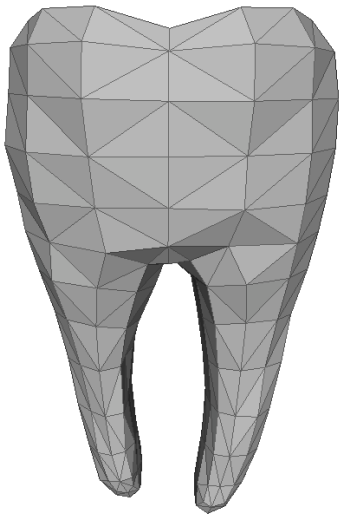
# Real-time Fragmentation of Solid Material using the Finite Element Method

Master's Thesis




Aarhus University
Department of Computer Science

October 1, 2009


Christian P. V. Christoffersen
cpvc@cs.au.dk
20050879

Carsten Nørby
tic@cs.au.dk
20051359



Primary Supervisor
Associate Professor
Ole Østerby

Secondary Supervisor
Associate Professor
Thomas Sangild Sørensen

# Summary

This thesis is within the field of simulated surgery and motivated by the current demand for an educational tool teaching the common procedure of wisdom tooth extraction. Attention is focused on how to simulate the fragmentation that will separate the crown of the tooth from its roots. We present a general method for crack prediction and propagation in volumetric solids based upon real-time structural analysis. The analysis is conducted using the finite element method and the Total Lagrangian Explicit Dynamics solving technique with a parallel approach. The stress and strain analysis are based on the theoretical laws of physics and the crack prediction is based on the theory of maximum principal stress from fracture mechanics. The failure surface as predicted by the crack tracking algorithm looks very promising. The location and curvature of the failure surface corresponds to the stress analysis and the intuition of how an object would actually fracture. Benchmarking the simulation model reveals great potential towards real-time interaction and visual feedback.

# Acknowledgement

We greatly appreciate all who contributed to this thesis, especially the following people who we would like to give a special thanks to:

Associate Professor, Ole Østerby for supervision, guidance, and for always leaving his door open, both literary and figuratively. We are grateful for all the time he spent carefully explaining theories to us and for his comprehensive reviews. Associate Professor Thomas Sangild Sørensen for supervision, especially within the field of simulated surgeries and for all the inspiring talks, ideas, references, reviews, and guidance throughout this entire thesis.
PhD student, Karsten Østergaard Noe, for sharing his knowledge and a prototype solver for structural analysis.

PhD Jesper Mosegaard and Peter Trier, from The Alexandera Institute, for introducing the problem domain, their cooperation and for establishing collaboration with people from the Aarhus School of Dentistry, Aarhus University.

Professor, dr. odont. Søren Schou, Aarhus School of Dentistry, Aarhus University, for his enthusiasm and all the time he spent on carefully explaining how dental surgical procedures are performed. We are thankful for all the material he provided including books, videos, dental instruments, wisdom teeth and for granting us the permission to use the dental pictures included in this thesis (marked by †).

PhD student, Lau Brix, for guidance concerning hardware configuration for a high performance computer suitable for parallel execution and the Siemens foundation (Siemensfonden) for funding this hardware. Christian Esbo Agergaard, 3D artist, for creating a three-dimensional surface mesh of a wisdom tooth, which is suitable for our simulation scenarios. The OpenEngine community for providing the portable framework used as the foundation for the simulator implementation.

# Readers Guide

This thesis consists of three parts. Part I presents analytical theories. This part includes statics, dynamics, continuum, and fracture mechanics. Futhermore, elasticity theory and basic linear algebra is introduced. Part II presents discrete theories. Here the fundamental equilibrium framework and the finite element method are explained in detail and used to construct a simulation model. Part III explains how the simulation software is designed and implemented. Parallel execution is introduced and the simulation software constructed is evaluated. The following is a brief summary of each chapter:

Chapter 1 introduces the reader to the field of simulated surgery and the potential of computer aided simulations in general. Here the motivation behind this thesis is stated. Chapter 2 explains the surgical procedure of wisdom tooth removal step by step. The physical phenomena of interest are identified and the concepts real-time and simulator are defined. The main objectives in the thesis is stated and related work are presented.

## Part I - Analytic Theories

Chapter 3 introduces the fundamental physics used. Work, energy, and equilibrium are defined before moving on to continuum mechanics where the concepts deformation, elasticity, stress and strain are introduced. Chapter 4 explains the basic linear algebra used.

## Part II - Discrete Theories

Chapter 5 introduces the fundamental structure for equilibrium problems in general. Chapter 6 describes the finite element method. The discretization of the continuum body and how to assemble the system equations are explained in detail. Chapter 7 explains how to apply the finite element method and a concrete solver technique is presented.

## Part III - Implementation

Chapter 8 presents the simulation software developed. Chapter 9 introduces the concept of parallel execution and explains how we utilize NVIDIA's CUDA technology. Chapter 10 explains the developed tools for interacting with the simulation software and how to visualise tensors. Chapter 11 presents the results obtained. Chapter 12 is a discussion on possible improvements. Chapter 13 summarizes and concludes on the results obtained.

# Contents

# List of Figures

Figures marked by the following symbols are from external sources:

† Image provided by Søren Schou, Aarhus School of Dentistry.
‡ From NVIDIA's CUDA Programming Guide (Version 2.3.1).

# List of Tables

# Chapter 1

# Introduction

The research field of computer simulated surgery has been a topic of increased interest in recent years. The current research has reached a level that facilitates real-time simulation of deformations in soft organic materials. Using computer aided simulations as an educational tool or an interactive training facility has a lot of potential when it comes to imparting new knowledge to students or trainees. Surgical procedures can be studied and practiced iteratively through extended feedback including any imaginable visual information. While performing the simulated surgery instructions and guidance could be presented, any potential risk could be pointed out, and unexpected critical scenarios could occur. Simulated surgery has the potential to enrich the learning process on so many levels. From the perspective of the student the main objective of using simulated surgery is to acquire certain, often very difficult, skills in a risk-free environment. Introducing surgical certificates based on simulations is an option within the near future. Professional approval based on simulated scenarios has been used for many years when educating and certifying pilots.

The field of simulated surgery still faces great challenges especially when it comes to the sense of touch. It is very difficult to simulate the correct feedback on the tool being used e.g. the weight, friction, momentum, manoeuvrability etc. Simulation models based on the laws of physics will always be idealized and respond as predicted by theory. Sometimes theory and practice is not consistent.

The development within computer aided visualization has been tremendous during the last decades. The research within this field has benefited heavily from the huge investments made by the entertainment industry constantly striving towards improving the level of realism. Simulating organic material like human tissue or organs require a model capable of representing the complex structures and the theoretical laws of behaviour. In the field of applied mathematics a generalized method for conducting numerical modeling of physical systems was first well defined in the late 1960's and early 1970's. Although these mathematical models are capable of capturing complicated problems they tend to get very complex and computationally expensive. Within recent years the introduction of multi-core hardware and the development in GPU programming languages, suddenly has made these models, based on the actual laws of physics, realisable. By taking advantage of the parallel architecture of the GPU, the computational resources have increased by orders of magnitude assuming the problem can be solved in a parallel environment.

This thesis is motivated by the current demand for an educational tool teaching a specific dental procedure. Inspired by related simulated surgeries The Aarhus School of Dentistry has requested a simulator for teaching the common procedure of wisdom tooth extraction. Performing the

surgical procedure requires certain skills and must be performed with caution to prevent serious nerve damage. All surgery involves risks, in this case the patient's tongue or part of the lip could become permanently numb due to nerve damage. This emphasizes the importance of a proficiency before operating on patients. Besides lots of lessons and videos teaching how to perform the operation, students practise on a rubber doll known as a *phantom*. Operating on a phantom is the last step before the students are expected to perform supervised operations on patients. According to Professor, dr. odont. Søren Schou, Aarhus University, there is a huge educational gap between operating on a phantom and a real patient.

This thesis deals with the development of a model for simulating one specific part of the complete surgical procedure of wisdom tooth removal. Attention is focused on how to simulate the fragmentation that will separate the crown of the tooth from its roots. By solving this particular problem we get one step closer towards the development of a complete simulator, handling the entire surgical procedure, hereby providing an educational tool with the purpose of minimizing the gap between operating on a phantom and a real patient.

The simulation model is based on the finite element method which is a widely accepted method for structural analysis. The physical laws from the field of continuum and fracture mechanics concerning deformation and fragmentation of solids are applied to obtain as realistic results as possible with the computational resources available.

Basically the simulator consists of three main parts. A visual, an interactive, and a computational part. The visual part is responsible for delivering real-time three-dimensional images on the screen. The interactive part handles the user interaction and simulates the dental tools accordingly. The computational part is responsible for solving the equations defined by the method used for the structural analysis.

It requires an interdisciplinary approach to develop a framework for simulating deformations and possible fractures in solid objects. From the field of dentistry, knowledge on how to perform the actual surgical procedure, is needed. Simulating real world phenomena of fracturing solid objects involves theoretical physics and mathematics, as well as engineering disciplines of predicting how solids behave under stress. Constructing a computational model representing the complex structures that incorporates the laws of physics suitable for high performance parallel execution is a computer science discipline. Implementing the simulator using the proper technologies, benchmarking the software model and validating the results are all within the field of computer science.

# Chapter 2

# Problem Domain

## 2.1 The Surgical Procedure

To gain fundamental knowledge on how the surgical procedure is performed in real life, the following section will explain each step involved. This section is based on an interview with Professor, dr. odont. Søren Schou, Aarhus University, who carefully took his time to explain the procedure in detail.

First of all when it comes to patients with wisdom tooth compilations, there are not two of a kind, but basically all complications can be divided into four scenarios. Within each scenario the location of the tooth can vary but the approach used to remove it is the same. The four scenarios are illustrated in figure 2.1 on the next page. The different scenarios require slightly different approaches when it comes to fragmenting the tooth. The following steps are carried out in each of the four scenarios.

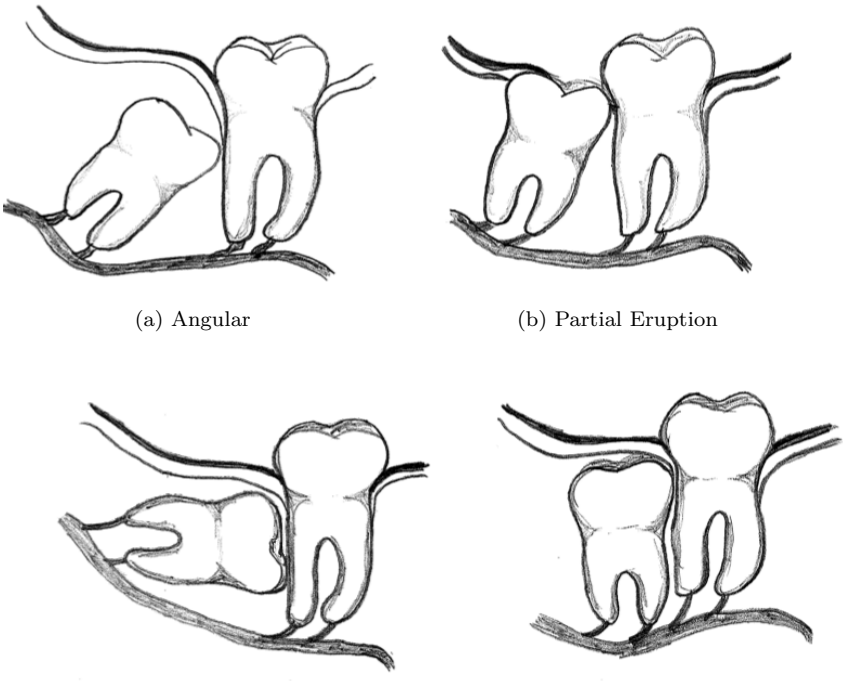
As a preoperative task the dentist carefully examines x-ray images of the wisdom tooth to determine the risk of nerve damage. It is crucial that the nerves are left undamaged since this can lead to complete and permanent numbness of the tongue and lip.

### Anaesthesia

The surgical procedure can be performed under local anaesthesia, leaving the patient conscious during the entire operation. The local anaesthesia is carefully injected directly into the nerves located on the inside of the jaw, making the tongue, part of the lip and the wisdom tooth numb. The anaesthesia substance contains adrenaline which contracts the blood from the blood vessels, hereby minimizing the bleeding from the wound. Shortly after the anaesthetic injection, the dentist pinches the surrounding tissue to ensure the area is numb.

### Incision

The soft tissue surrounding the wisdom tooth, on the outside of the jaw, needs to be loosened to expose the tooth. The cut starts from the back of the jaw and is carefully placed along the outside of the first and second molar, illustrated as the red line in figure 2.2 on the following page. The yellow dot is where the wisdom tooth is located, currently covered by soft tissue. It is crucial not to damage the nerve located on the inside of the jaw, therefore the incision is always performed on the outside of the molars. The cut is made slowly while any blood from the wound



(a) Angular (b) Partial Eruption

(c) Horizontal (d) Vertical

Figure 2.1: The four basic scenarios showing how the wisdom tooth can be located in the jaw.

is sucked away with a small tube to keep visibility high and to prevent it from running down the patients throat. When the incision has been made, the soft tissue can be pulled to the side hereby revealing the wisdom tooth and part of the roots of the first and second molar. Often the wisdom tooth is located deep down the jawbone so only the crown of the tooth is visible when the soft tissue has been removed.

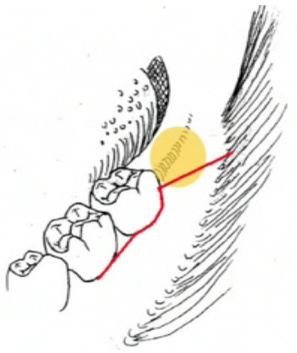


Figure 2.2: The red line illustrates where to place the incision. †

### Bone Removal

Now that the soft tissue is removed the next step is to remove the surrounding jawbone. This is done using a drill on the outside of the jaw as illustrated with the red line on figure 2.3. Drilling away the surrounding jawbone will expose the wisdom tooth down below its crown. When drilling, the point of contact between drill and jawbone, is constantly cooled with water to prevent the nearby soft tissue from heating up potentially causing damage.

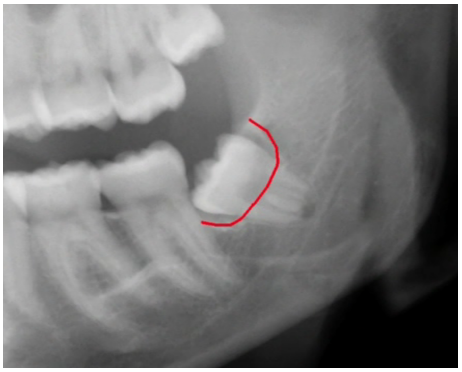


Figure 2.3: The red line illustrates where the jawbone must be drilled away. †

### Drilling a Groove

Now that the wisdom tooth has been exposed from both the surrounding tissue and jawbone, the dentist can move on and prepare a controlled fragmentation of the tooth. The fragmentation must carefully separate the crown of the tooth from its roots as illustrated by the red line in figure 2.4. A straight groove is drilled along the red line, as illustrated on figure 2.4. The groove is approximately 5 millimeters deep, or one third of the tooth, and located right below the crown.

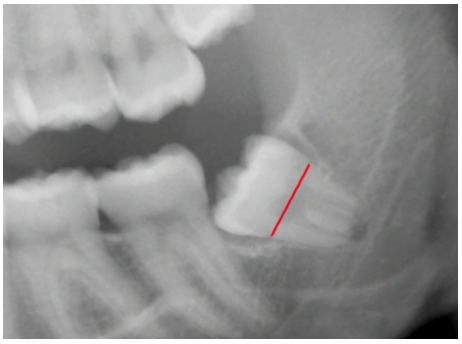


Figure 2.4: The red line illustrates where the groove should be drilled. †

The dentist always tries to avoid drilling down the enamel on the crown of the tooth, because it is so hard that the drill will be worn down too fast.

### Fragmentation

A special tool known as an *elevator*, illustrated in figure 2.5, is now inserted into the groove and twisted to separate the crown of the tooth from its roots. Sometimes, due to the location of the wisdom tooth, it is necessary to perform further fragmentation to either the crown or the roots.

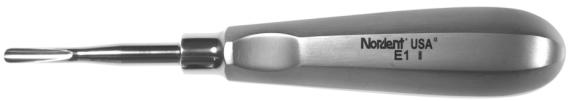



Figure 2.5: Dental tool known as an elevator. †

### Fracture Removal

Now that the tooth has been fragmented the individual parts must be removed carefully. It is very important to make sure all of the fragments are removed properly. If a small piece is left behind it can cause complications such as wound infections. Usually wisdom teeth are removed only if they cause some sort of complications. A common complication is when other teeth prevent the natural development, often causing an small infection around the wisdom tooth. The infected tissue must be removed from the wound as well.

### Smooth Off the Rough Edges

The dentist has to smooth off the rough edges in the jawbone to prevent complications with the soft tissue surrounding it. The crater from where the wisdom tooth was located, must be cleaned thoroughly with water to wash out every tiny bit of bone from the drilling.

### Stitching the Wound

The last step in the surgical procedure is to stitch the wound together. Usually two stitches is enough but this depends on the size of the wound. When the stitching is done a small cloth is inserted in the cheek to absorb any remaining blood. The surgical procedure is now done.

## 2.2 Simulator

In context of computer science a simulation is a computational model that approximates a real phenomenon or procedure. A simulation defines rules of behaviour reflecting those appearing in the real world. The results produced by a simulator can be used for evaluation or prediction of the phenomenon or procedure simulated. Challenges arise due to the almost unlimited complexity of any real world phenomena no matter how simple it seems. Specific parts of the phenomenon being simulated must be prioritised, but even then only a certain level of detail can be computed. If time and computational resources was no issue, designing a model that strictly represents solid objects by the smallest unit possible, would be a solution. However intuition tells us that it is probably impossible to represent and simulate the physical laws for such small units. Let us consider atoms as the basic unit of matter, the nucleus diameter of an atom is about $10^{-15}$ meter. Even for a small object like a tooth the amount of units to be represented would be billions, easily

exceeding the computer resources available. A discrete method must be used in order to facilitate computation of the problem domain.

### Real-time

It is crucial that the simulator gives the impression of a surgical procedure performed in real-time. The concept of simulated real-time can be considered from two different perspectives. One way is to measure real-time as the speed of average sensory perception. By measuring real-time this way we consider how fast the system is responding to an event such as user interaction. In this sense real-time is more of an illusion where users should perceive the simulation feedback at a rate that corresponds to the actual scenario in real life.

> *" It doesn't really matter whether the deformation that the surgeon sees in the virtual environment is accurate as long as it seems realistic! Just as important is that the model is robust and shows a consistent and predictable behavior over time"*
>
> (Morten Bro-Nielsen [BN98, page 8])

Another way of defining real-time is to consider the extrapolate of simulation time contra the computation time used. When simulating a phenomenon time must be discretized into time steps that represents the difference in time from the current configuration to the next. If it is possible to compute the next configuration faster than the time extrapolation it represents we can predict the behaviour of the phenomenon in real-time. If not we can pre-compute the behaviour and inspect them at a real-time rate afterwards, but this rules out real-time interaction which is an essential part a surgical simulator.

To accommodate the demand for real-time execution appropriate solving techniques must be used. A variety of limitations and approximations must be taken into account to keep the result plausible. When the level of simulation details increases so does the computational efforts. Making the right decisions in the trade-off between accuracy and speed is essential. Each time an approximation is made the margin of error must be considered to prevent the final result from ending wide of the mark.

### Phenomena

Before designing a model that represents the processes we would like to simulate, we need to identify the physical phenomena. It is no surprise that different materials behave very differently. Dropping a glass vase on the floor would probably break it while a rubber ball would bounce back without a scratch. What might be a surprise to some, is the rather complex physical theory behind this. The answer to why some materials are able to undergo change in form without breaking and others are not, lies in the structure of the molecules. Since representing a material on the level of molecules would exceed the computational resources, this phenomenon must somehow be represented on a higher level.

A rubber ball can deform into an ellipsoid and still return to its original form. In physics elasticity is the property that makes a material deform due to external forces, but still makes it return to its original form when the external forces are removed. This phenomenon requires a model where external forces somehow are transformed into elastic energy.

A homogeneous material contains only identical molecular structures and is therefore a pure material with the exact same properties everywhere. In the real world this is rarely the case, most materials are a composition of others. A tooth has different layers each with different material properties. From that perspective modelling heterogeneous materials would probably be preferable.

It is common knowledge that most material properties change as a result of change in temperature. When steel is heated its strength decreases, and rubber cooled down to extremely low degrees can shatter like glass. Decision must be made whether or not to represent this phenomenon in the simulation model.

When things break in the real world it often happens in a blink of an eye. A rule of thumb says that a crack propagates through a brittle material with the speed of sound. Observing the propagation would require a high speed recording of the phenomenon for later replay in slow motion. Intuition tells us that the crack must have a point of origin. As the crack propagates through the material it must somehow choose its path based on the material properties and the external forces acting upon the material.

## 2.3 Problem Statement

As already mentioned the surgical procedure of removing a wisdom tooth involves various steps of cutting tissue, drilling, fracturing dental material, stitching tissue, etc. Simulating each step requires a different approach and the use of distinct theory. It is out of scope of a single masters thesis to handle all of the different step.

Considering the relatively limited amount of work published within the field of fracturing volumetric models in real-time compared to the other fields of interest from the various simulation steps, attention will be focused on solving this particular problem.

This thesis deals with simulating the specific parts of the dental procedure that involves fragmentation of the tooth. Separating the crown of the tooth from its roots involves estimation of the internal stress and prediction of crack propagation through a solid material. The estimation of internal stress will be based on the finite element method which is a widely accepted method used for structural analysis. Using the finite element method to conduct real-time analysis is a fairly new concept developed within recent years. It has proved to be one of the most precise methods available. The method for predicting the point of origin and the propagation of the crack will be based upon the theory of maximum principal stress. To the best of our knowledge the parallel approach of conducting real-time structural analysis in conjunction with crack propagation, based upon maximum principal stress is new to the field.

The main objective in this thesis is to create a framework for real-time simulations of deformation and crack initialization in solid objects. With point of reference in the laws of physics we will construct a simulation model that strictly complies with relevant theory. The context of use is simulated surgery therefore speed and robustness of the model will be favoured over accuracy. External forces will be applied through a simulated dental tool, similar to the elevator actually used by dentists. The stress and strain analysis will be based on the theoretical laws of physics and the crack prediction will be base on the theory of maximum principal stress direction from fracture mechanics. We will construct a model aimed towards real-time crack detection based upon analysis of stress and strain behaviour. If the internal level of stress exceeds a material-specific threshold the solid object will fracture. Once the point of origin of the crack has been predicted

the determination of the crack surface is not considered time critical.

## 2.4 Related Work

The purpose of solving the particular problem of how to fracture solid material in real-time, is to establish a theory that will bring the development of a complete dental surgical simulator one step closer to reality. The simulation model presented in this thesis is base upon the combination of numerous theories and existing methods. The following articles presents closely related work and methods we have derived our simulation model from.

Morten Bro-Nielsen was among the pioneers within real-time simulation of deformable models used for simulated surgery. Morten Bro-Nielsen [BN97][BN98] presents a method base on a Fast Finite Element (FFE) method and linear elasticity. The interior nodes are removed from the equations to improve performance. Even though the interior nodes are excluded from the calculations the predicted behaviour of the surface nodes are exactly the same a if interior node was taking into account.

Miller et al. [MJLW07] present an efficient numerical method for computing soft tissue in real-time, based upon finite element analysis. Their method is based on total Lagrangian formulation relating stress and strain to the original configuration.

Zeike A. Taylor et al. [TCO08] present an improved method based directly upon the work conducted by [MJLW07]. The improvements made are primarily on how to solve the finite element equations in parallel using the GPU hereby gaining a significant increase in performance.

Thomas Sangild Sørensen and Jesper Mosegaard presents methods on how surgical simulations can accelerate deformable models through the parallel nature of the GPU[SM06b]. They show that deformable models based on the spring-mass or finite element method can be implemented using parallel programming hereby gaining a significant increase in performance.

Depending on how much the wisdom tooth is developed, fragmentation of the tooth can be necessary. Teeth are primarily made of a material called dentin which consists of calcium, phosphorus, and other mineral salts. Dentin is a very hard and brittle material. Techniques for simulating fracture initialization and propagation in brittle materials has been presented by O'Brian et al. [OH99]. The method presented is based on finite element analysis and aimed towards pre-rendered simulations.

Matthias Müller et al. [MMDJ01] present a method for real-time deformation and fractures in stiff materials. The method presented introduces a hybrid approach where the simulation is alternating between simulating rigid body dynamics and the continuum model at the point of impact. The fragmentation method used includes model re-meshing which improves the realism of the crack surface.

A comparison of different fracturing techniques has been conducted by P. Jäger et al. [JSK08]. The article compares four different approaches in terms of standard quality measures such as efficiency, robustness, stability and computational cost.

Besides the closely related work, we have also paid attention to the following articles and PhD theses presenting methods that could be used for simulating the other parts of the complete simulator.

In relation to handling input the user, here being a surgeon, must be presented to a device capable of simulating the tools similar to the ones used in real life. Surgeons are used to manoeuvre tools freely in three-dimensional space. An ordinary computer mouse has only two degrees of freedom which makes it unfit. Instead of conventional user input like those of a keyboard or mouse, we would prefer *haptic* feedback. Haptic refers to the sense of touch. In the context of surgical simulations it means a device that interfaces with the user through force feedback. The haptic device serves as a three-dimensional joystick with the extra feature of delivering force feedback, when the virtual tool intersects with an object in the simulator. Handling user input immediately is crucial to prevent the surgeon from experiencing latency and hereby loosing the illusion of real-time. Thomas Sangild Sørensen and Jesper Mosegaard[SM06a] present efficient methods for handling high-frequently haptic feedback using the GPU.

If the wisdom tooth is not developed enough the dentist will have to cut the surrounding tissue to expose the wisdom tooth. As seen in figure 2.1 on page 6 the amount of tissue to cut open depends on the location of the tooth. Techniques for cutting soft tissue, based on a spring-mass model, has been presented as part of a cardiac surgery simulator by Jesper Mosegaard[Mos06].

When the wisdom tooth is exposed, the next step is to drill a hole into the tooth as a preparation for the fragmentation. A method for drilling in volumetric material in context of simulated surgery has been developed by Peter Trier et al. [TNSM08]. The method is base on a data set available as a three-dimensional image format, and visualization based on ray-castings. This technique has proved very promising.

# Part I

# Analytical Theories

# Chapter 3

# Physics

Physics is a science based on experiments where phenomena of nature are observed with the objective of finding patterns and principles. These patterns and principles are called theories or, when they are commonly accepted, *laws of physics* [YFFS00, page 1]. As already touched upon in section 2.2 on page 8, the simulator incorporates such physical laws to simulate deformation and fragmentation of solid material. This chapter introduces the physical theories needed to create a simulation model. The aim is to assemble a theoretical model for deformation and fragmentation based on the laws observed and generally accepted by physicists. Note that the physical laws used are idealized models, meaning that the models are simplified versions of real phenomena and that concepts of little importance for the overall result are neglected [YFFS00, page 3]. In this respect we do not model material at the level of particles as the effects of individual particles are not detectable when larger volumes of matter are considered. This is called large-scale mechanics in contrast to quantum mechanics, which deals with laws at the level of atoms. Large-scale mechanics is valid when the size of the material analysed is much larger than the size of an atom [Dil07, page 1].

The subgroups of special interest are *statics*, *dynamics* and *continuum mechanics*. As the theory of continuum mechanics is quite complex, we will start off gently by revisiting basic university physics in the context of statics and dynamics. Here we define various physical quantities, their corresponding units and how they are related.

## 3.1 Statics and Dynamics

In statics, we only analyse objects that do not move. This enables us to describe and relate various phenomena acting upon an object. The next step is dynamics, where objects are allowed to move but not to deform. Dynamics expands the analytical method by allowing movement and rotation. In statics an object is viewed as a single point which describes it's position. In dynamics objects are modeled by their position and orientation. Statics and dynamics makes these assumptions so objects are less complex to analyse. In continuum mechanics objects are represented by volumetric elements, which are more complex.

### Space and Time

Within statics and dynamics, objects and forces alone are not enough when analysing a physical system. The concept of an object at rest or in motion requires a *frame of reference*. A frame of reference is any kind of coordinate system which enables us to uniquely define the position of

a body in three-dimensional space. To describe motion the frame of reference must be fixed so only motion of the moving body is considered. By introducing time we can refer to an object at a given time, hereby describing the object's *velocity* and *acceleration*. Complying with the SI system for units, distance is measured in meters ($m$), time in seconds ($s$), velocity in $m/s$, and acceleration in $m/s^2$.

### Forces

In mechanics there are many different types of forces, therefore we need some basic terms to describe them. A force has a direction and a magnitude. The direction is referred to as the *line of action*. The magnitude determines the size of the force and is measured in newton ($N$). A force acts upon an object at a specific point as illustrated in figure 3.1: A force is depicted in a *free-body diagram* as an arrow mounted at the point in which it acts.

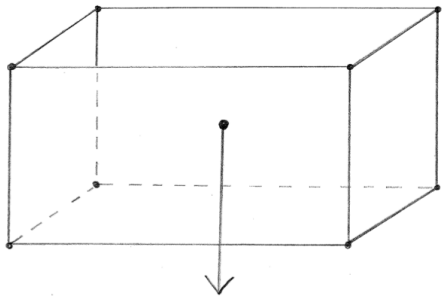


Figure 3.1: Free-body diagram of the gravitational force acting upon a box.

In mechanics an object is composed of two distinct concepts: a *surface* and a *body* (note that the entire object is often referred to as the body, which can be a little confusing). Forces can be divided into different types, the types used in this thesis are: body contra surface and internal contra external. External forces act upon a physical system from the outside and can not be generated by the system itself. An example of an external body force is the gravitational force as illustrated in figure 3.1. When two objects collide, a contact force occurs between the colliding surfaces. This is an example of an external surface force, and is illustrated in figure 3.2 on the next page. Internal forces do not occur in statics and dynamics because the various internal parts of an object cannot move and hereby interact. When modelling deformations, various parts of an object can interact, hence we need internal forces. In section 3.2 on page 26 internal forces will be introduced.

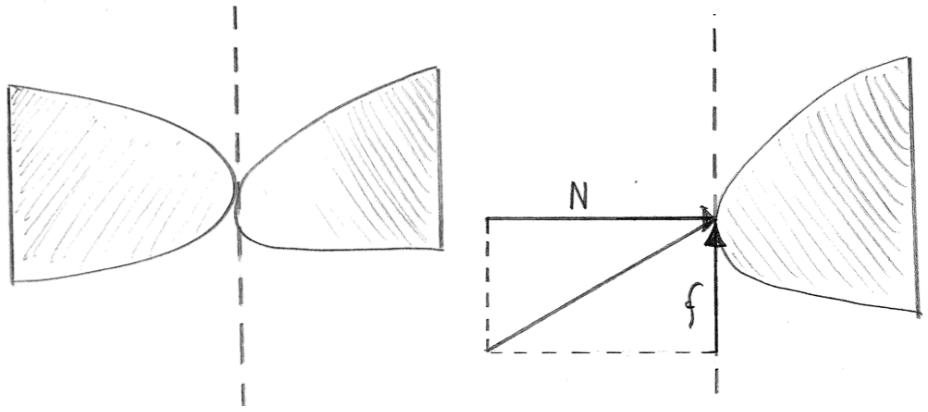



Figure 3.2: A contact force.

Forces can be represented by vectors and therefore be resolved into components. Resolving a force into its components sometimes eases the analysis of a problem. Figure 3.2 illustrates the contact force resolved into two components $N$ and $f$, defined as the force normal to the contact plane and a friction force along this plane, respectively [BF95b, page 80].

### Equilibrium

To analyse a problem in physics, for example a free-body diagram, the notion of equilibrium is used. Equilibrium in general means an unchanging state or a state of balance. In mechanics *equilibrium* is defined as:

**Definition 1.** *A body is in equilibrium if it is either at rest or moving in a straight line with constant velocity [YFFS00, page 97].*

If a body is in equilibrium we can use definition 1 to analyse entities influencing the object's behavior. Depending on what types of problems we consider and which kind of results we are seeking, this definition of equilibrium must be interpreted differently to meet the constrains imposed on a given problem. We now consider three interpretations of the definition, with increasing generality of the problems they capture, in each step lowering the number of constraints imposed on the problem.

#### Equilibrium for Fixed Objects

When more than one external force act upon an object, the forces can be added to yield the *sum of forces*. The sum of forces is the effective net force acting upon the body. We use this knowledge to establish equilibrium for fixed objects. Statics define equilibrium as *conservation of forces* defined by:

**Definition 2.** *A body that can be modeled as a particle is in equilibrium whenever the vector sum of the forces acting on it is zero. That is:* $\sum f = 0$ *[YFFS00, page 329].*

We can use this definition of equilibrium to analyse problems involving fixed objects. That is: We can calculate unknown forces. Imagine that we apply external gravitational force to an object

placed on a table. With the equilibrium equation we can calculate the force provided by the table to support the object.

### Equilibrium for Moving Objects

Equilibrium for fixed objects does not apply to moving bodies. In order to model motion, more theory is needed. The simplest moving body is a rigid body. A rigid body is allowed to move but not to deform. Movement has two distinct components: *translation* and *rotation*. When applying the theory of equilibrium on rigid bodies, translation and rotation must be included into the definition of equilibrium.

**Definition 3.** *In addition to the requirements of equilibrium for fixed objects, the sum of all torques due to all external forces acting on the body, with respect to any specified point, must be zero. That is:* $\sum f = 0 \wedge \sum m = 0$ *[YFFS00, page 329-330].*

*Torque* or *angular momentum* is a measure of a rotational force. Torque is defined as: $m = r \times f$, where the force $f$ is acting on a line of action perpendicular to the *lever arm*, in a distance of $r$ [YFFS00, page 294-296]. Note that equilibrium for fixed objects is a subset of equilibrium for moving objects, and is exactly the special case where all torques are zero.

### Equilibrium for Deformable Objects

When considering deformable objects, we no longer use forces as the quantity to define equilibrium. To define equilibrium for a deformable object, we will instead used the *principle of conservation of energy* as define in definition 4.

**Definition 4.** *The mechanical energy in a conservative system is always constant [BF95a, page 160].*

In order to understand definition 4 we need to introduce: work, kinetic energy, conservative systems, potential energy, and mechanical energy. Again we note, although this time not directly apparent, that equilibrium for moving objects is a subset of equilibrium for deformable objects. In section 3.2 on page 23, when the theory of continuum mechanics is elaborated, we will return to what equilibrium is for a deformable, in that section it will be applied to a continuum.

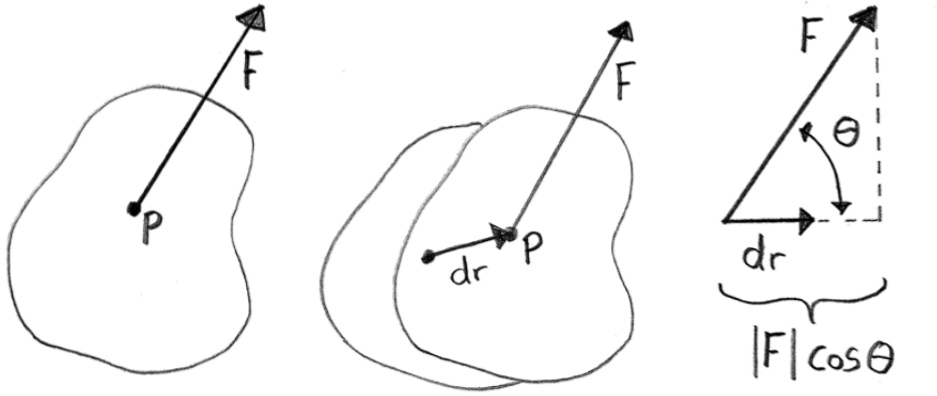
## Work

*Work* is the amount of energy transferred by a force acting over a distance. For a moving body where forces are not uniformly distributed work is defined in terms of *differential work*. So to calculate the total work $W$ we integrate $dW$, where $dW$ is the differential work done by $f$ as a result of the displacement $dr$. Work is measured in Newton meter ($Nm$) or Joule ($J$) [YFFS00, page 174-179].

$$dW = f \cdot dr = (|f|cos\phi)|dr| \qquad \Leftrightarrow \qquad W = \int dW = \int f \cdot dr \tag{3.1}$$

Consider a force $f$ acting on an object at point $P$ as illustrated in figure 3.3a on the facing page. Suppose the object undergoes an infinitesimal motion, so $P$ has displaced by vector $dr$ (see figure 3.3b on the next page). The differential work $dW$ done by $f$ as a result of the displacement $dr$ can then be illustrated as in figure 3.3c on the facing page. In this figure the force $f$ is not

parallel to $dr$ and therefore only a part of the force produces work. The precise amount of work produced by $f$ is equal to the length of $f$ projected onto $dr$ [BF95b, page 558].




(a) A force $f$ acting on an object (b) A displacement $dr$ of $P$ (c) $dW = (|f|cos\phi)|dr|$

Figure 3.3: Illustration of work.

### Energy

Although work and *energy* are related and both are measured in Joule (J), they are not entirely equal concepts. Work refers to a force acting over a distance. Energy on the other hand is used when referring to the amount stored or produced by an object. We use two concepts to describe an object's energy: *Kinetic energy* and *potential energy.*

#### Kinetic Energy

*Kinetic energy* is the energy of motion. Whenever an object moves it possesses kinetic energy. The amount of kinetic energy within an object with mass $m$ and velocity $v$ is defined as the work needed to accelerate the mass $m$ from rest to velocity $v$. The object maintains its kinetic energy as long as the velocity is constant. If the velocity increases or decreases so does the kinetic energy. It would require the same amount of work to decelerate an object to rest, as it took to accelerate it to velocity $v$. The force required to decelerate the object has to be in the opposite direction. The kinetic energy $E_k$ of an object is defined by [YFFS00, page 164-165]:

$$E_K = \frac{1}{2}mv^2 \tag{3.2}$$

Kinetic energy is related to work by the *work-energy theorem*, which states that work equals the change in kinetic energy [YFFS00, page 170]:

$$W = E_K^1 - E_K^0 = \Delta E_K \tag{3.3}$$

### Potential Energy

*Potential energy*, denoted $E_U$, is the energy stored within a physical system. An object can possess potential energy as a result of its position. If for example a heavy weight is elevated and held above the ground it possesses potential energy. The energy stored has the potential to be converted into a different form of energy, e.g. kinetic energy. Compressing a spring is another way of storing potential energy within an object. If the force compressing the spring is removed the spring's restoring force will make it return to its original position. Potential energy exists when an object is displaced and there are forces acting towards bringing the object back to its original position.

### Conservative and Non-conservative Systems

When kinetic energy can be converted to potential energy, and back again, we say that there is a two-way conversion of energy. Systems with the two-way conversion property are *conservative systems*, those without are *non-conservative systems*. A system where energy is turned into heat is an example of the latter [YFFS00, page 209-210]. Conservative systems are isolated and have the property that energy does not dissipate.

### Conservation of Mechanical Energy

When dealing with a conservative system the total amount of energy is refered to as the mechanical energy ($E_M$), and is defined by:

$$E_M = E_K + E_U \tag{3.4}$$

In a conservative system no energy dissipates therefore the mechanical energy within the system is constant. Kinetic energy can be stored by converting it to potential energy. The classic example is: When throwing a ball upwards, kinetic energy is converted into potential energy on the way up. On the way down the opposite happens. The relation that makes this possible is called *conservation of mechanical energy* and is a direct consequence of equation (3.4) for a conservative system [YFFS00, page 196]: If we consider an object that is deformed, it has an initial state (state 0), and a deformed state (state 1). In both states the energy can be calculated as:

$$E_M^0 = E_K^0 + E_U^0 \qquad E_M^1 = E_K^1 + E_U^1 \tag{3.5}$$

if the system is conservative, we know that:

$$\begin{aligned} E_M^0 &= E_M^1 && \Leftrightarrow \\ E_K^0 + E_U^0 &= E_K^1 + E_U^1 && \Leftrightarrow \\ E_K^0 - E_K^1 &= E_U^1 - E_U^0 && \Leftrightarrow \\ -(E_K^1 - E_K^0) &= E_U^1 - E_U^0 && \Leftrightarrow \\ -W &= E_U^1 - E_U^0 && \Leftrightarrow \\ -W &= \Delta E_U \end{aligned} \tag{3.6}$$

by setting the initial potential energy $E_U^0$ to zero, we obtain:

$$E_U^1 = -W \tag{3.7}$$

## 3.2 Continuum Mechanics

In continuum mechanics we assume that the substance of a body is a continuously distributed matter, that is distributed throughout and completely fills the space it occupies. We also assume that the substance is *homogeneous*, that is, the entire body consists of the same material. Continuum mechanics ignores the fact that matter is not continuous at the level of atoms. By ignoring this we can approximate physical quantities, such as energy and work, at the infinitesimal limit.

### Continuum

Continuum as a concept describes a material as a fixed region of space. This region forms the geometrical state of a body. We imagine the region completely covered by a set of infinitesimal volumetric elements, called *particles*. The position of all particles at a time $t$ is called a *configuration*. A particular configuration is chosen as the *reference configuration* and each particle in the reference configuration is identified by the position vector $X$. The reference configuration is often chosen as the configuration at time $t = 0$, where the body is in an undeformed state and at rest [Dil07, page 4].

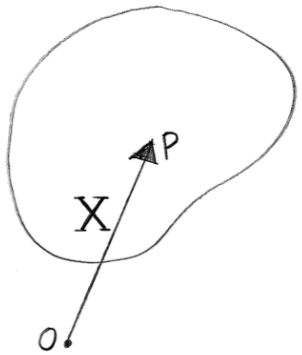



Figure 3.4: A continuum body.

