ISSN 0280-5316
ISRN LUTFD2/TFRT--5866--SE

Development of Interactive Simulator for Telepresence Robot in Surgical Applications

Christoph Haas

Department of Automatic Control
Lund University
June 2010

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* June 2010 *Document Number* ISRN LUTFD2//TFRT--5866--SE
Author(s) Christoph Haas	*Supervisor* Keita Ono at TU München, Germany Anders Robertsson Automatic Control,Lund Rolf Johansson Automatic Contro, Lundl (Examiner) *Sponsoring organization*

Title and subtitle
Development of Interactive Simulator for Telepresence Robot in Surgical Applications. (Utveckling av interaktiv simulator för robot med närvarokänsla i kirurgi)

Abstract
The purpose of this Diploma thesis is to develop and implement an interactive simulator for a surgical robot in a Telepresence application. The focus is on an incision procedure of a scalpel during an operation. The geometric deformation of the simulation is based on a Finite Element Method (FEM) model which has to cope with discontinuity due to the incision through the body. The FEM modelling can be done using e.g. FEM with remeshing method or XFEM which treats the cut in the body as a type of material discontinuity. A detailed analyse of the eXtended Finite Element Method (XFEM) and a comparison to the FEM with remeshing is made. For the implementation of simulation, a FEM remeshing method is used. A scalpel mounted to the end effector of a robot is controlled by a Haptic device and cuts through a silicone block. The deformation of the real test object are measured by a 2D scanning device and compared to the results of the simulation. The deviation of the reality and simulation were less than 1% based on the dimension of the body.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 75	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/

Meinen Eltern

Acknowledgement – Danksagung

Die vorliegende Arbeit entstand im Rahmen meiner Diplomarbeit am Lehrstuhl für Angewandte Mechanik der Technischen Universität München. Bei Herrn Prof. Dr.-Ing Dr.-Ing habil. Heinz Ulbrich bedanke ich mich für die Vergabe und Betreuung der Diplomarbeit. Besonderen Dank schulde ich Herrn Keita Ono, der mit der Betreuung meiner Diplomarbeit vertraut wurde. Durch seine engagierte Betreuung und stete Diskussionsbereitschaft mit vielseitigen Denkanstößen half er mir bei der Erstellung dieser Arbeit.

Ganz herzlich bedanke ich mich bei meinen Geschwistern Lena und Tillmann und meinem Freund Lucas für die intensive Auseinandersetzung mit meiner Arbeit gerade im Endspurt. Schließlich gilt ein besonderer Dank meinen Eltern Astrid und Joachim, deren Vertrauen und Unterstützung mir stets die Verfolgung meiner Ziele ermöglichen.

Contents

Introduction

This diploma thesis is part of the collaborative research center SFB453 “High-Fidelity Telepresence and Teleaction”. Telepresence means that a human operator feels present in another, remote, or not accessible environment over a technical connection, for instance internet or satellite. In a Teleaction application the human operator is not only passively present, but he can also actively intervene at the distant place. There exists many kinds of Teleaction application, but here a robot is used to interfere over a haptic device with the distant environment. A communication over a long distance comes along with time delays due to the connection. These delays can influence the feedback information of the system. Since the High-Fidelity has to be reached, the feedback perception should not be different from the reality. In our application, the feedback information consists of two parts, a force and visualisation of the surrounding of the distant place. The feedback force is given to the haptic device, so the operator feels the feedback information directly. The visual images are rendered at the monitor screen.

Robot systems over long distance can be found in many different areas. Here, a surgical operation scenario has been developed. A surgeon operates over a haptic device that is connected to the robot with the scalpel at the end effector. For some operation the knowledge or dexterous hands of specialists are needed. When these specialities cannot be in time at the place of emergency, the operation can be performed over the Telepresence system. A simulation environment for performing basic operation steps has to be design. Out of the simulation the operator get the visual image of the distance place and the forces acting on his scalpel. Different surgical simulators have been developed during the last years. The reasons for this simulator are not to operate over long distance, but to practice the skills of the surgeon inside a virtual environment before entering the operating room. These pre-operative training procedures lead to significant improvement in surgeries [1]. Ideas can be taken from the existing simulators. The most difficult task of this kind of simulators is realistically visualizing soft tissue behaviour in real time. A virtual cutting simulator should supply the following basic capabilities. Collision detection is needed for the control of the location, direction and orientation of the scalpel and for updating the intersection with the cut body. Another task is the updating of the simulation so the deformation of the incision is capture in real time. The physical model should reflect the physical behaviour as accurately as possible. By looking at the users interface, a haptic force feedback is invaluable in providing realistic interaction behaviour, both from the visual and the palpable point of view.

The goal of the work aims to support the mentioned purposes and focuses on the development of setting up a whole experimental platform to perform a cut with a human controlled robot over a haptic device. The main topics are to find a Finite-Element-Method (FEM) model that can deal with the discontinuity of the incision and develop a hardware platform for performing a cut and comparing it to the simulation.

The thesis is subdivided into six chapters. It begins with the presentation of the theoretical part of the project. It contains the description of the used physical model for the deformation of a body. First, the standard FEM is outlined and then the explanation of the eXtended-Finite-Element-Method (XFEM) follows. In the end of the chapter a conclusion follows about how to deal with strong discontinuities in a FEM mesh surrounding.

For the comparison of the simulation with the reality, an experiment is set up to measure the deformation occurring during a cut procedure. The hardware set up is explained in chapter three. A tension plate form is developed to give the test object a certain pre-tension before the cut is performed. The use of two different sensors, the force and the distance sensors, is also described. In the end, a CAD model of the whole experimental set is shown to explain how the sequence of the measurements during the cut process is.

Chapter four contains the procedures perform on the software side of the experiment. It comprehends the explanation of the Mesh generation with Matlab, the calculation of the FEM in the programming language C++ and the scanning of the scanControl Device.

After the presentation of the hardware and software side in the previous two chapters, the results of the simulation and the measurements are compared in chapter five. In different figures the advantages and disadvantage of the used simulations methods are pointed out. The last chapter summarizes results and gives an outlook for future works.

1 Principle of Finite Element Method

There are many ways to develop a model for elastic bodies. Thereby, the goal of the modelling is to reproduce the elastic behaviours of the body. The most popular approaches of modelling elastic bodies are the mass-spring system, modal systems and the Finite Element Method. Every approach has its advantages and disadvantages related to deformation effects and computation effort. Modal systems use certain eigenvalues for describing the whole deformation of a body. The method has normally a small number of degrees of freedom and therefore can be computed fast. The disadvantages are that nonlinear effects, huge deformation and nonlinear material properties are not included in the model. To describe these effects, it is better to use the Finite Element Method. More degrees of freedom are used, which leads to more computational time, but the model can cope with large deformation. So, the FEM has been chosen for the present work.

In section 2.1 a general formulation of the standard FEM is derived. The basic steps of the linear elasticity formulation of standard FEM are described. To capture singularities in an element, the eXtended Finite Element Method (XFEM) is introduced in section 2.2. As it can be seen from the name of the method, the XFEM is an extension of the standard FEM. The XFEM method is introduced in the 1D case to see how the shape functions behave. In the 2D case several deformation situation are compared and the effects of the XFEM are described. Instead of using the XFEM, a remeshing at the edges of the singularities gives also the deformation. A comparison between the two methods is given. In the end of section 2.2 the implementation of the 3D case is shown. Finally, a conclusion shows advantages and disadvantages. An advice for an application under certain circumstances is given.

1.1 Derivation of the Finite Element Method for 3D elasticity

In engineering mechanics, all physical phenomena are dealing with differential equations, and usually the problem is too complicated to be solved by classical analytical method. The finite element method (FEM) is a numerical approach by which partial differential equations with given boundary conditions can be solved in an approximate manner.

The most widely known model in solid mechanics is the linear elastic model. Based on this model, the FEM equations are derived in four basic steps according to [2]:

1. Establish the strong formulation of the problem
2. Obtain the weak form of the problem
3. Make an element wise approximation over the entire body of the unknown function
4. Choose the weight function in accordance with the Galerkin method

With the help of the Cauchy equation, the strong formulation of solid mechanics is obtained. When the strong formulation is multiplied by an arbitrary function, the weak formulation is found which is the starting point for every solution of a FEM routine. In the end, the

Galerkin method is chosen to replace the arbitrary function with a linear approximation over the elements.

1.1.1 Strong formulation of solid mechanics

The strong formulation of a mechanical problem is obtained summarizing all forces of a body. For this the body is considered as a standard cube, see Figure 1.1.

