, Winchester, UK, 1993.

[Far90] G. Farin. Surface over dirichlet tessellations. *Computer Aided Geometric Design*, (7):281–292, 1990. NorthHolland.

[Fre85] Heinrich Freyler. *Augenheilkunde für Studium, Praktikum u. Praxis*. Springer, Wien, 1985.

[Gib97a] S. Gibson. 3d chainmail: a fast algorithm for deforming volumetric objects. In *Symposium on Interactive 3D Graphics*, pages 149–154, 1997.

[Gib97b] Sarah F. F. Gibson. Linked volumetric objects for physics-based modeling. Technical Report TR97-20, MERL – Mitsubishi Electric Research Laboratory, Cambridge, U.S.A., 1997.

[Gib00] Sarah F. F. Gibson. Using linked volumes to model object collision, deformation, cutting, carving and joining. Technical Report TR-2000-24, MERL – Mitsubishi Electric Research Laboratory, Cambridge, U.S.A., 2000.

[GLM96] S. Gottschalk, M.C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings SIGGRAPH '96*, pages 171–180, 1996.

[GM97] S.F. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical Report TR-97-19, MERL – A Mitsubishi Electric Research Laboratory, 1997.

[GMTT89] J.P. Gourret, N. Magnenat-Thalmann, and D. Thalmann. Simulation of object and human skin deformations in a grasping task. In *Proc. of SIGGRAPH 89*, Computer Graphics Proceedings, Annual conference series, pages 21–30. ACM, 1989.

[Gol87] Herbert Goldstein. *Klassische Mechanik*. Aula-Verl., Wiesbaden, 1987.

[Gri00] Johannes Grimm. Simulation eines diffus schwellenden Gehirns mit der Methode der finiten Elemente. Master's thesis, Universität Mannheim, 2000.

[Har99] Ulrich Hartmann. *Ein mechanisches Finite-Element-Modell des menschlichen Kopfes*. PhD thesis, Universität Leipzig, 1999.

[HBHP90] X. Huang, M. M. Black, I. C. Howard, and E. A. Patterson. A two-dimensional finite element analysis of a bioprosthetic heart valve. *J. Biomechanics*, (23):753–762, 1990.

[Hel93] J. L. Helman. Designing VR systems to meet physio- and psychological requirements. ACM – Siggraph'93 Course 23 – Applied Virtual Reality, 1993.

[HHK92] W. Hsu, J. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, 26(2):177–182, July 1992.

[HLL+00] H.M. Huang, M.C. Lee, S.Y. Lee, W.T. Chiu, L.C. Pan, and C.T. Chen. Finite element analysis of brain contusion: an indirect impact study. *Medical & Biological Engeneering & Computing*, 38:253–259, 2000.

[HPP+00a] Karl Heinz Höhne, Bernhard Pflesser, Andreas Pommert, Kay Priesmeyer, Martin Riemer, Thomas Schiemann, Rainer Schubert, Ulf Tiede, Hans Frederking, Sebastian Gehrmann, Stefan Noster, and Udo Schumacher. *VOXEL-MAN 3D Navigator: Inner Organs. Regional, Systemic and Radiological Anatomy.* Springer-Verlag Electronic Media, Heidelberg, 2000. (3 CD-ROMs, ISBN 3-540-14759-4).

[HPP+00b] Karl Heinz Höhne, Bernhard Pflesser, Andreas Pommert, Martin Riemer, Rainer Schubert, Thomas Schiemann, Ulf Tiede, and Udo Schumacher. A realistic model of the inner organs from the Visible Human data. In Scott L. Delp, Anthony M. DiGioia, and Branislav Jaramaz, editors, *Medical Image Computing and Computer-Assisted Intervention, Proc. MICCAI 2000*, volume 1935 of *Lecture Notes in Computer Science*, pages 776–785. Springer, Berlin, 2000.