The components of $X$ in the chosen reference frame are referred to as the *material coordinates*. We say that the particle $P$ occupies the place $X$ as illustrated in figure 3.4. Suppose that the region covered by the body at *reference time* $t = 0$ is $R_0$. If the body moves and at time $t$ occupies $R$, then we describe this configuration in relation to the reference configuration by *spatial coordinates* $x$. This scenario is illustrated in figure 3.5 on the following page. The spatial coordinates are the material coordinates plus a displacement: $x = X + u$, where $u$ is referred to as the *displacement vector*. Spatial coordinates can also be calculated as a function $f$ of the reference configuration and time: $x = f(X, t)$ [Spe80, page 33-35]. Note that by the definition of material and spatial coordinates, there has to be a one-to-one correspondence between *material points* and *spatial points*.

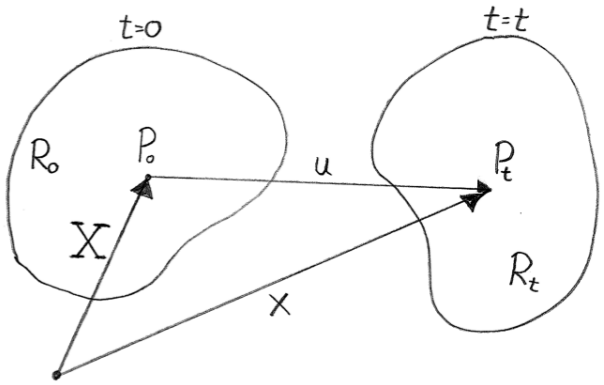



Figure 3.5: Displacement of a continuum.

### Deformation

If two configurations of a body are not the same, one or more of the particles have been displaced. A *displacement* can be divided into two distinct components: a *rigid body displacement* and a *deformation*. The difference being whether or not particles have moved in relation to each other. In a rigid body displacement the relative inter-particle distances are preserved. In a deformation they are not.

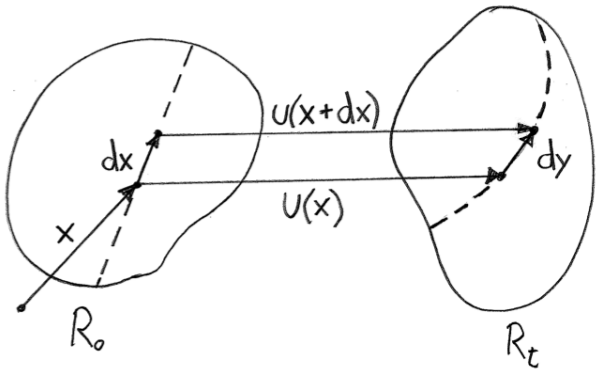



Figure 3.6: Deformation of an infinitesimal line segment.

Consider the straight line through the undeformed configuration as illustrated in figure 3.6. After the body has undergone a deformation the straight line has been mapped to a smooth curve as illustrated in the deformed configuration. Somehow we would like to represent the line or fiber deformation that takes place. Focus attention on the line segment $dx$ and imagine that the length of the line segment is much shorter than the radius of curvature. The small line segment is straight in the undeformed configuration and very close to straight in the deformed configuration. By decreasing the length of the line segments we improve the approximation of the curvature. This way we can describe even complex deformations of a body since it is reasonable to neglect curved material fibers as long as we only consider infinitesimal line segments. The infinitesimal line segments $dx$ and $dy$ are related by the ratio: $\frac{dy}{dx}$ [Bow09, section 2.1.2]. If this ratio is 1 the material is undeformed, between 0 and 1 it is compressed and if above 1 it is stretched.

### Strain

Strain is a measure of deformation. Strain represents the amount of stretch or compression by the relative distance between two particles in the material body. It is a dimensionless quantity, which can be expressed as a decimal fraction, or a percentage. In comparison with the ratio $\frac{dy}{dx}$ from section 3.2 on the facing page: A strain measure of zero means no deformation, below or above represents compression or stretching respectively. Strain can be decomposed into *normal strain* and *shearing strain*.

#### Normal Strain

Normal strain occurs when forces acting parallel to the axes of the reference frame pull equally throughout the body. Normal strain has the property of scaling the body in the axial directions. Figure 3.7 illustrates normal strain in one dimension. Here two external forces pull in opposite directions on a body hereby stretching it.

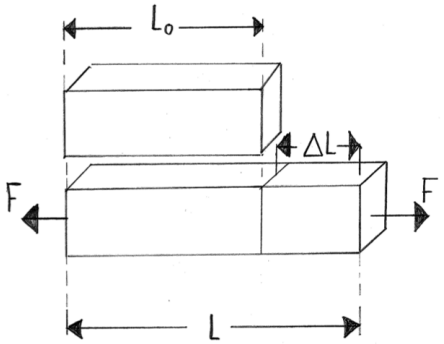



Figure 3.7: Normal strain.

For the simple case of a body axially loaded, the normal strain will be uniformly distributes through out the body and can be obtained by dividing the displacement (how much the body is stretched) by the initial length of the body.

$$\text{normal strain: } \frac{l - l_0}{l_0} = \frac{\Delta l}{l} \tag{3.8}$$

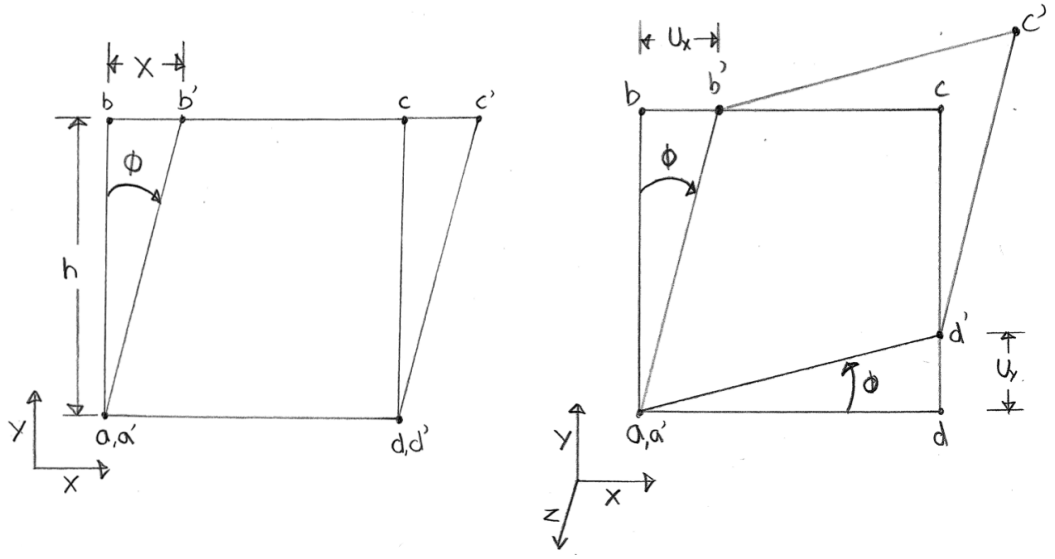
In general strain is not distributed uniformly over the entire body, therefore we define strain at a specific point within the body. Because continuum mechanics assume strain to be continues, differential calculus is used to define strain at any point. In three dimensions, we have a normal strain component for each direction in the reference frame. Written out as individual components this becomes: [HDSB01, page 659]

$$\varepsilon_x = \frac{\partial u_x}{\partial x} \qquad \varepsilon_y = \frac{\partial u_y}{\partial y} \qquad \varepsilon_z = \frac{\partial u_z}{\partial z} \tag{3.9}$$

were $u$ is the displacement vector with components $u_x$, $u_y$ and $u_z$ being the displacement along the $x$, $y$ and $z$ axis of the reference frame, respectively. Normal strain can be represented on vector form as: $\varepsilon = [\varepsilon_x,\ \varepsilon_y,\ \varepsilon_z]^T$.

### Shearing Strain

Shearing strain occurs when the forces are not aligned with the axes of the reference frame or if they are not uniformly distributed over the body. This results in distortion of the body's shape as illustrated in figure 3.8.




(a) Shearing strain in one direction.

(b) Shearing strain in two direction.

Figure 3.8: Shearing strain.

Shear strain is defined as the ratio of the displacement $x$ of the corner $b$ to the transverse dimension $h$ [YFFS00, page 343]:

$$\text{shear strain: } \frac{x}{h} = tan\phi \tag{3.10}$$

Figure 3.8a illustrates basic shearing strain in one direction. In three dimensions we are interested in defining the resulting shear strain between planes. Figure 3.8b illustrates the distortion from shear strain in two directions. To measure the resulting shear strain between the $xz$-plane and the $yz$-plane the two individual shearing strains is added together. The individual components of the shearing strain in three dimensions are [TG70, page 7]:

$$\gamma_{xy} = \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \qquad \gamma_{xz} = \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \qquad \gamma_{yz} = \frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y} \tag{3.11}$$

where $u_x$, $u_y$ and $u_z$ is the axial displacements along the $x$, $y$ and $z$ axis, respectively. Shearing strain can be represented on vector form as: $\gamma = [\gamma_{xy},\ \gamma_{xz},\ \gamma_{yz}]^T$.

## Internal Forces

Internal forces only exist inside an object, where one part of an object is subjected to forces by another part of the same object [BF95b, page 79]. Internal forces can be experienced simply by

stretching a rubber band. When stretching a rubber band the external force, responsible for the stretching, induces internal forces. When the external force is released the rubber band returns to its original shape hereby releasing the internal forces as well. The way a body behaves depends on its material properties. How different materials behave is elaborated in section 3.3 on page 29, but first stress is introduced as this is a representation of the internal forces.

### Stress

Stress is a measure of the average amount of force exerted per unit area ($N/m^2$). It is a measure of the intensity of the total internal forces acting within a body across imaginary internal surfaces, as a reaction to external applied forces.

$$\text{stress} = \frac{f}{A} \tag{3.12}$$

As with strain, stress can be decomposed into two separate components: *normal stress* and *shearing stress*. Consider an infinitesimal cubic element with edges parallel to the axis as illustrated in figure 3.9.

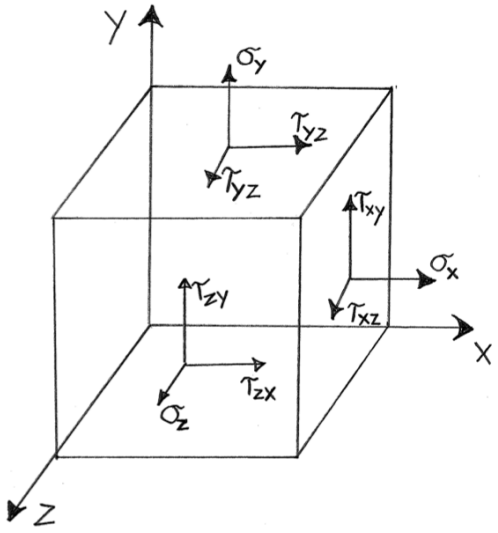



Figure 3.9: Stress in three dimensions.

The components of normal stress is denoted $\sigma_x$, $\sigma_y$ and $\sigma_z$, one for each axial stress acting perpendicular to a body face. Normal stress is either tension or compression, often causing change in volume. Shearing stress is denoted $\tau_{xy}$, $\tau_{yz}$ and $\tau_{zx}$, each acting parallel or tangential to a body face causing it to deform without particular volume change. The first letter in the double subscript indicates the *plane* in which the stress acts; whereas the second subscript indicates in which *direction* the stress acts. For example $\tau_{xy}$ has $x$ as its first subscript therefore the stress acts in the $yz$-plane because the coordinate direction $x$ is a normal to this plane. Second subscript is $y$ indicting the actual stress direction. We assume equilibrium of the element's volume, therefore $\tau_{xy}$ must be equal to $\tau_{yx}$ or else the element would not be at rest. This means we only need three shear stress components since $\tau_{xy} = \tau_{yx}$, $\tau_{yz} = \tau_{zy}$, and $\tau_{zx} = \tau_{xz}$. Altogether we only need

the six independent quantities $\sigma_x$, $\sigma_y$, $\sigma_z$, $\tau_{xy}$, $\tau_{yz}$ and $\tau_{zx}$ to completely describe the state of stress at a given point in the continuum [HDSB01, page 658]. When considering stress in three dimensions we refer to the measure as a force per volume, rather then the individual components as force per area. The normal stress can be represented on vector form as $\sigma = [\sigma_x,\ \sigma_y,\ \sigma_z]^T$ and the sharing stress as $\tau = [\tau_{xy},\ \tau_{xz},\ \tau_{yz}]^T$.

### Strain Energy

The potential energy stored within a deformed body is called *strain energy*. Strain energy is calculated the same way, we calculate work, but instead of relating force and displacement we use the stress and strain induced in the body. The body's internal potential strain energy is calculated as:

$$E_S = \int_V \varepsilon \cdot \sigma \ dV + \int_V \gamma \cdot \tau \ dV \tag{3.13}$$

where the strain and stress vectors dotted and integrated over the volume $V$ yields the internal potential strain energy. The integral insures that the internal forces are distributed uniformly throughout the volume [Str86, page 224]. Because stress is a force per volume measure, the strain energy is also a force per volume measure.

### Strain Energy Density

The *strain energy density* $dE_S$, is the strain energy per unit volume, which can be obtained by [TG87, page 625]:

$$dE_S = \int_0^{\varepsilon} \sigma \ d\varepsilon + \int_0^{\gamma} \tau \ d\gamma \tag{3.14}$$

and when stress-strain curves are introduced in section 3.3 on the next page, this constitutes the area below such a curve. Strain energy and strain energy density are directly related by the following equation:

$$E_S = \int_V dE_S \ dV \tag{3.15}$$

### Equilibrium for a Continuum

As described in section 3.1 on page 20, equilibrium for a deformable object is defined in terms of energy. Here we will specify this definition for a continuum. Recall from section 3.1 on page 22 that when an object has changed shape we consider two states of the object and their energy relation. Equation (3.7) is repeated below, it relates the change in potential energy to the work done be kinetic energy. Equilibrium for a continuum is precisely this, the relation between the work done by external force, and the resulting internal force measured as potential energy.

$$-W = \Delta E_U$$

As normally done we set the initial potential energy $E_U^0 = 0$, which means that the equation becomes: $-W = E_U^1$. The internal potential energy is the strain energy for a continuum and is defined in equation (3.13) as:

$$E_S = \int_V \varepsilon \cdot \sigma \ dV + \int_V \gamma \cdot \tau \ dV$$

The work done by the external forces is defined to be: surface work + body work. Forces acting upon the body (volume) are denoted $f_V$ and traction (surface forces) $f_S$, $u$ is the displacement vector, $V$ the volume and $S$ is the surface. The work is calculated by integrating force times displacement over the volume and surface, as described in section 3.1 on page 20.

$$\text{surface work: } \int_S u \cdot f_S \; dS \qquad \text{body work: } \int_V u \cdot f_V \; dV \tag{3.16}$$

A surface integral can be converted into a volume integral using the Gauss divergence theorem. By doing this the total external work can be expressed as [Mas70, page 49]:

$$\text{external work: } \int_V u \cdot f_V \; dV \tag{3.17}$$

Substituting these definitions into the equation: $-W = E_P^2$, we obtain:

$$-(\int_V u \cdot f_V \; dV) = (\int_V \varepsilon \cdot \sigma \; dV + \int_V \gamma \cdot \tau \; dV) \tag{3.18}$$

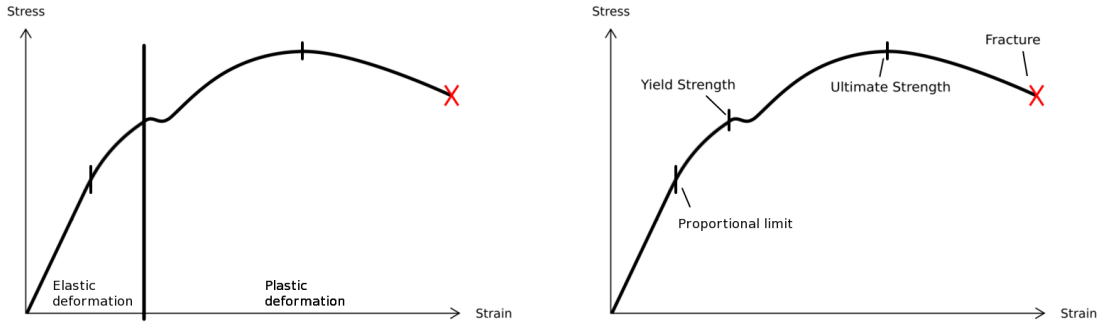
Consider equation (3.18), when external forces are applied, the body reacts by either adjusting the displacement or the level of internal energy.

## 3.3 Mechanical Properties

We will now elaborate on how stress and strain is related. Mechanical properties of a material describe how stress and strain are related in the given material. As the stress and strain increases with the external force applied, the material behaviour changes. Mechanical properties are theoretically divided into different phases based on how the material behaves. The first phase represents *elastic deformation*, here the material absorbs the forces and stores them as stress and strain. The second phase represents *plastic deformation*, during this phase the internal structures of the material changes permanently. If the amount of stress and strain exceeds the point of *fracture* the material will collapse. Elastic deformation is reversible, but the other deformation types are not. [YFFS00, page 344-345]

### Stress-strain Curves

To visualize how a material goes through the different types of deformation, we look at stress-strain curves. A stress-strain curve is a graph where measurements of stress as a function of strain, are plotted. The graph should be interpreted as follows: When moving away from origin along the x axis, the material gets more and more stretched (or compressed) depending on the direction. Normally the stress-strain curve only includes stretch, which is positive along the x axis. Isotopic materials yield the same stress-strain curve in any direction, therefore only one stress-strain curve is needed. To describe an anisotropic material more than one stress-strain curve is needed. The stress-strain curves in figure 3.10 on the next page, show an overview of how to interpret the curves. Figure 3.10a illustrates the different phases and figure 3.10b the names of the transition points between the phases. The transition point between linear and non-linear elasticity is called *proportional limit*. Between elastic and plastic deformation it is called *yield strength*, and the material will break when reaching the *fracture* point. The maximum amount of stress during plastic deformation is called the *ultimate strength* limit.




(a) Phases. (b) Points.

Figure 3.10: Stress-strain curves overview.

### Elastic Deformation

Elastic deformation or elasticity is the physical property of a material when it deforms due to external forces applied, but returns to its original shape when the forces are removed. The material returns to the original form because the applied forces are stored as stress and strain. The relation between stress and strain under elastic deformation can be either linear, non-linear or both depending on the material properties. *Linear elasticity* is when strain is directly proportional to stress. *Non-linear elasticity* is when the relationship between strain and stress is more complex and can be approximated by a non-linear function. A material with both properties is illustrated in figure 3.11.

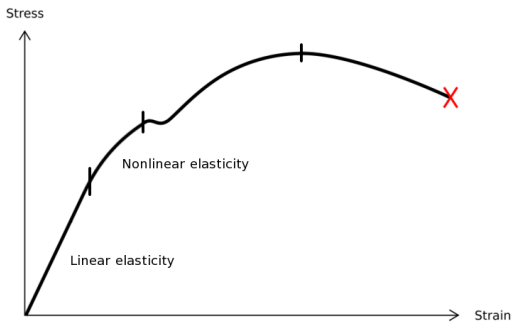


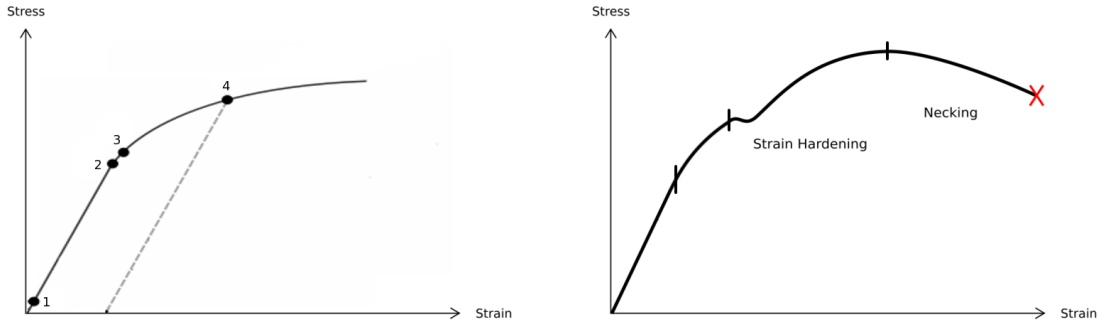

Figure 3.11: Stress-strain curve illustrating elastic deformation.

Elastic deformation is completely reversible. If the stress is reduced to its former value the strain falls back to the original level [ALRA90, page 183].

### Plastic Deformation

*Plastic deformation*, also known as *plastic flow*, is a irreversible deformation, which means that the body will not return to its original configuration when the external forces are released. When

the amount of external forces exceeds the yield strength the plastic deformation phase takes over. Plastic deformation changes the rest shape of a material, by changing its structure at the level of atoms. Atoms are bound in a grid structure. This grid structure is permanently changed by *dislocations* along *slip planes*. [ALRA90, page 191-205]. An important property of plastic deformation is that it preserves the volume of the material. An example of a plastic deformable material is metal. If metal is bent, the material's rest shape is permanently changed. The stress-strain curve in figure 3.12a shows a material that has gone through linear elastic deformation (stage 1-2), non-linear elastic deformation (stage 2-3) and plastic deformation (3 and onwards). If at a point (4), the external forces are released, the material has obtained a permanent deformation and therefore a new rest shape. The material returns to the new rest shape via the dotted line. If the material is, once more, subjected to external forces, the deformation of the material follows the dotted line until the point of yield strength is reached, and continues into plastic deformation if enough forces are applied [Dil07, page 159].




(a) Plastic deformation.

(b) Stress-strain curve of strain hardening and necking.

Figure 3.12: Stress-strain curves illustrating plastic deformation.

Plastic deformation covers many distinct physical processes. *Strain hardening* and *necking* are both examples of physical processes which can be seen on the stress-strain curve as illustrated in figure 3.12b. Strain hardening or *work hardening* is when a material becomes increasingly saturated with dislocations. When more and more dislocations are introduced the material starts to develop a resistance against new dislocations. This resistance shows itself as a strengthening of the material [ALRA90, page 194]. When a material is plastic deformed, and after the material's ultimate strength limit has been reached *necking* begins. The term *neck* refers to a point in the material where the material gets weaker than the rest of the material. Necking is part of the fracturing process for soft materials like metal [ALRA90, page 223].

### Fracture

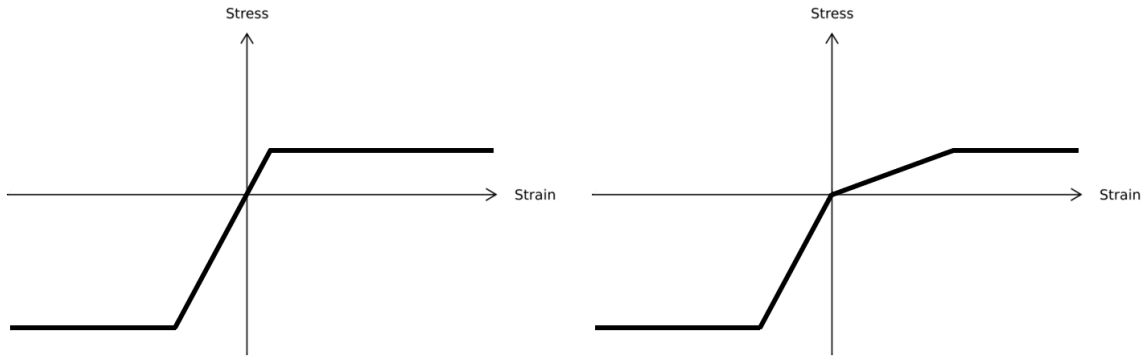
If deformation exceeds the point of fracture, the material collapses hereby releasing its internally stored forces. Material collapse, known as fracture, is a subject of its own: *Fracture mechanics*. Fracture mechanics is elaborated in section 3.5 on page 34.

### Brittle and Ductile Materials

Solid materials can be categorised according to their material properties. Before reaching the point of fracture, a material will undergo plastic deformations to some extent. If the amount of plastic deformation in a material is almost non-existing, as in the case of glass, it is reasonable to neglect this type of deformation. When this is the case the material is considered *brittle* and the point of yield strength ≈ fracture point. Materials, like metal, where plastic deformation is important are known as *ductile* materials [YFFS00, page 345].

### Variations in the Material Properties

Materials can have different properties when compressed compared to when stretched. Concrete for example, is much stronger when compressed than when stretched. Figure 3.13a illustrates this in one stress-strain curve by using that compressive strain is negative. Here the fracture point is much higher under compression that under stretch. The material can have a particular stress and strain relation under compression and another when stretched, hereby yielding two different slopes, as illustrated in figure 3.13b.




(a) Different fracture points. (b) Different slope.

Figure 3.13: Stress-strain curves of materials with different properties during compression contra stretch.

## 3.4 Linear Elasticity

When modeling brittle material, only the elastic deformation of a material is of interest, and we shall assume that the stress-strain relation is linear. Furthermore we shall assume that the material is *isotropic*, i.e. that it has the same properties is all directions. Linear elasticity in the simplest form is described by *Hooke's law* of elasticity. Hooke's law relates deformation and external forces by stating: *Extension of a material is in direct proportion with the forces acting upon it.* This is true as long as the force does not exceed the elastic proportional limit. The most commonly encountered form of Hooke's law is *the spring equation*, which relates the force exerted by a spring to the distance it is stretched or compressed by a spring constant $k$ measured in force per unit length $(N/m)$.

$$f = -k \cdot \Delta x \tag{3.19}$$

The negative sign indicates that the force exerted by the spring is in opposite direction of the displacement. To extend Hooke's Law into three dimensions we need to introduce more than one constant which relates forces and deformation. In keeping with generally used terms in theory of linear elasticity these constants are denoted moduli. More than one modulus enables us to relate the degrees of freedom in three dimensions. There are many different moduli or constants in linear elasticity but we only need two: the elastic modulus and Poisson's ratio, and how they relate stress and strain to cover all degrees of freedom.

### Modulus of Elasticity

The *elastic modulus* or *Young's modulus* $E$ relates normal strain and normal stress and is defined to be:

$$E = \frac{\text{normal stress}}{\text{normal strain}} = \frac{\sigma}{\varepsilon} \qquad \Leftrightarrow \qquad \varepsilon = \frac{\sigma}{E} \tag{3.20}$$

The elastic modulus is material dependent and can be found in reference books like [MW05]. Seen on a stress-strain curve a modulus is the slope of the linear piece of the graph. A high modulus indicates a hard or stiff material whereas a low modulus indicates a soft or bendy material.

### Poisson's Ratio

When a body is stretched or compressed in one direction, it tends to contract or expand in the opposite two directions. Poisson's ratio $\nu$ is a direct measurement of this phenomenon for isotropic materials. Consider a material in two dimensions, when it is compressed by a force along the $x$-axis it results in the strain $\varepsilon_x$. Due to the compression the material will expand $\varepsilon_y$ in the y direction, as follows:

$$\varepsilon_y = -\nu\varepsilon_x \qquad \Leftrightarrow \qquad \nu = -\frac{\varepsilon_x}{\varepsilon_y} = -\frac{\Delta x/x}{\Delta y/y} \tag{3.21}$$

Here $\nu$ is Poisson's ratio, relating the compression and expansion. To express this as stress, we use equation (3.20) and get:

$$\varepsilon_y = -\nu\varepsilon_x = -\nu\frac{\sigma_x}{E} \tag{3.22}$$

Like the elastic modulus, Poisson's ratio can be found in reference books.

### Relating Stress and Strain in Three Dimensions

To relate stress and strain in three dimensions, we need to combine the elastic modulus and Poisson's ratio in all dimensions. Consider a cubic element with tensile stress $\sigma_x$, $\sigma_y$ and $\sigma_z$ acting in the $x$, $y$, and $z$ direction, respectively. Due to the stress component, $\sigma_x$, the cube will stretch (if $\sigma_x > 0$) in the $x$ direction by the amount which according to equation (3.20) is $\varepsilon_x = \sigma_x/E$ and contract in the $y$- and $z$-direction by an amount of $-\nu\sigma_x/E$. Similarly for the remaining two stress components. The total deformation of the cube is now found by adding the various contributions [ALRA90, page 187]:

$$\begin{aligned}\varepsilon_x &= \frac{1}{E}\left[\sigma_x - \nu\left(\sigma_y + \sigma_z\right)\right] \\ \varepsilon_y &= \frac{1}{E}\left[\sigma_y - \nu\left(\sigma_x + \sigma_z\right)\right] \\ \varepsilon_z &= \frac{1}{E}\left[\sigma_z - \nu\left(\sigma_x + \sigma_y\right)\right]\end{aligned} \tag{3.23}$$

In equation (3.23) the relation between normal stress and strain are completely defined by the two constants $E$ and $\nu$. These constants can also be used to relate shearing stress and strain. The relationship between shearing stress and shearing strain is called the *Shear modulus* or *modulus of rigidity* and is denoted $G$.

$$G = \frac{\text{shear stress}}{\text{shear strain}} = \frac{\tau_{xy}}{\gamma_{xy}} \qquad \Leftrightarrow \qquad \gamma_{xy} = \frac{\tau_{xy}}{G} \tag{3.24}$$

The shear modulus can be deduced from $E$ and $\nu$ as done in [TG70, page 9-10], the relation is:

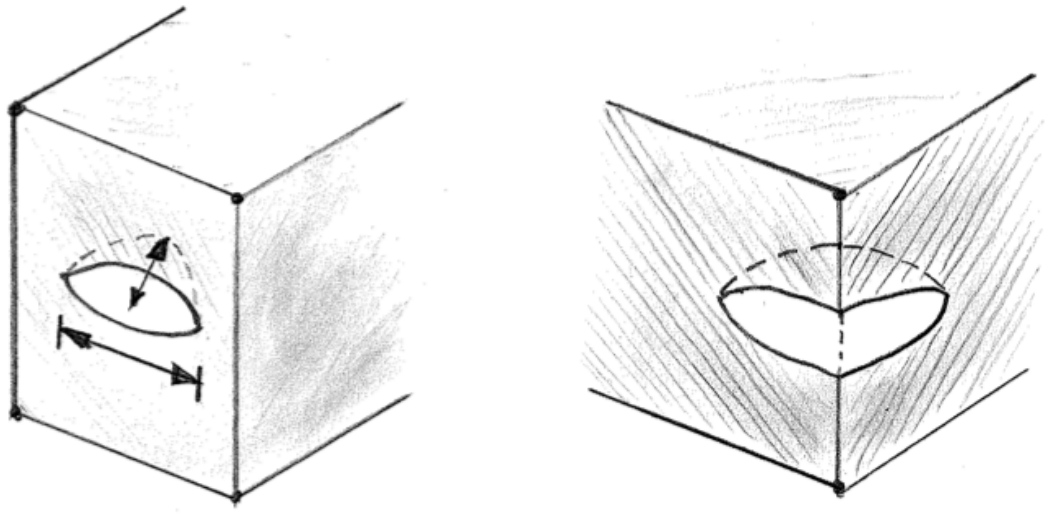
$$G = \frac{E}{2(\nu + 1)} \tag{3.25}$$

Shearing in one direction does not affect shear in other directions. Note that shear is independent of normal stress and strain [Dru67, page 266]. For shear in three dimensions, the equations are defined as:

$$\gamma_{xy} = \tau_{xy}/G \qquad \gamma_{yz} = \tau_{yz}/G \qquad \gamma_{zx} = \tau_{zx}/G \tag{3.26}$$

## 3.5 Fracture Mechanics

*Fracture mechanics* describes what happens to a material when the fracture point limit has been exceeded. When the material reaches this limit, a fracture is initiated at a specific *crack point* within the material. The fracture propagates by opening a crack inside the material along a *fracture plane*. The crack can be described by connected *crack surfaces*. The crack opens to release potential energy accumulated within the material. The internal potential strain energy, is used to create new *crack surfaces* by ripping part of the body into surfaces. This conversion of body to surface requires work, which is how the energy is released [ALRA90, page 218]. The *crack length* is determined by the amount of internally stored energy contra how much energy it takes to create the new surfaces. When considering an isotopic material in three dimensions, we idealize the crack surfaces by using two surfaces, which form a lens as illustrated in figure 3.14 on the facing page.



(a) Half lens. (b) Quarter of a lens.

Figure 3.14: Two crack surfaces forming a lens.

### Crack Initialization

There are different theories of how to calculate the crack initialization point, we use: *maximum principal stress* since this technique has proved suitable for brittle materials [Sin08, page 370]. *Principle stress* is a measure where both shearing and normal stress is incorporated, yielding three vectors. The longest vector of the three revels in which the direction the maximum stress points in a given point within the material. The method of maximum principle finds the maximum principle stress of all points in the material, yielding the precise point and direction of the stress. If this stress exceeds the fracture point ($\sigma_F$), the material breaks in this precises location. If a material have different fracture point limits when stretched or compressed, they are denoted: $\sigma_F^+$ and $\sigma_F^-$. How to calculate the principle stress based on normal and shearing stress will be elaborated in section 4.2 on page 47, when linear transformation has been introduced. The principle stress also contains information about how the crack surfaces will form. The vectors direction defines the normal of the crack plane.

### Crack length

The energy it takes to form the new surfaces is determined by the crack surfaces area and the *strain energy release rate*, which is a material property. The length of the crack is determined by the amount of strain energy stored in the point where the crack opens and the amount it takes to create the crack surfaces. The amount of energy used to create a surface in a give material depends upon the material itself and on the area of the crack surfaces. The area of the surfaces when using the lens form is further approximated by using a circle. This is done because the lens is much wider that thick and therefore a circle covers most of the area as illustrated in figure 3.15 on the following page, where the crack is seen from above. The circle area is calculated by: $A = \pi a^2$.

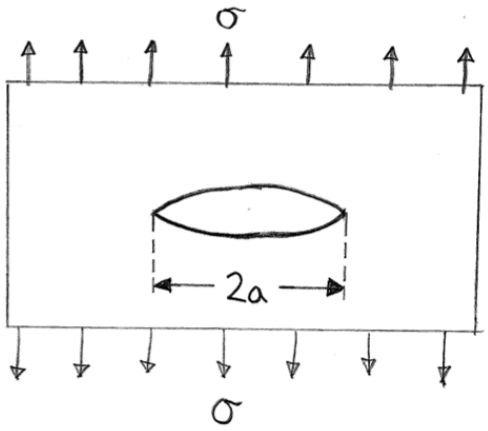
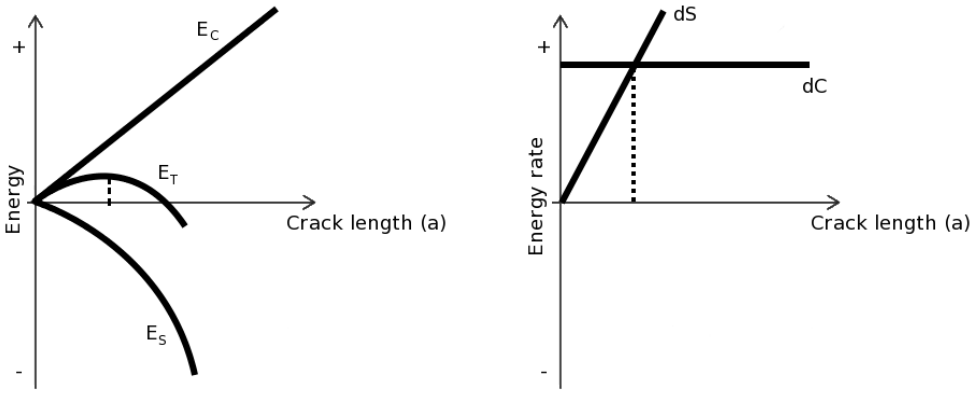



Figure 3.15: Crack surfaces seen from above.

The crack continues to grow until the strain energy cannot expand the crack surfaces further. Figure 3.16a illustrates an example of this. The graph $E_C$ depicts the amount of energy it takes to produce the crack surfaces as a function of the crack length $a$. The graph $E_S$ is the internal strain energy in the body. Focusing on graph $E_T$ this graph describes the total energy ($E_T = E_C + E_S$), here the fracture point is reached at the dotted line, and a crack is initialized, hereby releasing energy. Looking at the strain energy function $E_S$ we see that this energy rapidly increases as the crack starts to propagate. When the $E_T$ graph crosses the $x$ axis, there are no more energy left and the crack propagation stops.




(a) Strain contra surface energy.

(b) Derivatives of strain and surface energy.

Figure 3.16: Strain contra surface creation energy.

Figure 3.16b illustrates the energy derivatives of $E_C$ and $E_S$ from figure 3.16a. Here the point where the strain energy exceeds the surface creation energy becomes directly apparent [ALRA90, page 219-221]. In figure 3.16b, the following notation is used:

$$dS = \frac{\partial E_S}{\partial a} \qquad dC = \frac{\partial E_C}{\partial a}$$

### Crack propagation

Brittle material often have a single crack propagating through the entire material [Bow09, section 9.1]. Once the crack begin to propagate, the stress required for further propagation decreases as $a$ increases and therefore the crack accelerates rapidly [ALRA90, page 220].

# Chapter 4

# Mathematics

## 4.1 Linear Transformations

A linear transformation, or linear mapping, is a function $f$ that maps one vector space into another with the property that[Leo06, page 175]:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \tag{4.1}$$

where $\alpha$ and $\beta$ are scalars. An *affine* transformation from a Euclidean space into another is a mapping which preserves collinearity of points and ratios of distances between points on a straight line. In other words, points which lie on a straight line before the transformation continue to do so after the transformation is applied. Consider point $a$, $b$ and $c$, the ratio $\frac{|b-a|}{|c-b|}$ is preserved when using affine transformations. An affine transformation can be a translation, a rotation, a scaling, a shear or a combination hereof using matrix multiplication[WP01, page 2]. A common form of a linear equation in two variables, $x$ and $y$ is

$$y = Ax + b \tag{4.2}$$

where $A$ is a matrix representing rotation, scaling or shearing and $b$ represents translation. The set of solutions to this equation forms a straight line hence the name linear. Using *homogeneous* coordinates means representing an $n$-dimensional vector by $(n+1)$ components. This way we can include translation in the set of affine transformations and hereby get rid of $b$ in equation (4.2). The following examples shows commonly used affine transformations.

### Translation

Here is an example of an affine three-dimensional translation:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$t_x$, $t_y$ and $t_z$ are the axial translations in the $x$, $y$, and $z$ axis respectively. Only the rightmost column concerns the translation as opposed to scaling, rotation and shearing where only the upper-left $3 \times 3$ sub-matrix defines the transformation. Any affine transformation matrix can be applied to a vector $4 \times 1$ vector $v$ by using matrix multiplication $v' = T\ v$:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.4}$$

Notice that the fourth vector component has the value one since it is in homogeneous form. This way translation can be applied using matrix multiplication instead of the standard vector addition as shown in equation (4.2).

### Scaling

This is an affine scaling matrix:

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

$S_x$, $S_y$, and $S_z$ are the scaling factors in the $x$, $y$, and $z$ axis respectively.

### Shearing

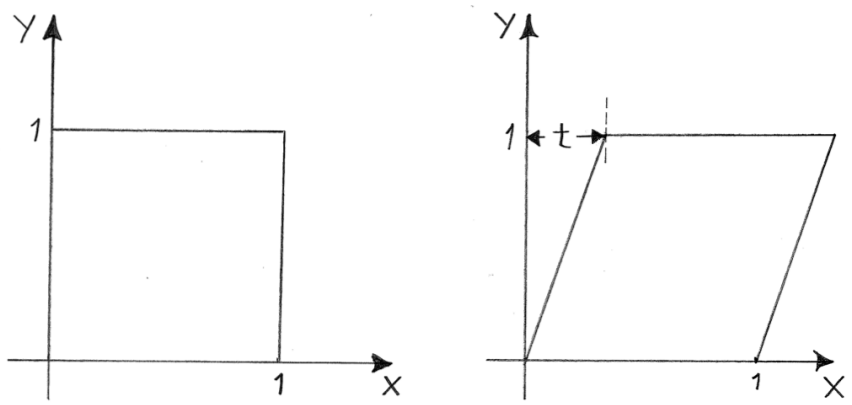
Generally shearing can be performed in an arbitrary plane. There are six basic shearing matrices [MH02, page 31]. One for each off-diagonal component in the upper-left $3 \times 3$ matrix as illustrated here:

$$H = \begin{bmatrix} 1 & S_{xy} & S_{xz} & 0 \\ S_{yx} & 1 & S_{yz} & 0 \\ S_{zx} & S_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

For simplicity the following example only illustrates shearing acting in planes orthogonal to axis of the coordinate frame. The following is an example of a shearing matrix in the $xy$-plane:

$$H_{xy} = \begin{bmatrix} 1 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

The result of applying the $H_{xy}$ shearing matrix to a unit square is illustrated in figure 4.1 on the next page.




(a) Before (b) After

Figure 4.1: Shearing in the xy-plane applied to a unit square.

### Rotation

Generally rotation can be applied around an arbitrary axis but for simplicity the examples here are limited to rotations around the $x$, $y$, and $z$ axis. The following three examples illustrate counter-clockwise rotation $\theta$ degrees around the $x$, $y$, and $z$ axis respectively:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

$$R_y = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

$$R_z = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

Constructing an affine rotation matrix requires certain matrix properties. To ensure preservation of collinearity and ratio of distances between points along lines the rotation matrix must be an *orthogonal* matrix. An orthogonal rotation matrix is by definition a square matrix with rows (or columns) that form an *orthonormal basis* [Leo06, page 258]. An orthonormal basis is a set of mutually perpendicular vectors all of magnitude one [Leo06, page 255] as illustrated in figure 4.2 on the following page.

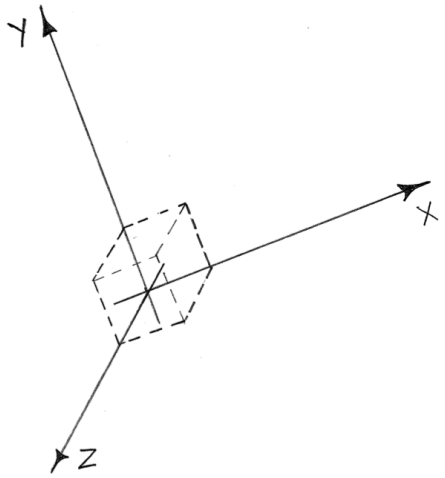



Figure 4.2: An orthonormal basis.