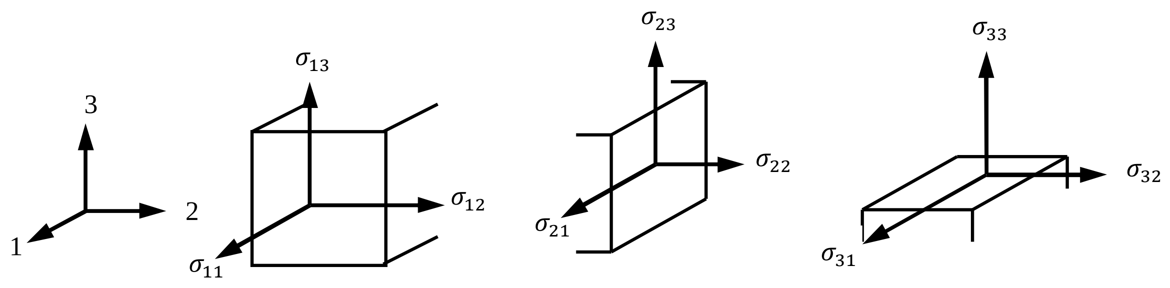


Figure 1.1: Stresses on standard cube

The derivative of the stress tensor is formulated for every direction of the coordinate system and summed up with the corresponding body forces b_i. The following equations are obtained:

$$\begin{aligned}
&\frac{\partial\sigma_{11}}{\partial x_1}+\frac{\partial\sigma_{12}}{\partial x_2}+\frac{\partial\sigma_{13}}{\partial x_3}+b_1=0,\\
&\frac{\partial\sigma_{21}}{\partial x_1}+\frac{\partial\sigma_{22}}{\partial x_2}+\frac{\partial\sigma_{23}}{\partial x_3}+b_2=0,\\
&\frac{\partial\sigma_{31}}{\partial x_1}+\frac{\partial\sigma_{32}}{\partial x_2}+\frac{\partial\sigma_{33}}{\partial x_3}+b_3=0.
\end{aligned} \tag{1.1}$$

These three equations are formulated in matrix notation, using the matrices $\tilde{\boldsymbol{\nabla}}^T$ and $\boldsymbol{\sigma}$,

$$\tilde{\boldsymbol{\nabla}}^T=\begin{bmatrix}\frac{\partial}{\partial x_1} & 0 & 0 & \frac{\partial}{\partial x_2} & 0 & \frac{\partial}{\partial x_3}\\ 0 & \frac{\partial}{\partial x_2} & 0 & \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_3} & 0\\ 0 & 0 & \frac{\partial}{\partial x_3} & 0 & \frac{\partial}{\partial x_2} & \frac{\partial}{\partial x_1}\end{bmatrix} \text{ and } \boldsymbol{\sigma}=\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{12}\\ \sigma_{13}\\ \sigma_{23}\end{bmatrix}. \tag{1.2}$$

Finally, the strong formulation in matrix form can be written as

$$\tilde{\boldsymbol{\nabla}}^{\mathrm{T}}\boldsymbol{\sigma}+\mathbf{b}=0. \tag{1.3}$$

1.1.2 The weak formulation of solid mechanics

The weak formulation is obtained in three steps. First, the strong formulation is multiplied by an arbitrary function v. In the next step the equation is integrated over the volume of the whole test object. In the last step, the Green-Gauss theorem is applied and the final equation states

$$\int_V (\tilde{\nabla}\mathbf{v})^{\mathrm{T}} \boldsymbol{\sigma} dV = \int_V \mathbf{v}^{\mathrm{T}}\mathbf{t} dS + \int_V \mathbf{v}^{\mathrm{T}}\mathbf{b} dV, \tag{1.4}$$

which is the weak formulation of the differential equations of equilibrium (1.1) subjected to the boundary conditions **t**, called the Neumann-boundary conditions.

As the weight vector **v** is arbitrary, equation (1.4) holds for any constitutive relation. The weak formulation is often termed the virtual work equation or virtual work principle.

1.1.3 Element wise approximation of the unknown function

Several elements can be used for the discretization of the unknown shape function. A detailed discussion of 3D elements is given in Hughes[3], Gallagher[4] and Zienkiewitz and Talyor[5]. To save up computation time, the linear tetrahedral is used.

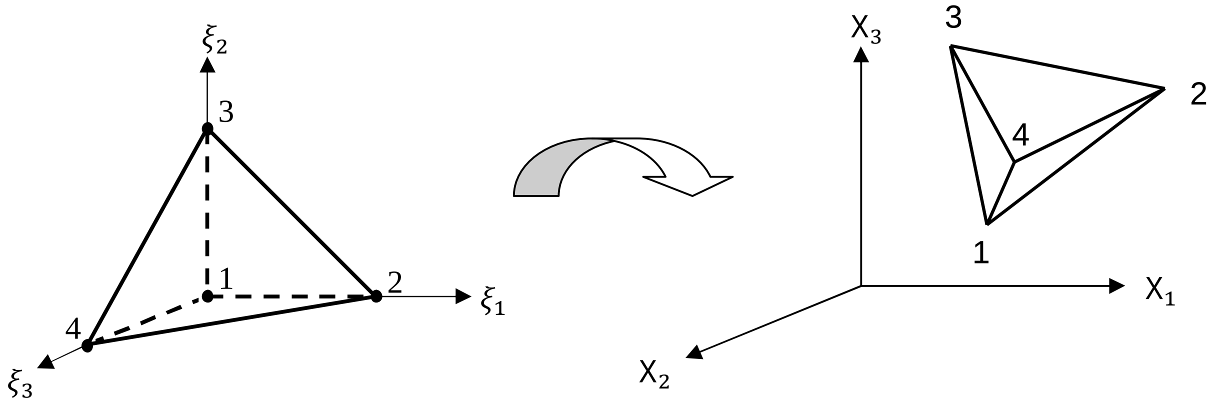


Figure 1.2: Isoparametric transformation

The concept of isoparametric finite elements is used to perform the integration over the volumes of the elements. The elements are first defined in local coordinates ξ_i and then transformed in global coordinates x_i by the Jacobi matrix, see Figure 1.2. This procedure was first introduced by Taig[6].

For a given nodal point i the shape functions Φ_i^e are then defined according to the Kronecker delta property

$$\Phi_i^e = \begin{cases} 1 & \text{at nodal point i} \\ 0 & \text{at all other nodal points.} \end{cases} \tag{1.5}$$

So the following shape functions of the tetrahedral in local coordinates are

$$\vec{\boldsymbol{\Phi}} = \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \end{bmatrix} = \begin{bmatrix} 1 - \xi_1 - \xi_2 - \xi_3 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{bmatrix}, \tag{1.6}$$

where the index of Φ_i describes the number of the node. Furthermore the derivatives of the shape functions are needed to perform the Galerkin method

$$\mathbf{\Phi}_\xi = \frac{\partial \mathbf{\Phi}_i}{\partial \xi_j} = \begin{bmatrix} \frac{\partial \Phi_1}{\partial \xi_1} & \frac{\partial \Phi_1}{\partial \xi_2} & \frac{\partial \Phi_1}{\partial \xi_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial \Phi_4}{\partial \xi_1} & \frac{\partial \Phi_4}{\partial \xi_2} & \frac{\partial \Phi_4}{\partial \xi_3} \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \tag{1.7}$$

To map the local shape function into the global, the Jacobi transformation is used. It is important to notice that numbering of the element cannot be chosen arbitrary due to the transformation. As it can be seen in the Figure 1.2, the local element is defined in a right handed coordinate system. The numbering of this element must be maintained in the global one. An easy right hand rule can be developed for controlling the numbering, see Figure 1.3.

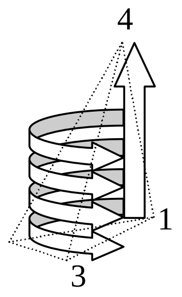


Figure 1.3: Right-Hand-Rule

The vertical arrow represents the thumb and the rest of the arrows the fingers. The nodes [1 2 3] present a plane on which the thumb points in the positive direction of the normal vector of the plane. In the same direction of the normal vector has to lie node 4.