[HvCAH91] J. M. Huygue, D. H. van Campen, T. Arts, and R. M. Heethaars. A two-phase finite element model of the diastolic left ventricle. *J. Biomechanics*, (24):527–538, 1991.

[JBM73] R. F. Janz, R. K. Bruce, and T.F. Moriarty. Deformation of the diastolic left ventricle - part ii: Non-linear geometric effects. *J. Biomechanics*, pages 509–516, 1973.

[JG73] R. F. Janz and A. F. Grimm. Deformation of the diastolic left ventrical – part i: Non-linear elastic effects. *Biophys. J.*, pages 689–704, 1973.

[JT95] S. Jianhua and D. Thalmann. Interactive shape design using metaballs and splines. In *Proc. Implicit Surfaces*, Grenoble, 1995.

[KBS80] A. D. Karaplan, M. P. Bienek, and R. Skalak. A mathematical model of lung parenchyma. *J. Biomech. Engng.*, (102):124–136, 1980.

[KCM99] U. Kühnapfel, H.K. Cakmak, and H. Maaß. 3d modeling for endoscopic surgery. In *Proc. IEEE Symposium on Simulation*,

pages 22–32, Delft University, Delft, NL, Oct 13 1999. ISBN: 90-804551-7-2.

[KGC+96] R.M. Koch, M.H. Gross, F.R. Carls, D.F. von Büren, G. Frankenhauser, and Y.I.H. Parish. Simulating facial surgery using finite element models. In *SIGGRAPH 96*, Anual Conference Series, pages 421–428, New Orleans, Louisiana, August 4–9 1996.

[KGKG99] Erwin Keeve, Sabine Girod, Ron Kikinis, and Bernd Girod. Deformable modeling of facial tissue for craniofacial surgery simulation. *Computer Aided Surgery*, 3(5):228–238, 1999.

[Kil96] Mark J. Kilgard. *OpenGL Programming for the X Window System*. Addison-Wesley Developers Press, 1996.

[Kil00] Mark J. Kilgard. Improving shadows and reflections via the stancil buffer. Technical report, nVidia Corporation, www.nvidia.com, 2000.

[KKH+97] U. Kühnapfel, Ch. Kuhn, M. Hübner, H.-G. Krumm, H. Maaß, and b. Neisius. The karlsruhe endoscopic surgery trainer as an example for virtual realita in medicla education. *Minimal Invasive Therapy and Allied Technologies (MITAT)*, (6):122–125, 1997.

[KWT88] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int. J. Comput. Vision*, 1(4):321–331, 1988.

[Lar86] W. Larrabee. A finite element model of skin deformation. *Laryngoscope*, pages 399–412, April 1986.

[LB95] R. Lieber and T. Burkholder. Musculoskeletal soft tissue mechanics. In Joseph D. Bronzino, editor, *The Biomechanical Engeneering Handbook*, pages 352–356. CRC Press, IEEE Press, 1995.

[LC87] William Lorensen and Harvey Cline. Marching cubes: A high resolution 3d surface reconstruction algorithm. *Computer Graphics*, 21(4):163–170, July 1987. handle with care; some bugs in the paper have been discussed in the scientific visualization community in the years since the paper first appeared.

[LL78] J. T. Liu and G. C. Lee. Static finite deformation analysis of the lung. *J. Engng. Mech. Div.*, (104):225–238, 1978.

[LTW93] Y. Lee, D. Terzopoulos, and K. Waters. Constructing physics-based facial models of individuals. In *Proc. of Graphics Interface*, pages 1–8, May 1993.

[LTW95] Y. Lee, D. Terzopoulos, and K. Waters. Realistic modeling for facial animation. In *Proc. of SIGGRAPH 95*, Computer Graphics Proceedings, Annual conference series, pages 55–62. ACM, 1995.

[LTY83] G. C. Lee, N. T Tseng, and Y. M. Yuan. Finite element modeling of lungs including interlobar fissures and the heart cavity. *J. Biomechanics*, 16(9):679 – 690, 1983.