The determinant of any orthogonal matrix is either $+1$ or $-1$. If the determinant equals $-1$ it forms a reflection matrix rather than a rotation matrix. If the determinant of the orthogonal matrix equals 1 it forms a rotation matrix and these are the ones we are interested in. A rotation matrix has the following useful properties:

- Columns (or rows) are orthogonal unit vectors.
- The transpose equals the inverse:

$$AA^T = A^T A = A^{-1}A = I \tag{4.11}$$

- The determinant equals $+1$.
- Closed under multiplication and the inverse operation.

The fact that the inverse of a rotation matrix is the same as the transpose is very useful. The transpose of a matrix can be found just by swapping rows with columns, whereas finding the inverse usually means solving a system of linear equations using Gaussian elimination or another solving technique, so generally transposing a matrix is computationally much faster.

## 4.2 Tensors

Tensors are a powerful abstraction proven very useful especially in physics and engineering. Because of its abstractness it is not an easy entity to describe. A tensor is a way of describing linear transformations according to certain rules. A tensor is in fact just a linear function, but in applied mathematics it sometimes seems like a “magic” entity.

The use of tensors are probably best motivated through an example. Consider the life of a pirate sailing on the ocean only powered by the wind. Dependent on the weather conditions the wind will have a certain speed and come from a certain direction. Wind could easily be represented as a vector $v$ where the magnitude of the vector represents the wind speed. When the wind hits the

sail it produces a force that will make the ship move. If the ship was always sailing in the same direction as the wind we could just represent the relation between wind and force with a simple scalar. The resulting force vector could simply be obtained by multiplying the wind vector with the scalar. You don't have to be a sailor to see that if the ship could only move in the direction of the wind only random sailing would be possible. Sailing is more complicated in real life. We need to represent the relation between wind and resulting force in a different way. We shall assume that this relation is linear, so doubling the wind speed will double the force. We could represent this linear relation between the wind and the force on matrix form:

$$T = \begin{bmatrix} \tau_{11} & \tau_{12} \\ \tau_{21} & \tau_{22} \end{bmatrix} \tag{4.12}$$

Here $T$ is an example of a tensor capable of relating two different quantities, one is the wind speed and direction the other is the force direction and magnitude. The resulting force could be expressed as:

$$f = T \cdot w \tag{4.13}$$

where $T$ is the tensor, $w$ and $f$ is the wind and force vector, respectively.

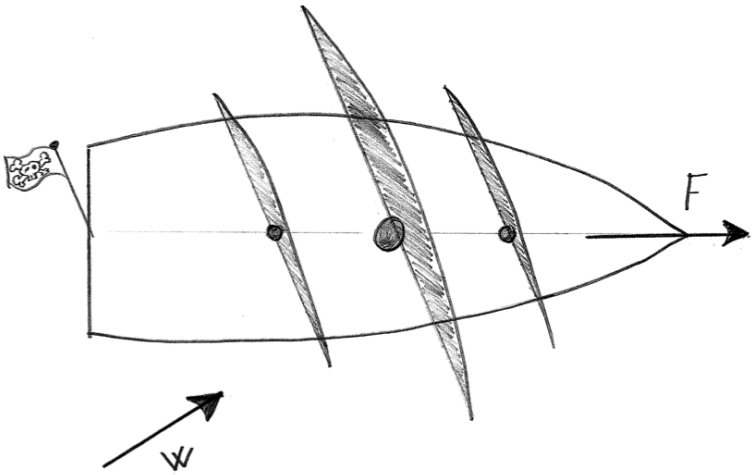



Figure 4.3: Wind from behind hits the sail and produces force.

As illustrated on figure 4.3 the pirate ship with its sail set, is going in a direction say 45° to the right of the wind direction. We represent the wind direction and speed as $w = [1, 1]^T$ hereby setting the wind speed to the length of the vector which equals $\sqrt{2}$. Lets say each wind speed unit produces two force units. A tensor representing this linear relation in direction and magnitude can be written in matrix form:

$$T = \begin{bmatrix} 2 \cdot (\cos\theta) & 2 \cdot (-\sin\theta) \\ 2 \cdot (\sin\theta) & 2 \cdot (\cos\theta) \end{bmatrix} \tag{4.14}$$

where $\theta$ defines the angle the vector will be rotated. The force produced by the wind can be obtained by:

$$f = \begin{bmatrix} 2 \cdot (\cos 45) & 2 \cdot (-\sin 45) \\ 2 \cdot (\sin 45) & 2 \cdot (\cos 45) \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2.828 \end{bmatrix} \tag{4.15}$$

Using tensor algebra we can begin to see why these abstract tensor entities are powerful. In the example we did only have one linear relation between wind and force, so we did only represent a single sail. A pirate ship probably has many sails with different size hence different linear relations between wind and force. The total force when three sails are set is obtained by:

$$f = R \cdot w + S \cdot w + T \cdot w \tag{4.16}$$

where $w$ is the wind and $R$, $S$, and $T$ are the three tensors each representing a linear relation between wind and force produced. Since wind with its direction and speed is constant equation (4.16) can be written as:

$$f = (R + S + T) \cdot w \tag{4.17}$$

Due to the fact that $R$, $S$, and $T$ are on matrix form and are linear they can be combined to form a new tensor $U$:

$$U = R + S + T \tag{4.18}$$

$$f = U \cdot w \tag{4.19}$$

Tensors are all about linear relations. Tensors of different orders exist we will classify tensors according to their order. All tensors can be represented by $n$-dimensional arrays where $n$ equals the tensor order. The number of indices needed to address each individual element in a tensor corresponds to the order. A scalar needs zero indices, a vector needs one, a matrix needs two etc.

A *zero order* tensor is simply a scalar and can be represented by a zero dimensional array e.g. just a single value. This scalar could represent the mass of a particle or the volume of an object. An example of a scalar could be the density of some fluid as a function of the position.

A *first order* tensor can be represented as a vector. Just like tensors a vector may be defined at a single point, or it may continuously vary from point to point hereby defining a vector field. Imagine how to model speed and direction of fluid through space. Each vector in the field has a direction and magnitude representing movement and speed respectively. In three dimensions a vector has three components, in four dimensions it has four, in n-dimensional space it has n components still representable as a first order tensor.

A *second order* tensor can be represented by a two-dimensional array, written out as a matrix. In three dimensions a second order tensor can be represented by a $3 \times 3$ matrix defining e.g. body stretch and rotation. If multiplying a vector by a second order tensor, the result is another vector by definition, so a second order tensor is a linear mapping of a vector onto another vector as in the example with the pirate ship. Second order tensors are very useful for describing movement and deformation of volumetric models. In continuum mechanics stress and strain are often best represented by second order tensors.

Tensors of all orders can be defined but as the order of the tensor increases so does the complexity. We will limit our discussion to second order tensors in three-dimensional space.

Before introducing an actual tensor definition it would be appropriate to introduce some basic terminology. Throughout the discussion second order tensors will be represented in the Cartesian

coordinate frame spanned by orthonormal vectors $e_i$, where $i \in \{1, 2, 3\}$. A vector $u$ in this frame is given by

$$u = \sum_{i=1}^{3} u_i e_i \tag{4.20}$$

Vector $u$ represented in the Cartesian coordinate frame is illustrated in figure 4.4.

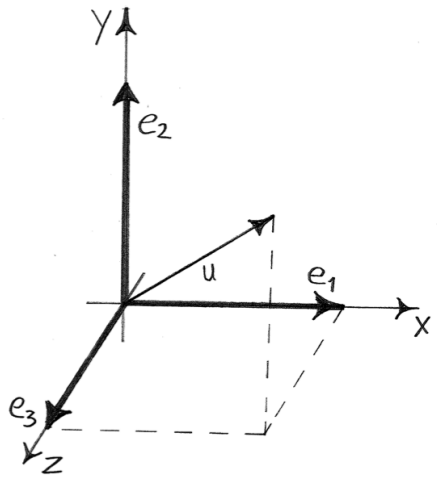



Figure 4.4: Vector $u$ in coordinate frame spanned by $e_i$, where $i \in \{1, 2, 3\}$.

In vector and especially tensor literature a special index notation if often used. If an index is repeated in a product of vectors or tensors, summation is implied over the repeated index. This is also known as the *Einstein Convention* and means the following is equivalent

$$\delta = a_i b_i \quad \equiv \quad \delta = \sum_{i=1}^{3} a_i b_i \quad \equiv \quad \delta = a_1 b_1 + a_2 b_2 + a_3 b_3 \tag{4.21}$$

**Definition 5.** *An entity is called a second-order tensor if it has nine components $T_{ij}$, $i \in \{1, 2, 3\}$, and $j \in \{1, 2, 3\}$, in the unprimed frame and nine components $T'_{ij}$ in the primed frame and if these components are related by the characteristic law, defined by:*

$$T'_{ij} = P_{il} P_{jm} T_{lm} \tag{4.22}$$

A tensor is defined as an entity, with components represented in a chosen coordinate frame, that can be represented in a different coordinate frame, related by the characteristic law.

It follows from definition 5 that if a tensor equation can be established in one coordinate frame then it must hold in any other coordinate frame [Bat07, page 502]. The components of a tensor changes if the coordinate frame changes. The characteristic law is what relates the representation in one coordinate frame to another and it can also be expressed on matrix form as:

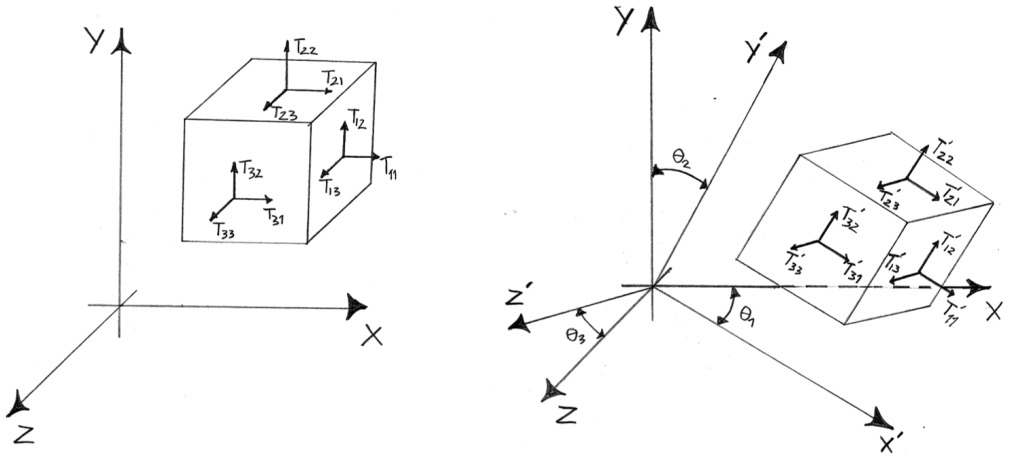
$$T' = P \; T \; P^T \tag{4.23}$$

where $P$ is the rotation matrix representing the change of basis from the primed to the unprimed coordinate frame. The tensor $T$ is represented in the unprimed coordinate frame and $P^T$ is the inverse rotation representing the change of bases from the primed back to the unprimed coordinate frame. $T'$ is the representation of the tensor in the primed coordinate frame.

Since $P$ is an orthogonal transformation matrix defining a rotation, the inverse of the transformation equals its transposed matrix $P^T$ as defined equation (4.11). So the stress tensor represented in the unprimed coordinate frame can be obtained by:

$$T = P^T \; T' \; P \tag{4.24}$$

According to definition 5 on the preceding page any tensor can be represented in a coordinate frame different from the initial coordinate frame.




(a) Initial unprimed coordinate frame

(b) Rotated primed coordinate frame

Figure 4.5: Transformation of tensor from unprimed to primed coordinate frame.

In addition to the unprimed frame, in figure 4.5a, consider a primed coordinate frame. The primed coordinate frame spans the same vector space as the unprimed but with different basis vectors $e'_j$ where $j \; \epsilon \; \{1, 2, 3\}$ as shown in figure 4.5b. Now consider any tensor $T$ represented in the unprimed coordinate frame. The tensor $T$ can be represented in the primed coordinate frame as illustrated in figure 4.5b through the relation defined in equation (4.23), on matrix form this expands to:

$$\begin{bmatrix} T'_{11} & T'_{12} & T'_{13} \\ T'_{21} & T'_{22} & T'_{23} \\ T'_{31} & T'_{32} & T'_{33} \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix} \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{bmatrix} \begin{bmatrix} P_{11} & P_{21} & P_{31} \\ P_{12} & P_{22} & P_{32} \\ P_{13} & P_{23} & P_{33} \end{bmatrix} \tag{4.25}$$

The diagonal elements of $T'$ represents the scale along the axis of the primed coordinate frame and the off-diagonal elements represents shearing.

### Strain Tensors

As already explained in section 3.2 on page 25 strain represents the amount of stretch or compression by the relative distance between two particles in the material body. Strain is a

dimensionless quantity which can be decomposed into normal and shearing strain. Normal and shearing strain can be represented by a second order tensor. Normal strain is the axial stretch or compression represented down the diagonal as if it was a scaling matrix. Shearing strain is represented in the off-diagonal components. Altogether we need six components to represent the entire strain measure, three normal strains and three shearing strains. As defined in equation (3.9) on page 25 normal strains are obtained by:

$$\varepsilon_x = \frac{\partial u_x}{\partial x} \qquad \varepsilon_y = \frac{\partial u_y}{\partial y} \qquad \varepsilon_z = \frac{\partial u_z}{\partial z}$$

and the three shearing strain, one for each angle between two planes are defined in equation (3.11):

$$\gamma_{xy} = \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \qquad \gamma_{xz} = \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \qquad \gamma_{yz} = \frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}$$

Normal and shearing strain can be represented as a second order tensor:

$$E_E = \begin{bmatrix} \varepsilon_x & \gamma_{xy} & \gamma_{xz} \\ \gamma_{yx} & \varepsilon_y & \gamma_{yz} \\ \gamma_{zx} & \gamma_{zy} & \varepsilon_z \end{bmatrix} \tag{4.26}$$

### Stress Tensors

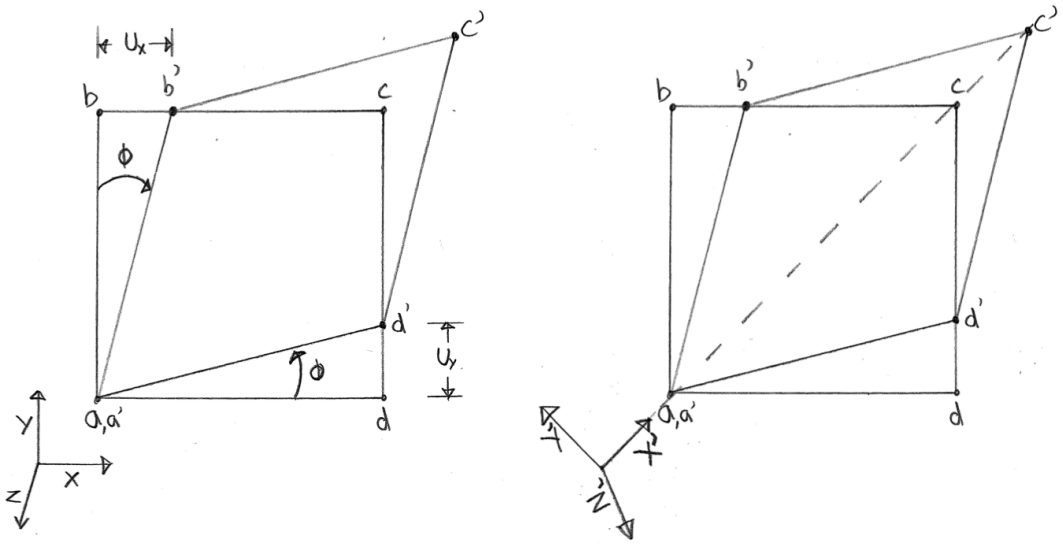
In continuum mechanics tensors are very useful for representing e.g. stress. As already explained in section 3.2 on page 27 stress can be decomposed into normal ($\sigma$) and shearing stress ($\tau$). In three dimensions the stress at any point in the continuum can be completely defined by the stress tensor:

$$S_E = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \tag{4.27}$$

where the three $\sigma$ components defines the normal stress and the six $\tau$ defines the shearing stress. Recall from section 3.2 on page 27 that since $\tau_{xy} = \tau_{yx}$, $\tau_{yz} = \tau_{zy}$, and $\tau_{zx} = \tau_{xz}$ the tensor becomes symmetric.

### Principal Values and Directions

It is often of our interest to determine the directions of the resulting strains also known as the *principal directions*. The sizes of the resulting strains acting in the principal directions are known as the *principal values*. Consider a strain tensor $E_E$ defining normal and shearing strain. When the tensor is represented on matrix form we recognize the normal strain in the axial directions down the diagonal just like a scaling matrix. The shearing is represented as the off-diagonal components introducing rotation capabilities into the matrix. Together the normal and shearing strain forms a transformation matrix mapping vectors from an unprimed to a primed coordinate frame. The base vectors spanning the primed vector space is exactly the principle directions we are interested in. Rotating the unprimed coordinate frame until its base vectors aligns with the base vectors of the primed coordinate frame, will effectively eliminate the rotation. The result from applying normal and shearing strain can hereby be represented combined as normal strain in the directions of the primed coordinate frame as illustrated in figure 4.6 on the next page.




(a) Unprimed coordinate frame with shearing strain. (b) Primed coordinate frame with normal strain.

Figure 4.6: Shearing strain represented as normal strain.

The base vectors of the primed coordinate frame defines the principal directions. The principal values defines the magnitude of the normal strains that act in the principal directions. The solution to the problem of finding the principal directions and values is equivalent to the solution of the *eigenproblem* as explained in section 4.3. The eigenvectors of any symmetric tensor $T$ is by definition three mutually perpendicular vectors. The eigenvectors defines the principal directions and the corresponding eigenvalues defines the principal values [Bat07, page 838]. The three eigenvectors are often referred to as the minimum, medium and maximum principal directions according to the size of their corresponding eigenvalues.

## 4.3 The Matrix Eigenproblem

Solving the matrix eigenproblem means finding eigenvalues and their corresponding eigenvectors for a given matrix. Finding the eigenvalues and eigenvectors reduces to the solution of linear equations [Leo06] as defined below

**Definition 6.** *Let* $\mathbf{A}$ *be an* $n \times n$ *matrix. A scalar* $\lambda$ *is said to be an* ***eigenvalue*** *or a* ***characteristic value*** *of* $\mathbf{A}$ *if there exists a nonzero vector* $\mathbf{v}$ *such that* $\mathbf{Av} = \lambda\mathbf{v}$*. The vector* $\mathbf{v}$ *is said to be an* ***eigenvector*** *or a* ***characteristic vector*** *belonging to* $\lambda$*.*

The equation $Av = \lambda v$ can also be written as

$$(A - \lambda I)v = 0 \tag{4.28}$$

According to definition 6 $\lambda$ is an eigenvalue of $A$ if and only if the corresponding vector $v$ is nonzero. Equation (4.28) has a nonzero solution of $v$ if and only if $det(A - \lambda I) = 0$ is satisfied,

this equation is also known as the *characteristic equation* for matrix $A$.

Expanding the determinant leaves us with an polynomial of $n$-degree for the $\lambda$ term. This polynomial is also known as the *characteristic polynomial*. The characteristic polynomial will have exactly $n$ roots being the eigenvalues from which we can find $n$ independent eigenvectors.

**Example**
Consider a matrix $A$ defined to be:

$$A = \begin{bmatrix} 1 & 1 \\ -2 & 4 \end{bmatrix}$$

Finding the eigenvalues and eigenvectors means solving equation (4.28). By insertion we obtain:

$$(A - \lambda I)v = \begin{bmatrix} 1 & 1 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

which is equivalent to:

$$(1 - \lambda)x_1 + x_2 = 0 \tag{4.29}$$
$$(4 - \lambda)x_2 - 2x_1 = 0 \tag{4.30}$$

With only two equations and three unknowns a third condition is needed. As previously stated $\lambda$ is an eigenvalue of matrix $A$ if and only if vector $v$ has a nonzero solution. If matrix $A$ is singular the equation (4.28) will have a nontrivial solution. From the characteristic polynomial we obtain a third condition:

$$det(A - \lambda I) = det \begin{pmatrix} 1 - \lambda & 1 \\ -2 & 4 - \lambda \end{pmatrix} = 0 \tag{4.31}$$

which expands to:

$$(1 - \lambda)(4 - \lambda) + 2 = 6 - 5\lambda + \lambda^2 = (\lambda - 3)(\lambda - 2) = 0 \tag{4.32}$$

Obvious $\lambda = 2$ and $\lambda = 3$ is a valid solution here. Standard techniques for finding roots in a given polynomial exists. Having established the two eigenvalues we still need to find their corresponding and independent eigenvectors satisfying the equation $Av = \lambda v$. By inserting $\lambda = 2$ into equation (4.29) and (4.30) we obtain:

$$-x_1 + x_2 = 0 \tag{4.33}$$
$$-2x_1 + 2x_2 = 0 \tag{4.34}$$

By looking at the coefficients and signs of the vector components $x_1$ and $x_2$ we see that the equation is satisfied with the base solution:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{4.35}$$

Any non-zero multiple of $[1, 1]^T$ is an eigenvector belonging to $\lambda = 2$. Eigenvector $[1, 1]^T$ is said to be a basis for the *eigenspace* corresponding to $\lambda = 2$. A scalar $\lambda$ with corresponding vector $v$ satisfying $Av = \lambda v$ defines an *eigenpair*.

The second independent eigenvector corresponding to $\lambda = 3$ is found just like before through insertion of $\lambda = 3$ into equation (4.29) and (4.30) hereby obtaining:

$$-2x_1 + x_2 = 0 \tag{4.36}$$
$$-2x_1 + x_2 = 0 \tag{4.37}$$

in order to satisfy the equation the solution must be:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{4.38}$$

We have now solved the eigenproblem for matrix $A$ by finding the two eigenvalues and their corresponding eigenvectors. The solution written as eigenpairs:

$$(2, \begin{pmatrix} 1 \\ 1 \end{pmatrix}), (3, \begin{pmatrix} 1 \\ 2 \end{pmatrix}) \tag{4.39}$$

Due to the fact that any second order tensor can be represented on matrix form, the eigenproblem can be solved for second order tensors by a similar approach to the one just shown. More general techniques for solving the eigenproblem exists but will not be further discussed here.

# Part II

# Discrete Theories

# Chapter 5

# The Equilibrium Framework

*Equilibrium problems* is the set of problems where the mathematical model describes conservation of something. It could be conservation of mass in fluids, conservation of current in electrical circuits, conservation of energy in solid mechanics, conservation of momentum in physics etc. Whenever the mathematical model describes a problem that involves an external conservation law, the particular problem is within the set of equilibrium problems. In the following discussion we will try to establish a fundamental framework for solving equilibrium problems.

A *steady state problem*, also known as a *static problem*, is a problem where only two configurations exists. The first one is the initial configuration describing the state of the system. If the problem is within the field of mechanics, the initial configuration will describe the systems masses, external loads, connecting springs and so forth. Finding the equations describing the behavior of the system, is also part of the initial configuration. Without introducing time, the system will react to the applied loads, springs will stretch etc., and eventually the system will reach its final state of equilibrium where everything stops moving, this is the second configuration.

## 5.1 The Structure of the Framework

The following discussion is limited to a one-dimensional steady state problem with masses connected by springs. This simple problem is one of the most basic problems in mechanics, yet it illustrates the general structure of the framework. The framework we are about to describe generalizes to all kinds of problems within the field of applied mathematics in $n$ dimensions.

Consider figure 5.1 on the following page. Three masses are connected by four springs. The top and bottom spring are fixed at one of their end points. This is the initial configuration. Gravity will pull down the masses forcing the springs to stretch or be compressed. Eventually the system will stop moving and hereby reach its final configuration. Equations describing the physical laws and the connection between forces acting and springs reacting to this, is exactly what we will try to find.

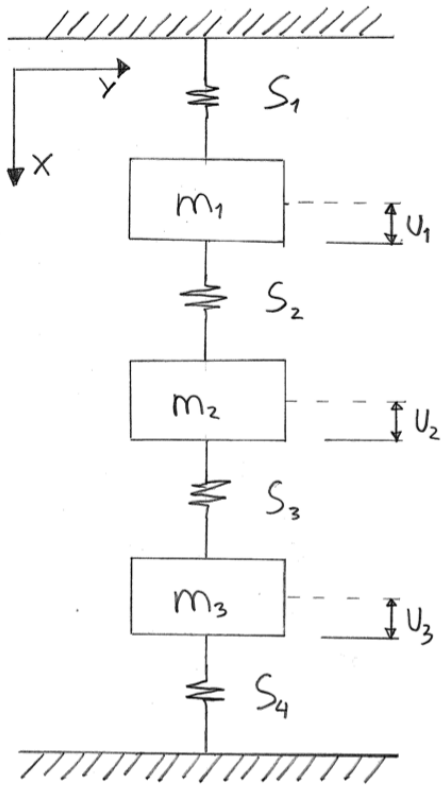



Figure 5.1: Basic spring mass problem.

Let's start by labeling the various quantities acting in the system. The three masses are represented by $m_1$, $m_2$, and $m_3$. The masses are connected by spring $s_1$, $s_2$, $s_3$, and $s_4$. Gravity $g$ acts on each mass hereby defining the external forces $f_1$, $f_2$, and $f_3$, defined to be positive downwards. The displacement of the masses, due to the forces acting on them, is represented by $u_1$, $u_2$, and $u_3$.

When the gravitational force acts on the masses it causes each separate spring to react. The internal spring forces are represented by $w_1$, $w_2$, $w_3$, and $w_4$. A positive $w_i$ value is defined to be stretch and a negative value is compression. For each spring we define a connecting quantity $e_1$, $e_2$, $e_3$ and $e_4$ one for each elongation of the spring.

By using vectors to represent each of the quantities $u$, $e$, $w$ and $f$ we can describe the problem by relating these as follows:

- There will be a matrix $A$ relating the displacement vector $u$ to a elongation vector $e$.
- There will be a matrix $C$ relating the elongation vector $e$ to a spring force vector $w$.
- There will be a matrix relating the spring force vector $w$ to the external force vector $f$.

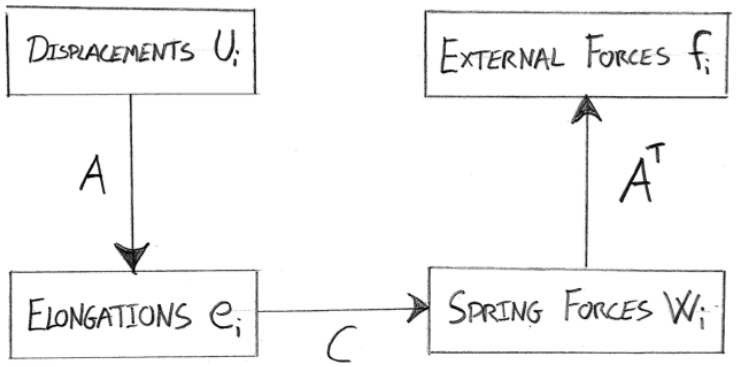



Figure 5.2: The structure of the framework.

Figure 5.2 illustrates how the different quantities can be related by matrices. This is what forms the significant structure of the framework used all over the field of applied mathematics. We are about to describe the fundamental framework for equilibrium steady state problems.

### Relating Displacements to Elongations

We will begin by constructing matrix $A$. This is the question of how much the springs stretch due to the displacement $u$. As seen on figure 5.1 on the facing page there are four springs and three mass displacements. The relation between the two quantities written on matrix form is:

$$e = A\ u \tag{5.1}$$

where vector $e$ equals $[e_1, e_2, e_3, e_4]^T$, vector $u$ equals $[u_1, u_2, u_3]^T$ and $A$ is the matrix relating the two quantities. The first spring $s_1$ stretches by the amount equal to the displacement of the first mass. So $e_1 = u_1$ because the spring is fixed at the top. The spring being fixed at one end introduces the first boundary condition. Later it will become clear why boundary conditions, limiting the solution space, are an important part of the framework. The elongation of the second spring equals the displacement $u_2$ subtracted by $u_1$ because we are only interested in the stretching of $s_2$. Looking at the third spring we isolate it by subtracting the springs already stretched above therefore $e_3 = u_3 - u_2$. The last spring $s_4$ is fixed at its end point causing it to compress since it can not move further downwards. The amount it is being compressed must be equal to the displacement of the last mass, note the negative sign due to compression, therefore $e_4 = -u_3$.

The equations are:

$$\begin{aligned} e_1 &= u_1 \\ e_2 &= u_2 - u_1 \\ e_3 &= u_3 - u_2 \\ e_4 &= -u_3 \end{aligned}$$

Written out in matrix form a certain pattern emerges

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{5.2}$$

As we shall see later, matrix $A$ contains information about the boundary conditions. In this example the boundary conditions are the two fixed springs, the one at the top and the one at the bottom.

### Relating Elongations to Spring Forces

We will now determine matrix $C$. Matrix $C$ is the one that relates the distances the springs have stretched to the internal spring forces $w$. Relating spring extension to a force is exactly what Hooke's law does. Hooke's law as defined in equation (3.19) on page 32:

$$f = -k\ \Delta x$$

where $\Delta x$ is the distance the spring has been stretched or compressed. $f$ is the restoring force and $k$ is the spring constant defining units of force it takes to stretch or compress the spring one unit length. The relation between the internal spring forces $w$ and the spring elongation $e$ is written as:

$$w = C\ e \tag{5.3}$$

where vector $w$ equals $[w_1, w_2, w_3, w_4]^T$, vector $e$ equals $[e_1, e_2, e_3, e_4]^T$ and $C$ is the $4 \times 4$ matrix relating the two quantities. Using Hooke's law we need to define a spring constant. When the spring constant is multiplied with the stretch $e$ the internal force $w$ are obtained. For each spring we define a spring constants $c_1$, $c_2$, $c_3$ and $c_4$. The internal spring forces is then obtained by:

$$\begin{aligned} w_1 &= c_1 e_1 \\ w_2 &= c_2 e_2 \\ w_3 &= c_3 e_3 \\ w_4 &= c_4 e_4 \end{aligned}$$

Equation (5.3) written on matrix form:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} c_1 & & & \\ & c_2 & & \\ & & c_3 & \\ & & & c_4 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} \tag{5.4}$$

where 0 is implied in all the blank entries of the $4 \times 4$ matrix $C$. Note how we just introduced an external law from the world of physics into the framework. The physical law was here introduced as a relation between two quantities in the framework.

### Relating Spring Forces to External Forces

The last relation that will take us all the way around the framework is the one between the internal forces $w$ and the external forces $f$. This is where the state of equilibrium is represented by a key equation, in this case the equation describes the conservation of forces as described by definition 2 on page 19.

We need to find the relation between the internal and external forces. When the system has reached its state of equilibrium all springs and masses have stopped moving, therefore the internal and external forces must cancel out. The only external force in this system is the gravity acting on each of the three masses, obtained by:

$$f_i = m_i \; g \tag{5.5}$$

where $g$ is the gravity and $i \; \epsilon \; \{1, 2, 3\}$.

Consider figure 5.1 on page 56 and look at the first internal force $w_1$ defined at spring $s_1$. When the system is in a state of equilibrium $w_1$ must cancel out the forces pulling down mass $m_1$. There are two forces pulling down mass $m_1$, the first one is $f_1$ as defined in equation (5.5). The second one is all the other masses below which equals whatever spring $s_2$ is holding defined by $w_2$. The first internal force $w_1$ is obtained by:

$$w_1 = f_1 + w_2 \tag{5.6}$$

which can be written as $w_1 - w_2 = f_1$. Now consider the internal force $w_2$, both $f_2$ and $w_3$ act downwards, therefore $w2$ must equal the sum of the contributions: $w_2 = f_2 + w_3$. The three relations between the internal forces and the external forces is obtained by:

$$\begin{aligned} w_1 - w_2 &= f_1 \\ w_2 - w_3 &= f_2 \\ w_3 - w_4 &= f_3 \end{aligned}$$

On matrix form these relations can be expressed as:

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \tag{5.7}$$

This is the crucial moment where we realize that the matrix relating the internal forces $w$ to the external forces $f$ is exactly $A^T$ - and it always is. No matter what kind of equilibrium problem we are trying to solve the structure as illustrated in figure 5.2 on page 57 always appears.

*"Nature produced $A^T$ there - we just sort of watched it happen."* (Gilbert Strang)

The fact that the relation between internal forces $w$ and external forces $f$ is the transpose of the relation between the displacements $u$ and the elongation $e$ is not by accident. What it means is: the change in internal stored energy when stretching the springs equals the amount of external work done at the masses (equation (3.6) on page 22).

## 5.2 Assembling the Equations

The equation $e = A\ u$ formed matrix $A$ and the equation $f = A^T w$ formed $A^T$. These two matrices appears from the geometry expressing how things are connected, how mass $m_1$ is connected to mass $m_2$ and so forth. There is always a matrix $C$ in the middle that represents the laws of physics or engineering, statistics or some other external law depending on the problem domain.

In this case the three equations that took us around the framework were:

$$e = A\ u \tag{5.8}$$

$$w = C\ e \tag{5.9}$$

$$f = A^T w \tag{5.10}$$

By substitution we can assemble the three equations and hereby obtain a single expression that connects forces $f$ and displacements $u$:

$$f = A^T C A\ u \tag{5.11}$$

The product of $A^T CA$ is what relates the input forces to the resulting displacements. Figure 5.3 illustrates the framework with the relations we have obtained.

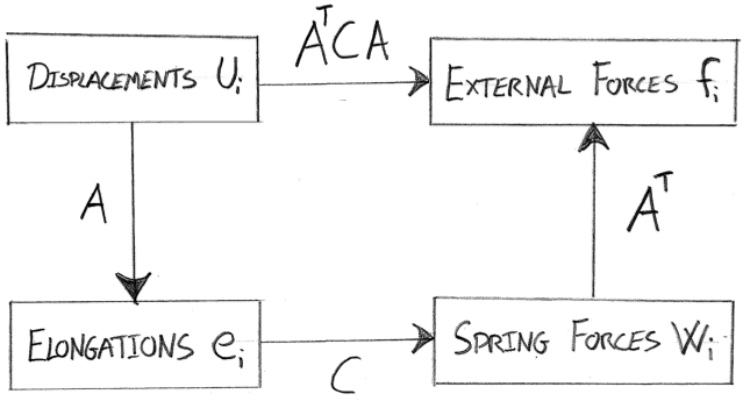



Figure 5.3: The framework with relations connecting each quantity.

Calculating the product of $A^T CA$ will lead to a special matrix. From the spring and mass example the following matrices were obtained:

$$A^T = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \qquad C = \begin{bmatrix} c_1 & & & \\ & c_2 & & \\ & & c_3 & \\ & & & c_4 \end{bmatrix}$$

Multiplying the $3 \times 4$ matrix $A^T$ by the $4 \times 4$ matrix $C$ and then multiplying the result by the $4 \times 3$ matrix $A$ equals a $3 \times 3$ matrix:

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_1 & & & \\ & c_2 & & \\ & & c_3 & \\ & & & c_4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} c_1 + c_2 & -c_2 & \\ -c_2 & c_2 + c_3 & -c_3 \\ & -c_3 & c_3 + c_4 \end{bmatrix}$$

## 5.3 The Stiffness Matrix

The matrix $A^T CA$ represents the connections and their properties in the system. In the domain of solid mechanics matrix $A^T CA$ is also known as the *stiffness* matrix. The stiffness matrix $K$ with spring constant $c_i = 1$ corresponding to the system as illustrated in figure 5.1 on page 56 is obtained by:

$$K = \begin{bmatrix} c_1 + c_2 & -c_2 & \\ -c_2 & c_2 + c_3 & -c_3 \\ & -c_3 & c_3 + c_4 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

### Positive Definite Stiffness Matrix

The resulting matrix $K$ has some interesting properties. First of all it is invertible. Remember it is the matrix that relates the displacement to the forces so physically it should be invertible because the system is well defined, so if the forces were known we could determine the displacements and the other way around. Secondly it is *positive definite* defined by [Str86, page 18]:

**Definition 7.** *$A$ is positive definite if $f = x^T Ax$ is always positive (for $x \neq 0$)*

where $A$ is a matrix and $x$ is a vector. In our case we are interested in showing that the stiffness matrix $K$, where $K = A^T CA$, is positive definite, this can be expressed as:

$$x^T A^T CA\ x > 0 \tag{5.12}$$

To prove that it is positive definite consider equation (5.12). Vector $x$ can be any vector so if it equals the displacement vector $u$ we obtain $A\ u$ which equals $e$ as defined in equation (5.8). The transpose of $A\ x$ is $x^T A^T$ so equation (5.12) can also be written as:

$$e^T C\ e > 0 \tag{5.13}$$

By taking the product of $e^T C\ e$ we obtain:

$$\begin{bmatrix} e_1 & e_2 & e_3 & e_4 \end{bmatrix} \begin{bmatrix} c_1 & & & \\ & c_2 & & \\ & & c_3 & \\ & & & c_4 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} c_1 e_1^2 + c_2 e_2^2 + c_3 e_3^2 + c_4 e_4^2 \end{bmatrix}$$

due to the square of $e$ the equation always comes out positive as long as the spring constants are positive (which they are by definition). The potential energy in a spring can be obtained by [YFFS00, page 175]:

$$E_U = \frac{1}{2} k \Delta x^2 \tag{5.14}$$

where $k$ is the spring constant ($c_i$) and $\Delta x$ is the stretch ($e_i$), multiplying equation (5.13) by $\frac{1}{2}$ we realise that the following is an expression of the potential energy in the springs

$$\frac{1}{2}c_1e_1^2 + \frac{1}{2}c_2e_2^2 + \frac{1}{2}c_3e_3^2 + \frac{1}{2}c_4e_4^2 = \frac{1}{2}e^TCe \tag{5.15}$$

Not only did we prove $A^TCA$ to be positive definite, $A\,x = 0$ only in the case where $x = 0$, we also showed that this equation is the relation between forces $f$ and displacements $u$, this leads to the important point that whenever there is movement, the potential energy is positive. $A\,x = 0$ only in the case where $x = 0$ so when there is no movement ($x = 0$), which is the case in the initial configuration, the potential energy equals 0.

### Positive Semi-definite Stiffness Matrix

The expression $A^TCA$ took us around the framework, in this case it represents an energy and was proved positive definite. The matrix $A$ and its transpose was constructed from the geometry of the problem. To see why the stiffness matrix $K$ has to be positive definite consider the stiffness matrix of the system illustrated in figure 5.4. Here we have three masses ($n = 3$) connected by two springs ($m = 2$). None of the springs are fixed so there are no boundary conditions.

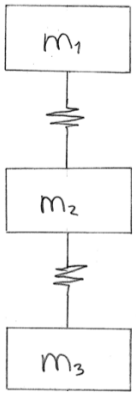



Figure 5.4: Spring mass problem without boundary conditions.

Recall how matrix $A$ was constructed, it is going to be a $n \times m$ matrix where the first row represents the stretching in the first spring. Matrix $A$ for this problem becomes:

$$A = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}$$

The stiffness matrix $K = A^TCA$ with each spring constant equal to 1 is obtained by

$$K = \begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

Here matrix $K$ is only semi-positive definite which means

$$x^TA^TCA\,x \geq 0 \tag{5.16}$$

where $x$ is any non-zero vector. So for instance

$$\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

so clearly matrix $K$ can take a non-zero vector to the *null space*. In this case it goes for all multiples of the vector $[1, 1, 1]^T$. Assume that vector $x$ is equal the displacement vector $u$, we are free to move all masses arbitrarily by the same amount, up or down, without causing any change in the potential energy. This is an example of a system that is not well defined due to the missing boundary conditions.

Describing the element connections, finding the equations and setting up the structure of the framework as explained here, is exactly what the *finite element method* is all about. The finite element method is one of the most powerful methods for conducting structural analysis. It is used not just in mechanics but all over the field of applied mathematics.

# Chapter 6

# The Finite Element Method

The *finite element method* (FEM) is a numerical analysis tool used for calculating approximate solutions and is employed to find solutions to a wide variety of different engineering and physics problems. The method, which is diverse and flexible, can be applied to complex problems where analytical solutions seldom exist or are to "expensive" to solve. As this is precisely the case with many problems in the broad field of continuum mechanics, where the complex shape of a continuum needs to be approximated, the method is used extensively within this field. The finite element method envisions the continuum body or *solution region* as an assembly of many small, interconnected subregions (elements), the same way continuum mechanics does. In the finite element method the size and number of the elements are finite in contrast to continuum theory, where the subregions are considered infinitesimal. A *finite element model* of a given problem is a piecewise approximation to the continuum body. That is: The basic premise of the finite element method is that the *solution region* can be approximated by an assemblage of discrete finite elements. Because the elements can have different shapes and can be assembled in a many ways, they can be used to approximate exceedingly complex geometrical shapes [HDSB01, page 3-5].

## 6.1 Basic Concepts and Fundamentals

When dealing with a continuum problem in any dimension the *field variable* or *nodal variable* is the unknown value that we are trying to find. Examples of field variables include: pressure, temperature, displacement, etc. and they can be either scalars, vectors, or higher-order tensors. Field variables capture infinitely many values because they are a continuous function over the continuum body. Therefore the problem has an infinite number of unknowns. The finite element discretization reduces the problem to one with a finite number of unknowns, this is done by dividing the continuum body into elements and expressing the unknown field variable in terms of approximation functions within each element. The *approximation functions*, also called *shape functions*, *basis functions*, or *interpolation functions*, are defined in terms of the field variable value at specified points in the body called *nodes* or *nodal points*. Nodes are usually located where adjacent elements are connected (*element boundaries*). In addition to the nodes located on the element boundaries, the element may also have other nodes. We distinguish between two different kinds of nodes: Nodes are called *exterior nodes* if they lie on the element boundary and *interior nodes* otherwise. To completely define the behaviour of the field variable within an element, the nodal values of the field variable and the interpolation functions for the element are used. But because we want to find this field variable the nodal values of the field variables become the unknowns. Once we have found the unknowns, the interpolation functions define the field variable

continuously throughout all elements. Clearly the precision of the approximation depends both on the size and number of elements and on the interpolation functions. As one would expect, we cannot choose the interpolation functions arbitrarily, certain *compatibility conditions* must be satisfied. Often the functions are chosen so the field variables or their derivatives are continuous across adjoining element boundaries hereby meeting the compatibility conditions, which is exactly what we will do. The most important feature of the finite element method, which distinguishes it from other numerical methods, is its ability to formulate solutions for individual elements before assembling these to represent the entire problem. This means that if we are analysing a problem like, for example, the one in chapter 5 on page 55 relating external forces to elongation of the springs, we first concentrate on modeling the problem for a single element, and then assemble these to model the entire problem. In essence, a complex problem is reduced to a series of greatly simplified ones. [HDSB01, page 5-6].