Using the definition of the local shape functions $\mathbf{\Phi}_\xi$ and applying the inverse of the Jacobi matrix, the global shape functions are given by

$$\mathbf{\Phi}_X = \frac{\partial \mathbf{\Phi}_i}{\partial \mathrm{X}_j} = \begin{bmatrix} \frac{\partial \Phi_1}{\partial \mathrm{X}_1} & \frac{\partial \Phi_1}{\partial \mathrm{X}_2} & \frac{\partial \Phi_1}{\partial \mathrm{X}_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial \Phi_4}{\partial \mathrm{X}_1} & \frac{\partial \Phi_4}{\partial \mathrm{X}_2} & \frac{\partial \Phi_4}{\partial \mathrm{X}_3} \end{bmatrix} = \frac{\partial \mathbf{\Phi}_i}{\partial \xi_k} (\mathbf{J})^{-1} \text{ with } \mathbf{J} = \mathbf{X}^{(\mathrm{e})} \mathbf{\Phi}_\xi . \tag{1.8}$$

The vector $\mathbf{X}^{(e)}$ contains the coordinates of the element and looks like:

$$\mathbf{X}^{(e)} = \begin{bmatrix} X_1^1 & \cdots & X_4^1 \\ X_1^2 & \cdots & X_4^2 \\ X_1^3 & \cdots & X_4^3 \end{bmatrix} . \tag{1.9}$$

The elements of the derivatives of the global shape functions $\mathbf{\Phi}_X$ constitute the B matrix

$$\mathbf{B} = \begin{bmatrix} \frac{\partial \Phi_1}{\partial X_1} & 0 & 0 & \frac{\partial \Phi_2}{\partial X_1} & 0 & 0 & \cdots \\ 0 & \frac{\partial \Phi_1}{\partial X_2} & 0 & 0 & \frac{\partial \Phi_2}{\partial X_2} & 0 & \cdots \\ 0 & 0 & \frac{\partial \Phi_1}{\partial X_3} & 0 & 0 & \frac{\partial \Phi_2}{\partial X_3} & \cdots \\ \frac{\partial \Phi_1}{\partial X_2} & \frac{\partial \Phi_1}{\partial X_1} & 0 & \frac{\partial \Phi_2}{\partial X_2} & \frac{\partial \Phi_2}{\partial X_1} & 0 & \cdots \\ 0 & \frac{\partial \Phi_1}{\partial X_3} & \frac{\partial \Phi_1}{\partial X_2} & 0 & \frac{\partial \Phi_2}{\partial X_3} & \frac{\partial \Phi_2}{\partial X_2} & \cdots \\ \frac{\partial \Phi_1}{\partial X_3} & 0 & \frac{\partial \Phi_1}{\partial X_1} & \frac{\partial \Phi_2}{\partial X_3} & 0 & \frac{\partial \Phi_2}{\partial X_1} & \cdots \end{bmatrix}. \quad (1.10)$$

1.1.4 The Galerkin method

In step 4, the Galerkin method is applied to the weak formulation. First, the displacement is approximated by the shape functions, multiplied by the displacement values at the nodes

$$\mathbf{u} = \boldsymbol{\phi}\mathbf{a}. \quad (1.11)$$

The Galerkin method means that the weight vector $\mathbf{v}$ is chosen according to

$$\mathbf{v} = \boldsymbol{\phi}\mathbf{c}. \quad (1.12)$$

The function $\mathbf{v}$ is arbitrary. So, the matrix $\mathbf{c}$ is arbitrary, which leads to

$$\tilde{\nabla}\mathbf{v} = \mathbf{Bc}, \text{ where } \mathbf{B} = \tilde{\nabla}\boldsymbol{\phi}. \quad (1.13)$$

Equation (1.13) insert in the weak FEM formulation gives

$$\mathbf{c}^{\mathrm{T}}\left(\int_V \mathbf{B}^{\mathrm{T}}\boldsymbol{\sigma}dV - \int_S \boldsymbol{\phi}^{\mathrm{T}}\mathbf{t}dS - \int_V \boldsymbol{\phi}^{\mathrm{T}}\mathbf{b}dV\right) = 0. \quad (1.14)$$

As the function $\mathbf{c}^{\mathrm{T}}$ is arbitrary, the function is cancelled out by the zero on the left side of equation (1.14) and the following equation is obtained

$$\int_V \mathbf{B}^{\mathrm{T}}\boldsymbol{\sigma}dV = \int_S \boldsymbol{\phi}^{\mathrm{T}}\mathbf{t}dS + \int_V \boldsymbol{\phi}^{\mathrm{T}}\mathbf{b}dV. \quad (1.15)$$

The relation between stresses and strains is called the constitutive relation and a variety of such relations have been established for instance elasticity, plasticity, viscoelasticity, etc. The simplest constitutive theory is the linear elasticity and is represented in 1D by Hook's law in 1676

$$\sigma = E \cdot \varepsilon\,, \quad (1.16)$$

where the material constant E is named Young’s modulus. This expression shows that the material response is path-independent, i.e. that there exists a one-to-one relation between stress and strain

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{13} \\ \sigma_{23} \end{bmatrix}; \mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & \cdots & D_{16} \\ D_{21} & D_{22} & \cdots & D_{26} \\ \vdots & \vdots & & \vdots \\ D_{61} & D_{62} & \cdots & D_{66} \end{bmatrix}; \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \gamma_{12} \\ \gamma_{13} \\ \gamma_{23} \end{bmatrix}. \tag{1.17}$$

For an isotropic material, properties are the same in all directions, and the following constitutive matrix is obtained

$$\mathbf{D} = \frac{E}{(1+v)(1-2v)} \begin{bmatrix} 1-v & v & v & 0 & 0 & 0 \\ v & 1-v & v & 0 & 0 & 0 \\ v & v & 1-v & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5(1-2v) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5(1-2v) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5(1-2v) \end{bmatrix}. \tag{1.18}$$

The coefficients $E$and v are Young’s modulus and Poisson’s ratio.

Applying linear elasticity $\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon}$ and $\boldsymbol{\varepsilon} = \tilde{\nabla}\mathbf{u} = \mathbf{B}\mathbf{a}$, the equation follows as,

$$\left(\int_V \mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}dV\right)\mathbf{a} = \int_S \boldsymbol{\phi}^{\mathrm{T}}\mathbf{t}dS + \int_V \boldsymbol{\phi}^{\mathrm{T}}\mathbf{b}dV, \tag{1.19}$$

$$\text{and } \mathbf{K} = \left(\int_V \mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}dV\right), \tag{1.20}$$

where **K** is the stiffness matrix. The boundary conditions are described in the traction vector **t** which is called the Neumann boundary condition and in the displacement vector **u** which is called the Dirichlet boundary condition [2]. These two boundary conditions are disjoint. That means that exactly one condition has to be valid at every boundary section.

1.2 General Formulation of Extended Finite Element Method

The extended finite element method (XFEM) is probably the most popular enriched method. It allows a local enrichment in subregions of the domains. In practice, it is mainly used in applications that involve discontinuities. A large number of such applications can be found: material interfaces, cracks, shocks, boundary layers, shear bands, etc. In this project, the XFEM method is used for describing a cut through an elastic body.

The general formulation of an XFEM approximation of a function u(x) is of the following form

$$u(x) = \sum_{i=1}^{n} \phi_i(x) u_i + \sum_{j=1}^{n} \phi_j^{*1}(x) \psi_j^1(x) a_j^1 + \cdots + \sum_{j=1}^{n} \phi_j^{*m}(x) \psi_j^m(x) a_j^m. \quad (1.21)$$

The first part of the sum is the standard FEM approximation with the shape function $\phi_i(x)$ and the displacements u_i at the nodal points. The rest of the term contains the Extended FEM part, where $\psi_j(x)$ are the discontinuous enrichment functions and a_j are the added nodal DOF. The shape functions of the added DOF $\phi_j^*(x)$ are not necessarily identical to the shape functions of the corresponding nodes. As starting point, the enriched shape functions are equal to the non-enriched ones [7].

The XFEM approximation consists of a standard FEM approximation plus additional enrichments. The XFEM is based on the partition of unity concept [2]. The functions $\phi_j^*(x)$ build a partition of unity in local parts of the domain

$$\sum_{i=1}^{n} \phi_j^*(x) = 1 \ \forall x \in \Omega_k^*, \forall k = 1, \ldots, m. \quad (1.22)$$

Using XFEM several kinds of discontinuities can be modeled. These discontinuities are divided in two main groups the strong ones and the weak ones. A weak discontinuity is characterized for instance by a kink in the displacement, i.e. a jump in the gradient. In our case, the cut is meant to be a strong discontinuity, because it has already a jump in the displacement.