[LWA] I.P. Logan, D.P.M. Wills, and N.J. Avis. Deformable objects in virtual environments. Department of Computer Science, Virtual Environment Research Centre, University of Hull, Hull, HU6 7RX, U.K., i.p.logan@dcs.hull.ac.uk.

[LWP94] P. Langley, A.J. Wilkinson, and S.H. Pulko. A three-dimensional transmission line model of transient elastic deformation. In *Proc. of the Twelfth IASTED International Conference APPLIED INFORMATICS*, pages 100–103, Annecy, France, May 1994. IASTED. ISBN: 0-88986-190-0.

[MJ96] R. MacCracken and K. Joy. Free-form deformations with lattices of arbitrary topology. In *Proc. of SIGGRAPH 96*, Computer Graphics, Annual conference series, pages 181–188. ACM, 1996.

[MPK+99] Michael I. Miga, Keith D. Paulsen, Francis E. Kennedy, Alex Hartov, and David W. Roberts. Model-updated image-guided neurosurgery using the finite element method: Incorporation of the falx cerebri. In Chris Taylor and Alan Colchester, editors, *miccai99*, volume 1679 of *Lecture Notes in Computer Science*, pages 900–909, Cambridge; UK, September 1999. Springer.

[MSBH94] Gordon D. Mallinson, Mark A. Sagar, David Bullivant, and Peter J. Hunter. A Virtual Environment and Model of the Eye for Surgical Simulation. In *Proc. of SIGGRAPH 94*, Annual conference series, pages 205–212, 1994.

[MT96] Tim McInerney and Demetri Terzopoulos. Deformable models in medical image analysis: a servey. *Medical Image Analysis*, 1(2):91–108, 1996. Oxford University Press.

[MT97] L. Moccozet and N. M. Thalmann. Dirichlet free-form deformations and their application to hand simulation. In *Proc. of Computer Animation '97*, pages 93–102. IEEE, 1997.

[MWMTT98] W. Maurel, Y. Wu, N. Magnenat-Thalmann, and D. Thalmann. *Biomechanical Models for Soft Tissue Simulation*. Basic Research Series. Springer, Berlin, Heidelberg, New York, 1998.

[ner98] Joachim Meißner. Biomechanische Simulaton mit finiten Elementen. Master's thesis, Universität Mannheim, 1998.

[NHK+85] Hitoshi Nishimura, Makoto Hirai, Toshiyuki Kawai, Toru Kawata, Isao Shirakawa, and Koichi Omura. Object modeling by distribution function and a method of image generation. *Transactions of the Institute of Electronics and Communication Engineers of Japan*, J68-D(4):718–725, July 1985. (translated into English by Takao Fujiwara while at Centre for Advanced Studies in Computer Aided Art and Design, Middlesex Polytechnic, England, 1989.). I could not get this one, but at least now you know where to find the original publication.

[NRBM83] A. Needleman, S. A. Rabinowitz, D. K. Bogen, and T. A. McMahon. A finite element model of the infarcted left ventricle. *J. Biomechanics*, (16):45–58, 1983.

[NSM98] Paul F. Neumann, Lewis L. Sadler, and Jon Gieser M.D. Virtual Reality Vitrectomy Simulator. In Alan Colchester, William M. Wells, and Scott Delp, editors, *Medical Image Computing and Computer- Assisted Intervention – MICCAI '98*, pages 910–917, Cambribge, MA, USA, October 1998. Springer.

[oGIaG] Medical College of Georgia & IMTC at GeorgiaTech. Simulation of a Catheract Surgery. `http://www.oip.gatech.edu/MMTLPROJ/eye.html`.

[PB81] S. Platt and N.I. Badler. Animating facial expressions. In *Proc. of SIGGRAPH 81*, volume 3 of *Computer Graphics*, pages 245–252, 1981.

[PBWP99] Alonso Pena, Malcom D. Bolton, Helen Whitehouse, and John D. Pickard. Effects of brain ventricular shape on periventricular biomechanics: A finite-element analysis. *Neurosurgery*, 45(1):107–118, 1999.