## 6.2 Constructing and Solving the Model

Solving a continuum problem via the finite element method always follows an orderly step-by-step procedure. In general this procedure can be summarized into a list of steps one must perform to construct and evaluate a problem. This list is shown below. In the following subsections, each of the steps will be carefully explained and in chapter 7 on page 83, we will go through how our model has been constructed using the same stepwise approach [HDSB01, page 7-8].

- Discretize the solution region
- Select interpolation function
- Find element properties
- Assemble the element properties to obtain the system equations
- Impose boundary conditions
- Solve the system equations
- Make additional computations

Note that some of these steps, like "Find element properties" and "Assemble the Element properties to obtain the system equation" essentially covers the same thing as the equilibrium framework discussed in chapter 5 on page 55. In this section we use a more structured approach for constructing and solving the system. Furthermore: We will adapt a more abstract view of what an element is and describe in detail how the interpolation functions work.

## 6.3 Discretize the Solution Region

Because the finite element method envisions the solution region as built up of many small elements the first step is to divide this region into elements. In chapter 5 on page 55, when we considered the equilibrium framework, our idea of what an element is was directly linked to physical phenomena of the problem. We imagined the elements to be individual segments or parts of the actual system, that is: A physical thing as for example a spring has a one to one correspondence to an element. In this case the nodes belonging to an element were part of the element itself and because the unknown field variable was displacement the nodes could move as the element deformed. We want to think of elements in less physical terms and instead use a more mathematical interpretation

of the concept. So instead of viewing an element as a physical part of the system, we think of it as a part of the solution domain. We imagine the solution domain as being divided into subregions sectioned by lines or planes. If the solution domain has curved boundaries, the curves are approximated by a series of straight line or flat plane segments. These lines or planes define the boundaries between elements and the elements are only interconnected at imaginary node points on the boundaries. In this way the solution domain is discretized into a patchwork of elements, as illustrated in figure 6.1 for a two-dimensional body.

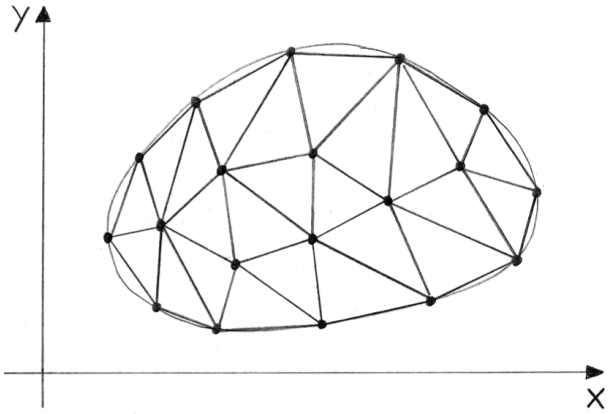
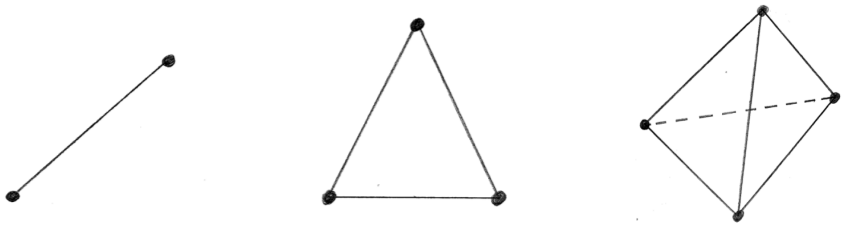



Figure 6.1: A discretized two-dimensional solution domain.

Mathematically a finite element mesh is interpreted as a spatial subdivision [HDSB01, page 86-87]. In one dimension the element covers a distance, in two the element covers an area and in three dimensions it is a volume. Depending on the number of spatial dimensions of the solution domain the elements have different characteristics. In one-dimensional space the most common element has two nodes which are connected by a line. This element type is called a bar, a truss or a beam element and is illustrated in figure 6.2a. So the simplest element in two dimensions, one that covers an area, is the three-node triangular element and in three dimensions the four-node tetrahedron as shown in figure 6.2b and figure 6.2c, respectively. The element types are denoted by how many nodal points they have. Note that the number of nodes has nothing to do with the element's geometrical shape, but is chosen because of other considerations (e.g. type of interpolation functions).



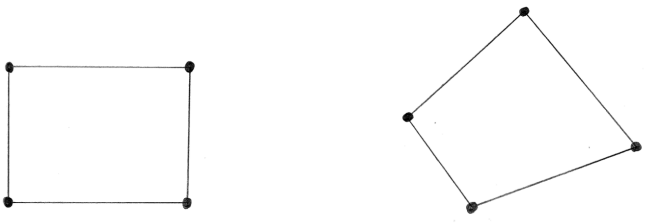
(a) Two-node line element.

(b) Three-node triangle element.

(c) Four-node tetrahedral element.

Figure 6.2: Simplest element types in one, two, and three dimensions.

When considering more than one dimension the range of different element types to choose from becomes larger. To get an idea of the range we show various kinds of two-dimensional elements with different shape and number of nodes. Figure 6.3 illustrates quadratic elements and figure 6.4 triangular elements.



(a) Four-node rectangle element.

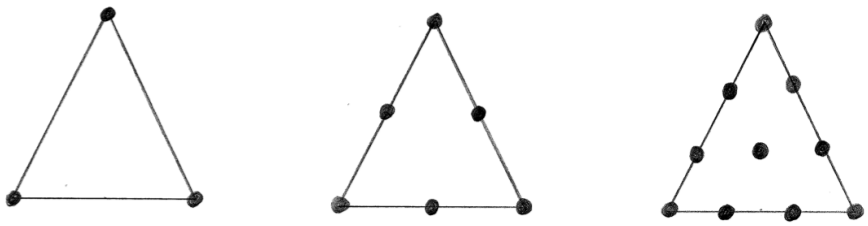
(b) Four-node quadrilateral element.

Figure 6.3: Examples of two-dimensional elements.

In addition to the shape, two other features characterizes a particular element type:

- The number of nodes assigned to the element
- The number and type of nodal variables chosen for it.

The number and type of nodal variables assigned to an element are called the elements *degree of freedom* (DOF). If for example we have a three-node element, where a two-dimensional displacement vector are the nodal variable, then the element have six degrees of freedom. The degrees of freedom for an element is the same as the number of independent variables [HDSB01, page 144].



(a) Three-node triangle element.

(b) Six-node triangle element.

(c) Ten-node triangle element.

Figure 6.4: Examples of members from the triangular element family.

Element types with the same shape but with different number of nodes are called an *element family*. An example of members from the triangular family is the three-node, six-node and ten-node triangular elements as shown in figure 6.4a, 6.4b, and 6.4c respectively. Note that the ten-node triangular element is an example of an element with one interior node.

By increasing the number of nodes in an element, the degrees of freedom increases hereby facilitating more complex interpolation functions. The nodal variables can be use to express non-linear interpolation functions. Consider using a two-node line element as the element type. The line with its two nodes have two nodal variables which can express a $1^{st}$ order polynomial equal to a linear interpolation:

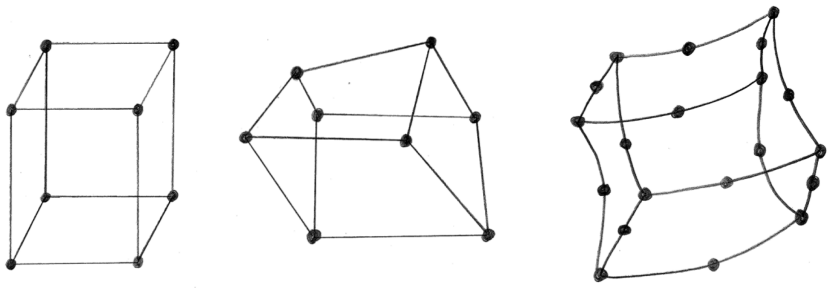
$$P(x) = ax + b$$

if we increase the number of nodes in the element by inserting an extra node on the midpoint of the line we get three nodal variables and can now use a $2^{nd}$ order polynomial to express a non-linear interpolation:

$$P(x) = ax^2 + bx + c$$

As the number of element nodes increases so does the order of the polynomial we use as the interpolation functions. Higher order interpolation functions gives more control over how the interpolation functions acts inside an element. But the interpolation functions must have convergence at the element boundaries hereby satisfying the compatibility conditions, and although we choose a higher order polynomial the interpolation functions are only continuous across element boundaries but not differentiable. The consequences of using a higher order interpolation functions with more nodes at each element is that the per element calculations get more complex and more time consuming. Instead of increasing the number of nodes for each element it is also an option to increase the number of elements, hereby approximating complex interpolation functions with more elements. Whether to choose fever elements and high order interpolation functions or more elements is up for discussion. Higher order polynomials as interpolation functions are further discussed in [HDSB01, page 144-146]. We chose the simplest of the two, that is we stick to $1^{st}$ order polynomials for each element in a high resolution mesh.

In three-dimensional space the range of element types becomes even larger and mixed with the fact that element boundaries can be curved, called ISO-parametric elements, this gives endless possibilities for combinations. In figure 6.5 some examples of three-dimensional elements are shown.



(a) Prism. (b) General hexahedron. (c) ISO-parametric hexahedron.

Figure 6.5: Examples of three-dimensional elements.

When assembling the elements to cover the solution region, different shape and mixed types of elements may be used. As this further increases the complexity of the governing equations we will stick to one element type.

### Element and Node Numbering

The elements share nodes therefore the logistics of keeping track of which nodes belong to a specific element and in which order the nodes should be used becomes a challenge. To keep track of this the common thing to do is to make a table containing the information. This table is called an *element table* an it relates *element numbers* and *local node numbers* to *global node numbers*. As this is best illustrated though an example, figure 6.6 shows a domain discretized into triangular elements, and table 6.1 the associated element table [BR98, page 15-16].

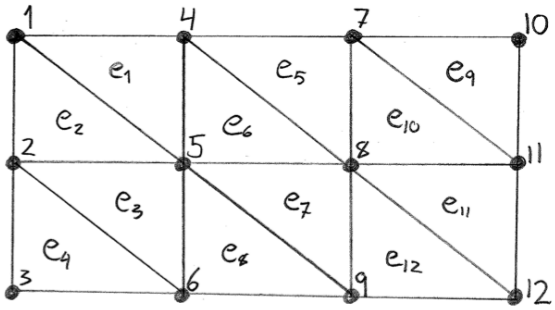



Figure 6.6: Global and local node numbering of domain discretized into triangular elements.

| n | 1 | 2 | 3 |
|---|---|---|---|
| $e_1$ | 4 | 1 | 5 |
| $e_2$ | 2 | 5 | 1 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| $e_{12}$ | 9 | 12 | 8 |

Table 6.1: Example of an element table.

The table indexes the elements by their number $n$ of element $e_n$ in the left column, and local node numbers 1, 2, and 3 in the top row. The table is then filled with the global node numbers for each element. An example element ($e_2$) is shown in figure 6.6, where local node number 2 corresponds to the global node number 5.

## 6.4 Selecting the Interpolation Functions

The next step is to choose the interpolation functions. The interpolation functions represent the field variable and its variation over an element. Polynomials are often selected as interpolation functions because they are easy to integrate and differentiate. The interpolation functions, make it possible to describe the field variable continuously over the volume of the element. That is, they describe the field variable at any given point within or at the boundary of the element. The collection of interpolation functions for the entire solution domain provides a piecewise approximation to the field variable.

### Example of a Piecewise Approximation

To illustrate this piecewise approximation we consider an example of a two-dimensional field variable $\phi(x, y)$. We will illustrate how $\phi$'s nodal values uniquely and continuously define $\phi(x, y)$ for the entire solution domain in the $x$-$y$ plane. At the same time, we will introduce the notation used for the interpolation functions. Suppose that we have a problem with the solution domain as shown in figure 6.1 on page 67. This domain has been sectioned into three-node triangular elements with exterior nodes at the vertices of the triangles connecting the elements. This type of domain discretization allow us to select $\phi$ to vary linearly over an element. Where the linear variation can be illustrated as in figure 6.7.

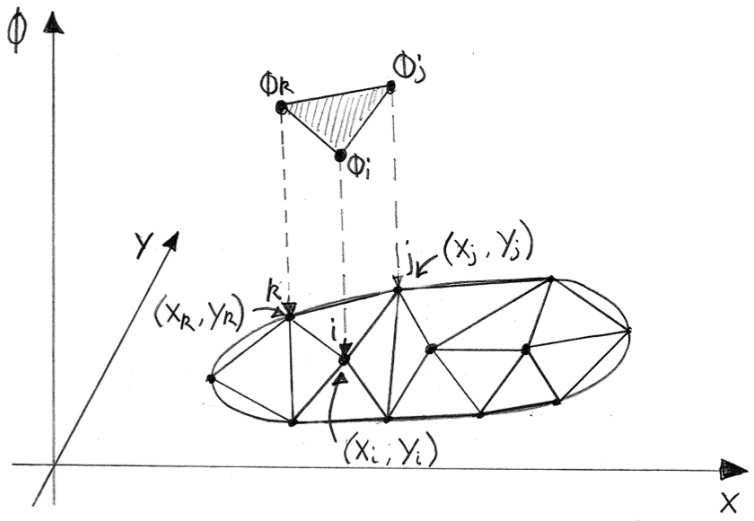



Figure 6.7: Subdivided domain and piecewise linear solution surface.

The first step in finding the interpolation functions is to mathematically describe the plane passing through the three nodal values of $\phi$ associated with element $e$. This is done in equation (6.1).

$$\phi^{(e)}(x, y) = \beta_1^{(e)} + \beta_2^{(e)} x + \beta_3^{(e)} y \tag{6.1}$$

Equation (6.1) can then be rewritten to express the constants $\beta_1^{(e)}$, $\beta_2^{(e)}$, and $\beta_3^{(e)}$ in terms of the global Cartesian coordinates at the element nodes and the nodal values of $\phi$. The rewriting is done by evaluating equation (6.1) at each node, which results in equation (6.2).

$$\begin{aligned} \phi_i^{(e)} &= \beta_1^{(e)} + \beta_2^{(e)} x_i + \beta_3^{(e)} y_i \\ \phi_j^{(e)} &= \beta_1^{(e)} + \beta_2^{(e)} x_j + \beta_3^{(e)} y_j \\ \phi_k^{(e)} &= \beta_1^{(e)} + \beta_2^{(e)} x_k + \beta_3^{(e)} y_k \end{aligned} \tag{6.2}$$

Separating the $\beta_i$'s gives:

$$\begin{aligned}
\beta_1^{(e)} &= \frac{\phi_i(x_j y_k - x_k y_j) + \phi_j(x_k y_i - x_i y_k) + \phi_k(x_i y_j - x_j y_i)}{2\Delta} \\
\beta_2^{(e)} &= \frac{\phi_i(y_j - y_k) + \phi_j(y_k - y_i) + \phi_k(y_i - y_j)}{2\Delta} \\
\beta_3^{(e)} &= \frac{\phi_i(x_k - x_j) + \phi_j(x_i - x_k) + \phi_k(x_j - x_i)}{2\Delta}
\end{aligned} \tag{6.3}$$

where

$$2\Delta = \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{vmatrix} = 2\begin{bmatrix}\text{area of triangle with} \\ \text{vertices } i, j, k\end{bmatrix} \tag{6.4}$$

and by substituting equation (6.3) into equation (6.1) and rearranging its terms, we get

$$\phi^{(e)}(x, y) = \frac{a_i + b_i x + c_i y}{2\Delta}\phi_i + \frac{a_j + b_j x + c_j y}{2\Delta}\phi_j + \frac{a_k + b_k x + c_k y}{2\Delta}\phi_k \tag{6.5}$$

where

$$a_i = x_j y_k - x_k y_i \qquad b_i = y_j - y_k \qquad c_i = x_k - x_j \tag{6.6}$$

the other coefficients are obtained through cyclic permutation of the subscripts $i$, $j$, and $k$. If i=1, j=2, and k=3, the coefficients are given explicitly by equation (6.7)

$$\begin{aligned}
a_1 &= x_2 y_3 - x_3 y_2 \qquad & b_1 &= y_2 - y_3 \qquad & c_1 &= x_3 - x_2 \\
a_2 &= x_3 y_1 - x_1 y_3 \qquad & b_2 &= y_3 - y_1 \qquad & c_2 &= x_1 - x_3 \\
a_3 &= x_1 y_2 - x_2 y_1 \qquad & b_3 &= y_1 - y_2 \qquad & c_3 &= x_2 - x_1
\end{aligned} \tag{6.7}$$

For each element $e$ we now define

$$N_n^{(e)} = \frac{a_n + b_n x + c_n y}{2\Delta} \quad n = i, j, k \tag{6.8}$$

and let

$$\phi^{(e)} = \begin{Bmatrix} \phi_i \\ \phi_j \\ \phi_k \end{Bmatrix} \qquad N^{(e)} = \begin{bmatrix} N_i^{(e)} & N_j^{(e)} & N_k^{(e)} \end{bmatrix} \tag{6.9}$$

where $\phi^{(e)}$ is a column vector, $N^{(e)}$ is a row vector. The functions $N^{(e)}$ are exactly the functions known as the interpolation functions. In matrix notation we write equation (6.5) as

$$\phi^{(e)}(x, y) = N^{(e)}\phi^{(e)} = N_i\phi_i + N_j\phi_j + N_k\phi_k \tag{6.10}$$

If the domain has been discretized into $M$ elements, the equations for the field variable over the entire domain is given by

$$\phi(x, y) = \sum_{e=1}^{M} \phi^{(e)}(x, y) = \sum_{e=1}^{M} N^{(e)}\phi^{(e)} \tag{6.11}$$

We see from equation (6.11) that if we know the nodal values of $\phi$, then we can represent the complete solution surface $\phi(x, y)$ as a series of interconnected triangles. The resulting many-faced surface has no gaps between elements and no discontinuities because the values of $\phi$ at

any boundary uniquely determines the linear variation of $\phi$ along the two nodes defining that boundary. Although we used a particular interpolation function and a particular element type to obtain equation (6.10) and (6.11), these equations are general. When using other element types and more complex interpolation functions the form of the equations remains the same, it is only the number of terms in the rows and columns of the matrices that differs. This means that we can represent the unknown field variable in each element as in equation (6.12) if a solution domain is subdivided into elements [HDSB01, page 87-90].

$$\phi = N^{(e)}\phi^{(e)} \tag{6.12}$$

### Global and Local Coordinates

Regardless of how we find the element properties, it is more convenient and easier to derive the matrix equations for the element properties in a coordinate system associated with the element (a local coordinate system). This is exactly the approach we followed for triangular elements above. Because the local coordinate system, in this case, is a function of the geometry and orientation of the element, the local coordinate system for each element may differ. When local coordinate systems are used, it becomes necessary to transform the element matrices to the common global coordinate system before the individual element matrices are assembled into the system equations. This must be done to preserve all element characteristics in relation to the other elements. Converting between two different coordinate systems is called *coordinate transformation* and is required when the nodal unknowns are components of a vector [HDSB01, page 37]. Note that displacements are typical field variables where the nodal unknowns are vectors, so we must be careful to insure that coordinate transformation occur. When we assemble the element properties, in section 6.5 on page 78, we will see that the coordinate transformations are part of each element stiffness matrix. Because of this the transformations is done for each element before the whole system is considered.

### Natural Coordinates

The definition of a local coordinate system relies on the element geometry and all coordinates (described in that local coordinate system) range between zero and unity, then the coordinate system is known as a *natural coordinate system*. A natural coordinate system has the property that when describing a point located precisely on one of the element's nodes, then one of the local coordinates of the point have unite value (the coordinate corresponding to this particular node), and the value zero in all its other local coordinates. Between nodes the values of the coordinates vary, but they always sum to unity. Natural coordinate systems can be constructed for a variety of elements including: two-node line element, three-node triangular elements, four-node quadrilateral elements, four-node tetrahedral elements and so on. Natural coordinates for a simplex (a triangle, a tetrahedron, etc.) are also called *barycentric coordinates*. A particularly advantageous benefit of natural coordinates is that: When we use these coordinates to derive the interpolation functions, then integration can be done in a special closed form making it easier to evaluate the integrals in element equations. Natural coordinates basically describes the location of any point inside an element in terms of that element's exterior nodes. The local natural coordinates within an element are denoted by: $L_i$ where $i \in \{1, 2, ..., n\}$ and $n$ is the number of external nodes of the element. There is always one coordinate associated with a node $i$ that has unit value in this particular node. By going through some examples it will become clear that local natural coordinates are functions of the global Cartesian coordinate system [HDSB01, page 151-157].

### Natural Coordinates in One Dimension

To define a natural coordinate system for a two-node line element, we select $L_1$ and $L_2$ as the natural coordinates. Then a point $x$ can be expressed as a linear combination of the nodal coordinates $x_1$ and $x_2$, as follows:

$$x = L_1 x_1 + L_2 x_2 \tag{6.13}$$

We may interpret the coordinates $L_1$ and $L_2$ as weighting functions relating the coordinates at the end nodes to the coordinate of any point inside the element. Furthermore these weighting functions cannot be independent, since they must obey

$$L_1 + L_2 = 1 \tag{6.14}$$

If we solve equation (6.13) and (6.14) simultaneously the result is the functions $L_1$ and $L_2$, as follows:

$$L_1(x) = \frac{x_2 - x}{x_2 - x_1} \qquad L_2(x) = \frac{x - x_1}{x_2 - x_1} \tag{6.15}$$

The functions $L_1$ and $L_2$ should be interpreted as simply being the ratio of lengths and are therefore also called *length coordinates*. The variation of $L_1$ and $L_2$ within a single element is illustrated in figure 6.8.

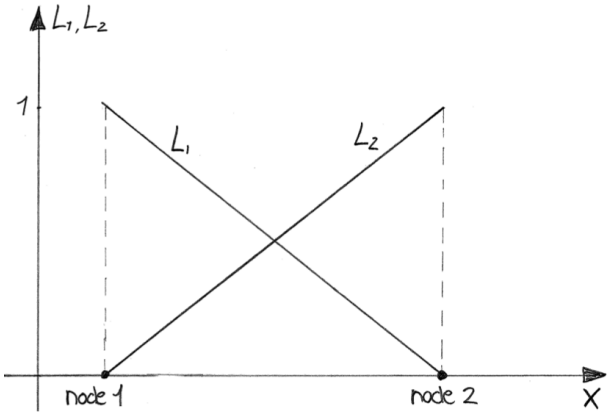



Figure 6.8: Variation of length coordinated within a line element.

Hence the linear combination of a field variable $\phi$ can be written as

$$\phi(x) = \phi_1 L_1 + \phi_2 L_2 \tag{6.16}$$

### Natural Coordinates in Two Dimensions

Developing natural coordinates for the three-node triangular element uses the same approach as in the one-dimensional case. Again, the goal of the procedure is to find the coordinates $L_1$, $L_2$, and $L_3$ that describe the location of any point $x$ within the element or on its boundary. The point $(x, y)$ is illustrated inside a triangular element in figure 6.9 on the next page.

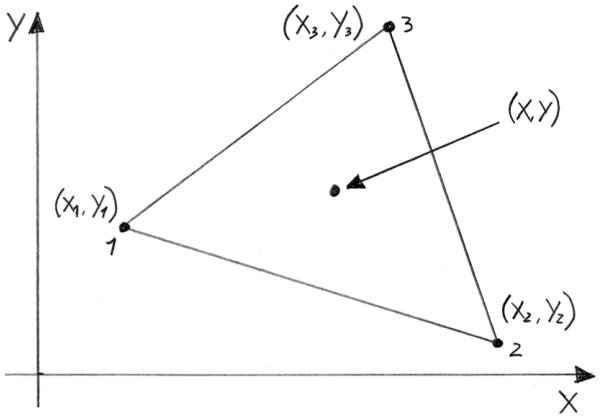



Figure 6.9: Triangular element with point ($x$,$y$).

The relationship between the global Cartesian coordinates of the point $(x, y)$ within the element, and the new local natural coordinates, that we are constructing, should have a linear dependency. The linear property holds for the following equations

$$\begin{aligned} x =& L_1 x_1 + L_2 x_2 + L_3 x_3 \\ y =& L_1 y_1 + L_2 y_2 + L_3 y_3 \end{aligned} \tag{6.17}$$

In addition to the equation above, requiring linear dependency, we again require that the weighting functions sum to unity:

$$L_1 + L_2 + L_3 = 1 \tag{6.18}$$

It is clear, from equation (6.18), that only two of the natural coordinates can be independent. Intuitively this must be the case, because if we describe any point as a linear combination of two points in the global coordinate system (two of an element's nodal points in global space), then there are only two independent coordinates if the described point lies inside the element. When solving equations (6.17) and (6.18) simultaneously for $L_1$, $L_2$, and $L_3$ as done in equation (6.19), the result gives the natural coordinates in terms of the global coordinates.

$$\begin{aligned} L_1(x, y) =& \frac{1}{2\Delta}(a_1 + b_1 x + c_1 y) \\ L_2(x, y) =& \frac{1}{2\Delta}(a_2 + b_2 x + c_2 y) \\ L_3(x, y) =& \frac{1}{2\Delta}(a_3 + b_3 x + c_3 y) \end{aligned} \tag{6.19}$$

Where $2\Delta$ is given by equation (6.4) on page 72. Recalling the linear piecewise approximation example from section 6.4 on page 71, we conclude that the natural coordinates $L_1$, $L_2$, and $L_3$ are identical to the linear interpolation functions. This means that: $N_i = L_i$ for the three-node triangular element when using a linear interpolation function. The natural coordinates for a triangle have an analogous interpretation to length coordinates. For both the line element and the triangular element $L_i$ is a ratio, in the case of the line it describes the ratios of lengths, but for the triangle it is a ratio of areas. Figure 6.10 on the next page illustrates how the natural coordinates for the triangular element, often called *area coordinates*, are related to areas. When a point is located on the boundary of the element, then one of the area segments vanish and hence

the appropriate area coordinate along the boundary is zero, hereby satisfying the compatibility conditions.

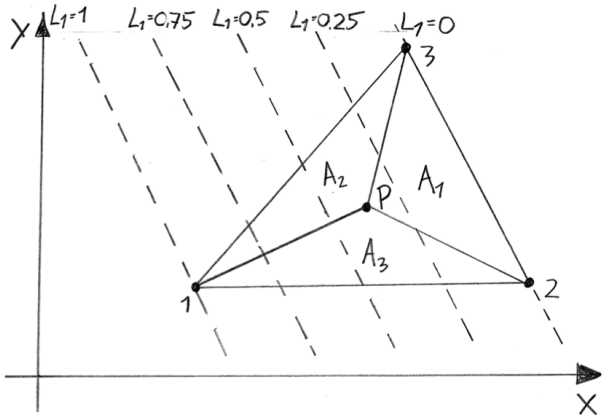



Figure 6.10: Area coordinates for a triangular element.

As with the length coordinates, the variation of area coordinates inside an element can be illustrated, this is shown in figure 6.11 for one element node.

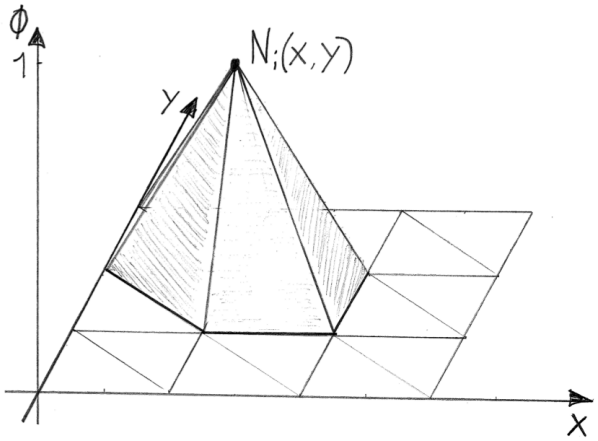



Figure 6.11: Variation of area coordinates for one node over the entire domain.

Note that in figure 6.8 on page 74 the interpolation functions are viewed at a particular element and that the graphs illustrates the interpolation functions for one element. In figure 6.11 however the interpolation functions are viewed at one node, which means that all interpolation functions for elements sharing this node are shown in the figure.

### Natural Coordinates in Three Dimensions

We will now extend natural coordinates from two to three dimensions for the four-node tetrahedron element. Natural coordinates for the four-node tetrahedron can be defined in a manner analogous to the procedure used for the three-node triangle.

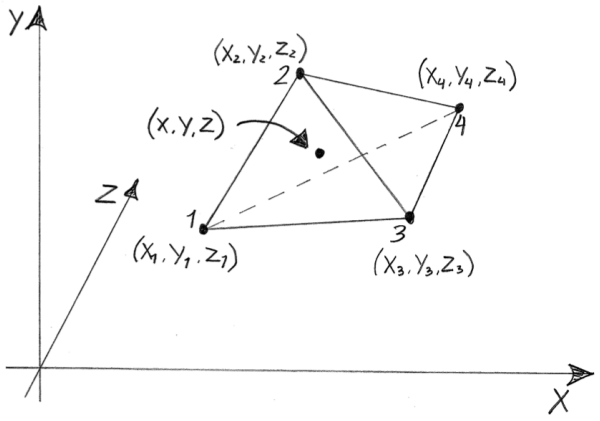



Figure 6.12: Tetrahedron element with global coordinates ($x$,$y$,$z$).

A typical four-node tetrahedron element can be see in figure 6.12, the figure also defines how the nodes are numbered. An element's global Cartesian coordinates and local natural coordinates are related by:

$$\begin{aligned} x &= L_1x_1 + L_2x_2 + L_3x_3 + L_4x_4 \\ y &= L_1y_1 + L_2y_2 + L_3y_3 + L_4y_4 \\ z &= L_1z_1 + L_2z_2 + L_3z_3 + L_4z_4 \\ 1 &= L_1 + L_2 + L_3 + L_4 \end{aligned} \tag{6.20}$$

Solving equation (6.20) gives:

$$L_i = \frac{1}{6V}(a_i + b_ix + c_iy + d_iz), \qquad i = 1, 2, 3, 4 \tag{6.21}$$

and

$$6V = \begin{vmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{vmatrix} = 6(\text{volume of the tetrahedron}) \tag{6.22}$$

where examples of the constants are:

$$a_1 = \begin{vmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{vmatrix} \quad c_1 = -\begin{vmatrix} x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \\ x_4 & 1 & z_4 \end{vmatrix} \quad b_1 = -\begin{vmatrix} 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix} \quad d_1 = -\begin{vmatrix} x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{vmatrix} \tag{6.23}$$

The other constants in equation (6.21) can be obtained by cyclic permutation of subscripts 1, 2, 3, and 4. The natural coordinates for the four-node tetrahedron are also called *volumetric coordinates* and can be physically interpreted as the ratio of volumes in a tetrahedron element. This physical interpretation is illustrated in figure 6.13 on the following page. Here the point $P$ inside the tetrahedron divides the tetrahedron volume $V$ into four sub-volumes $V_1$, $V_2$, $V_3$, and $V_4$. Each of these sub-volumes occupies a part of $V$ which is equal to the coordinate ratio. Also notice that together the sub-volumes completely covers $V$, which means that their ratios sum to unity.

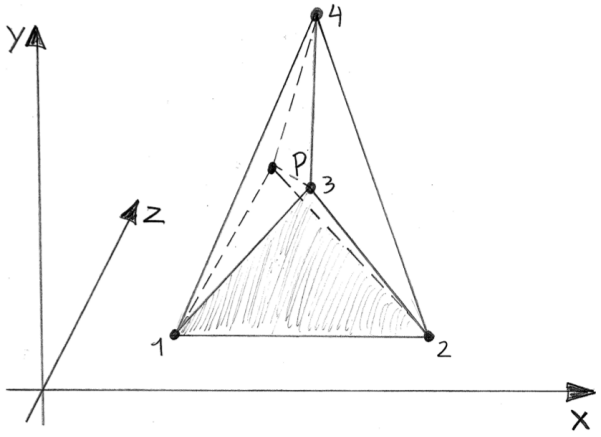



Figure 6.13: Illustration of volume coordinates.

We can represent the field variable $\phi$ as a function of $L_1$, $L_2$, $L_3$, and $L_4$ (instead of $x$, $y$, and $z$) as [HDSB01, page 158-159]:

$$\phi(x, y, z) = \phi_1 L_1 + \phi_2 L_2 + \phi_3 L_3 + \phi_4 L_4 \tag{6.24}$$

## 6.5 Finding the Element Properties

Once the elements and their interpolation functions have been selected we are ready to express the properties of the individual elements and hereby determine their matrix equations. Finding the element properties is tightly coupled with the kind of problem we are dealing with, but in general terms the nodal values are represented on matrix form and related by some physical laws as done in the example in chapter 5 on page 55 when explaining the equilibrium framework. The goal is to find the element stiffness matrix, which is the stiffness matrix for a single element.

## 6.6 Assembling the System Equations

In order to find the overall system properties, modeled by the patchwork of elements, we must combine all the element properties found in the previous step. In other words, the matrix equations expressing the behavior of the elements must be combined to form the matrix equations expressing the behavior of the entire system (the system equations). The assembly procedure relies on the fact that the exterior nodes of the elements are interconnected, which means that the value of the field variable in such a node is the same for all elements sharing the node. In this respect the finite element method possesses the unique feature: The system equations are constructed by assembling the individual element equations. Assuming that we by some means have found the equations necessary to describe the characteristics of the elements, then the next step is to combine these equations. Combining the element equations requires the same procedure regardless of the type and complexity of problem being considered. Even if a mixture of several different kinds of elements is used to model the system, the procedure remains the same. This construction procedure is, as already noted, based on the assumption that the unknown field variable values at shared nodes are the same for all elements connecting at that node. With this in mind it is quite easy to combine the element equations [HDSB01, page 40].

### The Construction Procedure

First the dimensions of the resulting system stiffness matrix is found. The dimensions are the same as the number of independent variables or degrees of freedom in the system. When the dimensions are known, then all element stiffness matrices are expanded with zero, to have the same dimensions. The last step is to add the expanded element stiffness matrices together. To better understand how this is done, we show a small example.

#### Example on Two Four-node Tetrahedron Elements

Consider the following example with two elements as illustrated in figure 6.14.

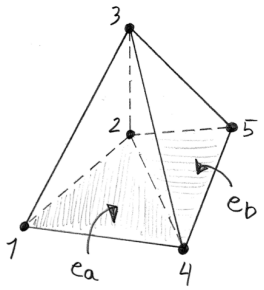



Figure 6.14: Example of two connected four-node tetrahedra.

With the following element table:

| n | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $e_a$ | 1 | 2 | 3 | 4 |
| $e_b$ | 2 | 3 | 4 | 5 |

Table 6.2: Element table for figure 6.14.

The element stiffness matrices are the:

$$K^a = \begin{bmatrix} K^a_{11} & K^a_{12} & K^a_{13} & K^a_{14} \\ K^a_{21} & K^a_{22} & K^a_{23} & K^a_{24} \\ K^a_{31} & K^a_{32} & K^a_{33} & K^a_{34} \\ K^a_{41} & K^a_{42} & K^a_{43} & K^a_{44} \end{bmatrix} \qquad K^b = \begin{bmatrix} K^b_{11} & K^b_{12} & K^b_{13} & K^b_{14} \\ K^b_{21} & K^b_{22} & K^b_{23} & K^b_{24} \\ K^b_{31} & K^b_{32} & K^b_{33} & K^b_{34} \\ K^b_{41} & K^b_{42} & K^b_{43} & K^b_{44} \end{bmatrix} \tag{6.25}$$

Element $e_a$ relate forces to displacements as follows:

$$F^a = K^a U^a \qquad \Leftrightarrow \qquad \begin{bmatrix} f^1 \\ f^2 \\ f^3 \\ f^4 \end{bmatrix} = \begin{bmatrix} K^a_{11} & K^a_{12} & K^a_{13} & K^a_{14} \\ K^a_{21} & K^a_{22} & K^a_{23} & K^a_{24} \\ K^a_{31} & K^a_{32} & K^a_{33} & K^a_{34} \\ K^a_{41} & K^a_{42} & K^a_{43} & K^a_{44} \end{bmatrix} \begin{bmatrix} u^1 \\ u^2 \\ u^3 \\ u^4 \end{bmatrix} \tag{6.26}$$

And element $e_b$ like this:

$$F^b = K^b U^b \qquad \Leftrightarrow \qquad \begin{bmatrix} f^2 \\ f^3 \\ f^4 \\ f^5 \end{bmatrix} = \begin{bmatrix} K^b_{11} & K^b_{12} & K^b_{13} & K^b_{14} \\ K^b_{21} & K^b_{22} & K^b_{23} & K^b_{24} \\ K^b_{31} & K^b_{32} & K^b_{33} & K^b_{34} \\ K^b_{41} & K^b_{42} & K^b_{43} & K^b_{44} \end{bmatrix} \begin{bmatrix} u^2 \\ u^3 \\ u^4 \\ u^5 \end{bmatrix} \tag{6.27}$$

The reason for combining the system stiffness matrix this way is directly apparent from the two element equations above. In these equations some of the field variables are the same. The way we combine them insures that these field variables will not be repeated in the final equations.

The expanded element stiffness matrices are then:

$$K^A = \begin{bmatrix} K^a_{11} & K^a_{12} & K^a_{13} & K^a_{14} & 0 \\ K^a_{21} & K^a_{22} & K^a_{23} & K^a_{24} & 0 \\ K^a_{31} & K^a_{32} & K^a_{33} & K^a_{34} & 0 \\ K^a_{41} & K^a_{42} & K^a_{43} & K^a_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad K^B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & K^b_{11} & K^b_{12} & K^b_{13} & K^b_{14} \\ 0 & K^b_{21} & K^b_{22} & K^b_{23} & K^b_{24} \\ 0 & K^b_{31} & K^b_{32} & K^b_{33} & K^b_{34} \\ 0 & K^b_{41} & K^b_{42} & K^b_{43} & K^b_{44} \end{bmatrix} \tag{6.28}$$

And the full system stiffness matrix:

$$K = K^A + K^B = \begin{bmatrix} K^a_{11} & K^a_{12} & K^a_{13} & K^a_{14} & 0 \\ K^a_{21} & K^a_{22} + K^b_{11} & K^a_{23} + K^b_{12} & K^a_{24} + K^b_{13} & K^b_{14} \\ K^a_{31} & K^a_{32} + K^b_{21} & K^a_{33} + K^b_{22} & K^a_{34} + K^b_{23} & K^b_{24} \\ K^a_{41} & K^a_{42} + K^b_{31} & K^a_{43} + K^b_{32} & K^a_{44} + K^b_{33} & K^b_{34} \\ 0 & K^b_{41} & K^b_{42} & K^b_{43} & K^b_{44} \end{bmatrix} \tag{6.29}$$

The final system equations now looks like this:

$$F = KU \qquad \Leftrightarrow \qquad \begin{bmatrix} f^1 \\ f^2 \\ f^3 \\ f^4 \\ f^5 \end{bmatrix} = K \begin{bmatrix} u^1 \\ u^2 \\ u^3 \\ u^4 \\ u^5 \end{bmatrix} \tag{6.30}$$

Note that the final system equations only include each force and displacement once.

## 6.7 Imposing the Boundary Conditions

Before we can solve the system equations they must be modified to include boundary conditions. The boundary conditions are employed to make sure that the system in fact has a solution as discussed in section 5.3 on page 61. If we have known nodal values, these are inserted, otherwise we have to constrain parts of the system in some way.

## 6.8 Solving the System Equations

The result of the assembly procedure is a set of equations that has to be solved simultaneously to obtain the unknown nodal values of the problem. In a static problem a set of linear or non-linear algebraic equations must be solved. If instead it is a dynamic problem then the nodal unknowns are functions of time, which generates a set of linear or non-linear ordinary differential equations to be solved.

## 6.9 Making Additional Computations

The solution found when solving the system equations is sometimes used to calculate or find other important parameters and values. For example this is where we detect if and when a crack should be propagated in our model as elaborated in section 7.7 on page 95.

# Chapter 7

# Applying the Finite Element Method

In this section, we follow the seven step procedure of creating a discrete finite element model introduced in section 6.2 on page 66. To summarize we have chosen the following which will be further elaborated in upcoming sections. At step four, assembling the system equations, no choise can be made so our choices are listed three steps at the time. We discretize the solution region into four-node tetrahedral elements, select linear volumetric coordinates as our interpolation function, and use elasticity theory and energy preservation to describe the element properties. At this point we assemble the element properties to obtain the system equation precisely as described in section 6.6 on page 78. Then we impose boundary conditions by fixing nodal positions, and construct an equation solver using the Total Lagrangian Explicit Dynamics technique. Lastly we add fracture mechanics as additional computations.

## 7.1 Discretize the Continuum

The continuum is a volumetric body. Therefore we need an element type which covers a volume. We have chosen the four-node tetrahedron, as this is the simplest volumetric element and because it is the most commonly used three-dimensional element type.

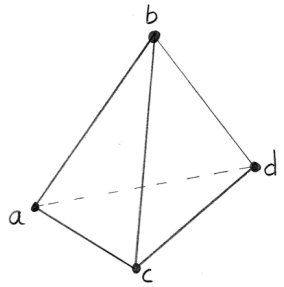



Figure 7.1: Four-node tetrahedron nodes with node names.

The four-node tetrahedron element has 4 nodes, 4 faces and 6 edges. For a tetrahedron with vertices: $a, b, c, d$, as illustrated in figure 7.1, the nodes are named according to the following rule: If a tetrahedron is defined in a right-hand Cartesian coordinate system, the nodes are named so that nodes $a$, $b$, $c$ are ordered counterclockwise when viewed from node $d$. By naming the nodes

this way we ensure that the following equation for calculating the volume always gives a positive value. [HDSB01, page 159]. The volume is then calculated as:

$$V = \frac{|(a-d)\cdot((b-d)\times(c-d))|}{6}. \tag{7.1}$$

Tetrahedron structures gives good approximations to the continuum body. As it is hard to imagine how tetrahedra fill a body figure 7.2 gives an example of how a box can be fully covered by five tetrahedra, which is the minimum number nessecery to fill a box.