The function ψ_j is called the global enrichment function. It defines the level sets of the discontinuity. Most widely used in XFEM is the Heaviside function which leads to identical results as the sign-function because they span the same approximation space,

$$\psi(x) = H(\phi(x)) = \begin{cases} 0 : \phi(x) \leq 0 \\ 1 : \phi(x) > 0 \end{cases}. \quad (1.23)$$

The enrichment of the nodes impacts all elements sharing the enriched nodes, i.e. the one ring of neighbours of the element containing the cut. Due to the additional enrichment terms the shape functions of the XFEM do not have the Kronecker delta property in general. Consequently, the displacement of the enriched nodes has to be computed as a sum of the components $u_i + \psi_i a_i$. This fact also complicates the treatment of the boundary conditions.

To avoid the problem, the shifted enrichment functions are used instead,

$$\psi_i(x) = \frac{H(x) - H_i}{2}, \tag{1.24}$$

where H_i is the value at the i-th node. The division by 2 ensures the Kronecker delta property. The effect of shifting is that the enrichment contributions only appear within the element, which leads to an enormous simplification of the implementation [8].

1.3 Comparison the XFEM method with the FEM

In this subchapter straight line cuts are performed in different meshes by using different approximation functions. Starting with the 1D case the course of the shape function over the cut can be visualized for better understanding. Afterwards 2D elements are used. Thereby some effects of how the XFEM method reaction to certain loadings can be seen. There is also done a comparison to standard FEM with remeshing and to deleting the cut element. Finally, 3D elements are cut. The same comparison is performed and the use of the XFEM can be seen.

1.3.1 1D elements

Starting point for the implementation of the XFEM is the 1D bar element. Trying to keep the solid problem only 1D leads to an unstable system. Putting bars in a row and cut one of them in the middle, the static balance of forces is not secured.

One easy way to solve this problem is to support the cut element at the end by two other bars. To get nearly a 1D case, the cut bar is situated on the x-axis and supported by two bars, one below and one above. It is demonstrated in the Figure 1.4.

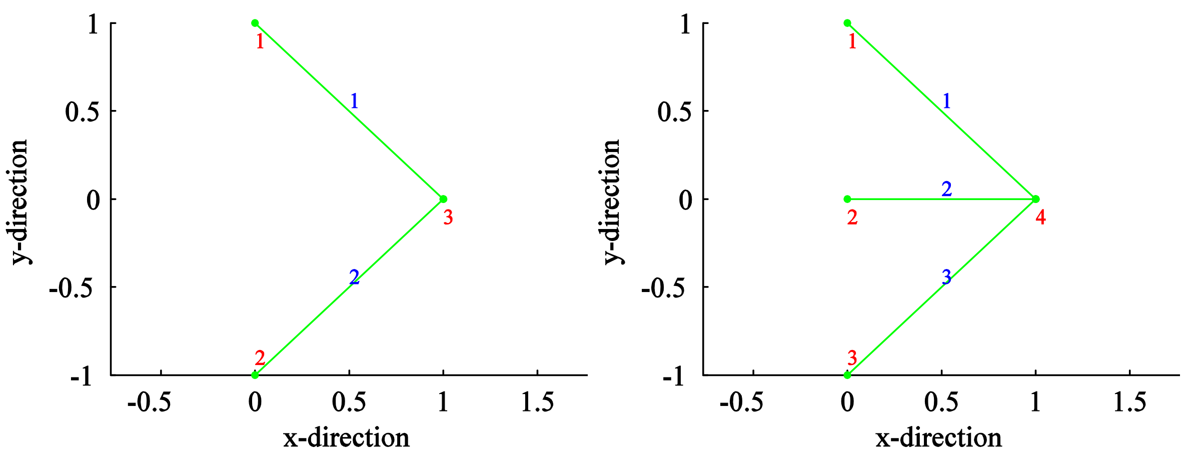


Figure 1.4: Truss structure, left standard FEM and right XFEM

By cutting a 1D bar in the middle, no load transmission will take place between the end nodes. From this it follows that the cut bar can be left out and then the other bars can be calculated after the standard FEM. In Figure 1.4, the XFEM mesh is shown on the right side, where the bar 2 will be cut in the middle. On the left side, the FEM mesh is plotted with the bar number 2 missing.

The XFEM is used with the Heaviside-enrichment function which is typically used for strong discontinuities and fully decouples the two parts of the domain.

The shape function of the enriched element 2 is given by,

$$\begin{aligned} M_1 &= N_1(x)\big[H\big(\phi(x)\big) - H\big(\phi(x_1)\big)\big], \\ M_2 &= N_2(x)\big[H\big(\phi(x)\big) - H\big(\phi(x_2)\big)\big], \end{aligned} \tag{1.25}$$

which are decomposed into a left, negative part, and a right positive part,

$$\begin{aligned} M_1^- &= N_1(x)\left[H\left(x-\frac{1}{2}\right) - H\left(x_1-\frac{1}{2}\right)\right] = (x-1)(0-0) = 0, \\ M_1^+ &= N_1(x)\left[H\left(x-\frac{1}{2}\right) - H\left(x_1-\frac{1}{2}\right)\right] = (x-1)(1-0) = 1-x, \end{aligned} \tag{1.26}$$

for the second node,

$$\begin{aligned} M_2^- &= N_2(x)\left[H\left(x-\frac{1}{2}\right) - H\left(x_2-\frac{1}{2}\right)\right] = (x)(0-1) = -x, \\ M_2^+ &= N_2(x)\left[H\left(x-\frac{1}{2}\right) - H\left(x_2-\frac{1}{2}\right)\right] = (x)(1-1) = 0, \end{aligned} \tag{1.27}$$

The enriched shape functions are plotted in Figure 1.5 and compared to the FEM shape functions.

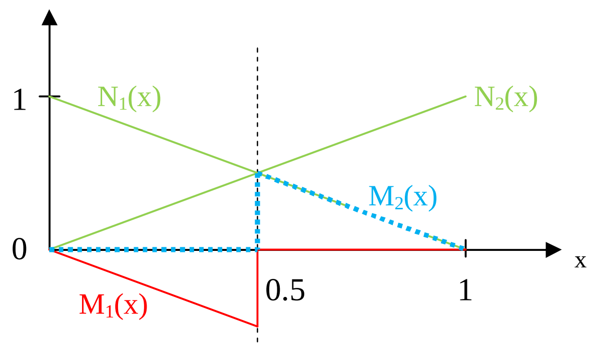


Figure 1.5: Shape function of the cut element

In the middle of element 2, the enrichment shape functions M_1 and M_2 have a step to zero. This represents the cut in the bar and decouples the two nodes from each others, so there is no longer a connection between the two nodes.

The derivatives of the shape function M_1 and M_2 are $N_{,x}^-(x) = [-1\ 1\ 0-1]$ and $N_{,x}^+(x) = [-1\ 1-1\ 0]$ and for the stiffness matrix follows,

$$\mathbf{K} = \frac{EA^-}{\mathbf{2}} \cdot M_1^T M_1 + \frac{EA^+}{\mathbf{2}} \cdot M_2^T M_2, \tag{1.28}$$

$$\mathbf{K} = EA^{-}\begin{bmatrix} 0.5 & -0.5 & 0 & 0.5 \\ -0.5 & 0.5 & 0 & -0.5 \\ 0 & 0 & 0 & 0 \\ 0.5 & -0.5 & 0 & 0.5 \end{bmatrix} + EA^{+}\begin{bmatrix} 0.5 & -0.5 & 0.5 & 0 \\ -0.5 & 0.5 & -0.5 & 0 \\ 0.5 & -0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

It should be denoted that this matrix is also singular after introduction of boundary conditions, because it fully decouples the right part of the bar and no Dirichlet boundary conditions are imposed there.