[PCRH78] Y. C. Pao, P. A. Chevalier, J. R. Rodarte, and L. D. Harris. Finite element analysis of the starinvariations in excised lobe of canine lung. *J. Biomechanics*, (11):91–100, 1978.

[PEHP93] D. J. Patin, E. C. Eckstein, K. Harum, and V. S. Pallares. Anatomic biomechanical properties of human lumbar dura mater. *Anesth Analg*, (76):535–540, 1993.

[PLDA00] Guillaume Picinbono, Jean-Christophe Lombardo, Herve Delingette, and Nicholas Ayache. Improving realism of a surgery simulator: Linear anisotropic elasticity, complex interactions and force extrapolation. Theme 3 4018, Institut National de Rechereche en Informatique et en Automatique, INRIA, 2000.

[PW89] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. In *Proc. of SIGGRAPH 89*, Computer Graphics Proceedings, Annual conference series, pages 215–222. ACM, 1989.

[Ree83] William T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Transactions of Graphics*, 2:91–108, April 1983.

[RK99] Cameron N. Riviere and Pradeep K. Khosla. Microscale tracking of surgical instrument motion. In Chris Taylor and Alan Colchester, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI '99*, volume 1679 of *Lecture Notes in Computer Science*, pages 1080–1087, Cambridge; UK, September 1999. Springer.

[RNP00] Arne Radetzky, Andreas Nürnberger, and Dietriech P. Pretschner. Elastodynamic shape modeler: A tool for defining the deformation behavior of virtual tissues. *RadioGraphics*, (20):865–881, 2000.

[Rog92] David Rogelberg, editor. *OpenGL Reference Manual*. Addison-Wesley Developers Press, 1992.

[Ruf00] Thomas Ruf. Entwurf, Aufbau und Evaluierung eines FPGA-basierten Farbmarker-Trackings für den Augenoperations-Simulator EyeSi2. Master's thesis, Universität Mannheim, 2000.

[Sch91] H. R. Schwarz. *Methode der Finiten Elemente*. Teubner, 1991.

[SGBM98] Markus Schill, Sarah Gibson, H.-J. Bender, and R. Männer. Biomechanical simulation of the vitreous humor in the eye using an enhanced chainmail algorithm. In Alan Colchester, William M. Wells, and Scott Delp, editors, *Medical Image Computing and Computer- Assisted Intervention – MICCAI '98*, pages 679–687, Cambribge, MA, USA, October 1998. Springer.

[Sim90] Karl Sims. Particle animation and rendering using data parallel computation. *Computer Graphics (SIGGRAPH'90)*, 24(4):405–413, August 1990.

[SP86] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. In *Proc of SIGGRAPH 86*, Annual Conference Series (Computer Graphics), pages 151–160. ACM, 1986.

[SRG+97] M. Schill, Ch. Reinhart, T. Günther, Ch. Poliwoda, J. Hesser, M. Schinkmann, H.-J. Bender, and R. Männer. Biomechanical simulation of brain tissue and realtime volume visualisation. integrating biomechanical simulations into the virim system. In

Proc. of the international Symposium on Computer and Communication Systems for Imageguided Diagnosis and Therapy, Computer Assisted Radiology, pages 283–288, Berlin, Germany, June 1997.

[SSBM96] M. Schill, M. Schinkmann, H.-J. Bender, and R. Männer. Biomechanical Simulation of the Falx cerebri Using the Finite Element Method. In *Proceedings of the 18. Annual International Conference, IEEE Engeneering in Medicine and Biology*, pages 455–456, Amsterdam, The Netherlands, Oct 1996.

[ST92] Richard Szeliski and David Tonnesen. Surface modeling with riented particle systems. *Computer Graphics*, 26(2):185–194, July 1992.

[SWH+99] 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 Colchester, editors, *Medical Image Computing and Computer- Assisted Intervention – MICCAI '99*, volume 1679 of *Lecture Notes in Computer Science*, pages 1166–1174, Cambridge; UK, September 1999. Springer.