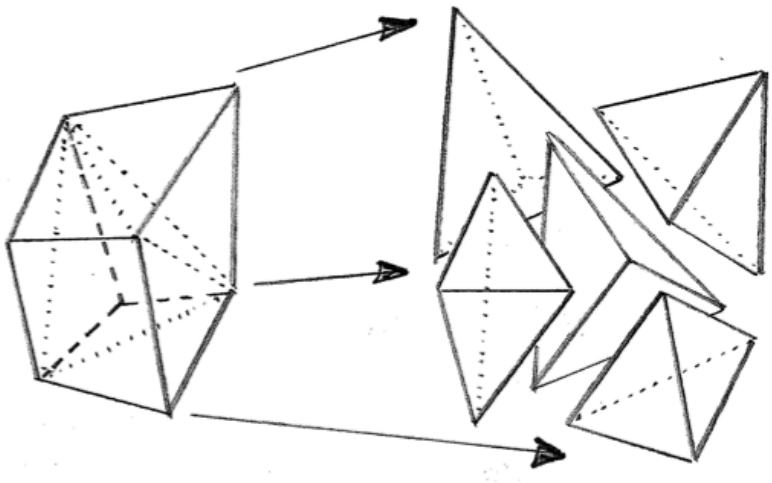


Figure 7.2: A general hexahedron decomposed into five tetrahedra.

As in the case of approximating a curve with triangular elements in section 6.3 on page 66, when a larger amount of geometrically smaller elements are used, we get a higher accuracy on the curve approximation.

## 7.2 Selecting the Interpolation Functions

As interpolation functions we use the volumetric coordinates as described in section 6.4 on page 76. We have chosen natural coordinates as interpolation functions by setting the interpolation functions equal to the natural coordinates.

$$N_i(x,y,z) = L_i(x,y,z) \tag{7.2}$$

The unknown field variable is in our case the displacements. We have a three component displacement vector at each of the four nodes, as illustrated in figure 7.3 on the next page, giving a total of 12 displacement variables.

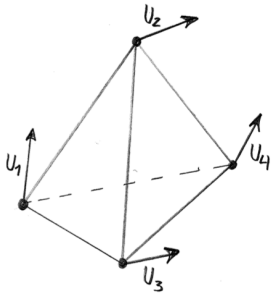



Figure 7.3: Displacement vectors in an four-node tetrahedron element.

The interpolation functions for our model define the element-wise displacement functions by interpolating the four nodal displacement vectors, with their 12 variables over the element. The 12 values are represented by $U$, where the right superscript denoted the node number and the right subscript the axial direction.

$$U^T = \begin{bmatrix} u_x^1 & u_y^1 & u_z^1 & u_x^2 & u_y^2 & u_z^2 & u_x^3 & u_y^3 & u_z^3 & u_x^4 & u_y^4 & u_z^4 \end{bmatrix} \tag{7.3}$$

and hence the continuous function $U(x, y, y) = NU$ where $N$, called the *interpolation matrix*, is defined by

$$N = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 \end{bmatrix} \tag{7.4}$$

which gives the displacement functions [HDSB01, page 262]:

$$U(x,y,z) = \begin{Bmatrix} U_x(x,y,z) \\ U_y(x,y,z) \\ U_z(x,y,z) \end{Bmatrix} = NU = \begin{Bmatrix} L_1(x,y,z)U_x^1 + L_2(x,y,z)U_x^2 + L_3(x,y,z)U_x^3 + L_4(x,y,z)U_x^4 \\ L_1(x,y,z)U_y^1 + L_2(x,y,z)U_y^2 + L_3(x,y,z)U_y^3 + L_4(x,y,z)U_y^4 \\ L_1(x,y,z)U_z^1 + L_2(x,y,z)U_z^2 + L_3(x,y,z)U_z^3 + L_4(x,y,z)U_z^4 \end{Bmatrix} \tag{7.5}$$

where $L_i$ is given by equation (6.21).

## 7.3 Finding the Element Properties

To determine the element properties we are going to use the elasticity theory discussed in section 3.4 on page 32. The element properties are related as follows: displacement to strain, stress to strain via the theory of elasticity, and finally we convert stress and strain into potential energy. The goal in this section, is to construct a function that describes the potential energy of an element in terms of its nodal displacement vector $U$. The equation for potential energy is defined to be the potential energy of the work done by the external forces ($E_V$) and the interal strain energy ($E_S$).

$$\prod(x, y, z) = E_S + E_V = E_S - W \tag{7.6}$$

Where $E_S$ is the strain energy and $E_V$ is the potential energy of the external forces. Where $E_V = -W$ as defined in equation (3.7) on page 22. From equation (3.13) and (3.17) we have that:

$$E_S = \int_V \varepsilon \cdot \sigma \; dV + \int_V \gamma \cdot \tau \; dV \qquad E_V = -W = -\int_V u \cdot f \; dV$$

Instead of using the dot product notation we will use transposed matrices from here on. Furthermore, the four $3 \times 1$ vectors, stress ($\sigma$ and $\tau$) and strain ($\varepsilon$ and $\gamma$), have been combined into two $6 \times 1$ vectors, and use the per element displacement and force vectors $U$ and $F$, which look like this:

$$\prod(x, y, z) = \int_V \begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix}^T \begin{Bmatrix} \sigma \\ \tau \end{Bmatrix} dV - \int_V U^T F \; dV \tag{7.7}$$

As $E_V$ is already a function of $u$, we concentrate on $E_S$. The material matrix $C$ from the elasticity theory relates stress to strain. Appendix A defines this material matrix and the matrix relation between stress and strain, repeated below for convenience:

$$\begin{Bmatrix} \sigma \\ \tau \end{Bmatrix} = C \begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix}$$

$$E_S(x, y, z) = \int_V \begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix}^T \begin{Bmatrix} \sigma \\ \tau \end{Bmatrix} dV = \int_V \begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix}^T C \begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix} dV \tag{7.8}$$

The final step is a little more complicated, here we will relate strain to displacement. Recalling from section 3.2 on page 25 that normal strain is defined by equation (3.9) as:

$$\varepsilon_x = \frac{\partial u_x}{\partial x} \qquad \varepsilon_y = \frac{\partial u_y}{\partial y} \qquad \varepsilon_z = \frac{\partial u_z}{\partial z}$$

and shearing strain by equation (3.11) as:

$$\gamma_{xy} = \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \qquad \gamma_{xz} = \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x} \qquad \gamma_{yz} = \frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}$$

on matrix form this looks like the following:

$$\begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{Bmatrix} = LU(x, y, z) = LNU = BU \tag{7.9}$$

where the *differential operator* $L$ is defined as [HDSB01, page 233]:

$$L = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \end{bmatrix} \tag{7.10}$$

and the *strain interpolation matrix* $B$ [HDSB01, page 235]:

$$B = LN = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} & 0 \\ \frac{\partial N_1}{\partial z} & 0 & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial z} & 0 & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial z} & 0 & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial z} & 0 & \frac{\partial N_4}{\partial x} \\ 0 & \frac{\partial N_1}{\partial z} & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial z} & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial z} & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial z} & \frac{\partial N_4}{\partial y} \end{bmatrix}$$

$$= \frac{1}{6V} \begin{bmatrix} b_1 & 0 & 0 & b_2 & 0 & 0 & b_3 & 0 & 0 & b_4 & 0 & 0 \\ 0 & c_1 & 0 & 0 & c_2 & 0 & 0 & c_3 & 0 & 0 & c_4 & 0 \\ 0 & 0 & d_1 & 0 & 0 & d_2 & 0 & 0 & d_3 & 0 & 0 & d_4 \\ c_1 & b_1 & 0 & c_2 & b_2 & 0 & c_3 & b_3 & 0 & c_4 & b_4 & 0 \\ d_1 & 0 & b_1 & d_2 & 0 & b_2 & d_3 & 0 & b_3 & d_4 & 0 & b_4 \\ 0 & d_1 & c_1 & 0 & d_2 & c_2 & 0 & d_3 & c_3 & 0 & d_4 & c_4 \end{bmatrix} \tag{7.11}$$

Returning to the equation for strain energy we have:

$$E_S(x,y,z) = \int_V \begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix}^T C \begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix} dV = \int_V U^T N^T L^T C L N U \; dV = \int_V U^T B^T C B U \; dV \tag{7.12}$$

and because the integral over the volume $V$ only includes constant terms [HDSB01, page 263]:

$$\begin{aligned} E_S(x,y,z) &= \int_V U^T B^T C B U \; dV = U^T B^T C B U \int_V dV \\ &= U^T B^T C B U V = U^T V B^T C B U \end{aligned} \tag{7.13}$$

now defining the stiffness matrix to be:

$$K^{(e)} = V B^T C B \tag{7.14}$$

the function becomes:

$$E_S(x,y,z) = U^T K U \tag{7.15}$$

inserting this into the function for potential energy and solving the integration of external force as above, gives:

$$\prod(x,y,z) = E_S(x,y,z) - U^T F \int_V dV = U^T K U - U^T F V \tag{7.16}$$

To get an impression and overview of the equation sizes we list the matrix sizes: $U$: $12 \times 1$, $F$: $12 \times 1$, $C$: $6 \times 6$, $L$: $6 \times 3$, $N$: $3 \times 12$, $B$: $6 \times 12$, and finally $K$: $12 \times 12$.

## 7.4 Assembling the System Equations

The system equations are assembled in a straightforward manner. We expand the element stiffness matrices and add them to yield the system stiffness matrix as illustrated by the example in

section 6.6 on page 79. To get an idea of the size of the complete system we consider a system with 100 four-node tetrahedron elements, with 70 global nodes. The size of the complete system will then have a force vector $F$: $70 \times 1$ and $U$: $70 \times 1$. The stiffness matrix for the system will then be $70 \times 70$ in size. The example above is relatively small compared with the models we are using. The number of elements in our models range from 1000 to 10000 elements.

## 7.5 Imposing the Boundary Conditions

Because the number of unknowns in the system are larger that then number of equations, we need to impose boundary conditions, so it is possible to solve the system. Recall that we aim at modelling fragmentation of a tooth, as part of the surgical procedure of removing a wisdom tooth. In this scenario a part of the tooth is located in the jaw, which effectively restrains the position of its roots. By fixing the position of nodes located in the jaw, we impose a boundary condition hereby restricting the solution domain.

## 7.6 Solving the System Equations

To solve the system equations we need to use the principle of minimum potential energy. To find this minimum potential energy the system equations have to be differentiated [Bat07, page 85-87].

$$\frac{\partial \prod}{\partial x} = 0 \qquad \frac{\partial \prod}{\partial y} = 0 \qquad \frac{\partial \prod}{\partial z} = 0 \tag{7.17}$$

in our case, repeated from equation (7.16):

$$\prod(x, y, z) = U^T KU - U^T FV$$

by applying equation (7.17) this becomes:

$$\partial \prod(x, y, z) = KU - F = 0 \tag{7.18}$$

which in turn can be stated as:

$$KU = F \tag{7.19}$$

This equation relates displacements to external forces via the stiffness matrix. The stiffness matrix describes physical laws of the problem we are modelling. The stiffness matrix could model any problem, making this equation general. Therefore equation (7.19) is called the *standard finite element equation*.

### Introducing Time Into the Equation

Equation (7.19) defines the standard finite element equation of equilibrium but only for static problems. In dynamic problems we need to introduce time into the equation. The most important equation in dynamic problems is based upon the static case as defined in equation (7.19) but now there is motion. According to Newton's second law motion is introduced as mass times acceleration. By introducing this into the equation we obtain

$$M\ddot{U} + KU = F \tag{7.20}$$

where $M$ is the mass matrix, the dot notation is used for time derivatives so $\ddot{U}$ is the acceleration equal to the second derivative of the displacement, $K$ is the stiffness matrix and $U$ is the

displacement vector. Equation (7.20) is equivalent to the equation for undamped harmonic motion. By measuring the actual dynamic response of structures it can be observed that energy vanishes during motion, preventing the structure from oscillating infinitely. Taking this into account damping is introduced into the equation [Bat07, page 166]:

$$M\ddot{U} + D\dot{U} + KU = F \tag{7.21}$$

where $D$ the constant damping matrix. Equation (7.21) is still a linear expression. By allowing the stiffness matrix $K$ to change according to the displacement $U$ we introduce non-linearity. The stiffness matrix can be expressed as a function of the displacement:

$$M\ddot{U} + D\dot{U} + K(U)U = F \tag{7.22}$$

### Time Integration Schemes

In general implicit or explicit methods are used when obtaining numerical solutions to time-dependent problems involving partial differential equations. Basically explicit methods calculate the next system state based on the current, whereas implicit methods solve an equation from the current system state. If $B(t)$ is the current system state at time $t$ and $B(t + \Delta t)$ is the next system state at time $t$ plus a time step $\Delta t$ then an explicit method would obtain the next system state by:

$$B(t + \Delta t) = F(B(t)) \tag{7.23}$$

whereas an implicit method would obtain it by the equation:

$$G(B(t), B(t + \Delta t)) = 0 \tag{7.24}$$

The implicit method requires extra computations when solving the equation but since $G$ is a function of both the current and the next system state, it is possible to find solutions using large time steps. Implicit integration methods tends to be unconditionally numerically stable (with a few exceptions) whereas the explicit method is not. The explicit methods are only conditionally stable because the numerical error increases along with the size of the time step. The *central difference method* is an explicit method that is very effective in the solution of certain problems. Based on the previous system state at time $t - \Delta t$ and the current at time $t$ the central difference method can be used to obtain an approximate solution for time $t + \Delta t$. The choice of integration scheme depends on the problem domain. See [Bat07, page 768] for a more detailed explanation of integration schemes.

### Total versus Updated Lagrangian

*Total Lagrangian* refers to the formulation of the finite element method used. Stress, strain, and deformation are all measured with respect to the initial (undeformed) configuration of the system in contrast to the *Updated Lagrangian* formulation where measures are with respect to the previous configuration [Bat07, page 522]

### Total Lagrangian Explicit Dynamic Solver

The solver applied is known as the *Total Lagrangian Explicit Dynamic Solver* (TLED), the following section will elaborate on the solving technique as presented in [TCO08]. Before getting into the details of the solving technique, we will start by defining what exactly we are trying

to solve. Given a volumetric body and using the finite element method we are interested in computing the deformations and the internal stresses as a result of the externally applied forces.

The solver is based upon a total Lagrangian formulation, so the stress and strain measures must comply with this. Here we use the *Green-Lagrange* strain tensor since it is a strain measure formulated with respect to the initial (undeformed) configuration and therefore complies with the total Lagrangian formulation. We also need to choose an appropriate stress measure to use with the strain measure. According to [Bat07, page 515] the *Second Piola-Kirchoff* stress tensor is work-conjugate with the Green-Lagrangian strain tensor. These two measures will be further elaborate in this section.

The central difference method is used for the explicit time integration scheme which allows us to perform calculations separately for each element. This makes the solving technique highly suitable for parallel execution.

### Solution Method

The solving technique used is developed for high-speed non-linear finite element analysis of soft materials by [TCO08], their work is based on methods presented by [MJLW07]. The key contribution made to the previous work done by [MJLW07] is mainly performance improvements, e.g. a solution scheme for solving finite element equations in parallel. To gain an overview of the solution method we will start by briefly introduce the equations used and how they are related.

The solver works in an iterative way and by the end of each iteration we obtain a solution to the equilibrium of the body as defined in equation (7.22). The solution is the displacement of each node in each element of the body.

### Notation

Throughout this section the following notation is used.

$$ {}^{t}_{0}X_{\_} \tag{7.25} $$

A left superscript indicates the configuration ($t$ = current) in which the quantity ($X$) is measured. The left subscript (0) indicates the configuration the quantity is measured with respect to (0 = initial configuration). If $e$ is the right subscript it indicates the element the quantity is measured in. If $i$ or $ij$ is the right subscript it refers to the $i^{th}$ row in a matrix or the matrix component on the $i^{th}$ row in the $j^{th}$ column.

### Displacement Derivatives

We will now use the shape functions to interpolate the current displacement matrix $U$ at time $t$. On matrix form the shape function for a tetrahedron is represented by a $4 \times 3$ matrix, three partial derivatives for each of the four nodes. We use the notation ${}_0\partial N$ for the initial partial derivatives of the shape function. The current element displacement derivatives are obtained by:

$$ {}^{t}_{0}\partial U_d = {}^{t}_{0}U_e^T \; {}_0\partial N \tag{7.26} $$

where ${}^{t}_{0}\partial U_d$ is a $3 \times 3$ matrix representing the updated displacement derivatives for an element. The current nodal displacement for element $e$ is represented in ${}^{t}_{0}U_d$ as a $4 \times 3$ matrix and ${}_0\partial N$ is

a $4 \times 3$ matrix of element shape function derivatives. The shape function is obtained as explained in 6.4 on page 76.

### Deformation Gradient Tensor

The element displacement derivatives for each element have been obtained, so now we can determine the deformation gradient tensor. This is the fundamental measure of deformation with components:

$$ {}^{t}_{0}X_{ij} = \frac{\partial\ {}^{t}x_i}{\partial\ {}^{0}x_j} \qquad i,j\ \epsilon\ \{1,2,3\} \tag{7.27} $$

The deformation gradient tensor is a second order tensor represented as a $3 \times 3$ matrix [Bat07, page 502]:

$$ {}^{t}_{0}X = \begin{bmatrix} \frac{\partial^t x_1}{\partial^0 x_1} & \frac{\partial^t x_1}{\partial^0 x_2} & \frac{\partial^t x_1}{\partial^0 x_3} \\ \frac{\partial^t x_2}{\partial^0 x_1} & \frac{\partial^t x_2}{\partial^0 x_2} & \frac{\partial^t x_2}{\partial^0 x_3} \\ \frac{\partial^t x_3}{\partial^0 x_1} & \frac{\partial^t x_4}{\partial^0 x_2} & \frac{\partial^t x_5}{\partial^0 x_3} \end{bmatrix} \tag{7.28} $$

The deformation gradient captures the stretches and rotations that the material fibres have undergone from the initial undeformed configuration and to a deformed configuration. Since we have already obtained the displacement derivatives equal to ${}^{t}_{0}\partial U_d$ we simply add the identity matrix $I$:

$$ {}^{t}_{0}X = {}^{t}_{0}\partial U_d + I \tag{7.29} $$

If there is no displacement ${}^{t}U_e$ is zero, therefore ${}^{t}_{0}\partial U_d$ is zero (7.26) and hence ${}^{t}_{0}X$ will be equal to the identity matrix. An important property of the deformation gradient is that it can always be decomposed into a unique product of two matrices, a symmetric stretch matrix ${}^{t}_{0}Q$ and an orthogonal matrix ${}^{t}_{0}R$ corresponding to a rotation such that[Bat07, page 508]:

$$ {}^{t}_{0}X = {}^{t}_{0}R\ {}^{t}_{0}Q \tag{7.30} $$

As explained in section 4.2 on page 47 shearing and normal strain can be represented as only normal strain if the coordinate axis aligns with the principal directions of the strain. By decomposing a tensor the principal directions are represented by the rotation matrix $R$ and the principal stretch in these directions is represented as an axial scaling matrix $Q$. It is important that the deformation gradient tensor maintains the transformational properties of stretch and rotation. This is the reason why the identity matrix is added in (7.29) or else the tensor would be all zeros as a result of zero displacement (${}^{t}\partial U_d = 0$). The zero-tensor would eliminating the transformational properties and as we shall see later result in a density of zero.

### Measure of Volume Change

The determinant of the deformation gradient is a measure of the volume change as a result of deformation. The density of an element will change with the deformation because the mass is constant but the volume can change, we refer to this as the density ratio given by:

$$ \frac{{}^{0}\rho}{{}^{t}\rho} = {}^{t}J = det({}^{t}_{0}X) \tag{7.31} $$

If the density ratio equals one it means the volume is unchanged, if above or below one it means the volume increases or decreases, respectively. Note that the volume of any deformed element must be positive due to the fact that no matter how much an element is compressed the material can not disappear. This is the reason why the deformation gradient tensor equals the identity matrix if there is no displacement as mentioned earlier. If the material is incompressible (e.g. fluids) the volume remains unchanged.

### Right Cauchy-Green Deformation Tensor

The measure of stress is evaluated from the measure of strain, so first we need a proper way of measuring strain. We define strain measure as the change in material length from the initial undeformed configuration to a deformed configuration. Although the deformation gradient defined previously is a measure of deformation from the initial configuration to a deformed configuration, it cannot be used directly for strain measures because it contains rigid body rotations. A deformation gradient has the same properties as a rotation matrix. A rotation followed by its inverse leads to no change, the inverse of an orthogonal rotation matrix equals its transpose, ($\mathrm{RR}^T = \mathrm{R}^T\mathrm{R} = 1$ see section 4.1 on page 41, so we can exclude the rotation in the deformation gradient tensor by multiplying it with its transpose. This leads to the *Right Cauchy-Green deformation tensor*, which represents a rotational-free strain measure, defined by:

$$ {}^{t}_{0}C = {}^{t}_{0}X^{T}\ {}^{t}_{0}X \tag{7.32} $$

The tensor represents a measure of how much a material fiber has been stretched or compressed from the undeformed initial configuration to a deformed configuration.

### Green-Lagrange Strain Tensor

The *Green-Lagrangian strain tensor* is a non-linear strain measure based on the Cauchy-Green deformation tensor. Multiplying the deformation gradient with its transpose (7.32) eliminates the rotation, but at the same time the change in strain length is squared which makes it a *non-linear* strain measure. Non-linear strain measures are suitable for both small and large deformations. If the time step is small (much smaller than unity) the non-linear terms in the Cauchy-Green deformation tensor becomes negligible and the strain measure reduces to a linear expression [DD08, page 215]. The Cauchy-Green deformation tensor is defined with respect to the initial configuration, hence the Green-Lagrangian strain tensor is on total Lagrangian formulation:

$$ {}^{t}_{0}E_{GL} = \frac{1}{2}({}^{t}_{0}C - I) \tag{7.33} $$

### Second Piola-Kirchoff Stress Tensor

With the Cauchy-Green deformation tensor and the Green-Lagrangian strain tensor in place we can derive a measure of stress. As mentioned earlier the *Second Piola-Kirchoff stress* is work-conjugate with the Green-Lagrangian strain tensor [Bat07, page 515], the second Piola-Kirchoff stress tensor is obtained by:

$$ {}^{t}_{0}S = \frac{\partial^{t}_{0}E_{s}}{\partial^{t}_{0}E_{GL}} \tag{7.34} $$

where $E_s$ is the *strain energy function*. A strain energy function is a function representing the internal stored energy that will make a perfect elastic material return to its original shape when

external forces are removed.

Using the neo-Hookean hyperelastic model, which is one of the simplest non-linear strain energy functions, and through differentiation of equation (7.34) with respect to the Green-Lagrangian strain, the stress components can be obtained by [TCO08, page 655]:

$$ {}^{t}_{0}S_{ij} = G(\delta_{ij} - {}^{t}_{0}C^{-1}_{ij}) + \lambda \, {}^{t}J \, ({}^{t}J - 1) \, {}^{t}_{0}C^{-1}_{ij} \tag{7.35} $$

where $G$ is defined by equation (3.25) on page 34 and $\lambda$ is defined by:

$$ \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{7.36} $$

In (7.35) $\delta_{ij}$ is the Kronecker's delta, define by:

$$ \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{7.37} $$

The inverse of the Cauchy-Green deformation tensor is denoted ${}^{t}_{0}C^{-1}_{ij}$, and ${}^{t}J$ is the determinant of the deformation gradient tensor at time $t$. The terms in equation (7.35) defining the second Piola-Kirchoff stress do not seems to have a direct relation to the physical interpretation [Spe80, page 134]. The intuition behind a strain measure can be related directly to the physical stretches of the material fibers, the second Piola-Kirchoff stress on the other hand is much more complex in its way of evaluating the stress. Once evaluated the stress itself is a standard quantity measured in force per volume.

Recall that only six independent entries are necessary to define the entire state of stress as explained in section 3.2 on page 27. Therefore the second Piola-Kirchoff stress (7.35) can be written on vector form now denoted ${}^{t}_{0}\hat{S}$:

$$ {}^{t}_{0}\hat{S} = [{}^{t}_{0}S_{11} \;\; {}^{t}_{0}S_{22} \;\; {}^{t}_{0}S_{33} \;\; {}^{t}_{0}S_{12} \;\; {}^{t}_{0}S_{23} \;\; {}^{t}_{0}S_{13}]^{T} $$

where ${}^{t}_{0}S_{11}$, ${}^{t}_{0}S_{22}$, and ${}^{t}_{0}S_{33}$ are the normal stress and ${}^{t}_{0}S_{12}$, ${}^{t}_{0}S_{23}$, and ${}^{t}_{0}S_{13}$ is the shearing stress.

**Strain Displacement Matrix**

The strain displacement matrix ${}^{t}_{0}B^{e}$ relates the strain in element $e$ to the element's nodal displacements as defined by equation (7.11) on page 87. The strain displacement matrix is defined for each element and represented as a $6 \times 12$ matrix as defined in (7.11) on page 87. The strain measure is defined for each element, while the strain displacement matrix is defined for each node in each element. The strain displacement matrix applies a linear transformation, with translation and rotation, to each node. The initial strain displacement matrix for the $a^{th}$ node in an element is defined as:

$$ {}_{0}B^{e}_{a} = \begin{bmatrix} {}_{0}\partial N_{a,1} & 0 & 0 \\ 0 & {}_{0}\partial N_{a,2} & 0 \\ 0 & 0 & {}_{0}\partial N_{a,3} \\ {}_{0}\partial N_{a,2} & {}_{0}\partial N_{a,1} & 0 \\ {}_{0}N_{a,3} & 0 & {}_{0}\partial N_{a,1} \\ 0 & {}_{0}\partial N_{a,3} & {}_{0}\partial N_{a,2} \end{bmatrix} \quad (a = 1, 2, ...., n) \tag{7.38} $$

where ${}_0\partial N_{a,i}$ with subscript $a$ and $i$ denotes the partial derivative of the shape function for node $a$ with respect to the $i^{th}$ spatial dimension of the node position. Note that the constant strain displacement matrix ${}_0B_a^e$ is calculated for each node, therefore $n$ equals four in the case of a tetrahedron.

The strain displacement matrix is a function of the element's geometry. In non-linear analysis the geometrical function can change over time, hereby allowing the element's nodes to move in non-linear paths. Since the geometrical function can change over time we need to update the strain displacement matrix. Using the constant strain displacement matrix ${}_0B_a^e$ and the current deformation gradient ${}_0^tX$ we obtain the updated strain displacement matrix ${}_0B_a^e$ by:

$$ {}_0^tB_a^e = {}_0B_a^e \; {}_0^tX^T \qquad (a = 1, 2, ...., n) \tag{7.39} $$

### Nodal Force Contributions

The internal stresses and the strain displacements have been obtained so now we can derive the nodal force contributions $\tilde{P}$. The nodal force is calculated for each node in each element. The four nodal forces are obtained by:

$$ {}^t\tilde{P}^e = {}^0V^e \; {}_0^tB_L^{e\,T} \; {}_0^t\hat{S}^e \tag{7.40} $$

where ${}^t\tilde{P}^e$ is a $12 \times 1$ column matrix, ${}^0V$ is the initial volume, ${}_0^tB^e$ is the $6 \times 12$ strain displacement matrix and ${}_0^t\hat{S}$ is a $6 \times 1$ row matrix defining the stress. All quantities are defined for each element $e$.

### Nodal Displacement

The final step is to calculate the displacement matrix $U$ for each element. When elements share a node, each element will have force contributions to that particular node (each obtained by equation 7.40) and the result is the sum of all contributions denoted $P_i$ where $i$ is the global node index.

Using the central difference method and equation (7.22) on page 89 the component-wise displacement for the configuration at time $t + \Delta t$ can be obtained by:

$$ {}^{t+\Delta t}U_i = \frac{{}^tF_i - {}^tP_i + \frac{2M_{ii}}{\Delta t^2} \, {}^tU_i + (\frac{D_{ii}}{2\Delta t} - \frac{M_{ii}}{\Delta t^2}) {}^{t-\Delta t}U_i}{\frac{D_{ii}}{2\Delta t} + \frac{M_{ii}}{\Delta t^2}} \tag{7.41} $$

where $M$ is the constant mass matrix, $D$ is the constant damping matrix, and $F$ is the externally applied load. It is assumed that ${}^tU$ and ${}^{t-\Delta t}U$ are known, ${}^tU$ and ${}^{t-\Delta t}U$ being the current and previous displacement matrix, respectively. Since the mass and damping matrices are constant the following vectors on page 95 can be pre-calculated hereby simplifying equation (7.41):

$$ {}^{t+\Delta t}U_i = A_i({}^tR_i - {}^tF_i) + B_i \, {}^tU_i + C_i \, {}^{t-\Delta t}U_i \tag{7.42} $$

$$
\begin{aligned}
M_{ii} &= \frac{1}{4}\rho V \\
D_{ii} &= \alpha M_{ii} \\
A_i &= \frac{1}{\frac{D_{ii}}{2\Delta t} + \frac{M_{ii}}{\Delta t^2}} \\
B_i &= \frac{\frac{2M_{ii}}{\Delta t^2}}{\frac{D_{ii}}{2\Delta t} + \frac{M_{ii}}{\Delta t^2}} = \frac{2M_{ii}}{\Delta t^2} A_i \\
C_i &= \frac{\frac{D_{ii}}{2\Delta t} - \frac{M_{ii}}{\Delta t^2}}{\frac{D_{ii}}{2\Delta t} + \frac{M_{ii}}{\Delta t^2}} = \frac{D_{ii}}{2\Delta t} A_i - \frac{B_i}{2}
\end{aligned}
\tag{7.43}
$$

## 7.7 Discrete Fracture Mechanics

In addition to solving the system of equations we are also interested in determining if the level of stress causes any fractures. We can assume certain properties when working with brittle materials. When a crack emerges we can assume it will propagate all the way through the material due to the huge energy release. Furthermore we can assume that the crack propagation will happen in the blink of an eye due to the fact that cracks propagate approximately with the speed of sound in brittle materials [Gdo05, page 242]. A variety of different crack tracking schemes exists and basically they can all be categorised into four types of tracking; *fixed tracking*, *local tracking*, *non-local tracking* and *global tracking*. Determination of the continuous crack plane in three-dimensions is the key challenge and handled in a distinct way for each crack tracking category.

### Crack Tracking Strategies

*Fixed tracking* is based upon the assumption that the failure surface is *a priori* known. A typical example of predefined failure surfaces could be in structures with obvious weak interfaces between substructures like in weldings. Knowing the possible failure surface beforehand makes the algorithm very fast and stable but not very flexible.

*Local tracking* is based upon loading history of the material. Here the failure surface is not *a priori* known. As the load on the material increases so do the internal stress eventually exceeding the fracturing limit $\sigma_F$. When the limit has been exceeded there are many different strategies for determining the orientation of the crack plane. One such strategy is based upon the maximum principal stress direction where the direction of the stress is normal to the crack plane. The plane orientation is modified according to any cracked neighbouring elements to ensure a continuous crack surface. The drawback of the local tracking scheme is the fact that any new crack plane is severely restricted by the local neighbouring elements already cracked. If two edges are already determined by neighbours then the crack plane is fully determined no matter the direction of the maximum stress.

*Non-local tracking* is an extension to the local tracking. To circumvent the obvious drawback of the local tracking, this algorithm considers more than just local neighbours when determining the crack plane. All neighbours within a certain radius are taken into account often interpolating the different maximum stress directions. This method requires extra computational costs and it strongly relies on the number of existing cracked neighbours. If the number of cracked neighbours is low this type of algorithm doesn't do very well [JSK08, page 1341].

*Global tracking* is based upon finding a global solution. A global system of equations must be found and solved to determine the continuous failure surface. Finding and solving the system of equations requires a lot of efforts compared to the other methods. The computational cost of this type of algorithm is the highest of all four methods, but the algorithm is also the most general and stable.

### The Crack Tracking Algorithm

The following crack tracking algorithm is based upon [AB05] and uses a local crack tracking scheme. The local tracking scheme is computationally fast and it produces great results with brittle materials since it tends to produce relatively flat failure surfaces. Since the orientation of new crack planes are severely restricted by any cracked neighbour elements, the failure surface tends to be neither concave nor convex shaped. This conforms with the fact that cracks in brittle materials tends to travel the shortest path to the surface. An element $e$ has between one and four neighbouring elements, here defined as those nearby elements that shares a triangular face with $e$.

It is assumed that the crack initiation only occurs within a single element. If more than one element exceeds the fracturing limit only the element with the highest principal stress is fractured hereby initiating the crack. Finding the maximum principal stress means solving the eigenproblem as explained in section 4.2 on page 47. The maximum principal stress direction defines the normal vector to the initial crack plane. This means the orientation of the initial crack plane is only determined by the maximum principal stress direction.

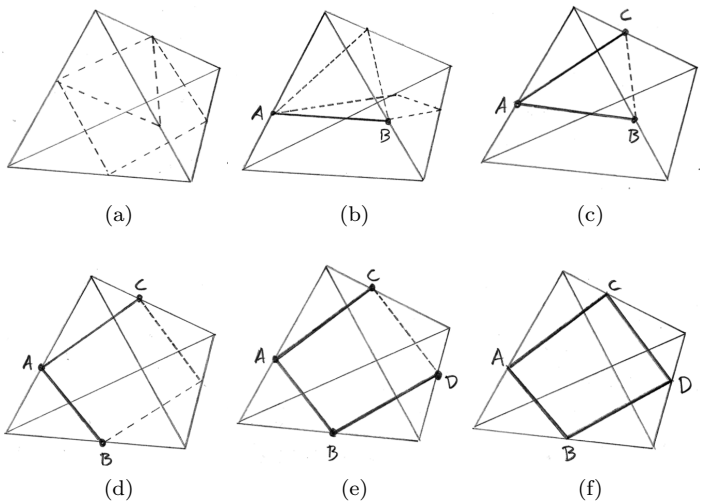



Figure 7.4: Crack configurations.

As depicted in figure 7.4 on the facing page case (a) the crack plane has no predefined neighbouring edges and is therefore determined by the maximum principal stress direction alone. When the crack has been initiated the following crack planes depend on the element's neighbourhood. Case (b) illustrates an element about to be cracked. In this case the cracked neighbour defines two crack points that must be located on the crack plane we are about to determine. The new crack plane is only free to rotate around the edge defined by the two neighbouring crack points. The new normal vector $n'$ to the plane is determined by the principal stress direction $n$ and the shared edge defined between crack point $A$ and $B$, is obtained by:

$$n' = n - \left[\frac{n \cdot (A - B)}{|A - B|^2}\right](A - B) \tag{7.44}$$

In case (c-f) the crack plane normal is determined by the pre-existing neighbouring crack planes. With two or more neighbouring edges the plane normal is completely determined and the element's maximum principal stress direction is not considered. Case (a) initiates the crack which only happens once. From here the crack propagates by cracking all neighbours in the next iteration, and their neighbours in the next etc. This way the failure surface will propagate like rings in the water, from the initial cracked element and out.

# Part III

# Implementation

# Chapter 8

# The Simulation Model

The *simulation model* is here defined to be the fundamental components driving the simulation. The fundamental components are responsible for the user interaction, solving the system of equations and visualizing the results in a three-dimensional scene. The following section will introduce the fundamental components of the simulation model. We will elaborate on the responsibility of each component and the technologies used. An overview of the simulation model is illustrated in figure 8.1.

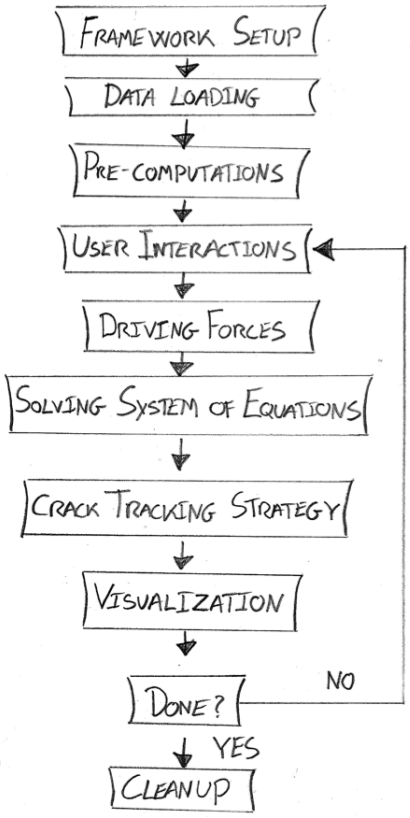



Figure 8.1: Overview of the simulation model.

### Framework Setup

The simulator model is implemented using the OpenEngine framework[1] which is a lightweight visualization framework with a small kernel and an extension system easy to use. The framework is written in C++[2] and the build system cmake[3] makes is easy to build on Windows, Linux and Mac OS X. The authors of this thesis are also the co-authors of the OpenEngine framework so the choice was obvious.

The first step is to set up the OpenEngine framework. The framework handles the main loop of the application. All components that need process time must be inserted into the framework as modules. The modules will be processed in a sequential manner in each main loop iteration. Setting up the framework includes adding nodes to the scene graph, setting the position and orientation of the camera, adding input handlers, and setting the path to disk resources like models and textures.

### Data Loading

Some resources have to be loaded from disk. The primary resource loaded is the solid model we are interested in simulating. The solid model is loaded into memory using a TetGen loader. TetGen[4] is an open source project capable of generating a quality tetrahedral mesh from a surface representation of a solid model. The data model representing the model consists of a vertex-pool, a surface mesh, and a body mesh. The vertex-pool is an one-dimensional array containing vertices represented in global space as vectors. Each component of a vertex vector is represteted as a floating point number. The surface and body meshes are one-dimensional arrays containing integer indices, pointing into the vertex-pool. The surface mesh bundles three indices pointing to the three vertices from the vertex-pool defining a surface triangular face. The body mesh bundles four indices pointing to the four vertices defining a tetrahedron. The surface mesh is used when the body surface is visualised and the body mesh identifies the nodes used when simulating the physics.

### Pre-computations

The pre-computations are an important part of the Total Lagrangian Explicit solver as explained in section 7.6 on page 94. The equation for displacement as defined by (7.41) on page 94 has been simplified through pre-computation of the mass, the damping, and the three matrices $A$, $B$, and $C$ as defined in equation (7.43) on page 95. The pre-computations are essential to the performance of the solver. By pre-computing as much as possible more resources are made available at run-time. The shape function derivatives are an important part of the finite element method and can also be pre-computed. The shape function derivatives are obtained as explained in section 6.4 on page 70.

Explicit time integration operators such as the central difference method used is only conditionally stable. The main concern on the approximating method is the fact that the error increases with the time step size. It is essential to keep the time step size small or else the approximation error will grow out of proportion causing numerical instability. There is a complicated relation between the maximum allowed time step, the geometry of the elements and the material properties. To

[1] http://www.openengine.dk
[2] http://en.wikipedia.org/wiki/C++
[3] http://www.cmake.org/
[4] http://tetgen.berlios.de/index.html

prevent the explicit integration from becoming numerically unstable we must determine the size of the critical time step [TCO08, page 654]:

$$\Delta t_{cr} = \frac{L_e}{c} \tag{8.1}$$

where $L_e$ is the smallest edge length from the set of elements and $c$ is the dilatational wave speed which defines how fast sound travels through the material defined by:

$$c = \sqrt{\frac{E(1-\nu)}{\rho(1+\nu)(1-2\nu)}}$$

where $E$ is Young's modulus, $\nu$ is Poisson's ratio and $\rho$ is the density. These values are all material dependent and usually determined by table look up.

In order to improve the performance of the crack tracking algorithm, a neighbouring list is pre-computed. Since it is a precondition that all elements are interconnected, each element must have at least one neighbour and maximum four. By pre-computing the neighbouring list the problem of finding neighbouring elements reduces at run-time from an $O(n^2)$ to an $O(1)$ problem by table look up.

Finally the initial volume of each element is pre-computed. The initial volume is used when calculating the nodal force contributions as explained in section 7.6 on page 94.

### User Interactions

The user interaction is handled by the OpenEngine framework. The module handling the user interaction is attached as a listener to the device registering the input. The device could be a keyboard, mouse, joystick, or haptic device. When input is registered all attached listeners are notified and proper action is taken by the receivers. Since the OpenEngine framework is single threaded, the event queues are only flushed once during each main loop iteration. The simulation must produce a minimum of 25 frames per second in order to accommodate the demand for real-time execution. Since each main loop iteration produces one frame the user interaction will be handled with the frame rate.

### Driving Forces

The initial state of the body is equilibrium. When there are no external forces acting on the body there are no internal forces. As explained in section 3.2 on page 28 the body is in equilibrium when the external forces equals the internal forces (equation 3.18). So in order for the system to react we have to apply external driving forces. The driving force can be applied to the entire domain, like the gravitational force would be, or it can be applied to only parts of the domain. To affect the system we have implemented different types of modifiers. The concept of modifiers and the different types are explained in section 10.1 on page 115. Modifiers like the *force modifier* and the *displacement modifier* change the equilibrium of the system by adding external forces or by changing the displacements, respectively.

### Solving the System of Equations

The system of equations must be solved in each iteration to constantly converge towards equilibrium. The solving technique is explained in detail in section 7.6 on page 89. The explicit solver is

a straightforward algorithm that in a fixed number of steps calculates displacements as a result of the externally applied forces.

As mentioned in the acknowledgement PhD student Karsten Noe kindly shared his knowledge and prototype solver. The solver is implemented in a straightforward manner according to the TLED article [TCO08]. Instead of reimplementing his work we incorporated and re-factored parts of the solver. We have reused the pre-computation of the shape function but improved the calculation of the nodal masses from an equally to a weighted distribution. Furthermore we have improved the precision of the shape function simply by increasing the floating point precision in the intermediate calculations from single to double. We have reused parts of the core matrix calculations, primarily the part where quantities like the deformation gradient and the strain and stress tensors are updated through matrix multiplications. We added all the modifiers used for controlling the driving forces as elaborated in section 10.1 on page 115. The eigenproblem solver and all components related to the crack tracking algorithm have been added. All debug and visualization tools were implemented from scratch (elaborated in chapter 10 on page 115).