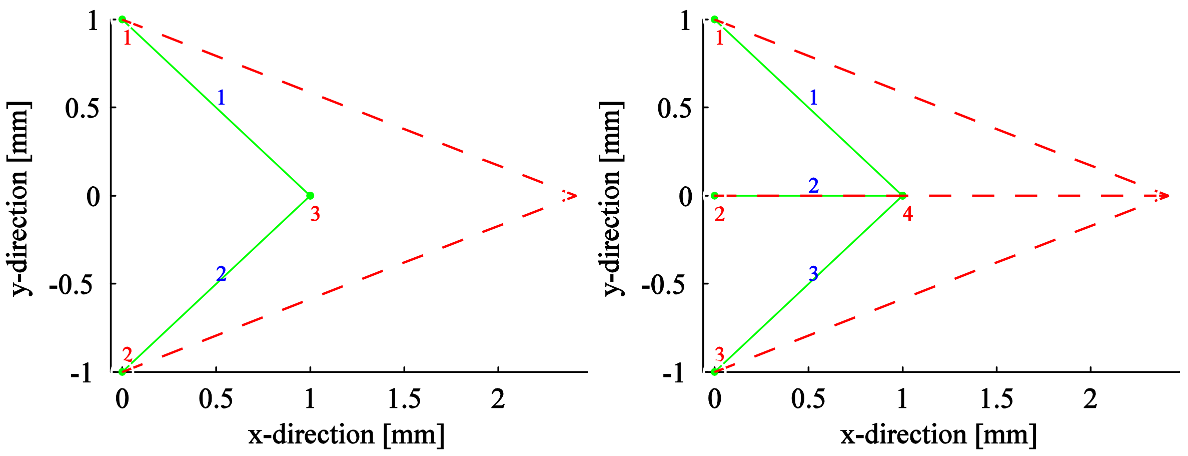


Figure 1.6: Deformed truss structure, left standard FEM and right XFEM

To obtain a non-singular stiffness matrix new Dirichlet boundary conditions at the enriched element stiffness matrix have to be added. After the cut, the enriched part only can act in x-direction which means that the degrees in y-direction are equal to zero. Using this added boundary condition the global stiffness matrix is non-singular and can be solved for every applied force [9].

1.3.2 2D elements

With the 2D elements, the different techniques can be shown how to implement the XFEM method. Rectangle elements are used as 2D elements. The shape functions are easy to derive and this type of element is axially symmetrical, which will lead to symmetric deformation, when a symmetric load is applied. This is an easy check whether our FEM programme is working correctly.

Another reason for the use of rectangles is that the shape of the elements matches with the projection of a block in a plane. As the goal of the present work is to cut in a straight line through a body, the 2D mesh consisting of rectangles comes equates in some situations the 3D simulation.

1.3.2.1 Creation of the meshes

Three different meshes are used in this subchapter, a XFEM mesh, a FEM mesh where the cut is remeshed and a FEM mesh where the cut element is completely delete.

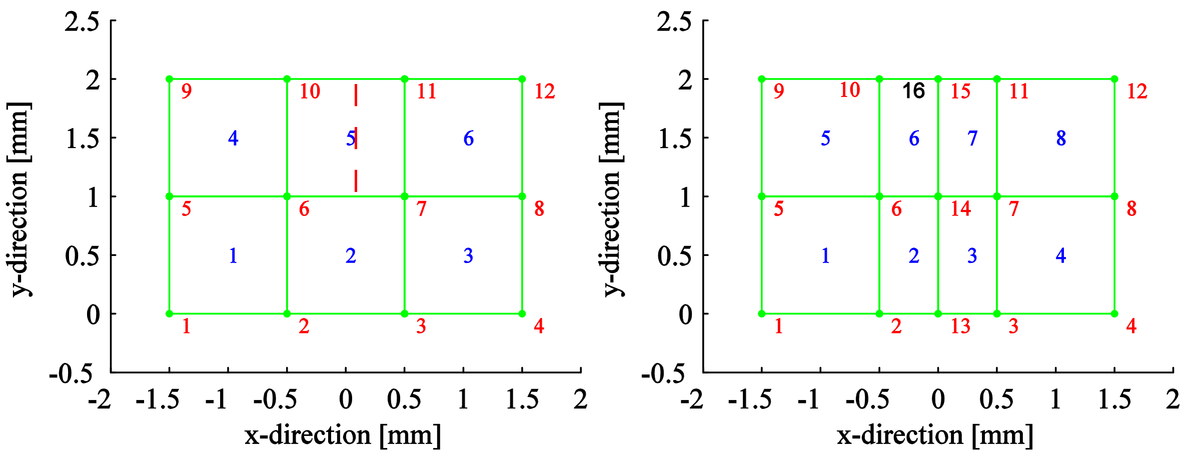


Figure 1.7: Mesh, left XFEM and right FEM

In Figure 1.7, on the left hand side the XFEM mesh is plotted and on the right hand side the FEM mesh with remeshing. There are six elements connected to each other for the XFEM case. The red numbers represent the numbering of the nodes and the blue numbers the numbering of the elements. At the nodes [1 5 9], a deformation in the negative x-direction and at the nodes [4 8 12] on the opposite x-direction takes place. Through all tests, the offset of the deformation is kept equal at 0.4 mm. The cut will take place in the middle of element 5 along the y-axis, represented by the red dashed line.

To develop the same model using conventional FEM several steps have to be performed. A remesh is needed to describe the discontinuity at the cut edges, so there has to be added additional element around the cut surface. When only dividing element 5, the convergence criterion is violated. It is postulated that the approximation must be continuous over the boundaries of every element, see [2], so element 2 has to be also divided. By adding a new node 16 at the node 15 a gap between the elements 6 and 7 can occur, see Figure 1.7 on the right hand side.

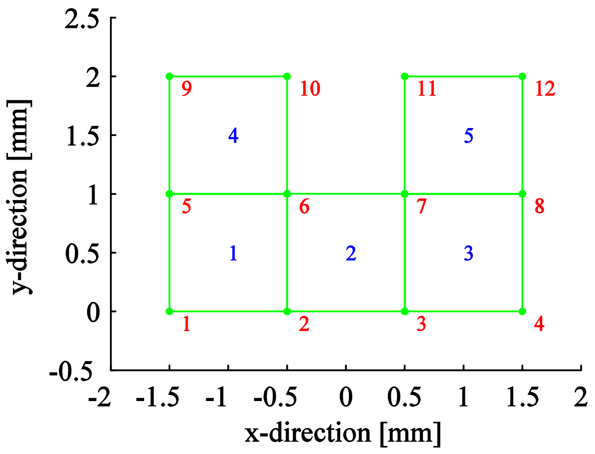


Figure 1.8: Mesh, FEM element missing

In the last mesh, the same mesh is taken as for the XFEM calculation, but the five element is deleted out of the mesh, see Figure 1.8.

1.3.2.2 Comparison between the different meshes

Comparison between the meshes should be done with caution. As the FEM in general represents an approximation over the whole test body, the size and number of the elements have an influence on the result.

For instance, more nodes and more elements are used in the FEM with remeshing compared to the XFEM mesh. To get an idea, how huge the difference regarding to the different meshes are, the problem is stated without the cut. This means that the element 5 is not enriched for the XFEM and for the FEM node 16 is not added to the mesh, so there will be no cut present in the meshes. The difference of the deformation regarding to the different mesh techniques are shown in Figure 1.9.

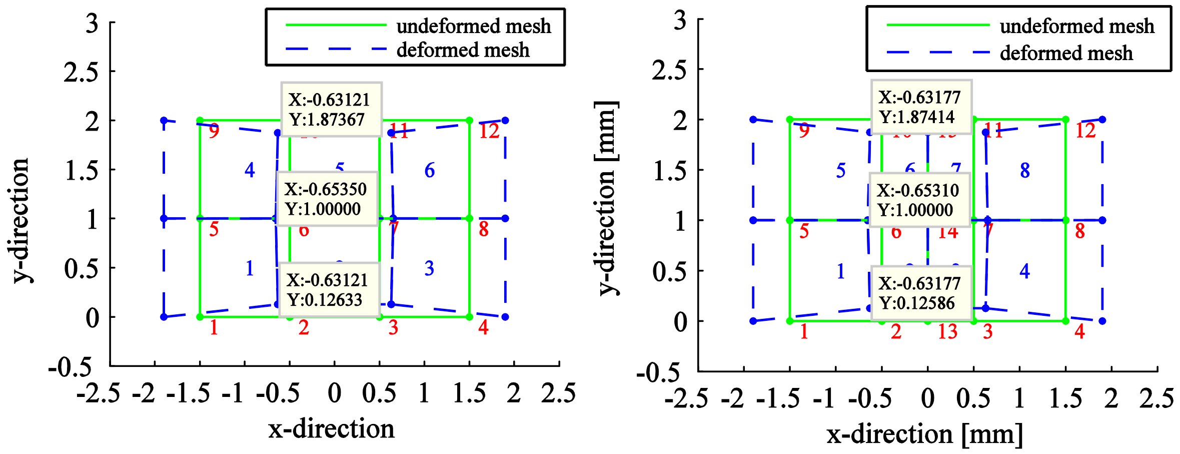


Figure 1.9: Deformed mesh, left FEM and right XFEM

The deformation is symmetric around the y-axis. The nodes [2 6 10] are taken for the comparison. There are slight differences in the deformation. In Figure 1.10 the two different deformed meshes are plotted over each other.