[Tab91a] L. A. Taber. On a non-linear theory for muscle shells – part i: Theoretical development. *J. Biomech. Engng*, (113):56–62, 1991.

[Tab91b] L. A. Taber. On a non-linear theory for muscle shells – part ii: Application to the beating of the left ventricle. *J. Biomech. Engng*, (113):63–71, 1991.

[TK95] Nadia Magnenat Thalman and Prem Kalra. The simulation of a virtual tv presentor. *Technical Report, MIRALab, University of Geneva, PG95*, 1995. `http://www.miralab.unige.ch/ARTICLES/PG95.html`.

[Ton91] David Tonnesen. Modeling liquids and solids using thermal particles. *Graphics Interface '91*, pages 255–262, 1991.

[TPF89] Demetri Terzopoulos, John Platt, and Kurt Fleischer. From gloop to glop: Heating and melting deformable models. In *Proc. Graphics Interface*, pages 219–226, June 1989.

[Tsa87] Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, 1987.

[TSBM94] Hideo Takizawa, Kazuaki Sugiura, Motoki Baba, and J. Douglas Miller. Analysis of intracerebral hematoma shapes by numerical computer simulation using the finite element method. *Neurol Med Chir (Tokyo)*, (34):65–69, 1994.

[TW90] D. Terzopoulos and K. Waters. Physically based facial modeling, analysis and animation. *J. Visual. Comp. Anim.*, (1):73–80, 1990.

[TW91] D. Terzopoulos and K. Waters. Techniques for realistic facial modeling and animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Proc. Computer Animation '91*, Tokyo, 1991. Springer.

[Vaw] D. L. Vawter. A finite element model for macroscopic deformation of teh lung. In R. H. Gallagher, B. R. Simon, P. C. Johnson, and J. F. Gross, editors, *Finite Elements in Biomechanics*. John Wiley & Sons, Chichester, UK.

[vdB97] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphical Tools*, 1997.

[Wat87] K. Waters. A muscle model for animating three-dimensional facial expression. In *Proc. of SIGGRAPH 87*, Computer Graphics Proceedings, Annual conference series, pages 17–24. ACM, 1987.

[Wat92] K. Waters. A physical model of facial tissue and muscle articulation derived from computer tomography. In *Proc. of Visualization in Biomedical Computing*, volume 1808, pages 574–583. SPIE, 1992.

[WBMT99] Yin Wu, Pierre Beylot, and Nadia Magnenat-Thalmann. Skin aging estimation by facial simulation. In *Computer Animation 1999 (CA'99)*, Geneva, Switzerland, 26–28 May 1999. IEEE Computer Society.

[Whi94a] Ross T. Whitaker. Volumetric deformable models: Active blobs. Technical report, ECRC, European Computer-Industry Research Centre GmbH (Forschungszentrum), 1994. also in: [Whi94b].

[Whi94b] Ross T. Whitaker. Volumetric deformable models: Active blobs. In Richard A. Robb, editor, *Proc. of Visualization In Biomedical Computing*, pages 122–134, Rochester, Miesota, USA, Oct 1994.

[Wil95] Reg Willson. Tsai camera calibration software. `http://www.cs.cmu.edu/afs/cs.cmu.edu/user/rgw/www/TsaiCode.html`, 1995. Code Revision 3.0b3.

[WMW86] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Animating soft objects. *The Visual Computer*, 2(4):235–242, 1986.

[WND97] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison-Wesley Developers Press, 1997.

[WT91] K. Waters and D. Terzopoulos. Modeling and animating faces using scanned data. *Visualization and Computer Animation*, (2):123–128, 1991.

[WTT95] Yi Wu, Daniel Thalmann, and Nadia Magnenat Thalmann. Deformable surfaces using physically-based particle systems. In *VHCGI95*, Switzerland, 1995. downloaded from: `http://www.miralab.unige.ch/ARTICLES/art95.html`.