### Crack Tracking Strategy

The *crack tracking strategy* is responsible for determining *when* the material will crack and *how* the crack will propagate. The crack tracking strategy used here is based on a local tracking scheme and the principle of maximum stress direction as explained in section 3.5 on page 35. Finding the maximum principal stress means solving the eigenproblem as explained in section 4.2, therefore we need to solve the eigenproblem for each tetrahedral element and compare the maximum eigenvalue with the fracturing limit defined for the material. The eigenproblem is solved for each stress tensor in each iteration, using a library, but the problem can be solved by hand as explained in section 4.3 on page 48.

**Algorithm 8.0.1:** Crack Tracking Algorithm()

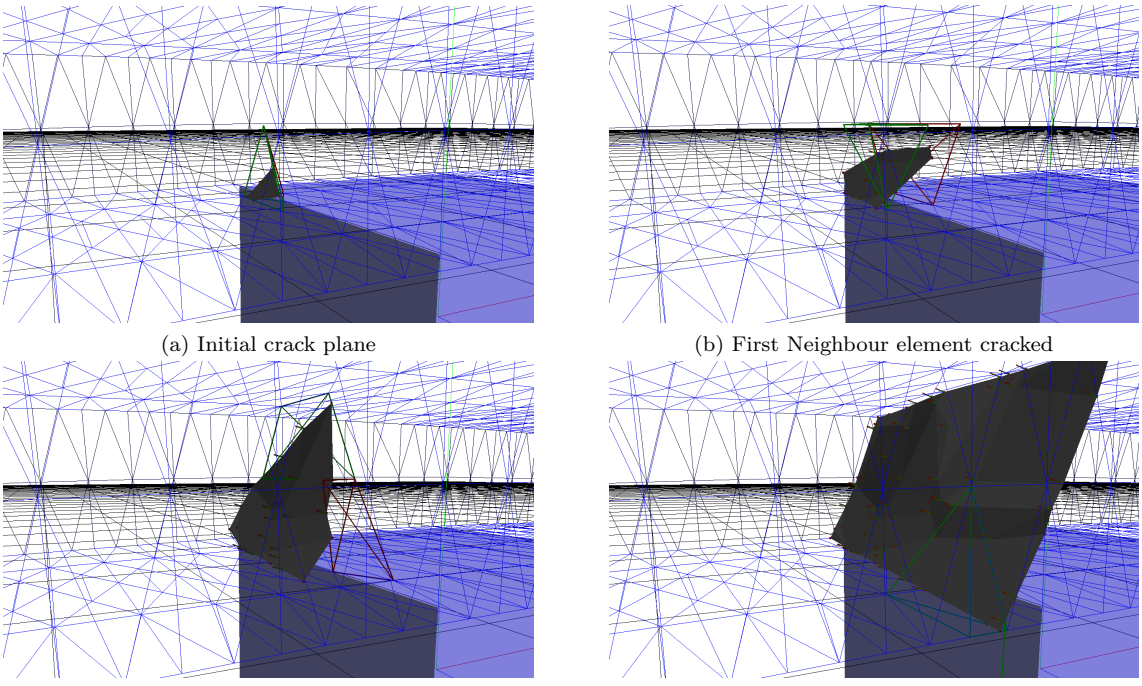
$MaxStress \leftarrow 0$
**for each** $e \in E$
**do**
- solve eigenproblem for e
- **if** $eigenvalue > MaxStress$
  - **then** $MaxStress \leftarrow eigenvalue$

**if** $MaxStress > limit$
**then**
- determine crack plane orientation from eigenvector
- add element to crack front set C

**while** $|C| > 0$
**do**
- lookup neighbouring elements
- determine crack plane from principal stress direction and cracked neighbours
- add uncracked neighbours intersecting crack plane to C
- remove cracked element from C

If the fracturing limit has been exceeded in one or more elements the simulation is stopped and we switch from parallel to sequential execution. The crack tracking algorithm is performed sequentially since it would not benefit from a parallel approach because the crack is propagating in a sequential manner. The element with the maximum eigenvalue will be the first one to crack. Since the first element has no cracked neighbours the orientation of the first crack plane is determined only by the maximum principle stress direction as explained in section 7.7 on page 96 on page 96.

In the algorithm $E$ is the set of tetrahedral elements and $C$ is the crack front set. When the crack has been initiated the crack tracking strategy will proceed by cracking neighbour elements until the failure surface has propagated all the way through the solid object. The crack tracking operates in a iterative way. When an element has been cracked, all its neighbouring elements, intersecting with the crack plane, are added to the crack front set. The crack front set is the set of elements that will be cracked in the next iteration. As the crack propagates the size of the crack front set usually increases rapidly since most elements has more than one uncracked neighbour. But when the failure surface reaches the surface of the solid object the crack front set decreases.



(a) Initial crack plane

(b) First Neighbour element cracked

(c) After the first iteration

(d) Crack tracking done

Figure 8.2: Crack propagation.

To illustrate the crack tracking algorithm consider figure 8.2. A concrete beam is dropped on a solid box, gravity is the only external force acting on the beam. Figure 8.2a illustrates the initial crack plane, where the orientation of the plane is determined by the maximum principal stress direction. Figure 8.2b and 8.2c illustrate the propagation of the crack. Finally the failure surface has reached the surface of the object and the crack tracking has been completed as illustrated in figure 8.2d.

### Visualization

The visualization is based upon a scene graph and a renderer. The scene graph is a tree structure of entities that can be visited breadth-first or depth-first by the renderer as defined by the visitor pattern [GHJV95, page 331]. The renderer is responsible for rendering the geometry on the screen, decoupling the rendering technology from the representation of the geometry. The default renderer is implemented using OpenGL[5] due to its cross platform libraries.

### Cleanup

When the simulation is terminated the memory allocated must explicitly be freed. This goes for the main memory as usual but also for the memory allocated on the graphics card.

[5] `http://www.opengl.org/`

# Chapter 9

# Parallel Execution

The Total Lagrangian Explicit Dynamics solving technique (described in section 7.6), used to solve the finite element equations is highly suited for parallel execution. So to get optimal performance when implementing this technique, we need to implement it for a parallel architecture. The range of parallel technologies and platforms to choose from is large, ranging from cluster farms, grid computers, graphics cards to multi-cored central processing units (CPUs). Motivated by the surgical scenario, and the request from densists to use the simulator in educational training, we have chosen a hardware platform, which is commonly available, inexpensive, and does not require a large storage facility. Two different platforms and accompanying APIs have been discussed, both available as consumer products and affordable. The first candidate was the Cell architecture, which is commonly available in Sony's PlayStation 3. The PlayStation 3 is suited for this kind of development because it allows custom installation of an alternative operating system like Linux, and because both an API and programming tools are freely available for Linux. The second platform is an ordinary PC with a high performance graphics card. Graphics cards nowadays are relatively powerful when considering the computational power contra price. We have chosen to base our implementation on multi-cored graphics cards as this is a widely spread technology, and because we find the provided programming abstractions and tools well suited for rapid prototype development.

## 9.1 Programming a Graphics Card

The processing unit on a graphics card is called the graphics processing unit (GPU), and general purpose programming of the GPU is called GPGPU programming. GPGPU programming has been done for quite a while now. The beginning was a sort of misuse of the original APIs for the GPU. Programmers wrote their applications in shader languages, which originally were intended for producing graphical effects as part of the visualization. As GPGPU programming has spread, new initiatives towards making a general GPGPU API, like for example CUDA, has started to surface. We have chosen the CUDA programming language developed by NVIDIA. This programming API has a higher level of abstraction, it comes with useful libraries and is easy to use. The drawback is that it is hardware dependent, and can therefore only be used with NVIDIA graphics cards.

## 9.2 Basic Description of CUDA

The Compute Unified Device Architecture (CUDA) is an extension to the C programming language, which enables the programmer to use the NVIDIA GPUs for GPGPU programming (this only apply for NVIDIA 8000 or newer GPUs). The extension includes special syntax, allowing programmers to explicitly determine which parts of the program that is going to be executed on the device. In CUDA terminology a *host* refers to the CPU with access to the main memory, and the *device* refers to a specific graphic card with access to the available memory, known as *global memory*, on the graphics card. In CUDA the kernels are launched with parameters defining the *block* and *grid* size. Threads are executed in blocks which are wrapped in a grid, as illustrated in figure 9.1.

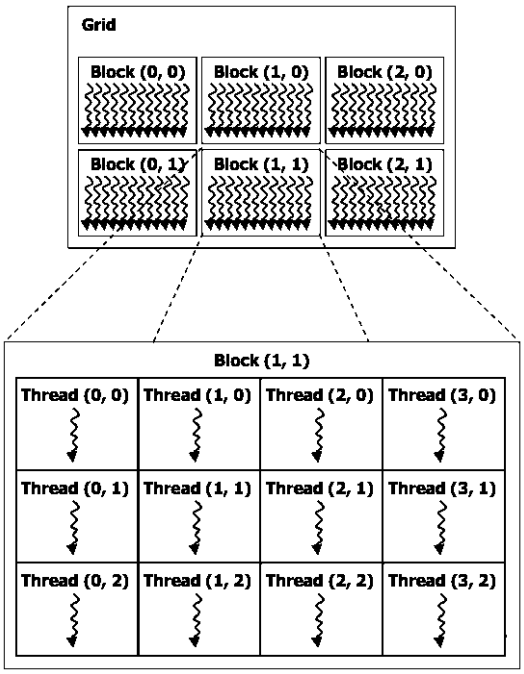



Figure 9.1: Illustration of the CUDA grid, block, and thread abstractions. ‡

When a kernel is launched the grid is set for execution. When executing the grid each block is scheduled one by one and 16 threads from the block are executed simultaneously by the hardware in what is known as a *half warp*. The execution order of the blocks and warps is non-deterministic and can be different every time the program is executed. The execution and scheduling of threads is performed by the device. The extension also includes special keywords which can be used inside kernels to access the dimensions and identifiers of the thread, block and grid.

## 9.3 Our Implementation

The CUDA technology allows code to be executed in parallel. The parallel execution used here is based on the *Single Instruction, Multiple Thread (SIMT)* technique used to achieve data level parallelism. This technique is closely related to the wide known *Single Instruction, Multiple Data*

*(SIMD)*. It basically means that multiple threads are executing the same program (same set of instructions) in parallel but with different data. When executing parts of the program in parallel we use blocking calls to start kernels on the GPU, this means that either the host or device is active. Switching from the host to the device, must be done explicitly in the code by executing a kernel. The host and the device have separate memory spaces. If data needs processing on both the host and the device it needs to be copied from one memory space to the other.

We will now elaborate on how the finite element system equations are solved by element-wise computations and how the individual solutions are assembled by the end of each iteration. Consider the element equation (6.26) for one tetrahedron, as explained in section 6.6 on page 79. The equation is repeated below:

$$F^a = K^a U^a \qquad \Leftrightarrow \qquad \begin{bmatrix} f^1 \\ f^2 \\ f^3 \\ f^4 \end{bmatrix} = \begin{bmatrix} K^a_{11} & K^a_{12} & K^a_{13} & K^a_{14} \\ K^a_{21} & K^a_{22} & K^a_{23} & K^a_{24} \\ K^a_{31} & K^a_{32} & K^a_{33} & K^a_{34} \\ K^a_{41} & K^a_{42} & K^a_{43} & K^a_{44} \end{bmatrix} \begin{bmatrix} u^1 \\ u^2 \\ u^3 \\ u^4 \end{bmatrix}$$

These equations can be easily computed in parallel for each element. It is when elements are connected and hereby interacting it gets more complex. When elements are connected they share nodal points. The shared nodal points can be seen explicit by looking at the final system equation. Consider the example from section 6.6 on page 79 again, here we directly see each element's contribution to the shared nodal variables. They are seen as weights ($K^e_{ij}$) added together in the matrix. The system matrix from equation (6.29) in the example is repeated below for convenience:

$$K = K^A + K^B = \begin{bmatrix} K^a_{11} & K^a_{12} & K^a_{13} & K^a_{14} & 0 \\ K^a_{21} & K^a_{22} + K^b_{11} & K^a_{23} + K^b_{12} & K^a_{24} + K^b_{13} & K^b_{14} \\ K^a_{31} & K^a_{32} + K^b_{21} & K^a_{33} + K^b_{22} & K^a_{34} + K^b_{23} & K^b_{24} \\ K^a_{41} & K^a_{42} + K^b_{31} & K^a_{43} + K^b_{32} & K^a_{44} + K^b_{33} & K^b_{34} \\ 0 & K^b_{41} & K^b_{42} & K^b_{43} & K^b_{44} \end{bmatrix}$$

Since the element equations are added together we can skip the construction of the system stiffness matrix, and instead add the individual nodal contributions from each element. This is the approach used by the TLED solver which can be summarized as:

- The external nodal forces are applied by the user interaction module and processed by the finite element solver. This data is located in the main memory and copied to global device memory as the first step.
- We then start the main solver kernel (TLED solver) called `calculateForces_k`. This kernel is executed for each element, and performs all calculations as explained in section 7.6 on page 89. The resulting list of nodal force contributions are saved in global memory. Furthermore this kernel solves the eigenproblem for each element, and detects if the material dependent fracturing limit has been exceeded. If so it raises a flag.
- The nodal force contributions are now being added together for each node to determine the resulting force and displacement. This kernel is named `updateDisplacements_k`. The resulting displacements are stored in global memory, one displacement vector for each node.
- The final kernel is named (`updateBodyMesh_k`) and it updates the surface and body mesh by adding the displacement vector to the initial position for each node. Updating the global nodal positions is only done for visualization purposes. Due to the small simulation time step performed in each iteration we solve the system equations multiple times in between the visualizations.

The finite element solver is executed every iteration of the main-loop, but the visualization of the surface, does not need to be updated this often. The visualization is set at approximately 30 frames per second (FPS), guaranteeing fluent motion of the visual animations. The visualization of the surface is done via the CUDA OpenGL interoperability API. This API allows OpenGL to access the memory used by the CUDA programs and directly visualizes it without the need of copying the data back and forth between main and global memory.

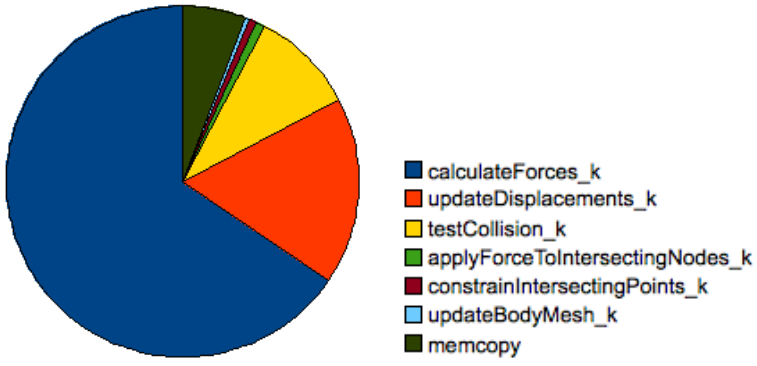
### Execution Time

To get an overview of how time consuming the different kernels are, we have made time measurements of each kernel executed. The test was performed on the laptop described in appendix D.2 on page 159 with version 1.2 of the CUDA Visual Profiler, using the tooth mesh described in appendix B.2. Table 9.1 shows the result from the profiler.

| kernel name | # calls | GPU microseconds | % GPU time |
|---|---|---|---|
| calculateForces_k | 52325 | 10968700 | 65.45 |
| updateDisplacements_k | 52325 | 2893260 | 17.26 |
| testCollision_k | 54417 | 1605060 | 9.57 |
| memcopy | 25115 | 966304 | 5.76 |
| applyForceToIntersectingNodes_k | 54417 | 131965 | 0.78 |
| constrainIntersectingPoints_k | 54417 | 125802 | 0.75 |
| updateBodyMesh_k | 2093 | 67304 | 0.40 |
| precalculateShapeFunctionDerivatives_kernel | 1 | 22 | 0.00 |
| loadArrayIntoVBO_k | 8 | 18 | 0.00 |
| applyTransformation_k | 2 | 9 | 0.00 |
| precalculateABC_kernel | 1 | 6 | 0.00 |

Table 9.1: Profiling results.

To get an visual overview of the percentages, the following diagram illustrated kernels that is part of one iteration, and not pre-computational kernels.




(a) Diagram. (b) Information.

Figure 9.2: GPU time spent on the individual kernels.

About 2/3 of the computational resources on the device is spent on processing the finite element solver which is not surprising considering the complexity compared to the other tasks.

## 9.4 Optimizing Performance

To get optimal performance from the device through CUDA we need to overcome the memory latency limitation imposed. The memory bus on the graphics cards has a high bandwidth but also a large latency when fetching from the global memory. When programming in CUDA one needs to be aware of the fact that memory fetching must be done in correspondence to the alignment and coalescing rules as described in [NVI09]. The alignment and coalescing constraints allow the GPU to effectively hide the memory latency. We want the simulation to run in real-time, therefore all code inside the main-loop has been optimized for performance in relation to these guidelines As most of this code is executed on the device, the main concerns are the memory latency as described.

### Using float4 Instead of float3

CUDA devices are capable of reading 4-byte, 8-byte or 16-byte words, corresponding to 32-bit, 64-bit, and 128-bit operations, from global memory into thread registers when executing threads [NVI09, page 81]. In our simulation we often use three consecutive floats. For example when representing vectors in space, like global coordinates, displacements, etc. If we represented this by three `float` variables, CUDA would generate two memory operations to fetch the memory, one 32-bit and one 64-bit. Instead we use one `float4`, which corresponds to a single 128-bit memory operation, resulting in faster memory fetching. The increased performance is at the expense of increased memory consumption.

### Coalesced Memory Access

Global memory bandwidth is used most efficiently when the simultaneous memory accesses by threads in a half-warp, can be coalesced into a single memory transaction of 32, 64 or 128 bytes [NVI09, page 82]. This means that if the threads are using memory cells located next to each other in an array the memory for all the threads can be fetched in one transaction or block copied of the memory. To facilitate this we use the same memory allocation scheme as [RMS08, page 55-56]. Instead of allocating tetrahedral elements side by side, we use arrays of individual variables related to all tetrahedra. This means that arrays are allocated for storing all densities, all masses, all nodal indices etc.

# Chapter 10

# Helper Tools

In this chapter we will introduce the helper tools implemented. The different kind of modifiers will be introduced followed by the tensor visualization tool.

## 10.1 Modifiers

The initial state of the physics system is equilibrium. In order to simulate a reaction something has to act on the system forcing it out of equilibrium. All interaction with the system is done through modifiers of different types. Each modifier consists of a nodal selection tool and an action that will be applied to the selected nodes. Any volumetric shape can be used as a selection tool. By default box selection is used. Nodes inside the volumetric shape are in the set of selected nodes. During simulation a specified action is applied to the set of selected nodes. The modifiers vary in the way they interact with the physics system. The following modifiers types will be introduced:

- Fixed Displacement Modifier
- Force Modifier
- Displacement Modifier
- Restrictive Displacement Modifier
- Projective Displacement Modifier

### Fixed Displacement Modifier

The *fixed displacement modifier* restrains the displacement of all selected nodes by fixing their position in space. The positional restriction is applied to all selected nodes. Usually it is necessary to restrict part of the object when external forces are applied. Otherwise the object would just move instead of causing deformation, but of course this depends on the scenario. The fixed displacement modifier is like a clamp holding the object in place while working on it.

Figure 10.1 on the next page illustrates a beam fixed in one end. The fixed displacement modifier is always visualized with a green color to indicate that the selected nodes are neutral to any stress and strain. The restriction is applied simply by overwriting the new displacement by the current hereby resetting any nodal movement.

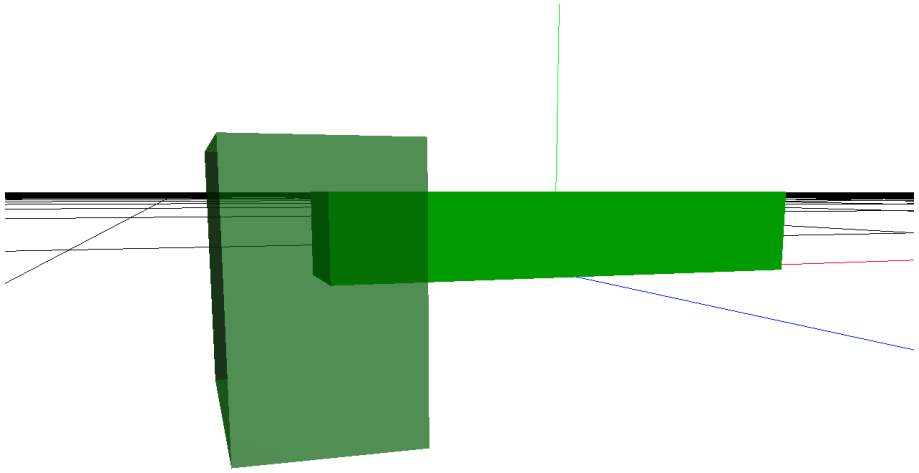


Figure 10.1: Beam fixed at one end.

### Force Modifier

The *force modifier* applies a force with any specified direction and magnitude to all selected nodes. This is very useful for applying loads to selected parts of the object being simulated. During simulation new nodal selection can be made hereby gradually increasing or decreasing the load.

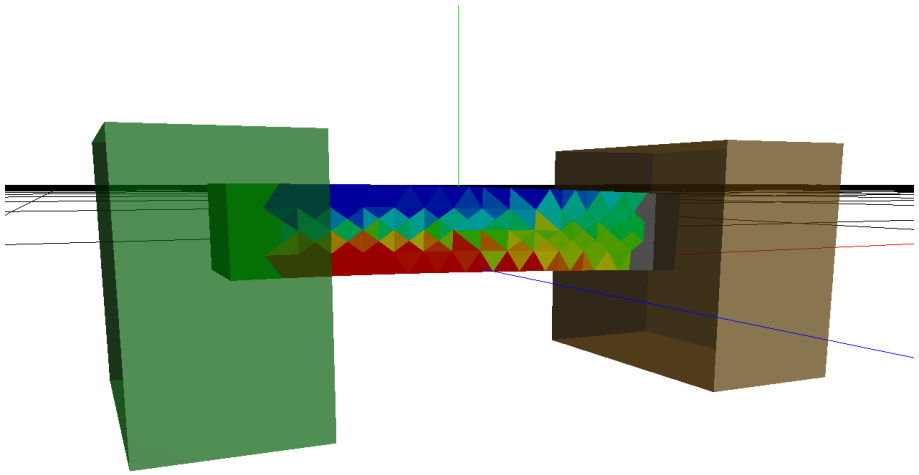


Figure 10.2: Supported beam with load applied to its end point.

Figure 10.2 illustrates a beam supported at the left end and applied with external forces at its right end. The selection tool for the force modifier is always visualized by a brown color with the

selected nodes grayed out. As illustrated in figure 10.2 on the preceding page only the end of the beam is selected and the force modifier applies a force acting downwards which causes tension on top of the beam and compression at the bottom.

### Displacement Modifier

The *displacement modifier* adds a specified displacement vector to all nodes selected. Furthermore it restricts the selected part of the object from any deformation hereby preventing it from absorbing any stress or strain.

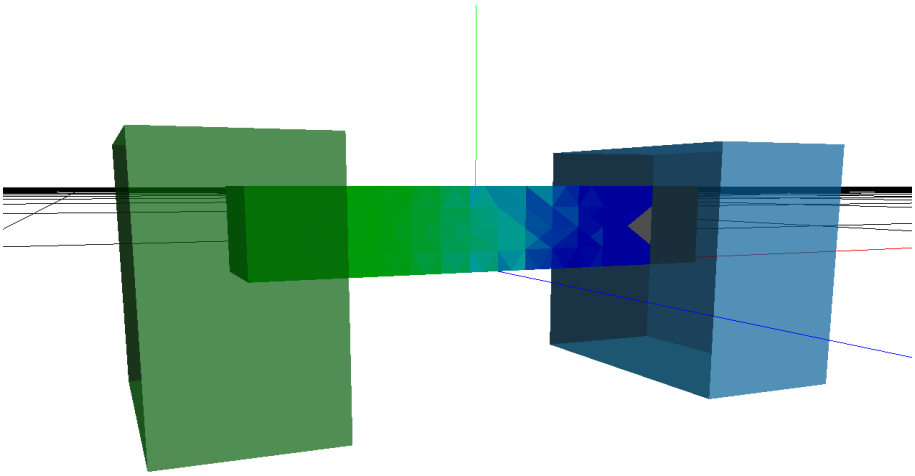


Figure 10.3: The beam is fixed at the left end while stretched by the modifier at the other end.

Figure 10.1 illustrates a beam with its left end fixed while a displacement modifier is applied at the right end. The displacement modifier stretches the beam horizontally causing tension in the beam. The stresses and strains occurring as a result to this, can only occur in the mid subsection of the beam where none of the two modifiers are applied. The selection tool for the displacement modifier is always visualized by a blue color with the selected nodes grayed out.

### Restrictive Displacement Modifier

The *restrictive displacement* modifier simply restricts any nodes from entering the space occupied by the selection tool. This modifier can be use for simple collision detection between an object and the modifier. The restrictive displacement modifier can be used as a ground restriction in the scene or simply as a colliding object as illustrated in figure 10.4 on the next page. If a node is about to enter the space occupied by the selection tool it will be moved back to its previous location. The drawback of this very simple restriction is that the method only applies under the assumption that a previous nodal position exist. If an object does not move, all nodal positions in the current configuration at time $t$ equals the nodal positions in the previous configuration at time $t - \Delta t$. In other words the object has to intersect with the modifier and not the other way around. Due to the simplicity this type of modifier is computationally fast in contrast to the projection displacement modifier as explained next.

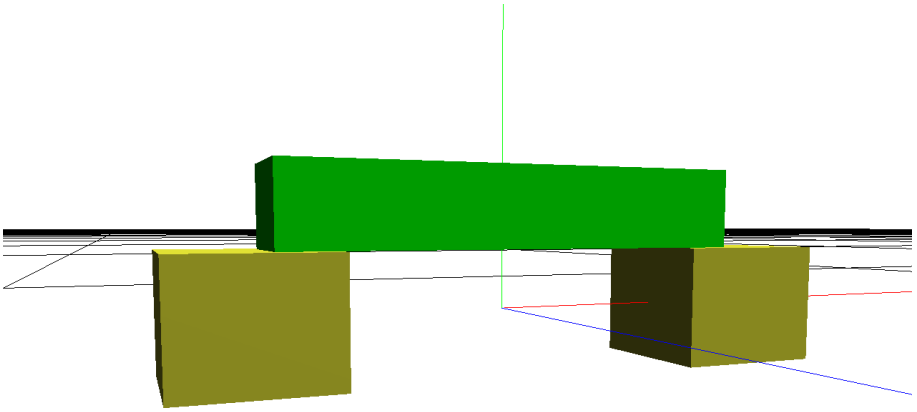


Figure 10.4: Gravitational force acts on the beam supported in both ends.

Figure 10.4 illustrates the gravitational force acting on the beam supported by two restrictive displacement modifiers. The selection tool occupying the restrictive space is always visualized by a yellow color.

### Projection Displacement Modifier

The *projection displacement modifier* is an extension to the restrictive displacement modifier, the only difference being the algorithm used when correcting the nodal displacements.

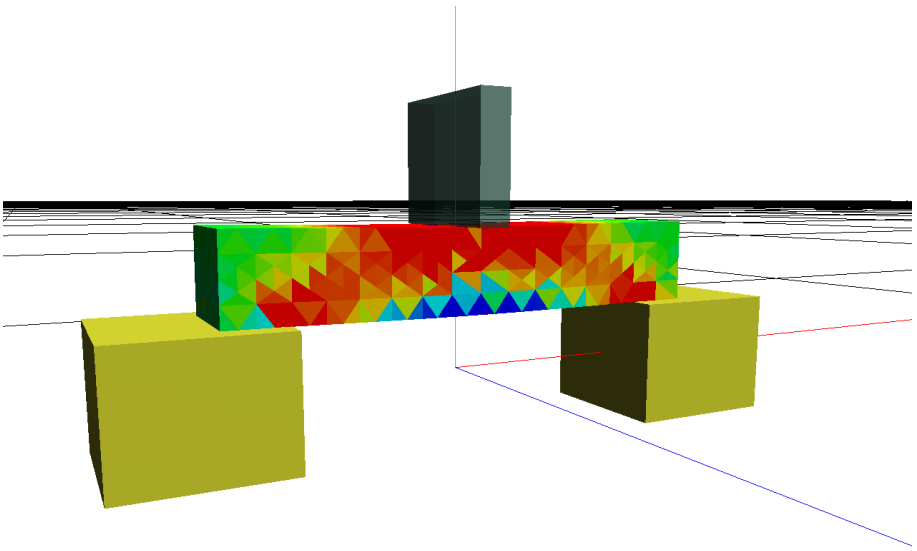


Figure 10.5: Center nodes at the top of the beam are forced down causing the beam to bend.

As already mentioned the restrictive displacement modifier only works when the object intersects with the modifier. We need the ability to interact with the object in the way a craftsman uses his tools on a material. When the modifier intersects with an object we need to calculate a new displacement for the intersecting nodes. The default strategy implemented relies on vector projection. Any node intersecting the space occupied by the selection tool is projected onto the closest plane from the geometry defining the selection tool. By default a box is implemented as the selection tool. If a node is located inside the box, the distance to each of the six planes defining the box is calculated. The node is then projected onto the plane closest to the node.

Figure 10.5 on the preceding page illustrates a supported beam being subjected to a projection displacement modifier forcing the midpoint nodes down causing the beam to bend. The selection tool for the restrictive displacement modifier is visualized by a dark blue-gray color.

## 10.2 Tensor Field Visualization

Visualizing the stresses and strains would reveal very interesting information e.g. how stress and strain propagates through the simulated object and where potential strengths and weaknesses would be located within the material. Stress and strain tensors are defined for each element. The number of elements depends on the size of the problem domain but it is not unusual to have hundreds or thousands of elements. The data set, containing a tensor for each element forming a tensor field, can easily become relatively large. The challenge of tensor field visualization is to transform this large data set into a single image, easy to interpreted. A method for this has been presented by [Wün08]. The method presented is based upon *data transformation* and *data reduction*.

Data transformation is necessary since the data set consists of tensors on matrix form which cannot be visualized directly. We are interested in retrieving the relevant information from the data set and represent it differently. As already explained in section 4.2 on page 47 the eigenproblem of a second order symmetric tensor defines the principal eigenvalues and corresponding eigenvectors. Retrieving the principal values and directions from a second order tensor is an example of data transformation.

Data reduction is necessary since the data set can become relatively large dependent on the size of the problem domain. A way of reducing the data set could be to only extract the largest eigenvalue and the corresponding eigenvector since these define the maximum principal value and direction, respectively. Experiments with stress visualization indicate that when external forces are applied to an object the amount of stress heading in the maximum principal stress direction is by far the most dominant in comparison to the other two stress directions. Recall that the tensor quantity (here being stress or strain) can be resolved into three mutually perpendicular vectors known as the maximum, medium and minimum principal directions.

The following tensor field visualization is based upon solving the eigenproblem for each stress tensor and reduce this by only considering the largest eigenvalue (maximum principal stress) and the direction of the corresponding eigenvector (maximum principal stress direction). The eigenvalue is visualized by mapping it to a color scheme and the eigenvector is visualized by a small visual entity pointing in the same direction as the eigenvector.

### Vizualizing the Eigenvalues

To visualize the maximum principal stress, we use a color scheme. Since the eigenvalue is mapped to a color scheme which has three components the conversion is not straightforward. We need to use a color ramp to do the mapping. A *color ramp* is a function which maps a range of scalar values to colors. We use a linear scaled color ramp known as the *hot-to-cold color ramp*, ranging from blue over green to red as illustrated in figure 10.6. This color ramp is ideal when visualising values above and below zero.

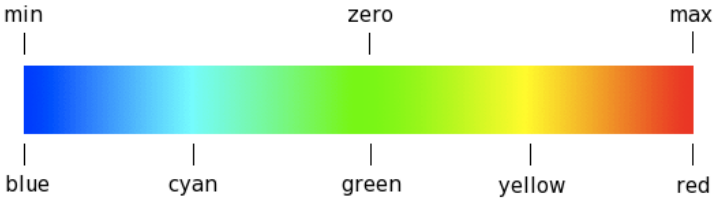



Figure 10.6: The hot-to-cold color ramp.

To define such a function we first have to specify how to represent colors in the simulation. We used the three component color scheme, with red, green, and blue (RGB) represented as floating point values ranging from 0 to 1. Any color representable is a mixture of these three components. The three components are bundled in a $3 \times 1$ vector. So red for example is represented as (1,0,0), green (0,1,0), and blue (0,0,1). A color ramp can be illustrated as in figure 10.7. The edge length of the box is one hereby limiting each RGB component range to be between 0 and 1. The black emphasized line in the figure illustrates the hot-to-cold color ramp. All values are mapped to this line hereby being mapped a scalar value to three vector components.

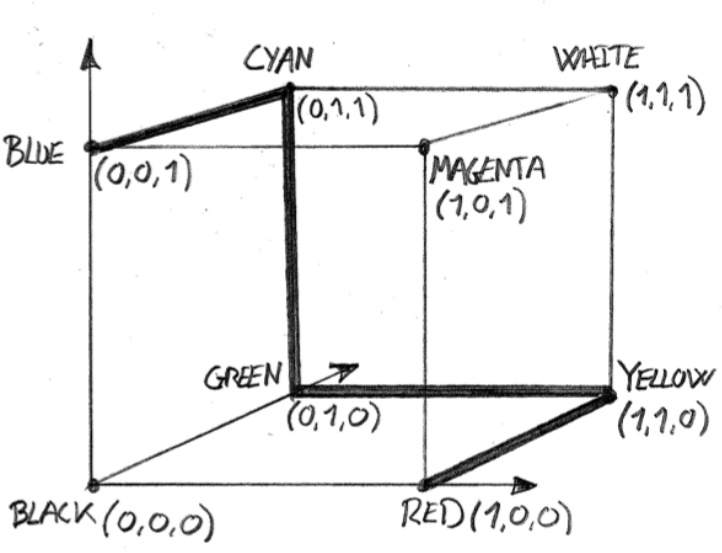



Figure 10.7: Illustration of how to interpolate color values for the hot-to-cold color ramp.

The principal stress is mapped to at color within the range of the material dependent tensile strength ($\sigma_F^+$) and compressive strength ($\sigma_F^-$). Zero stress is green, stress above $\sigma_F^+$ is clamped to blue and stress below $\sigma_F^-$ to red. Two linear interpolation functions are used in between, one for stretch and one for compression. Each interpolation function is determined by the material properties entered.

### Visualizing the Eigenvectors

The shape of the visual entity is important, it has to clearly illustrate the direction and location of the stresses without requiring too many resources. A relatively simple visual entity must be used since the size of the tensor field grows linearly with the number of visual elements. Figure 10.8 illustrates the shape of the visual entity we constructed. It is suitable for visualizing the directions of the stress tensors due to its simple representation and the obvious orientation.

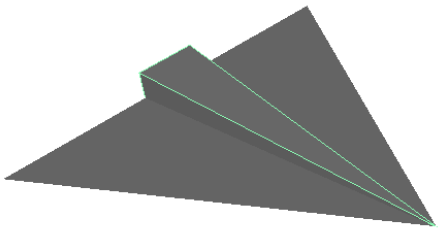


Figure 10.8: A visual entity shaped like an airplane.

The nose of the airplane is pointing in the maximum principal direction and the wings are aligned the medium principal direction. Any polygon model can be used for visualizing the stress tensors, but since the absolute coordinates of the visual entity for every tensor is being represented the memory usage quickly becomes an issue. The visual entity for each stress tensor is represented separately to facilitate parallel execution of the matrix calculations necessary for orienting and locating the geometry in space. By doing so the calculations and the performance of the visualization is improved at the expense of an increased memory consumption.

The first step is to load the preferred geometry that is going to be used as the visual entity. The visual entity is represented by triangular faces, each with three points. Each point is inserted into the model vertex buffer $V$ as a column vector:

$$V = \begin{bmatrix} v_x^0 & v_x^1 & & v_x^n \\ v_y^0 & v_y^1 & \cdots & v_y^n \\ v_z^0 & v_z^1 & & v_z^n \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{10.1}$$

where $n$ is the number of vertices in the model. The airplane as illustrated in figure 10.8 is constructed from seven triangles each represented by three distinct vertices, which gives a total of 21 vertices. Each vertex is of type `float4` which occupy 16 bytes each (therefore a zero row in $V$). By default the airplane model representation takes up 336 bytes of memory. To improve the visualization we take the lighting conditions into account by calculating the correct normal

vectors for each polygon. The normal vectors are represented by matrix $N$ in the same way as the vertices in matrix $V$:

$$N = \begin{bmatrix} n_x^0 & n_x^1 & & n_x^n \\ n_y^0 & n_y^1 & \cdots & n_y^n \\ n_z^0 & n_z^1 & & n_z^n \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{10.2}$$

No matter how many stress tensors we are about to visualize matrices $V$ and $N$ are only represented once in the memory. But since each tensor has its own orientation and position in space we need to construct a transformation matrix for each tensor. Each stress tensor is represented as a $3 \times 3$ symmetric matrix from which we construct a $4 \times 4$ transformation matrix. The transformation matrix is an affine matrix representing both rotation and translation as defined in section 4.1 on page 39. The rotation matrix must align its x- and y-axis with the maximum and medium principal direction of the stress tensor, respectively. As explained in section 4.2 on page 47 finding the principal values and directions means solving the eigenproblem for the stress tensor. The solution to the eigenproblem for an $n \times n$ symmetric matrix is $n$ eigenvalues and $n$ mutually perpendicular eigenvectors. By normalising each eigenvector we obtain an orthogonal set of unit vectors also known as an *orthonormal basis*. The three vectors in the orthonormal set represent the base vectors that span the vector space oriented according to the principal directions. From the orthonormal basis we can construct an affine linear rotation matrix simply by inserting the eigenvectors into a transformation matrix $T^e$. Assuming that the nose of the airplane points in the positive x-axis the vector representing the maximum principal direction is used as the base vector defining the x-axis. The maximum principal direction is inserted as row vector $[e_{11}, e_{12}, e_{13}]$ in matrix $T^e$, $[e_{21}, e_{22}, e_{23}]$ and $[e_{31}, e_{32}, e_{33}]$ become the medium and minimum principal direction, respectively. The center of mass for each tetrahedral element is used as the position of the geometry and inserted as the fourth column vector $[t_x^e, t_y^e, t_z^e]^T$ hereby adding translation to matrix $T^e$ which becomes:

$$T^e = \begin{bmatrix} e_{11} & e_{12} & e_{13} & t_x^e \\ e_{21} & e_{22} & e_{23} & t_y^e \\ e_{31} & e_{32} & e_{33} & t_z^e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10.3}$$

$T^e$ is constructed for each stress tensor. All together this forms the *matrix array* $T$, which contains a $4 \times 4$ transformation matrix $T^e$ for each element $e$:

$$T = \left[ \left[ T^0 \right] \left[ T^1 \right] \left[ T^2 \right] \cdots \left[ T^e \right] \right] \tag{10.4}$$

By applying the individual transformation matrices to the vertex buffer $V$ the geometry of the model is rotated and translated into the absolute positions. The transformation matrix $T^e$ is applied to each vertex in buffer $V$:

$$V_i^e = T^e \; V_i \tag{10.5}$$

where $V_i$ is the $i^{th}$ column vector in $V$ in equation (10.1). On matrix form equation (10.5) becomes:

$$\begin{bmatrix} V^e_{i,x'} \\ V^e_{i,y'} \\ V^e_{i,z'} \\ 0 \end{bmatrix} = \begin{bmatrix} e_{11} & e_{12} & e_{13} & t^e_x \\ e_{21} & e_{22} & e_{23} & t^e_y \\ e_{31} & e_{32} & e_{33} & t^e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_{i,x} \\ V_{i,y} \\ V_{i,z} \\ 0 \end{bmatrix} \tag{10.6}$$

Each vertex in the model, is transformed for each element and stored in buffer $V'$ representing the absolute coordinates ready for visualization:

$$V' = \left[ \left[ V^0 \right] \left[ V^1 \right] \left[ V^2 \right] \cdots \left[ V^e \right] \right] \tag{10.7}$$

The normal vectors are transformed the exactly same way and stored in the $N'$ array:

$$N' = \left[ \left[ N^0 \right] \left[ N^1 \right] \left[ N^2 \right] \cdots \left[ N^e \right] \right] \tag{10.8}$$

The matrix operation of applying the transformation matrix $T^e$ to each vertex point $V^e_i = [x, y, z]$ for each element $e$ can be carried out separately which makes it suitable for parallel execution. The following example illustrates the $10 \times 10 \times 10$ test beam with 964 tetrahedra as described in appendix B. If the airplane, with its 21 vertices, is used as the visual entity for each of the 964 tensors about to be visualized, the location and orientation of the entity is determined by a total of $964 \times 21 = 20244$ threads. If normal vectors are included we have twice as many threads.

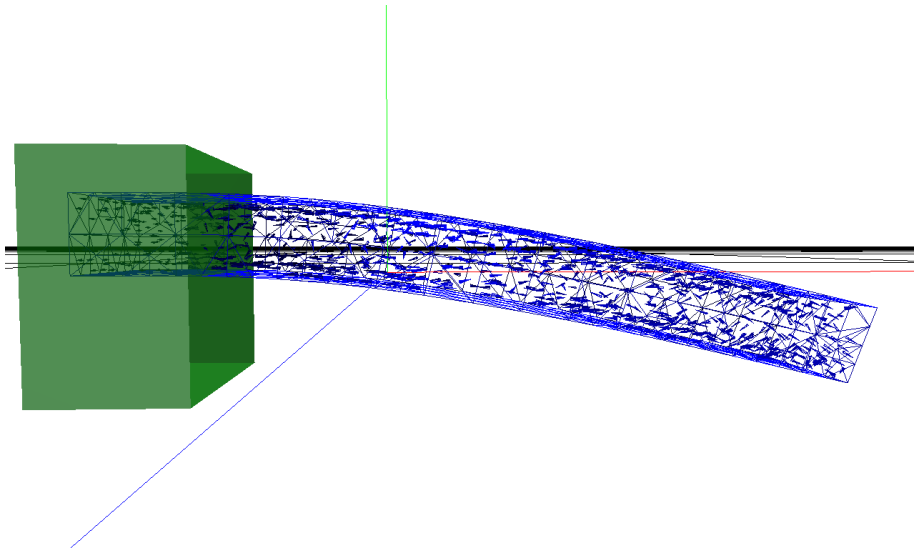


Figure 10.9: Beam with tensor visualization.