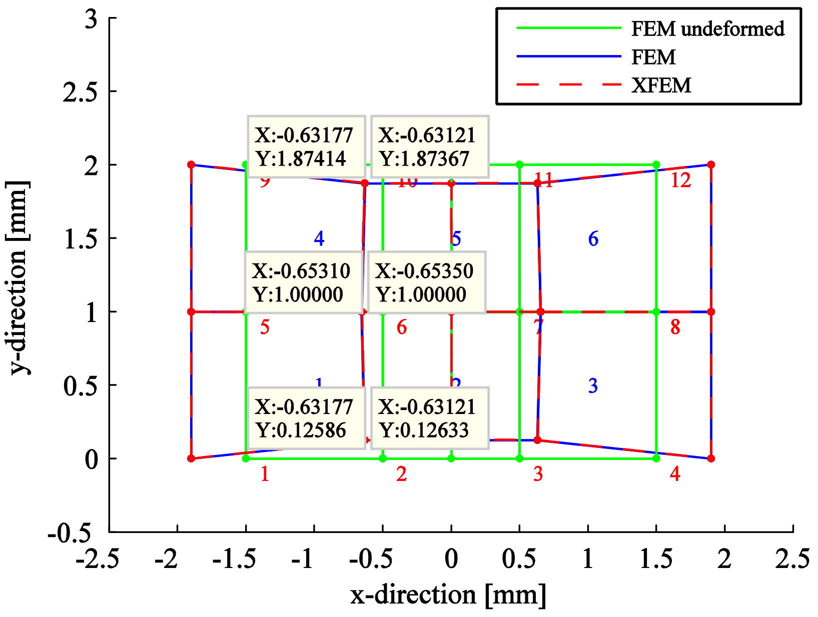


Figure 1.10: Comparison of FEM and XFEM without cut

To investigate the error, the absolute and the relative deviation is calculated.

$$e_{abs} = e_1 - e_2 \tag{1.29}$$

$$e_{rel} = \frac{e_1 - e_2}{e_2} \tag{1.30}$$

The absolute deviation is the difference of the two displacements. The relative is based on the deviation. In Table 1.1, the absolute deviation is on the right side and on the left the relative.

Absolute Deviation [mm]				Relative Deviation to displacement at the node [%]			
Node	2	6	10	Node	2	6	10
x-direction	0.00055696	-4.02E-04	0.00055696	x-direction	0.42448	0.2617	0.42448
y-direction	0.000474	5.63E-18	-0.000474	y-direction	0.37519	41.05	0.37519

Table 1.1: XFEM compared to FEM

The values of the absolute deviations are small. Looking at the relative deviations, the only value which stands out is the relative deviation of node 6 in y-direction. At that node, the absolute deviation has the potency of -18. In numerical terms, it means that the displacement is zero. Comparing the zero displacement to the absolute value does not make sense in this context.

The XFEM method is also compared to a different remeshing method. Instead of creating new nodes around the edges of the cut, the cut element is completely left out.

Three different meshes (XFEM, FEM with remeshing around the edges, FEM with deleting the cut element) are now obtained and compared to each other to see the effects of the deformations.

The first comparison is done between the two FEM meshes,

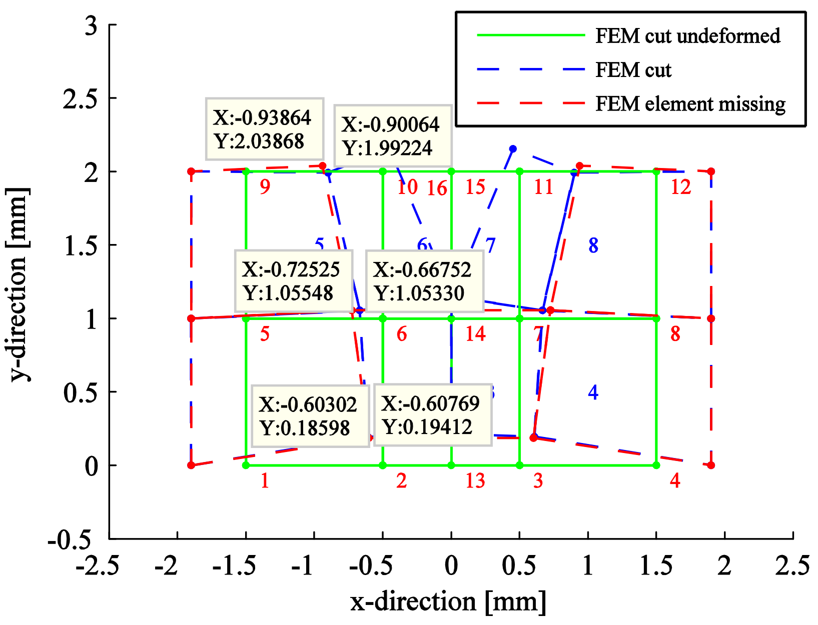


Figure 1.11: Comparison FEM with cut and FEM element missing

Absolute Deviation [mm]				Relative Deviation to displacement at the node [%]			
Node	2	6	10	Node	2	6	10
x-direction	-0.0046612	0.05773	0.038005	x-direction	4.3285	34.461	9.4862
y-direction	0.0081407	-0.0021815	-0.046444	y-direction	4.1937	4.0929	598.29

Table 1.2: Absolute/relative deviation between FEM with cut and FEM element missing

The biggest difference between the two deformations is in node 10. This means that the element 6 of the FEM mesh with cut has the same stress situation regarding to directions as the element 4 in the other mesh. This matter can also be seen, by looking at the movements of the free edge 16 in the FEM mesh with remeshing compared to the node 10 of the other mesh. As the middle element is missing, the nodes 10 have be compared. But the node ten is not a free edge in the FEM mesh with remeshing, so two different situations are modelled which leads to two different solutions. Later in this chapter, the important of the free edges will be more outlined.

In the second comparison, the deformed XFEM mesh is plotted together with the defomed FEM mesh with cut and as reference mesh is the not deformed FEM with cut given,

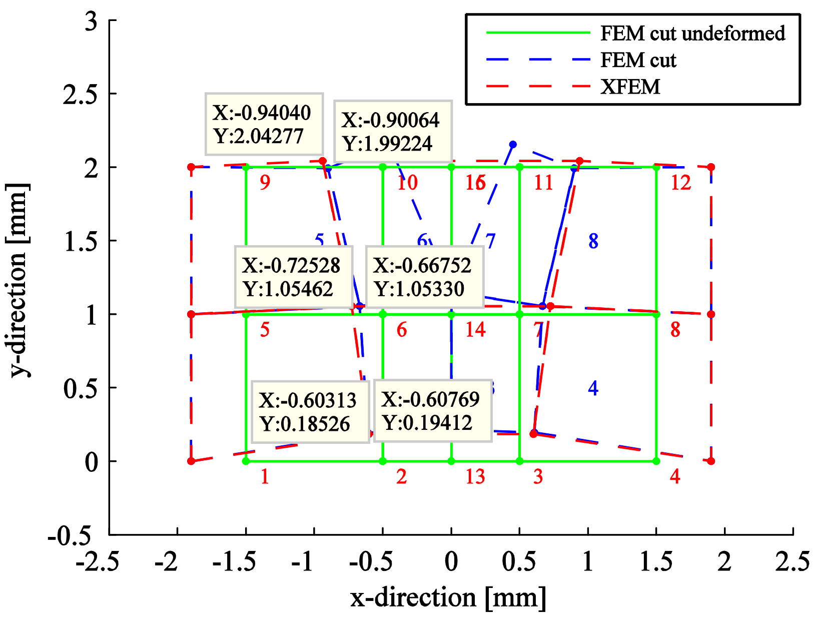


Figure 1.12: Comparison FEM cut to XFEM cut

Absolute Deviation [mm]				Relative Deviation to displacement at the node [%]			
Node	2	6	10	Node	2	6	10
x-direction	-0.0045596	0.057761	0.039761	x-direction	4.2342	34.48	9.9245
y-direction	0.0088591	-0.0013216	-0.050537	y-direction	4.5638	2.4796	651.01

Table 1.3: Absolute/relative deviation between FEM with cut and XFEM with cut

In the table above, the biggest value for the relative deviation occurs at node ten in the x-direction. Comparing the two Table 1.2 and Table 1.3 shows that the deviation is similar in all values. It seems that the XFEM method cancels out the element and creating a free edge at node 10, which leads us to the next comparison between the deformed XFEM mesh and the deformed FEM mesh with the middle element missing, see Figure 1.13.