[WW89] Brian Wyvill and Geoff Wyvill. Field functions for implicit surfaces. *Visual Computer*, 5:75–82, 1989.

[WW92] A. Watt and M. Watt. *Advanced animation and rendering techniques*. ACM press New York, New York. Addison Wesley, Wokingham, England, 1992.

[YXR+00] D. Yu, C. Xu, M. Rettmann, D. Pham, and J. Prince. Quantitative validation of a deformable cortical surface model. In *SPIE International Symposium on Medical Imaging*, February 2000.

[YYFS94] J. Yamashita, H. Yokoi, Y. Fukui, and M. Shimojo. A virtual surface modeler for direct and regional free form manipulation. In *Proc. of ICAT 94*, The Fourth International Conference on Artificial Reality and Tele-Existence, pages 35–42, 1994.

[Zie75] O. C. Zienkiewicz. *Methode der finiten Elemente*. Carl Hanser Verlag, München-Wien, 1975.

A

Modal Analysis

The changing of the basis from p generalized nodal displacements to q generalized modal displacements with ($q \ll p$) brings a significant improvement with respect to the resolution of the algebraic equations system. The change of the basis is performed by expressing the nodal displacement vector in terms of the modal displacement vector:

$$\vec{U} = \Phi\vec{V} \qquad with \qquad \Phi_{p\times q} = \left[\vec{\phi_1} \ldots \vec{\phi_q}\right] \tag{A.1}$$

with:
$\vec{U}$ the generalized *nodal* displacement vector.
$\vec{V}$ the generalized *modal* displacement vector.
$\vec{\phi_i}$ the the eigenvectors of the linearized eigenproblem.
$\Phi_{p\times q}$ the modal transfer matrix which is composed of the column eigenvectors.

With the mass matrix M, the stiffness matrix K and the damping matrix D the corresponding eigenvalue problem is

$$K\Phi = M\Phi\Omega^2 \qquad with \qquad \Omega^2 = \begin{bmatrix} \omega_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \omega_q^2 \end{bmatrix} \tag{A.2}$$

where the generalized eigenvalues $\omega_i = 2\Pi f_i$ and the f_i are the natural frequencies of the system ($i = 1, \ldots, q$) [MWMTT98]. Because M and K are generally symmetric positive definite and D is typically a linear combination of M and K, they can be simultaniously diagonalized by Φ.

$$\begin{aligned} \Phi^T M\Phi &= \tilde{M} \\ \Phi^T K\Phi &= \tilde{K} \\ \Phi^T D\Phi &= \tilde{D} \end{aligned} \tag{A.3}$$

Using equ. (A.1) on equ. (3.9) and multipying by Φ^T yields:

$$\Phi^T M \Phi \ddot{\vec{V}} + \Phi^T D \Phi \dot{\vec{V}} + \Phi^T K Phi \vec{V} = \Phi^T \vec{F} \tag{A.4}$$

With equs. (A.3) and the definition $\tilde{F} = \Phi^T \vec{F}$ this leads to

$$\tilde{M}\ \ddot{\vec{V}} + \tilde{D}\ \dot{\vec{V}} + \tilde{K}\ \vec{V} = \tilde{F} \tag{A.5}$$

In equ. (A.5) the system equations are lineary independent. Each equation describes a vibrational mode of the object. In 3D, six vibrational modes represent the possible rigid body motion (three translational and three rotational degrees of freedom). Additional modes account for linear strain, quadratic strain and higher order deformations.

For simulations that requir a real time system response the vibrational modes can be sorted in such a way that the rigid body motions are calculated first and deformations are only calculated up to the degree that system time is available. This allows a setup with different levels of detail for the deformation depending on the available computational time [GM97].

B

Element Matrices

The following passage introduces the matrices used for the finite element analysis described in section 6.2. The matrices for the *BV-triangle* are given as a combination of the matrices of the *B-triangle* andf the *V-triangle*.