It can be difficult to interpret the tensor visualization especially from a distance as illustrated in figure 10.9 where the scenario is a beam fixed in one end with external forces pulling it down at the other end. But a closer look often reveals a pattern as illustrated on figure 10.10 on the following page. Notice that we cannot determine the sign of the maximum principal stress direction, this explains the opposite directions of the visual entities. The principal directions of the stress tend to point in a horizontal direction in the top and bottom of the beam which corresponds to the tension and compression in these directions. In the middle of the beam the

stress directions seem undetermined which in fact is the case when we consider the magnitude of the stress alternating around zero in this area.

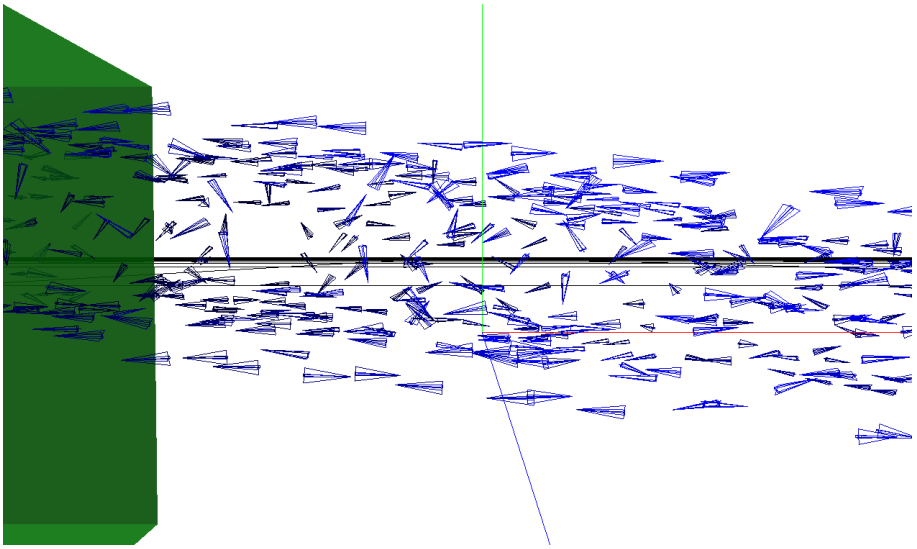


Figure 10.10: A closer look of the tensor visualization reveals a pattern.

# Chapter 11

# Results

In this section we will present and discuss the results obtained from conducting a series of different test scenarios. The test scenarios can be subdivided into two groups. Test scenarios in the first group are aimed towards testing general capabilities of the simulation model, such as performance and scalability. In the second group of test scenarios focus is only on evaluating the crack tracking strategy. The evaluation of the crack tracking strategy is primarily done through visual inspection. Since it is very difficult, if not impossible, to predict exactly how an object would break in the real world, we have to rely on our intuition of how objects should behave. As mentioned in the introduction to this thesis, one of the critera of success is to produce plausible results. If the outcome of a simulation scenario is consistent with the intuition we consider the result plausible. We have tried to construct the test scenarios in a way that should induce an obvious intuition for the expected outcome. In order to reproduce the following results obtained see appendix E.2 for instructions.

## 11.1 Scalability

The first test is conducted to see how the simulation time scales with the size of the problem domain. The execution time for a single iteration of the physics module is highly dependent on the geometry, but independent of the material parameters being used. To illustrate the relation between the execution time of the physics module and the size of the problem domain we have measured the time it take to perform $10^6$ iterations with different models varying in number of body elements. The average execution time for a single iteration recorded and plotted as a function of the number of elements. The results obtained are listed in table 11.1 on the next page. The test was conducted on the performance desktop setup described in appendix D using the models described in appendix B. The execution time was measured using the high precision timer (`Utils::Timer`) from the OpenEngine framework.

| quantity | # nodes | # tetrahedra | $\Delta t_{sim}$ |
|---|---|---|---|
| tetrahedron | 4 | 1 | 28.97 |
| bar $20 \times 20 \times 20$ | 113 | 263 | 36.85 |
| box | 14 | 17 | 36.91 |
| bar $10 \times 10 \times 10$ | 373 | 964 | 39.34 |
| tooth (simple) | 371 | 1168 | 42.60 |
| tooth with 33% slice (simple) | 483 | 1652 | 52.78 |
| test tooth | 646 | 1902 | 56.26 |
| tooth | 2264 | 667 | 61.09 |
| tooth with 33% slice | 3796 | 1049 | 96.24 |
| bar $5 \times 5 \times 5$ | 1619 | 4658 | 137.57 |
| sphere | 119 | 2393 | 256.11 |
| test tooth high resolution | 2763 | 8819 | 262.38 |
| bar $2.5 \times 2.5 \times 2.5$ | 6297 | 21809 | 726.29 |

Table 11.1: Simulation time in microseconds per iteration for different models.

Figure 11.1 shows the time measurements from table 11.1, plotted as a function of the number of elements.

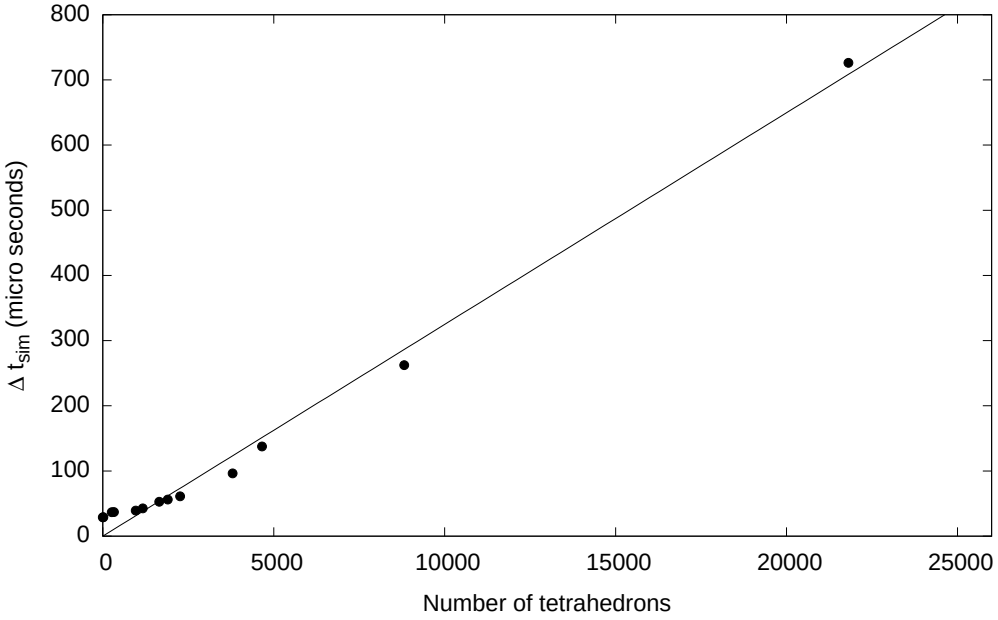



Figure 11.1: Graph of the mesh size contra simulation time.

Using the best-fit tool from GNU Plot[1] the relation between simulation time and number of elements can be approximated with a straight line ($y = \alpha\ x$), where $\alpha \approx 0.032$ with a standard asymptotic error of 2.619%. The graph clearly shows that we have a linear dependence between

[1] `http://www.gnuplot.info/`

the geometry size and the time it takes to execute an iteration of the physics module. We expect this linear dependence to hold until the mesh data exceeds the memory limit of the graphics card.

## 11.2 Real-time Analysis

As stated in section 8 on page 102, we have a stability requirement on the governing equation. The requirement imposes a critical time step limit, denoted $\Delta t_{cr}$, which must be met for the system to be stable. This means that the times step we use to extrapolate the physics module must be smaller than $\Delta t_{cr}$, for the physics module to be stable. If we also want the physics module to run in real-time, we must compute one iteration of the physics faster than $\Delta t_{cr}$. The conclusions is: If we want the physics module to be stable and run in real-time, then we must have: $\Delta t_{sim} \leq \Delta t_{cr}$. From the critical time step $\Delta t_{cr}$ calculated by equation (8.1) on page 103 we obtain:

$$\Delta t_{cr} = \sqrt{\frac{1}{c^2}} L_e = \alpha L_e$$

$$\frac{1}{c^2} = \frac{\rho}{M} \qquad M = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$$

As seen in the equations $\Delta t_{cr}$ is a function of the three material parameters: $E$, $\nu$, and $\rho$. $\Delta t_{cr} = \alpha L_E$ describes $\Delta t_{cr}$ as a function of the minimum edge length $L_e$ in the model. In the following example we use the material parameters for dentin. Teeth are primarily made of dentin which basically consists of calcium, phosphorus and mineral salts. Dentin is considered a brittle material since it has very little tendency to deform before fracturing. By calculating the P-Wave modulus $M$, and $\alpha$ for dentin, where $E = 12\ GPa$, $\nu = 0.32$, and $\rho = 2580\ kg/m^3$ as listed in section B.1 on page 153 in appendix B, we obtain:

$$M = \frac{12 \cdot (1 - 0.32)}{(1 + 0.32)(1 - 2 \cdot 0.32)} \approx 17.17\ GPa \tag{11.1}$$

$$\frac{1}{c^2} = \frac{\rho}{M} = \frac{2580 \frac{kg}{m^3}}{\frac{1700}{99} G \frac{N}{m^2}} \approx 150 \cdot 10^{-9} (\frac{s}{m})^2 \tag{11.2}$$

$$\alpha = \sqrt{\frac{1}{c^2}} \approx 387 \cdot 10^{-6} \frac{s}{m} \tag{11.3}$$

Where the units are converted by the following definitions:

$$N = \frac{kg \cdot m}{s^2} \qquad Pa = \frac{N}{m^2} \tag{11.4}$$

To confirm the dimensions of the tooth, we borrowed an actual wisdom tooth from the School of Dentistry, Aarhus University, and modeled this as a mesh in millimeters. An axis aligned bounding box around the tooth has the dimensions: width × height × depth, $10.7 \times 20.1 \times 10.1$. Calculating $\Delta t_{cr}$ for the tooth model, which has a $L_e = 0.14\ mm = 0.14 \cdot 10^{-3}\ m$:

$$\Delta t_{cr} = \alpha \cdot L_e = 387 \cdot 10^{-6} \cdot 0.14 \cdot 10^{-3} \approx 55.6 \cdot 10^{-9} s \tag{11.5}$$

which means time progresses with the size of 55.6 nanoseconds in each iteration. The constant execution time $\Delta t_{sim}$ for this model, was measured in section 11.1 on page 127 to be $\Delta t_{sim} = 42.6$ microseconds. This means it takes much more computational time to predict the next configuration

at time $t + \Delta t$ that the actual time step $\Delta t$ represents. In other words if it takes 20 seconds to foresee 10 seconds into the future you will always be behind. This illustrates that with the given object and material properties we cannon achieve real-time performance which strictly complies with the theoretical laws of physics.

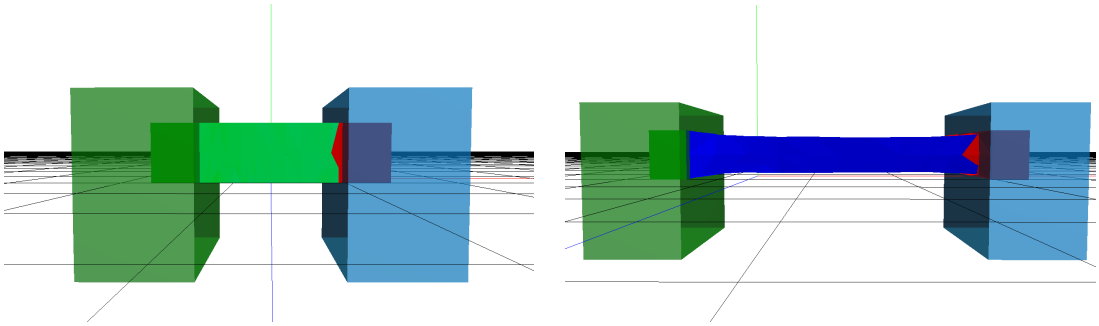
## 11.3 The Elasticity Theory

In this test scenario we are interested in comparing the simulated deformation with the theoretical deformation calculated by hand. The scenario is a beam fixed in one end and stretched by a displacement modifier in the other end. The direction of the stretch is parallel to the x-axis.

As explained in section 3.3 on page 30 elastic deformation is the physical property of a material when it deforms due to external forces applied, but returns to its original shape when the forces are removed. The modulus of elasticity $E$ is used for defining the elastic property of a given material. The modulus of elasticity is defined by (equation 3.20 on page 33), repeated below:

$$E = \frac{\text{normal stress}}{\text{normal strain}} = \frac{\sigma_x}{\varepsilon_x} \qquad \Leftrightarrow \qquad \varepsilon_x = \frac{\sigma_x}{E}$$

As mentioned in section 7.6 on page 89 we use the Green-Lagrange strain measure ${}^t_0E$ with the second Piola-Kirchoff stress measure ${}^t_0S$ instead of normal stress and strain. In section 3.4 on page 32 we assumed that the relation between stress and strain is linear since brittle materials are of our main concern. Let us start by measuring the actual relation between stress and strain to see if the linear assumption is correct. The following experiments are all conducted with a $20 \times 40 \times 150$ millimeter beam and $E = 12\ GPa$ corresponding to dentin. The internal stresses and strains are caused by the stretching alone, no other external forces are applied. The setup is illustrated in figure 11.2. Notice how the beam becomes thinner according to Poisson's ratio as explained in section 3.4 on page 33.



(a) (b)

Figure 11.2: Beam is being stretched to conduct stress-strain measures.

While the beam is being slowly stretched we measure the Green-Lagrange strain tensor as defined in equation (7.33) on page 92:

$${}^t_0E = \frac{1}{2}({}^t_0C - I)$$

By using the technique described in section 4.2 on page 47 we determine the maximum principal strain value by solving the eigenproblem for the symmetric tensor ${}^t_0E$. By applying axial stretch

along the positive x-direction in the global coordinate frame, the maximum principal strain direction will also be oriented this way.

We will now construct a stress-strain curve as the ones introduced in 3.3 on page 29. It is the maximum principal strain value we are interested in relating to the maximum principal stress value. The second Piola-Kirchoff stress measure is used as defined in equation (7.35) on page 93, repeated below:

$$ {}^{t}_{0}S_{ij} = \mu(\delta_{ij} - {}^{t}_{0}C^{-1}_{ij}) + \lambda \, {}^{t}J \, ({}^{t}J - 1) \, {}^{t}_{0}C^{-1}_{ij} $$

Using a similar approach as with the strain value, we solve the eigenproblem for the symmetric stress tensor ${}^{t}_{0}S$ hereby determining the maximum principal stress value. The stress and strain tensors are defined for each tetrahedral element in the body. A random element located somewhere near the center of the body has been pre-selected. Once the simulation starts the stress and strain measurements are conducted every $50^{th}$ iteration. The result is illustrated in figure 11.3.

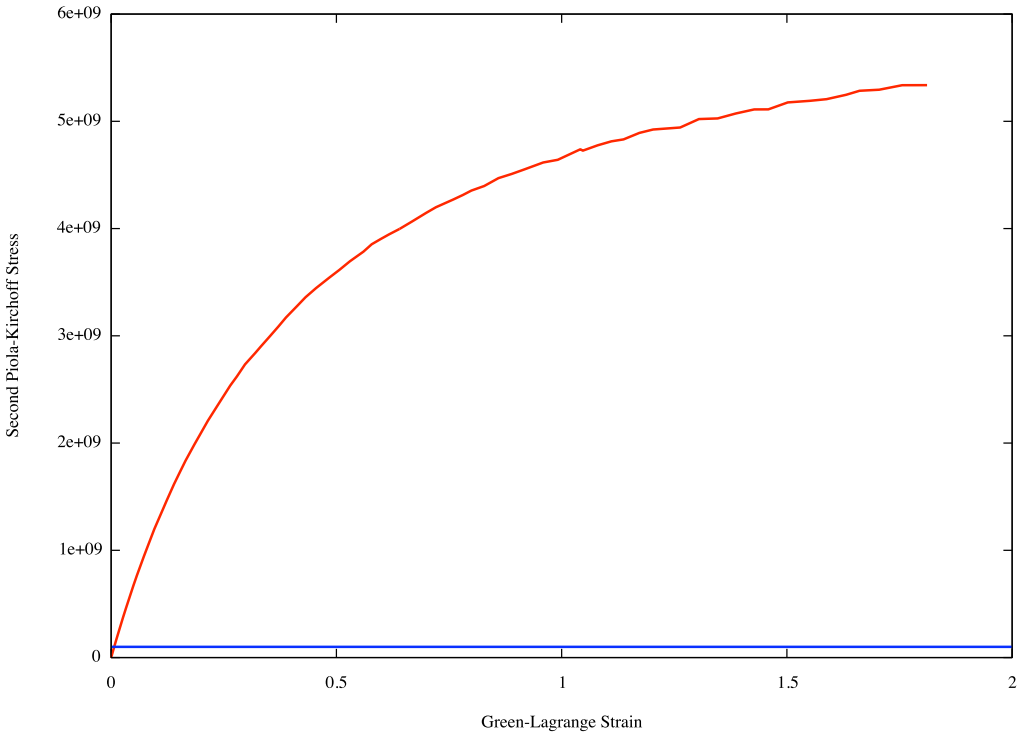



Figure 11.3: Relation between Green-Lagrange strain and second Piola-Kirchoff stress.

The relation is obviously non-linear, but as seen on the x-axis the Green-Lagrange strain measure reaches 1.5. The Green-Lagrange strain measure is non-linear so physically a measure of 1.5 means the beam has been stretched twice its original length. A brittle material like dentin cannot be stretched this far since its tensile strength is approximately 100 $MPa$ ($\sigma_F^+ = 100 \times 10^6 \ Pa$) as described in appendix B. The fracturing limit is illustrated by the horizontal blue line in figure 11.3. The maximal strain at the point of fracture is obtained by equation (3.20):

$$\varepsilon = \frac{\sigma_F^+}{E} = \frac{100 \times 10^6}{12 \times 10^9} \approx 0.008333$$

Considering the 150 millimeter beam this corresponds to a maximal stretch of 1.25 millimeters before the point of fracture is reached. The theoretical elastic modulus $E$ is expressed as a linear relation between stress and strain. It is reasonable to use the non-linear relation between the Green-Lagrange strain and the second Piola-Kirchoff stress in correlation with the linear elastic modulus, because the non-linear relation is approximately linear below the point of fracture. Figure 11.4 illustrates how close to linear the simulated relation between stress and strain is.

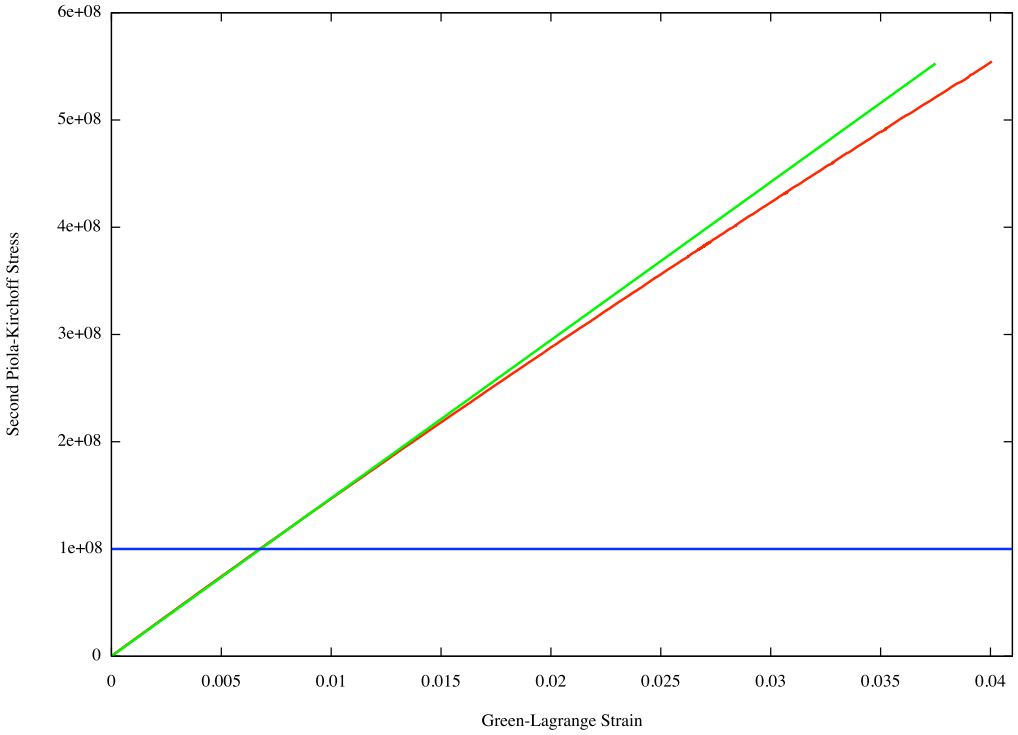


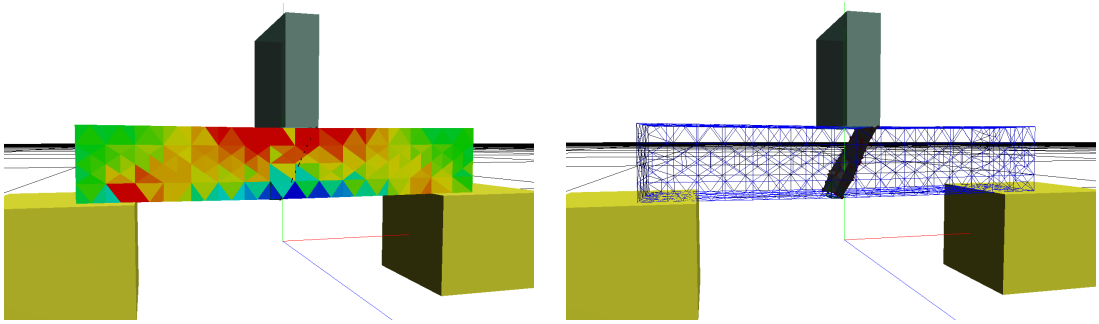

Figure 11.4: Below fracture point the simulated stress-strain relation is approximately linear.

Using the best-fit tool in GNU Plot the measured relation between stress and strain, below the point of fracture, can be approximated with a straight (green) line ($y = \alpha\ x$), where $\alpha = 1.4730 \times 10^{10}$ with a standard asymptotic error of 0.2%. Brittle materials are defined to be materials that can be model by a linear stress-strain relation (described in section 3.3 on page 32). As illustrated although we are using the non-linear neo-Hookean elasticity theory, we have an approximate linear dependence between stress and strain for brittle materials (dentin) below the fracture point.

## 11.4 Fragmentation of Supported Beam

This is the first scenario with the purpose of testing the crack tracking algorithm. The main purpose is to see if the crack tracking algorithm determines a failure surface according to our

intuition of how the concrete beam would break. The scenario is simple; a beam is located the top of two boxes supporting each end of the beam as illustrated in figure 11.5a. A third box is slowly being moved downwards hereby forcing on top of the beam down until the beam fractures. Concrete is considered a brittle material so the beam is expected to fracture vertically somewhere around its center where the internal forces will peak.



(a) (b)

Figure 11.5: Fragmentation of supported beam.

The beam bends as the force increases gradually causing the internal stress to peak around the center of the beam. As seen on 11.5a the beam is compressed on the top and stretched at the bottom peaking around the center. When the material dependent point of fracture ($\sigma_F^-$ or $\sigma_F^+$) is exceeded the crack is initialized and starts to propagate through the solid object. As illustrated in figure 11.5b the crack tracking algorithm determines a failure surface located around the center of the beam propagating vertically as expected. At the top of the beam the failure surface is located slightly to the right of where the external forces are applied. As expected the crack propagates towards the center of the beam here creating a slightly curved failure surface. The point of origin and the curvature depends on the amount of force being applied and especially the rate at which they are applied.

## 11.5 Mesh Independent

When it comes to approximating the shape of an object the number of elements determines the quality of the approximation. As explained in section 6.3 on page 66 any solution domain with curved boundaries can be approximated by a series of straight lines or flat planes. When discretizing a complex shaped object we must choose an appropriate element size in order to make a reasonable approximation. This is not the case with simple shaped objects like a beam. It is a rectangular box which can be discretized by a minimum of five tetrahedra as illustrated in figure 7.2 on page 84. Discretizing a beam into five elements would probably not be advisable due to the quality of the failure surface. A minimum of five elements are enough for approximating the shape but not for a reasonable approximation of the failure surface. In general we are interested in how the number of elements, hence the quality of the mesh, affects the outcome of the crack tracking algorithm. For benchmark purposes we have constructed four beams with the exact same dimensions but with gradually increasing number of elements as described in section B.2 on page 154 in appendix B. The beam is fixed in one end while external forces are applied at the other end. The external forces pulling down are uniformly distributed on the selected nodes (elements colored gray). Each beam setup is exactly the same, illustrated in figure 11.6 on the following page.

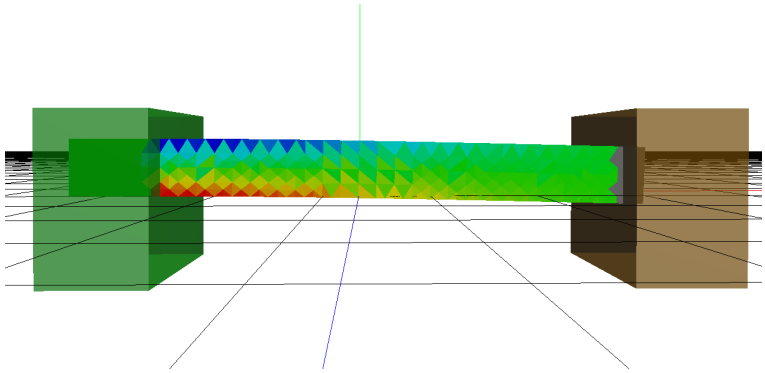
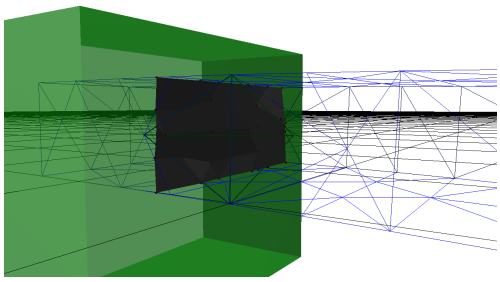
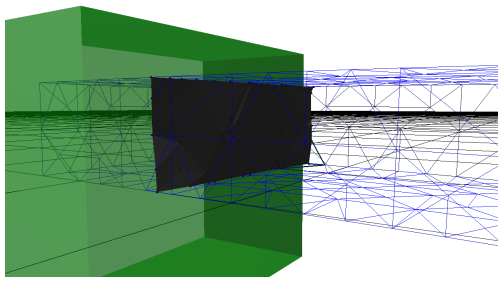


Figure 11.6: Benchmark setup for testing mesh dependency.

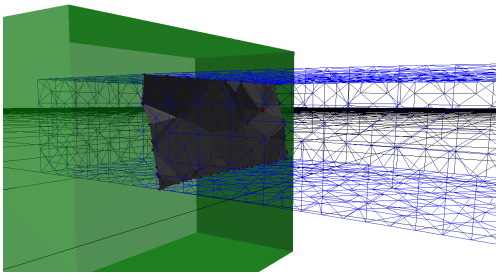
Each beam was tested one at the time and as seen on figure 11.7 the resulting failure surfaces determined by the crack tracking algorithm are fairly similar. In all four cases the point of origin of the crack was located slightly to the right of the supported area and the orientation of the initial crack plane was vertical. Although the four failure surfaces are slightly different they are all equally located and oriented. By increasing the mesh quality we also increase the number of crack planes and hereby the details in the continuous failure surface. As seen on figure 11.7d the failure surface is more curved than on figure 11.7a but the overall location and vertical curvature remains the same. In this scenario the crack tracking algorithm and the failure surface determined seems unaffected by the increasing quality of the mesh.



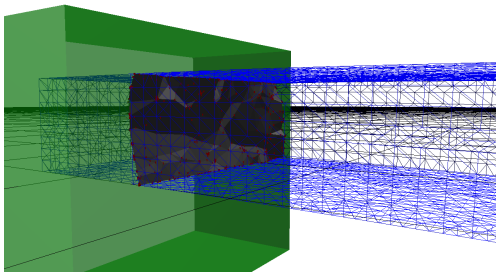
(a) Beam with 263 elements.



(b) Beam with 964 elements.



(c) Beam with 4658 elements.

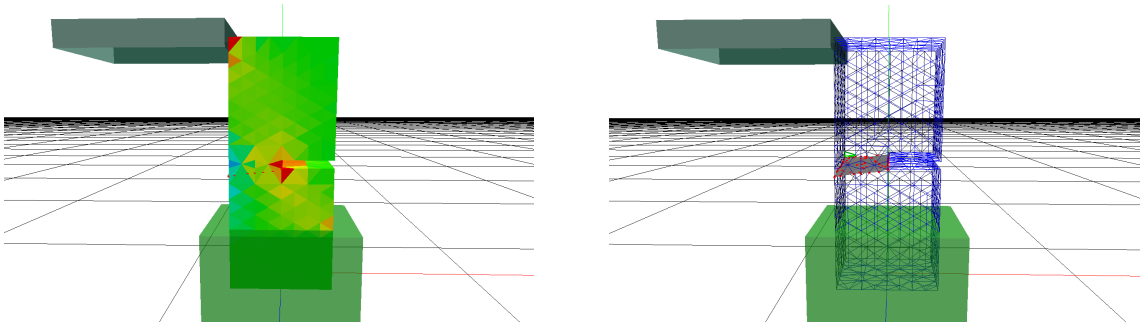


(d) Beam with 21809 elements.

Figure 11.7: Crack tracking with beam in different mesh qualities.

## 11.6 Fragmentation of Tooth Mock-up

The purpose of this scenario is once again to see if the crack tracking algorithm determines a failure surface according to our intuition. The solid object being used in this scenario is a mock-up of a tooth with a groove drilled halfway through. Due to the groove the object must be weakened in this area and is therefore expected to fracture somewhere close to this. The bottom of the tooth mock-up is fixed while forces are being applied gradually at the top pushing the object to the right. Again we use dentin as the material ($E = 12 \cdot 10^9$):



(a) (b)

Figure 11.8: Fragmentation of tooth mock-up.

As seen on figure 11.8a the modifier pushes the top of the tooth mock-up to the right, which causes compression near the groove. The pressure is gradually increased until the point of fracture ($\sigma_F^-$ or $\sigma_F^+$) is exceeded. The failure surface as seen on figure 11.8b started at the bottom of the groove and propagated to the left slightly curving downwards. The beam is fragmented around its weak midpoint near the groove as expected.

## 11.7 Fragmentation of Tooth

In this scenario a tooth model is fragmented using a simulated dental tool known as an elevator, as depicted in figure 2.5 on page 8. When the groove has been drilled in the tooth, the elevator is inserted into the groove and twisted in order to deform the brittle dentin material just enough to cause fracture. The tip of the elevator tool is actually slightly curved which makes it suitable for elevating both the crown and the roots once they are separated. In this scenario attention is only focused on fracturing the tooth and not on how to elevate the separated parts from the jawbone. The representation of the elevator tool is therefore simplified to a non-curved plane as illustrated in figure 11.9. The tooth model is fixed by its roots similar to the actual scenario. Performing the drilling of the groove is beyond the scope of this simulator, therefore the tooth model has a predefined groove with a depth that corresponds to 1/3 of the tooth's width.

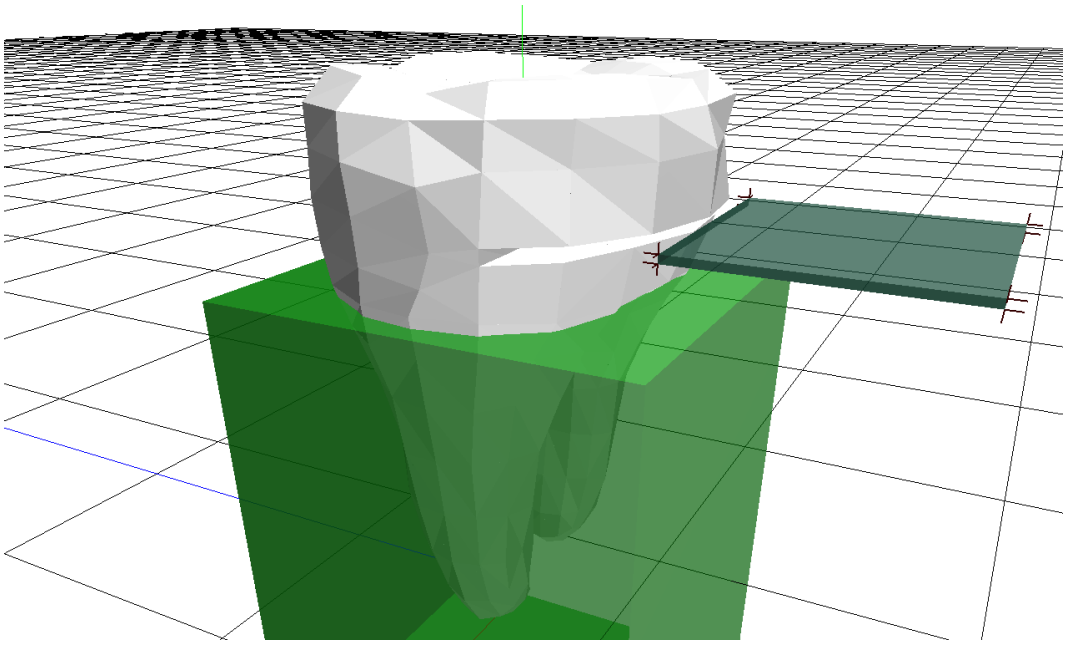


Figure 11.9: The tooth model with a predefined groove.

The elevator tool in inserted into the groove and slowly twisted until the dentin material has been deformed beyond its fracture point. External forces are applied in a somewhat different way in comparison with the other scenarios. Since the elevator tool is twisted all forces are applied to a relatively small area. When the tool is twisted the crown is being forced upwards, while the roots are being forced downwards. The twisting motion of the elevator tool causes maximum tension at the bottom of the groove as expected. As seen on figure 11.10a on the facing page the crack has its origin at the bottom of the groove. As the crack propagates in all directions from its origin, the orientation of the failure surface becomes clear. Due to the restriction on the roots the only part free to move is the crown, therefore it slightly tilts backwards as the groove is gradually being widened. The stress starts to point downwards on the opposite side of where the groove is located. This reaction explains why the failure surface tends to curve downwards eventually reaching the surface of the material slightly lower compared to the horizontal location of the groove. The curvature of the failure surface starts to emerge early in the propagation as

seen in figure 11.10c.

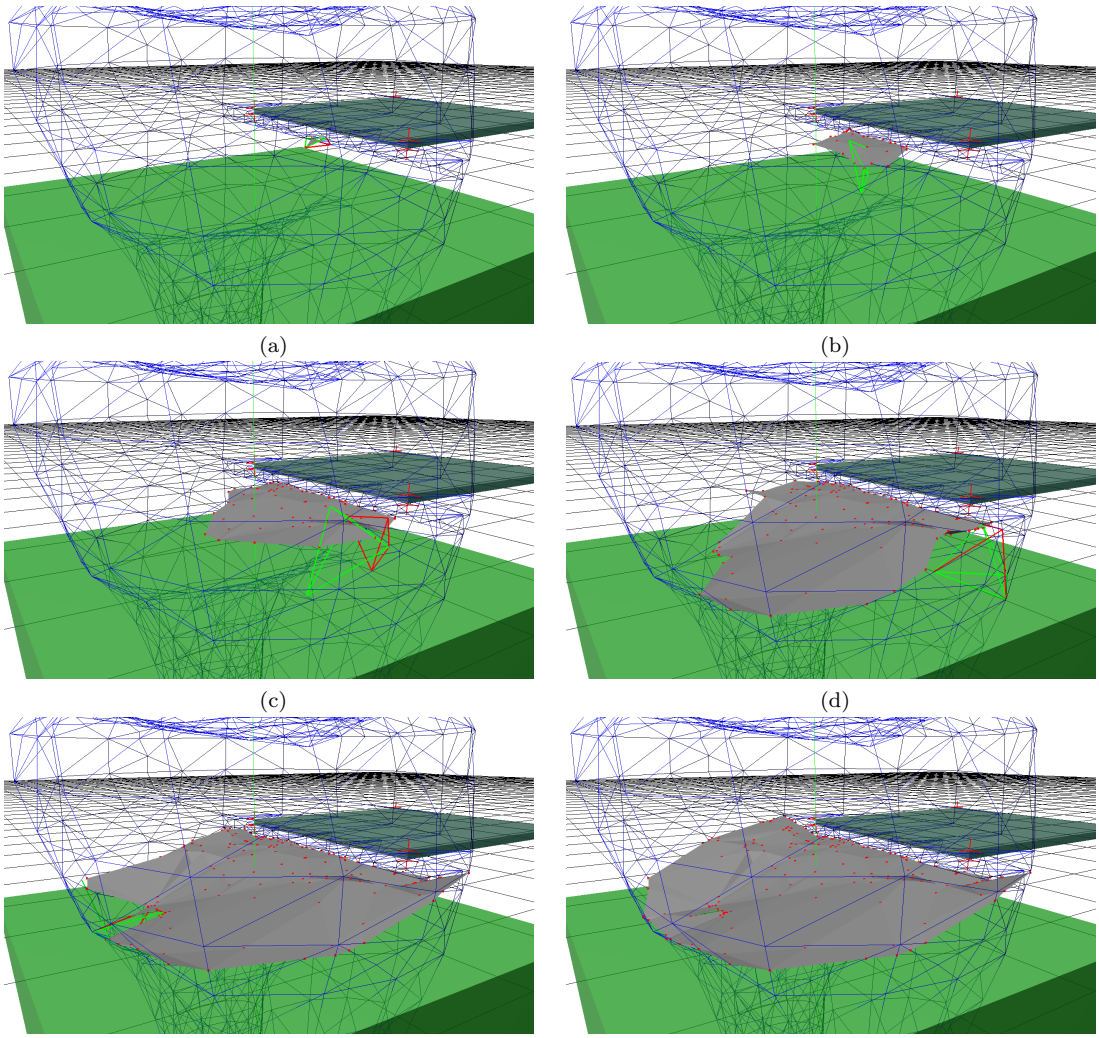


Figure 11.10: Crack propagation through tooth model.

## 11.8 Fragmentation with Varying Density

As illustrated in section 11.2 on page 129 we do not have the computational resources required to perform real-time simulation of the given model if we strictly comply with the theoretical laws of physics. Assuming the simulated model and the hardware used is unchanged, the computational time required for performing a single iteration ($\Delta t_{sim}$) is constant. The question is how can we increase the size of the simulated time step without violating the boundary condition that refrains the explicit time integration from numerical instability. We need to increase performance at the expense of accuracy.

$$\Delta t_{cr} = \sqrt{\frac{1}{c^2}} L_e = \alpha L_e$$

$$\frac{1}{c^2} = \frac{\rho}{M} \qquad M = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$$

Consider the equation for determining the critical simulated time step $\Delta t_{cr}$. We do not wish to change the geometrical representation of the model assuming it is reasonable, so $L_e$ is left unchanged. The other option is to change $\alpha$ which depends on $E$, $\nu$, and $\rho$. If possible we do not want to change the elastic modulus $E$ because this would affect the relation between stress and strain. As seen from the equations we would need to decrease the elastic modulus to obtain a larger critical time step hereby softening the material. Changing Poisson's ratio $\nu$ is not an option since this only effects the contraction ratio during deformation. With very small deformations the effect will be minimal. Changing the density $\rho$ is an option. By increasing the density by a factor of $10^6$ we obtain a critical time step size $\Delta t_{cr}$ of:

$$\frac{1}{c^2} = \frac{\rho}{M} = \frac{2580 \cdot 10^6 \frac{kg}{m^3}}{\frac{1700}{99} G \frac{N}{m^2}} \approx 150 \cdot 10^{-3} (\frac{s}{m})^2 \qquad \wedge \qquad \alpha = \sqrt{\frac{1}{c^2}} \approx 387 \cdot 10^{-3} \frac{s}{m} \tag{11.6}$$

$$\Delta t_{cr} = \alpha \cdot L_e = 387 \cdot 10^{-3} \cdot 0.14 \cdot 10^{-3} \approx 55.6 \cdot 10^{-6} s \tag{11.7}$$

With a constant computational time of $\Delta t_{sim} = 42.6 \cdot 10^{-6}$ and a critical time step $\Delta t_{cr} = 55.6 \cdot 10^{-6}$ the real-time condition if fulfilled ( $\Delta t_{sim} \leq \Delta t_{cr}$), but how will this effect the stress and strain behaviour and hereby the prediction of the failure surface.

The effect of increasing the density, besides obtaining a larger $\Delta t_{cr}$ time step, is increased mass and damping matrices in the system due to equation (7.43) on page 95 where the damping factor is determined from a constant factor $\alpha$ and the mass $M$ as $D_{ii} = \alpha M_{ii}$. The high-density object will have increased damping but the stress and strain behaviour seems unaffected. Recall that the rate of deformation before reaching the point of fracture is very low in brittle materials like dentin (below 1% as calculated in section 11.3 on page 130). Damping the very limited motion has almost no effect neither on the stress and strain nor visually. The failure surface determined in the high-density material corresponds to the failure surface determined in the material with the correct density. Although the simulated response of the material is not theoretically correct the failure surface in the high-density material is considered plausible and still in correspondence with the intuition. In the sense of physical interpretation the change in density is drastic and out of proportions. But in the sense of simulated response the prediction of the crack surface seems realistic. With the constant computational time of $42.6 \cdot 10^-6$ the stress and strain analysis is conducted about 20.000 times pr second facilitating real-time interaction and visualization.