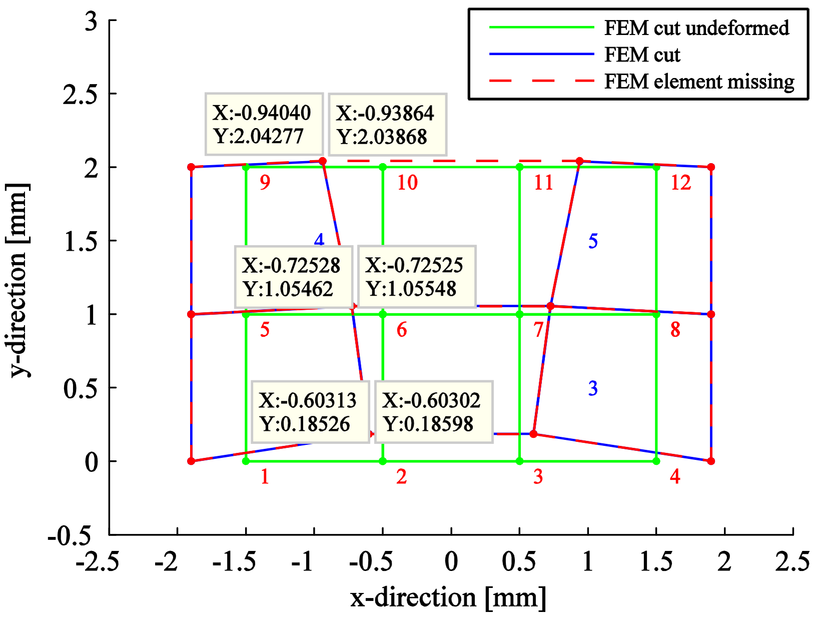


Figure 1.13: Comparsion XFEM cut to FEM element missing

Absolute Deviation [mm]				Relative Deviation to displacement at the node [%]			
Node	2	6	10	Node	2	6	10
x-direction	0.0001016	3.06E-05	0.0017557	x-direction	0.098617	0.013597	0.40026
y-direction	0.00071842	0.00085991	-0.0040924	y-direction	0.3863	1.5499	10.58

Table 1.4: Absolute/relative deviation between FEM and XFEM

Having a look in the table above, all nodes are concurrent. Only the node ten has a relative deviation of 10.58. But it must be taken into account that this value was before by around 600, so it is a better conformance now. It seems that the free edge of the FEM mesh conforms to the missing element more the XFEM method than to the remeshing method.

1.3.3 3D elements

As in the 2D case, there will be performed the same comparison in the 3D case. In the 3D case it is more difficult to develop a FEM mesh using the “remeshing method” which can be compared to the XFEM. A cube with the side length of one is taken and divided up into five tetrahedral, see Figure 1.14.

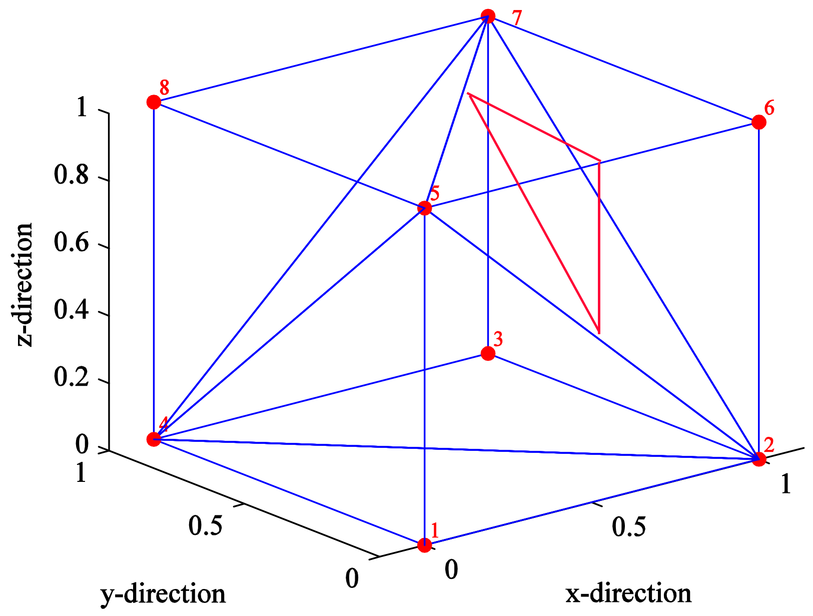


Figure 1.14: Standard cube divided into 5 tetrahedrons

The cut is performed in the element with the edges [5 2 7 6]. The cut plane is situated in the surface of the y-, z-axis with the offset of 0.5. The cut surface of the element is represented by the red triangle in Figure 1.14. As described in the previous chapter, the convergence criterion has to be valid. This means that in the remeshing situation, new nodes are added around the edges of the cut. The neighbour elements have to be also subdivided due to the boundary continuity law.

When the cut surface coincides with an edge of an element, a new node is created which can be seen in Figure 1.15,

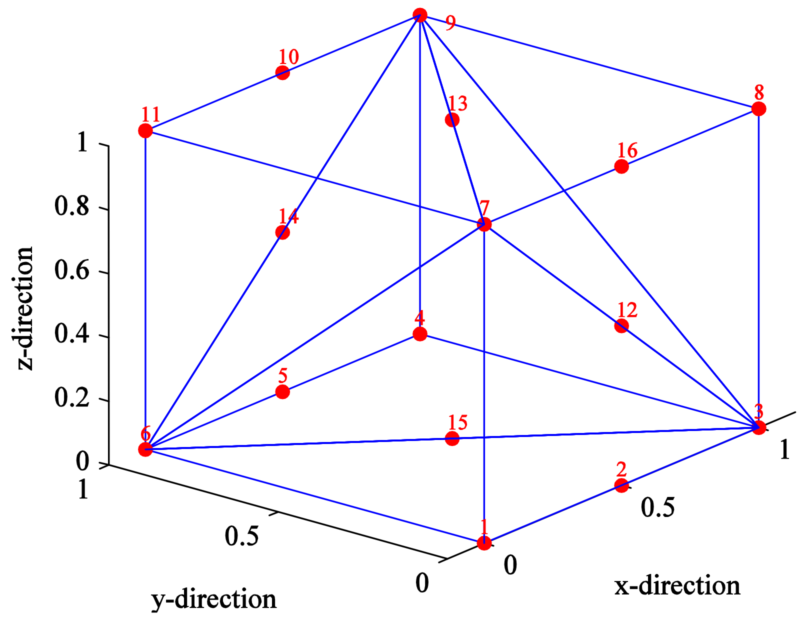


Figure 1.15: Standard cube with added nodes in cut surface

Out of the mesh shown in Figure 1.15, the three different mesh(XFEM, FEM with remeshing and FEM with cut elment missing) can be created. For the XFEM case the node

16 is left out and the cut element possesses the nodes [12, 13, 7, 8], shown in Figure 1.16. The sides of the cut element are marked red.

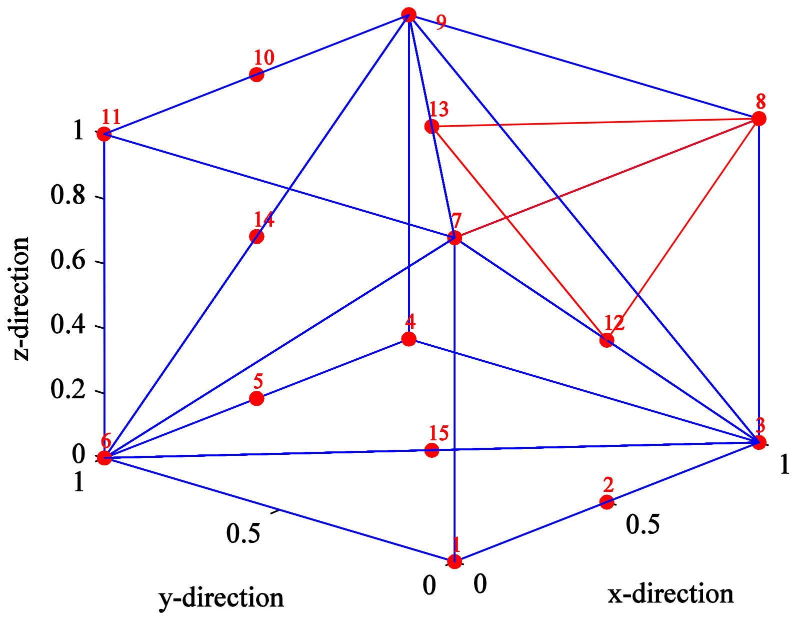


Figure 1.16: XFEM mesh

For the remeshing situation the node 16 is add on the edge of the corners 7 and 8. As in the 2D case the cut element is subdivided into two elements [12, 13, 7, 16] and [8, 13, 12 16]. By adding the new node 17 at the node 16, a gap between the two elements are possible.