Triangular Element: *BV-triangle*

The used element is a combination of a triangular element which considers in-plane strain (*V-triangle* (figure B.2) and one that considers bending *B-triangle* (figure B.3). We call the combined element *BV-triangle* (figure B.1.

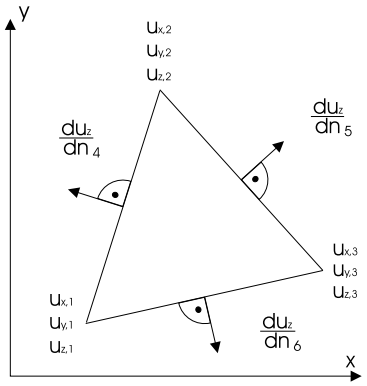

Figure B.1: Node variables for the *BV-triangle* considering in-plane strain and bending

The node variables of the *BV-triangle* are a combination of the node variables of the *B-triangle* and *V-triangle*.

$$\vec{u} = \left(u_{x,1}, u_{y,1}, u_{x,2}, u_{y,2}, u_{x,3}, u_{y,3}, u_{z,1}, u_{z,2}, u_{z,3}, \left(\frac{\partial u_z}{\partial n}\right)_4, \left(\frac{\partial u_z}{\partial n}\right)_5, \left(\frac{\partial u_z}{\partial n}\right)_6 \right)^T \tag{B.1}$$

Analogous is the stiffness matrix of the *BV-triangle* a combination of the stiffness matrices of the *B-triangle* and *V-triangle*.

$$\tilde{S}_{BV} = \begin{pmatrix} \tilde{S}_V & 0 \\ 0 & \tilde{S}_B \end{pmatrix} \tag{B.2}$$

By exchanging rows and collums $\tilde{S}_{BV}$ is brought to the more convinient form S_{BV} which corresponds to the following node variable vector:

$$\vec{u} = \left(u_{x,1}, u_{y,1}, u_{z,1}, u_{x,2}, u_{y,2}, u_{z,2}, u_{x,3}, u_{y,3}, u_{z,3}, \left(\frac{\partial u_z}{\partial n}\right)_4, \left(\frac{\partial u_z}{\partial n}\right)_5, \left(\frac{\partial u_z}{\partial n}\right)_6 \right)^T \tag{B.3}$$

V-triangle

The V-triangle lies in the xy-plane. The three corners, each contributes a node variable: $\vec{u}_1 = (u_{x,1}, u_{y,1})^T$, $\vec{u}_2 = (u_{x,2}, u_{y,2})^T$ und $\vec{u}_3 = (u_{x,3}, u_{y,3})^T$ The

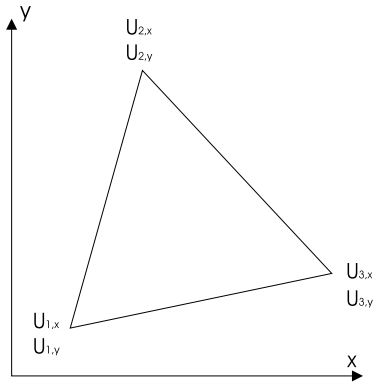

Figure B.2: Node variables for the *V-triangle* that considers in-plane strain.

V-triangles stiffness matrix is:

$$\tilde{S}_V = A^T S_V A \tag{B.4}$$

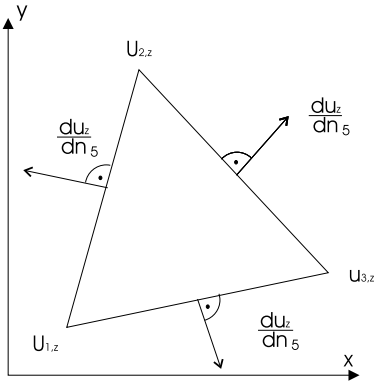

Figure B.3: Node variables for the *B-triangle* considering bending.