The following test scenario illustrates how changing the density affect the outcome of the crack tracking algorithm. The illustrations to the left in figure 11.11 on the next page show the crack propagation in a tooth with the correct density simulated in somewhat slow-motion. The illustrations to the right are conducted with a density increase of $10^6$:

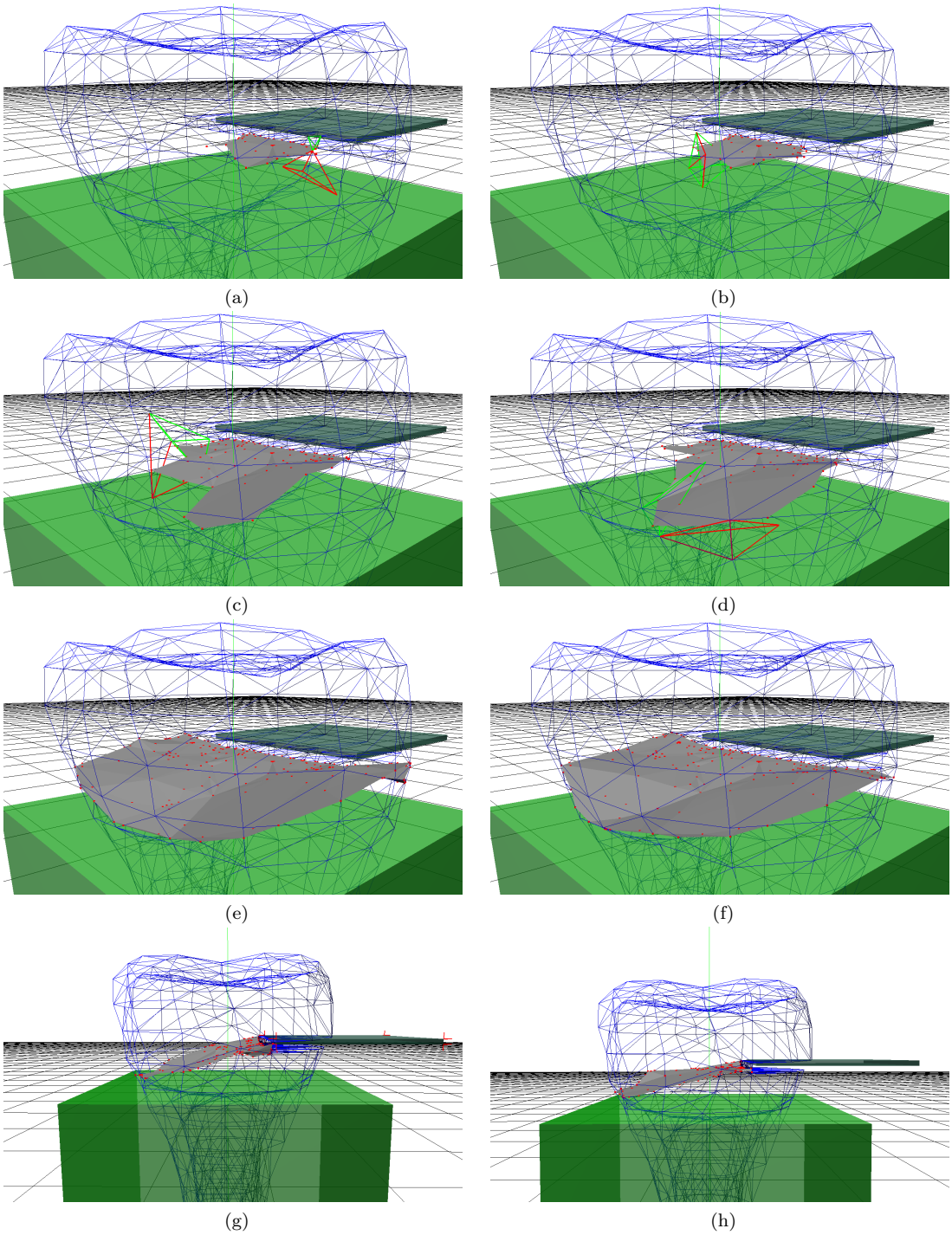


Figure 11.11: Crack propagation through normal contra high density material.

# Chapter 12

# Future Work

During the development of the simulation prototype lots of improvements came to mind. When considering the endless interdependent components of the final prototype, ideas, questions or improvements arise to each of the individual components. It seems like whenever we look into how things are connected and how changes would propagate though the system new challenges arise. In this section we have limited the discussion of any future work by only considering improvements of the prototype that will lead towards a simulator that can be evaluated by dentists. To form a decent evaluation basis for a dentist, improvements must be made. It would be very helpful to get evaluation feedback from a dentist especially on the sense of touch when the fragmentation is performed. Analysing the fragments of the tooth by comparing the simulated fragmentation to actual tooth fragments could be very interesting.

### Improving the precision on the pre-calculations

The pre-calculation of the shape function derivatives is based on the geometry of the elements. Currently the global position of the elements can cause significant floating point inaccuracy leading to inaccurate shape function derivatives. To minimize the error the shape function derivatives could be pre-calculated in local space coordinates instead of global. The geometry of the elements are represented by floating point numbers which have a relative representation error[Øst02, page 9]. So the error increases with the distance from origo of the global coordinate system. By translating the geometry of the elements into origo and scaling it to unity the error can be minimized.

### Improving how to apply forces

An essential part of the simulator is the interaction. To simulate a reaction something has to act on the system changing the state of equilibrium. As described in section 10.1 on page 115 all interaction is carried out through the modifiers. The Projective Displacement modifier is the modifier used for simulating interactive tools like the elevator from the test scenarios. As implied by its name, the modifier interacts with the system through nodal displacements. So when there is detectable collision between tool and object the colliding surface nodes of the object are displaced until they are free of collision. At the point of contact between tool and object all forces are applied through deformation of the object's surface. Using brittle materials even the slightest deformation causes a huge amount of internal stress which makes the interaction between tool and object difficult. When the tool touches the surface of the object in a way that

seems very gently on the screen, the small surface deformation is actually causing the object to fracture immediately. This would probably be the correct response if we did hit the object with a sharp tool using great force, but this is not the case here. Basically the problem is the method used when applying the forces as displacement. By displacing the surface elements at the point of contact the internal forces peak in a single iteration before they get the chance to propagate to neighbouring elements over multiple iterations. This has to do with the parallel approach used. Since each element is solved in parallel they only interact when nodal force contributions are added at the end of each iteration. Therefore forces needs to propagate through the elements over multiple iterations. A more gentle method for applying forces could improve the interaction between tool and object. The displacement could be gradually applied to the surface elements or distributed over multiple elements within a certain distance to compensate for the propagation.

### Improving the method for measuring internal stress

In relation to how forces are applied into the system the method for determining the maximum level of internal stress needs improvement. The current method determines the maximal level of stress by considering each individual element separately. If the internal stress peaks in a single element the fracturing limit is exceeded upon impact causing the object to start fracturing before the stress has propagated properly through the element mesh. Instead the maximum level of stress could be measured by interpolating the stresses from neighbouring elements hereby reducing peak values.

### Improving the crack tracking algorithm

In general we have obtained good results from the local crack tracking algorithm implemented. But there are improvements to be made. If the crack initialization has been initiated due to an internal stress peak in a single element the algorithm performs poorly. It performs poorly because the internal stress has not propagated properly from the point of impact and throughout the element mesh. As a result the magnitude of the principal stress values is alternating around zero. This causes the maximum principal stress direction to alter between the three principal directions in a more or less random manner. The crack tracking algorithm relies on the principal stress directions so as a consequence of the randomness the failure surface determination can 'spin out of control'. Since every element only can crack once and the number of elements is finite, the algorithm is deterministic at all times. But the crack propagation can in rare cases form a spiral like a winding staircase. This only occurs when the crack is propagating through parts of the body with internal principal stress values around zero and could be avoided by improving the stress measure as mentioned earlier or by improving the crack tracking algorithm to perform better with seemingly random stress directions.

### Re-meshing of the fractured body

When the crack tracking algorithm has determined a failure surface the intersecting elements should be separated accordingly. The prototype does not handle the actual separation of the fragments. To facilitate a proper evaluation of the size and shape of the fragments, separation has to be performed. How each element has been cracked is registered as crack points determining the intersection between the crack plane and the element edges. This representation is used for visualizing the crack surface and can be used to separate the element mesh.

### Support composite materials

A wisdom tooth consists of different materials, primarily dentin. The material properties of dentin are not the same throughout a tooth, different material subgroups like superficial, middle, and deep dentin have different material properties. Besides dentin a layer of enamel is covering the crown, this material is also divided into subgroups [GSdC04, page 324]. Currently the prototype only supports homogeneous materials which means that all elements have the same material properties. By implementing support for different material properties in each element, inhomogeneous and composite materials could be simulated. It would require extending the mesh file format, the internal data structures and making small changes to the finite element solving technique. It could be very interesting to evaluate and test how the crack tracking performs in composite materials.

### Improve tooth mesh

The tooth mesh used in the test scenarios is modelled by hand roughly shaped to form the real tooth borrowed. Since only the surface of the real tooth has been taken into consideration the model does not include any internal cavities. Instead a model could be created from a volumetric x-ray image by registering the surface curvature in each layer and assemble these to form a volumetric contour of the tooth which includes cavities. The internal cavities will effect the propagation of internal stress and strain making the tooth simulation more compliant with the actual scenario.

### Improving the graphical user interface

The prototype suffers from a lot of hard coded properties and setups. Few parameters, like which mesh and material should be used, can be passed as command-line arguments at start-up. Other than that, the code needs to be changed and recompiled. A graphical user interface capable of changing the simulated scenario at run-time is necessary if non-technical personal should evaluate the simulator.

### Support for haptic devices

To facilitate a proper evaluation of the simulated fragmentation process the input devices must be improved. Feedback from dentists on how the tools manoeuvre in space, how the twisting action of the elevator tool complies to the actual scenario etc. would be helpful. Using a standard keyboard and mouse is not sufficient if we are interested in evaluating the simulated response of user interactions. When using the simulated tool the user tends to lose the sense of how much force he or she actually applies to the object. Especially with brittle materials where even a very small deformation causes huge amounts of internal stress. Implementing support for a haptic device with force feedback would be necessary for the dentists to evaluate the sense of touch and manoeuvrability in the simulator.

# Chapter 13

# Conclusion

The main objective of this thesis was to construct a simulation model capable of conducting real-time structural analysis of stress and strain to be used for crack surface prediction.

For the structural analysis we constructed a simulation model based upon the finite element method and a solver based on the Total Lagrangian Explicit Dynamics technique aimed towards parallel execution. The finite element method with its pure style is so elegant and yet so complex it takes years to fully master. Once the method has been implemented as the fundamental framework for representing and solving the problem domain its true potential emerges. According to our experiences the method turns out to be very stable, predictable and beneficial when analysing the behaviour of stress and strain.

Recall from the problem statement that in the context of simulated surgery the demand for a robust model, real-time execution, and plausible results is favoured over accuracy. Challenges arise when simulating small objects like a human tooth made of a brittle material like dentin. When using an explicit solver we need to respect the boundary conditions on the size of the time step since this is crucial for the numerical stability of the method. The critical time step is effectively reduced when increasing the material stiffness or decreasing the smallest edge length in the object simulated. The real-time analysis conducted in section 11.2 on page 129 illustrates that with the given object size and material we cannot achieve real-time simulation which strictly complies with the theoretical laws of physics. If we compromise the theoretical laws of physics by increasing the material density of the tooth we are allowed to extrapolate simulation time by a larger time step hereby facilitating real-time execution. In the sense of physical interpretation the change in density, and hereby the external forces required for deformation, is out of proportions. But in the sense of simulated response the prediction of the crack surface looks very promising. The crack surface determined seems unaffected since the same material stiffness is used as elaborated in section 11.8 on page 137.

Benchmarking the simulation model with the fragmentation scenario as described in section 11.7 on page 136 reveals great potential. The system equations representing the tooth model can be solved approximately 20.000 times per second simulated on hardware as described in appendix D.1 without fine-tuning the CUDA implementation. This easily accommodates real-time interaction and visual feedback at real-time frame rates ($> 25$ FPS).

We have presented a local crack tracking algorithm based upon relevant theory from fracture mechanics in particular the principle of maximum stress direction. Though improvements can be made as explained in section 12 on page 141 we believe the method applied shows great potential

towards fracturing solid brittle materials. In general the failure surface as predicted by the crack tracking algorithm looks very promising. The location and curvature of the failure surface corresponds to the stress analysis and the intuition of how the object would actually fracture.

Using this crack tracking scheme in direct correlation with the stress and strain analysis conducted did cause a few shortcomings. As pointed out in section 12 on page 141 the approach used for applying forces and measuring stress needs improvement to obtain a less sensitive interaction. To improve this method any further we need feedback on how actual teeth fracture in the given surgical scenario. A thorough comparison of simulated contra actual tooth fragments, requires research beyond the scope of a single master thesis.

# Part IV

# Appendices

# Appendix A

# Hooke's Law in Three Dimensions

This section provides the general relationship between stress and strain for elastic material in three dimensions on matrix form. The general relationship between stress and strain for a homogeneous material can be expressed by a $3 \times 3 \times 3 \times 3$ symmetric fourth-order tensor with 81 components, this tensor is called the *material tensor* and denoted $C$. But because both the stress and strain tensors are symmetric $3 \times 3$ matrices the material tensor reduces to a $6 \times 6$ tensor (matrix) with 36 components for general anisotropic material, this matrix is called the *material matrix* [Str86, page 220]. When considering isotropic materials only 21 of the 36 components differ because of symmetry [HDSB01, page 659-661]. Using vectorization of the stress and strain matrices and Voigt notation for the material matrix, the relationship looks as follows:

$$\begin{Bmatrix} \sigma \\ \tau \end{Bmatrix} = C \begin{Bmatrix} \varepsilon \\ \gamma \end{Bmatrix} \tag{A.1}$$

or on matrix form:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{Bmatrix} = \begin{bmatrix} C_{11} & \cdots & \cdots & \cdots & \cdots & C_{16} \\ \vdots & \ddots & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & \ddots & & \vdots \\ \vdots & & & & \ddots & \vdots \\ C_{61} & \cdots & \cdots & \cdots & \cdots & C_{66} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{Bmatrix} \tag{A.2}$$

Depending of if we use linear or non-linear elasticity theory, the material matrix will look different, here we present the material matrix for linear elasticity. Recall equation (3.23) from section 3.4 on page 33 relating normal stress and normal strain, repeated here for convince.

$$\varepsilon_x = \frac{1}{E} \left[ \sigma_x - \nu \left( \sigma_y + \sigma_z \right) \right]$$

$$\varepsilon_y = \frac{1}{E} \left[ \sigma_y - \nu \left( \sigma_x + \sigma_z \right) \right]$$

$$\varepsilon_z = \frac{1}{E} \left[ \sigma_z - \nu \left( \sigma_x + \sigma_y \right) \right]$$

Rewriting this on matrix form the normal strain relation looks as follows:

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{bmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{bmatrix} \tag{A.4}$$

For the shearing stress and strain we recall equation (3.26) from section 3.4 on page 33:

$$\gamma_{xy} = \tau_{xy}/G \qquad \gamma_{yz} = \tau_{yz}/G \qquad \gamma_{zx} = \tau_{zx}/G$$

and equation (3.25) relating $G$ and $E$

$$G = \frac{E}{2(1+\nu)}$$

and by substituting $G$ the equations becomes:

$$\gamma_{xy} = \tau_{xy}\frac{2(1+\nu)}{E} \qquad \gamma_{yz} = \tau_{yz}\frac{2(1+\nu)}{E} \qquad \gamma_{zx} = \tau_{zx}\frac{2(1+\nu)}{E} \tag{A.5}$$

on matrix form, it looks like this:

$$\begin{bmatrix} \gamma_x \\ \gamma_y \\ \gamma_z \end{bmatrix} = \frac{1}{E} \begin{bmatrix} 2(1+\nu) & 0 & 0 \\ 0 & 2(1+\nu) & 0 \\ 0 & 0 & 2(1+\nu) \end{bmatrix} \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{A.6}$$

from the two matrices we assemble the inverse material matrix $D$.

$$D = C^{-1} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ -\nu & 1 & -\nu & 0 & 0 & 0 \\ -\nu & -\nu & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2(1+\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & 2(1+\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(1+\nu) \end{bmatrix} \tag{A.7}$$

the material matrix is then:

$$C = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{A.8}$$

Remark that if we should construct the material matrix for the non-linear elasticity theory, then $C$ would be a function of strain [HDSB01, page 659-662]

# Appendix B

# Test Data

## B.1 Materials

Table B.1 lists the material properties that were used during testing. The material properties are measured at room temperature and at 1013.25 hPa (1 atm) pressure.

| name<br>unit | $E$<br>GPa | $\nu$ | $\rho$<br>$kg/m^3$ | $\sigma_F^+$<br>MPa | $\sigma_F^-$<br>MPa |
|---|---|---|---|---|---|
| concrete | 41 | 0.21 | 2400 | 5 | 40 |
| dentin | 12 | 0.32 | 2580 | 100 | 250 |

Table B.1: Data for test materials.

When $E$ is the elasticity modulus, $\nu$ is Poisson's ratio, $\rho$ the density, $\sigma_F^+$ the tensile strength, and $\sigma_F^-$ the compressive strength. The data for concrete was found on the web[1]. The data for dentin was found in the following references: $\nu = 0.32$ [KMM03, page 19], $\rho = 2580kg/m^3$ [BH98, page 25], $E = 12GPa$ [BH98, page 29], $\sigma_F^+ = 100GPa$, and $\sigma_F^- = 250GPa$ [APM01, page 71].

## B.2 The Meshes

We have three different kind of objects that we want to test. The first kind are simple objects, these objects are small models that can be used to produce a minimal amount of output data, which makes analysis easier. The second object kind is called a bar, which is a rectangular object with dimensions $160mm \times 40mm \times 20mm$. The third kind of objects are teeth. These are tightly coupled to the surgical scenario that we are trying to simulate.

[1] `http://www.engineeringtoolbox.com/concrete-properties-d_1223.html`

### The simple objects

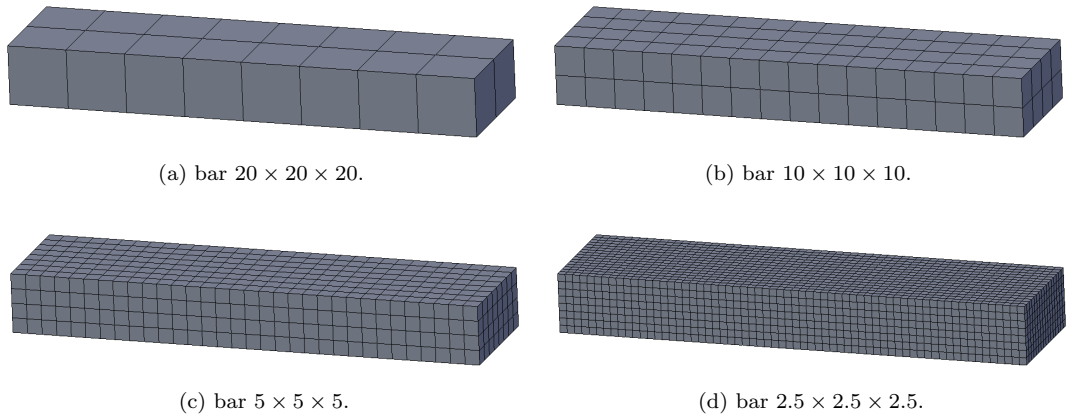
Table B.2 described the simple objects we have been using.

| name | # nodes | # surface triangles | # body tetrahedra |
|---|---|---|---|
| tetrahedron | 4 | 4 | 1 |
| box | 14 | 24 | 17 |
| sphere | 119 | 212 | 328 |

Table B.2: Data for the simple meshes.

### The Bar Object

To model the bar object, we have four meshes with different resolutions. The models have been constructed using the Blender modelling tool, which is a tool for constructing surfaces meshes. The models have been constructed as quadratic surface elements in Blender by subdividing the bar as shown in figure B.1 for the four meshes.



(a) bar $20 \times 20 \times 20$.

(b) bar $10 \times 10 \times 10$.

(c) bar $5 \times 5 \times 5$.

(d) bar $2.5 \times 2.5 \times 2.5$.

Figure B.1: The four different bar resolutions.

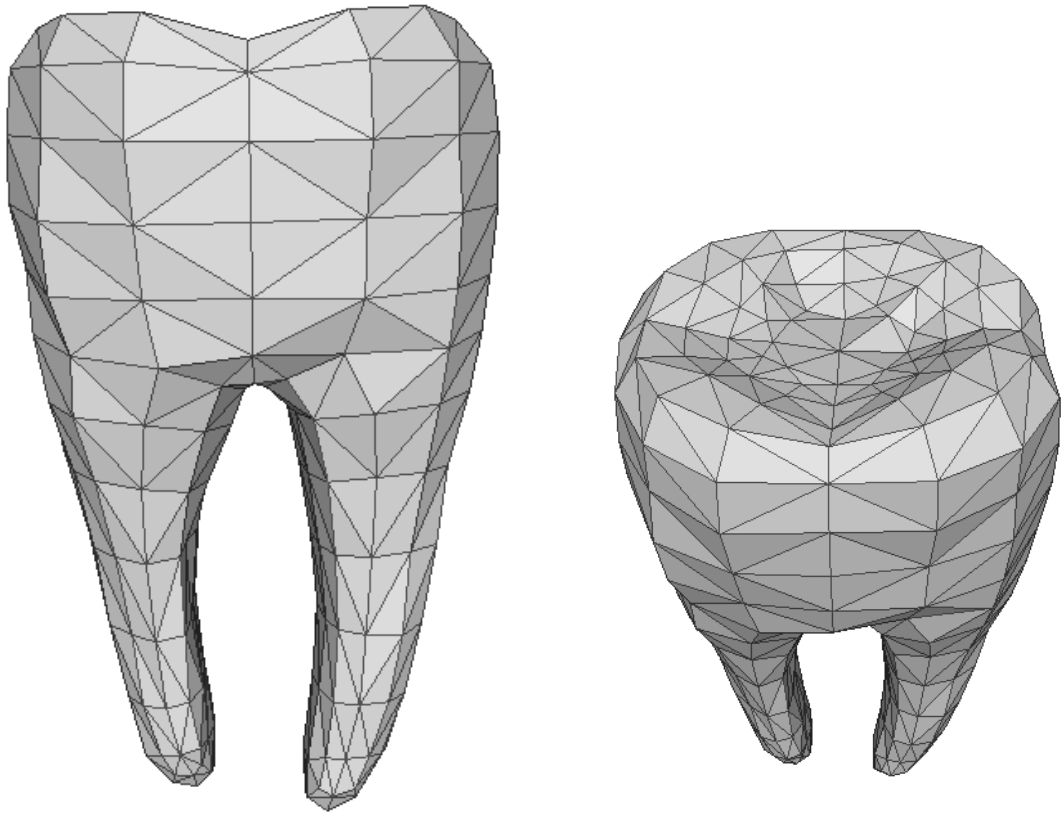
Afterwards the quadratic meshes have been automatically triangularized by Blender. Then the surface meshes must be converted into a volumetric mesh via TetGen, the whole process is described in appendix C. The final stats of the four meshes can be seen in table B.3.

| name | # nodes | # surface triangles | # body tetrahedra |
|---|---|---|---|
| bar $20 \times 20 \times 20$ | 113 | 210 | 263 |
| bar $10 \times 10 \times 10$ | 373 | 708 | 964 |
| bar $5 \times 5 \times 5$ | 1619 | 3004 | 4658 |
| bar $2.5 \times 2.5 \times 2.5$ | 6297 | 10214 | 21809 |

Table B.3: Data for the different bar resolutions.

### The Tooth Object

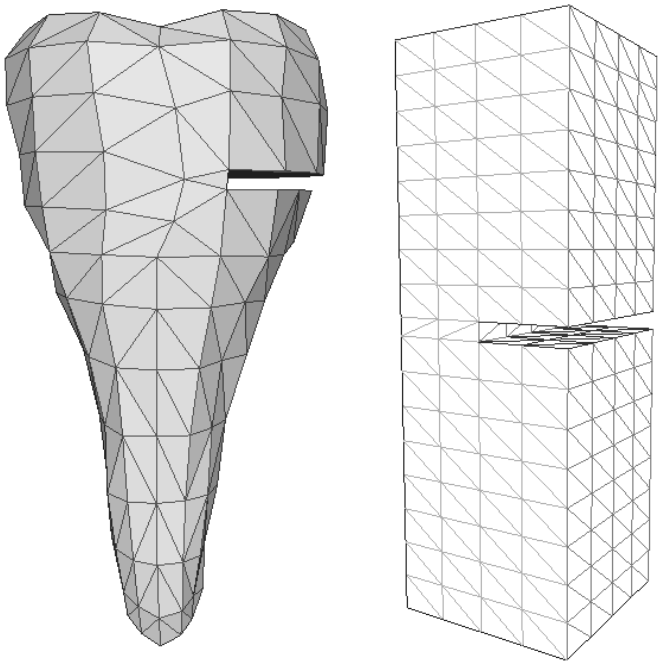
The tooth object is modeled with a few different kinds of meshes depending on in which kind of scenario we are using it. Basicly we have two models, a test tooth and a model of a real wisdom tooth. The test tooth is a rectangular model like the bar, but this model has a slice 50% through the middle of the object. The test tooth is illustrated in figure B.3b on the following page. The real tooth object comes in two variants, one with a slice 33% through the body, modelling a drill hole, and another without the slice. These two can be seen in figure B.4 and B.3a on the next page respectively. We also have these two variants in different resolutions, the original model and a simplified version. The simple version has been generated using the Quadric Edge Collapse Decimation tool in MeshLab.



(a) Tooth from the front.

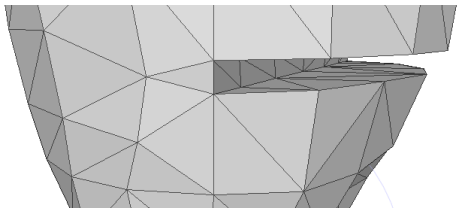
(b) Tooth above.

Figure B.2: The tooth test model.



(a) Tooth with 33% slice.

(b) Test tooth with 50% slice.



(c) Closeup of the drill hole in tooth with 33% slice.

Figure B.3: Tooth models with slice.

The stats for the tooth models are described in table B.4.

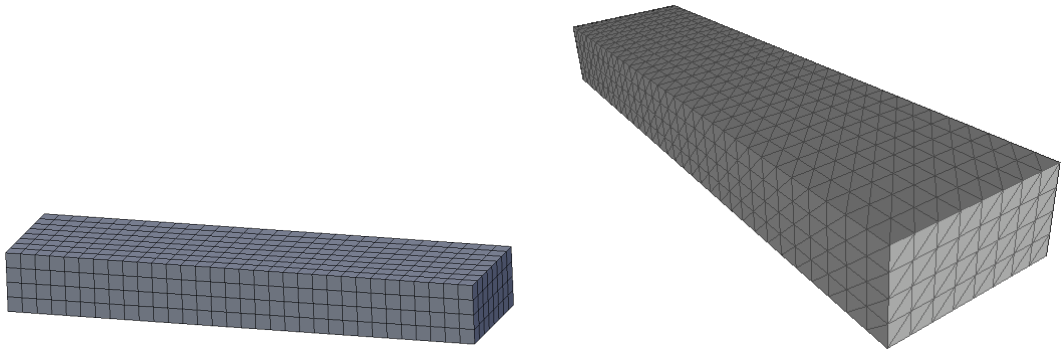
| name | # nodes | # surface triangles | # body tetrahedra |
|---|---|---|---|
| tooth | 667 | 1100 | 2264 |
| tooth with 33% slice | 748 | 1198 | 2677 |
| tooth (simple) | 371 | 680 | 1168 |
| tooth with 33% slice (simple) | 483 | 826 | 1652 |
| test tooth with 50% slice | 646 | 1180 | 1902 |
| test tooth high resolution with 50% slice | 2763 | 4808 | 8819 |

Table B.4: Data for the different tooth meshes.

# Appendix C

# Constructing Volumetric Meshes

Suppose that we have constructed the model shown in figure B.1c on page 154 (repeated below in figure C.1a), and want to convert this model into a volumetric mesh for use in the simulator.



(a) bar $5 \times 5 \times 5$.

(b) bar $5 \times 5 \times 5$ triangularized.

Figure C.1: The bar $5 \times 5 \times 5$ as quadratic and triangularized surface mesh.

To export this surface mesh and hereby triangularize it, as illustrated in figure C.1b, choose the following from the menu inside Blender (tested on Blender version 2.48a): [File] → [Export], and choose the [Stanford PLY] format.

```
output from Blender:  bar5x5x5.ply
```

Open this file with MeshLab (tested with version 1.2.2) and convert the file using the menu items: [File] → [Save as...]. A dialog appears, choose the [STL File Format]. In the next dialog uncheck the checkbox named "Binary encoding". It is importent to uncheck this checkbox because TetGen only wants ASCII files as input.

```
output from MeshLab:  bar5x5x5.stl
```

Generate the volumetric body mesh using TetGen (tested with version 1.4.2), which is a command line tool. Enter the following into the command line:

```
./tetgen -pqAzF bar5x5x5.stl

output from TetGen:  bar5x5x5.1.ele, bar5x5x5.1.node, and bar5x5x5.1.smesh
```

the `.node` files contains a vertex-pool, the `.ele` the body mesh, and the `.smesh` the surface mesh, as described in section 8 on page 102. It is these three files we load into our program. Note that the number of surface triangles generated by TetGen from the original surface model, is a order of magnitude larger than in the original model. This is caused by Blender's way of triangulating the surface, which does not take into account that we as a final goal want to generate a volumetric mesh.

The applications used are all cross platform and open source applications and can be downloaded from the following web sites: `http://www.blender.org/`, `http://meshlab.sourceforge.net/`, and `http://tetgen.berlios.de/`.

# Appendix D

# Test Machines

## D.1 Performance Desktop

Software: Ubuntu 9.04, 64bit with CUDA 2.0 toolkit, SDK and NVIDIA driver version 180.44. The hardware was sponcered by the Simens Foundation.

| Hardware components | Type |
|---|---|
| Asus GeForce PCI-E GTX295 1792MB DDR3 | Graphics card |
| Asus RAMPAGE II EXTREME X58 CrossFire & SLI | Motherboard |
| Intel Core i7 920 2.66 GHz 8MB Box | CPU + FAN |
| Corsair DDR3 PC3-1600 6GB CL9 kit XMS3 Classic | RAM |
| Hitachi Deskstar 7K1000.B 320GB 16MB 7200RPM | Harddrive |
| Pioneer BDC-S02BK BD-ROM/DVD-RW retail black m. sw | DVD-RW |
| Logitech OEM U96 Optical Wheel Mouse USB Black | Mouse |
| Logitech OEM Deluxe 250 USB keyboard black | Keyboard |
| Antec Performance One P193 | Enclosure |
| Corsair HX Series 1000W HX10000W 12cm SLI | PSU |
| PNY NVIDIA Tesla C1060 | GPU |

Table D.1: Table showing performance desktop hardware.

## D.2 Laptops

Two different Mac Book Pro version 3.1 were used, one with a graphics card with 128 MB RAM and one with 256 MB. Both were running Apple OSX, Leopard 10.5, running CUDA 2.2 toolkit and SDK, with graphic card drivers included in the toolkit.

| Hardware components | Type |
|---|---|
| NVIDIA 8600M GT, 128 MB or 258 MB | Graphics card |
| Intel Core 2 Duo 2.2 GHz | CPU |
| 2 GB | RAM |

Table D.2: Table showing laptop hardware.

# Appendix E

# Simulator Software

## E.1 CD-ROM

On the enclosed CD-ROM you will find the following:

- The source code for the entire simulation software.
- The source code for the OpenEngine framework.
- The volumetric meshed as described in appendix B.
- Videos and screenshots from the simulations performed.
- This thesis in pdf-format.

## E.2 OpenEngine Installation

The OpenEngine framework must be obtained and installed along with a few other tools and libraries to reproduce the results obtained in this thesis. On the enclosed CD the entire source code can be found along with the OpenEngine framework and all resources such as test data and scenario setups. Setting up the different test scenarios requires rebuilding the source since the implementation is only a prototype. The build system is based on cmake and tested on Windows Vista, Ubuntu Linux and Mac OS X. All dependent tools and libraries and free of use and most are open source.

## E.3 Getting Started

To get started check out the main web site `http://www.openengine.dk`. All OpenEngine resources can be accessed from the main web page. The site is wiki based and holds the source code, documentation and useful information regarding how to get started.

## E.4 Latest Version

See how to obtain a copy of the OpenEngine source code at `openengine.dk/wiki/Darcs`. Here you will find a guide on how to checkout the source code through the version control system darcs (`www.darcs.net`).

### Documentation

For the official OpenEngine source code documentation see: `http://openengine.dk/doc/`. To keep the documentation as updated as possible nightly doxygen builds are published here.

### Community

The website is the main source of new information so start looking here when questions come to mind. The ongoing discussion on all kinds of relevant OpenEngine work takes place on the mailing list, see `openengine.dk/wiki/MailingList` on how to join the list. The mailing list is the best place to ask any OpenEngine related question.

# Bibliography


[AB05] Pedro M. A. Areias and Ted Belytschko. Analysis of three-dimensional crack initiation and propagation using the extended finite element method. *International Journal for Numerical Methods in Engineering*, 63(5):760–788, March 2005. ISSN 1097-0207. URL `http://dx.doi.org/10.1002/nme.1305`.

[ALRA90] J. C. Anderson, K. D. Leaver, R. D. Rawlings, and J. M. Alexander. *Material Science*. Chapman and Hall, 4th edition, 1990. ISBN 0748740775.

[APM01] Erik Asmussen, Anne Peutzfeldt, and E. Christian Munksgaard. *Dentalmaterialer: Almen Del*. Odontologisk Boghandel & Forlag, København, 2001. ISBN 8774930648.

[Bat07] Klaus-Jürgen Bathe. *Finite Element Procedures*. Prentice Hall, 2007. ISBN 9780979004902.

[BF95a] Anthony Bedford and Wallace L. Fowler. *Dynamics: Engineering Mechanics*. Addison-Wesley, Reading, Massachusetts, 1995. ISBN 0201581973.

[BF95b] Anthony Bedford and Wallace L. Fowler. *Statics: Engineering Mechanics*. Addison-Wesley, Reading, Massachusetts, 1995. ISBN 0201581930.

[BH98] Jonathan Black and Garth Hastings. *Handbook of Biomaterial Properties*. Springer, 1st edition, 1998. ISBN 9780412603303.

[BN97] Morten Bro-Nielsen. Fast finite elements for surgery simulation. *Studies in health technology and informatics*, 39:395–400, 1997. ISSN 0926-9630. URL `http://www.ncbi.nlm.nih.gov/pubmed/10173064`.

[BN98] Morten Bro-Nielsen. Finite element modeling in surgery simulation. *Proceedings of the IEEE*, 86(3):490–503, March 1998. ISSN 0018-9219. URL `http://dx.doi.org/10.1109/5.662874`.

[Bow09] Allan F. Bower. *Applied Mechanics of Solids*. CRC, Boca Raton, Florida, 2009. ISBN 1439802475.

[BR98] Vincent Allan Barker and Jan Reffstrup. *The Finite Element Method for Partial Differential Equations*. Department of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, 1998.

[DD08] Prakash Mahadeo Dixit and Uday S. Dixit. *Modeling of Metal Forming and Machining Processes: By Finite Element and Soft Computing Methods*. Engineering materials and processes. Springer, London, 1st edition, 2008. ISBN 1848001886.

[Dil07] Ellis Harold Dill. *Continuum mechanics: Elasticity, Plasticity, Viscoelasticity*. CRC, Boca Raton, Florida, 2007. ISBN 0849397790.

[Dru67] Daniel Charles Drucker. *Introduction to Mechanics of Deformable Solids*. McGraw-Hill, New York, 1967.

[Gdo05] E. E. Gdoutos. *Fracture Mechanics: An Introduction*, volume 123 of *Solid mechanics and its applications*. Springer, Dordrecht, the Netherlands, 2nd edition, 2005. ISBN 1402028636.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley professional computing series. Addison-Wesley, Reading, Massachusetts, 1995. ISBN 0201633612.

[GSdC04] Marcelo Giannini, Carlos Jose Soares, and Ricardo Marins de Carvalho. Ultimate tensile strength of tooth structures. *Dental Materials*, 20(4):322–329, 2004. ISSN 0109-5641. URL `http://dx.doi.org/10.1016/S0109-5641(03)00110-6`.

[HDSB01] Kenneth H. Huebner, Donald L. Dewhirst, Douglas E. Smith, and Ted G. Byrom. *The Finite Element Method for Engineers*. John Wiley and Sons, New York, 4th edition, 2001. ISBN 0471370789.

[JSK08] P. Jäger, P. Steinmann, and E. Kuhl. Modeling three-dimensional crack propagation - a comparison of crack path tracking strategies. *International Journal for Numerical Methods in Engineering*, 76(9):1328–1352, 2008. URL `http://dx.doi.org/10.1002/nme.2353`.

[KMM03] J. H. Kinney, S. J. Marshall, and G. W. Marshall. The mechanical properties of human dentin: A critical review and re-evaluation of the dental literature. *Critical Reviews in Oral Biology and Medicine*, 14(1):13–29, 2003.

[Leo06] Steven J. Leon. *Linear Algebra with Applications*. Pearson Prentice Hall, Upper Saddle River, New Jersey, 7th edition, 2006. ISBN 0131857851.

[Mas70] George E. Mase. *Schaum's Outline of Theory and Problems of Continuum Mechanics*. Schaum's Outline Series. McGraw-Hill, New York, 1970. ISBN 0070406634.

[MH02] Tomas Möller and Eric Haines. *Real-time Rendering*. AK Peters, Natick, Massachusetts, 2nd edition, 2002. ISBN 1568811829.

[MJLW07] Karol Miller, Grand Joldes, Dane Lance, and Adam Wittek. Total lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. *Communications in Numerical Methods in Engineering*, 23(2):121–134, 2007.

[MMDJ01] Matthias Müller, Leonard McMillan, Julie Dorsey, and Robert Jagnow. Real-time simulation of deformation and fracture of stiff materials. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, pages 113–124. Springer-Verlag, New York, USA, 2001. ISBN 3-211-83711-6.

[Mos06] Jesper Mosegaard. *Cardiac Surgery Simulation - Graphics Hardware meets Congenital Heart Disease*. PhD thesis, Department of Computer Science, University of Aarhus, Denmark, October 2006.

[MW05] W. Martienssen and H. Warlimont. *Springer Handbook of Condensed Matter and Materials Data*. Springer, Berlin, 2005. ISBN 3540443762.

[NVI09] NVIDIA Corporation. *NVIDIA CUDA, Programming Guide, version 2.3.1.*, 2009.

[OH99] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 137–146. ACM Press/Addison-Wesley, New York, USA, 1999. ISBN 0-201-48560-5. URL `http://dx.doi.org/10.1145/311535.311550`.

[Øst02] Ole Østerby. *Numerical Analysis, Supplementary Notes.* Department of Computer Science, Aarhus University, 2002.

[RMS08] Allan Rasmusson, Jesper Mosegaard, and Thomas Sangild Sørensen. Exploring parallel algorithms for volumetric mass-spring-damper models in cuda. *Biomedical Simulation*, pages 49–58, 2008. URL `http://dx.doi.org/10.1007/978-3-540-70521-5_6`.

[Sin08] D. K. Singh. *Strength of Materials.* CRC, Boca Raton, Florida, 2008. ISBN 9781420069167.

[SM06a] Thomas Sangild Sørensen and Jesper Mosegaard. Haptic feedback for the gpu-based surgical simulator. *Studies in health technology and informatics*, 119:523–528, 2006. ISSN 0926-9630. URL `http://view.ncbi.nlm.nih.gov/pubmed/16404113`.

[SM06b] Thomas Sangild Sørensen and Jesper Mosegaard. An introduction to gpu accelerated surgical simulation. In Matthias Harders and Gabor Szekely, editors, *Third International Symposium, ISBMS 2006*, volume 4072 of *Lecture Notes in Computer Science*, pages 93–104. Springer, Berlin, 2006. URL `http://dx.doi.org/10.1007/11790273_11`.

[Spe80] Anthony James Merrill Spencer. *Continuum Mechanics.* Longman mathematical texts. Longman, London, 1980. ISBN 0582442826.

[Str86] Gilbert Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, Wellesley, Massachusetts, 1986. ISBN 0961408804.

[TCO08] Zeike A. Taylor, Mario Cheng, and Sebastien Ourselin. High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. *IEEE Transactions on Medical Imaging*, 27(5):650–663, May 2008. ISSN 0278-0062. URL `http://dx.doi.org/10.1109/TMI.2007.913112`.

[TG70] Stephen Prokofyevich Timoshenko and J. N. Goodier. *Theory of Elasticity.* McGraw-Hill, New York, 3rd edition, 1970. ISBN 0070858055.

[TG87] Stephen Prokofyevich Timoshenko and James M. Gere. *Mechanics of Materials.* Van Nostrand Reinhold, Boston, Massachusetts, 2nd edition, 1987. ISBN 0278000401.

[TNSM08] Peter Trier, Karsten Østergaard Noe, Mads Sølvsten Sørensen, and Jesper Mosegaard. The visible ear surgery simulator. *Studies in health technology and informatics*, 132:523–525, 2008. ISSN 0926-9630. URL `http://view.ncbi.nlm.nih.gov/pubmed/18391361`.

[WP01] Alan H. Watt and Fabio Policarpo. *3D games: Real-time Rendering and Software Technology.* SIGGRAPH series. ACM Press, New York, 1st edition, 2001. ISBN 0201619210.

[Wün08] Burkhard Wünsche. *The Visualization of 3D Stress and Strain Tensor Fields*. Department of Computer Science, University of Auckland, 92019, New Zealand, 2008.

[YFFS00] Hugh D. Young, Roger A. Freedman, A. Lewis Ford, and T. R. Sandin. *Sears and Zemansky's University Physics*. Addison-Wesley, San Francisco ; Harlow, 10th edition, 2000. ISBN 0201603365.