with:

$$S_V = hF\frac{E}{(1+\nu)(1-2\nu)}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \nu \\ 0 & 0 & \frac{1-\nu}{2} & 0 & \frac{1-\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1-\nu}{2} & 0 & \frac{1-\nu}{2} & 0 \\ 0 & \nu & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{B.5}$$

and

$$A^{-1} = \begin{pmatrix} 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_1 & y_1 \\ 1 & x_2 & y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_2 & y_2 \\ 1 & x_3 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_3 & y_3 \end{pmatrix} \tag{B.6}$$

B-triangle

The B-triangle considers energy generated by bending the triangle out of the xy-plane. In addition to the node translations $u_{z,i}, i = 1, 2, 3$, we also need to consider the first derivative of nodes on the middle of the three edges $(du_z dn_j), j = 4, 5, 6$. The B-triangles stiffness matrix is:

$$\tilde{S}_B = A^T S_B A \tag{B.7}$$

with:

$$S_B = \frac{1}{3}h^3 F \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 4\nu \\ 0 & 0 & 0 & 0 & 2(1-\nu) & 0 \\ 0 & 0 & 0 & 4\nu & 0 & 4 \end{pmatrix}. \tag{B.8}$$

and:

$$A^{-1} = \begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 \\ 1 & x_3 & y_3 & x_3^2 & x_3 y_1 & y_3^2 \\ 0 & \cos\phi_4 & \sin\phi_4 & 2x_4\cos\phi_4 & x_4\sin\phi_4 + y_4\cos\phi_4 & y_4\sin\phi_4 \\ 0 & \cos\phi_5 & \sin\phi_5 & 2x_5\cos\phi_5 & x_5\sin\phi_5 + y_5\cos\phi_5 & y_5\sin\phi_5 \\ 0 & \cos\phi_6 & \sin\phi_6 & 2x_6\cos\phi_6 & x_6\sin\phi_6 + y_6\cos\phi_6 & y_6\sin\phi_6 \end{pmatrix} \tag{B.9}$$

Tetrahedron element

The tetrahedron element (figure B.4) used in section 6.2 has the following stiffness matrix. The tetrahedron's stiffness matrix is given by:

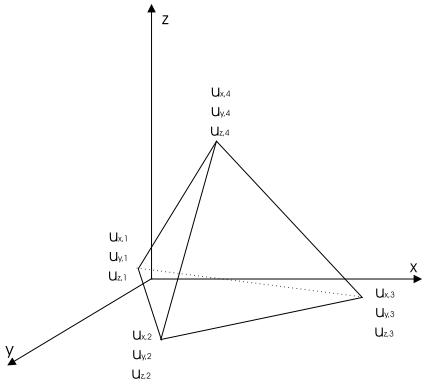

Figure B.4: The tetrahedron element used for simulating the swelling brain.

$$\tilde{S} = A^T S A \tag{B.10}$$

with:

$$S = \frac{E}{(1+v)(1-2\nu)}$$

$$\begin{pmatrix}
1-\nu & 0 & 0 & 0 & 0 & \nu & 0 & 0 & 0 & 0 & \nu & 0 \\
0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & 0 & 0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\nu & 0 & 0 & 0 & 0 & 1-\nu & 0 & 0 & 0 & 0 & \nu & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 & \frac{1}{8}-\frac{1}{4}\nu & 0 & 0 \\
\nu & 0 & 0 & 0 & 0 & \nu & 0 & 0 & 0 & 0 & 1-\nu & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

and:

$$A^{-1} = \begin{pmatrix}
1 & x_1 & y_1 & z_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & x_1 & y_1 & z_1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_1 & y_1 & z_1 \\
1 & x_2 & y_2 & z_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & x_2 & y_2 & z_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_2 & y_2 & z_3 \\
1 & x_3 & y_3 & z_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & x_3 & y_3 & z_3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_3 & y_3 & z_3 \\
1 & x_4 & y_4 & z_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & x_4 & y_4 & z_4 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_4 & y_4 & z_4
\end{pmatrix} \tag{B.11}$$