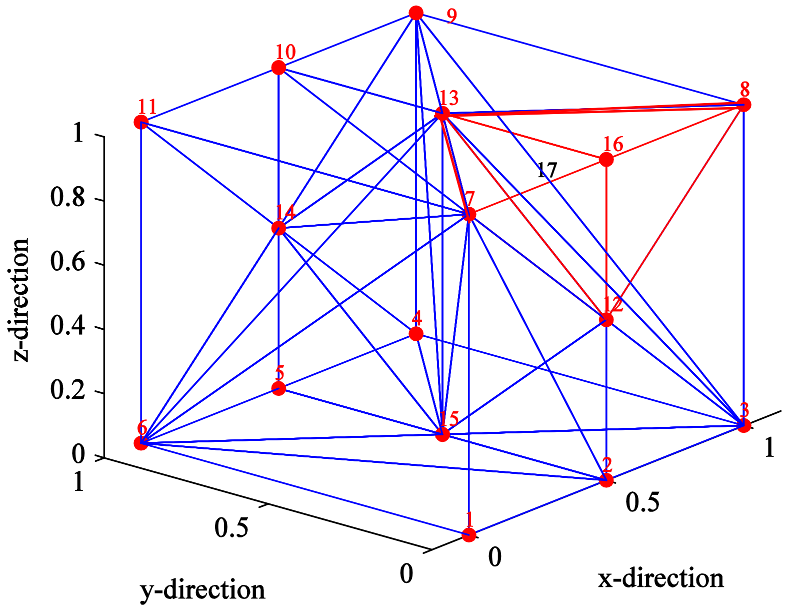


Figure 1.17: FEM mesh with cut element

For the last mesh the FEM with the element missing the cut element [13, 12, 7, 8] is left out. Now the nodes where x equals one are pulled in x-direction by 0.3 mm, but they still can move freely in y- and z-direction. First a comparison between the two FEM methods is performed,

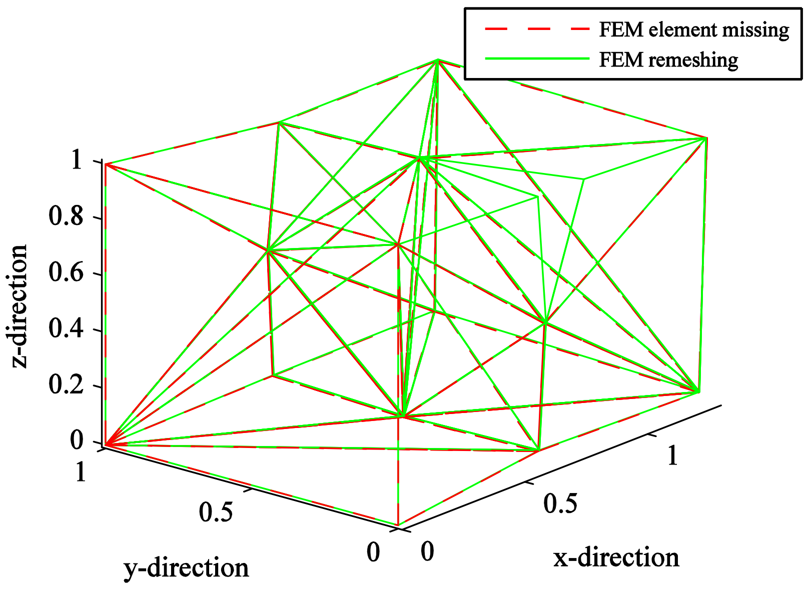


Figure 1.18: Comparsion FEM element missing to FEM cut

Absolut Deviation [mm]											
Node	2	3	4	5	8	9	10	12	13	14	15
x-dir.	-0.0002	0.0000	0.0000	0.0009	0.0000	0.0000	-0.0019	-0.0013	-0.0027	0.0003	0.0001
y-dir.	-0.0040	-0.0030	-0.0034	-0.0035	0.0040	-0.0027	-0.0023	-0.0077	-0.0029	-0.0031	-0.0037
z-dir.	0.0047	0.0029	0.0052	0.0039	-0.0012	0.0058	0.0048	0.0058	0.0077	0.0039	0.0051
Relativ Deviation to displacement at the node [%]											
Node	2	3	4	5	8	9	10	12	13	14	15
x-dir.	0.1178	0.0000	0.0000	0.5734	0.0000	0.0000	1.2681	0.8992	1.6573	0.2422	0.0477
y-dir.	8.8145	4.8241	9.7397	11.3430	10.1910	5.9507	5.0375	19.5800	16.7860	8.0146	31.0840
z-dir.	11.9950	20.7890	11.5300	10.7200	1.5982	10.1870	8.1124	53.2320	14.6700	70.3960	19.1300

Table 1.5: Absolute/relative deviation between FEM with element missing and FEM cut

In the 3D case it is hard to see effects of the deformation straight from the visualization of the mesh. It is better to compare the absolute and relative deviations. In Table 1.5 the absolute deviation for the FEM remeshing method compared to the FEM without the cut element are compared. All absolute deviations which are higher than 0,004 are highlighted yellow. For the relative deviation, the nodes with higher value of 20, are highlighted. The differences of the relative deviation are especially huge in z-direction.

In Figure 1.19 a comparison is made between the XFEM method and the FEM with remeshing. The red line represents the XFEM mesh and the green the FEM with remeshing.

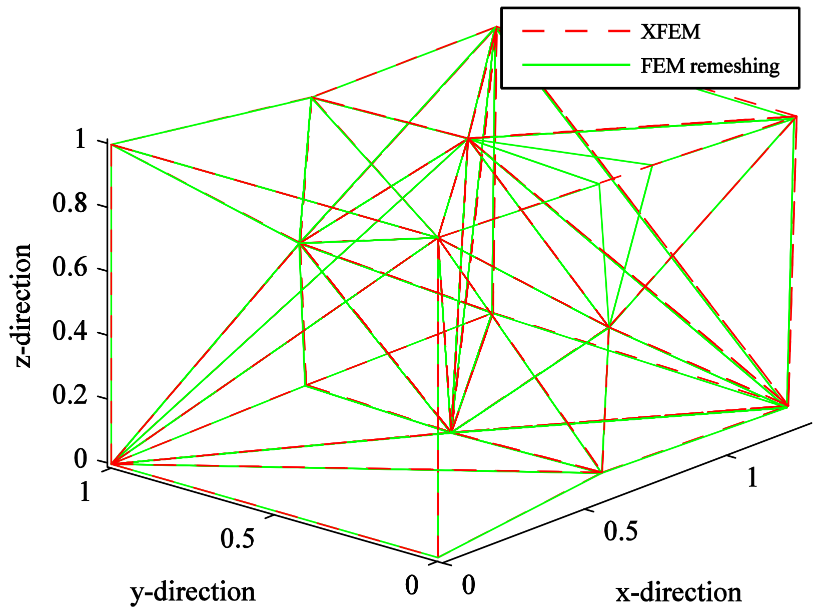


Figure 1.19: Comparison XFEM to FEM remeshing

Absolut Error [mm]											
Node	2	3	4	5	8	9	10	12	13	14	15
x-direction	0.0008	0.0000	0.0000	-0.0015	0.0000	0.0000	0.0028	0.0047	0.0048	-0.0005	0.0001
y-direction	0.0061	0.0076	0.0078	0.0058	0.0040	0.0079	0.0045	0.0089	0.0056	0.0055	0.0058
z-direction	-0.0073	-0.0105	-0.0081	-0.0057	-0.0088	-0.0088	-0.0069	-0.0095	-0.0100	-0.0057	-0.0076
Relativ Error to displacement at the node [%]											
Node	2	3	4	5	8	9	10	12	13	14	15
x-direction	0.6194	0.0000	0.0000	1.0389	0.0000	0.0000	1.8445	3.1099	2.9459	0.4594	0.0684
y-direction	12.4420	11.8770	24.6890	21.3060	11.2470	18.5780	10.5270	18.9910	39.3320	15.3850	37.4180
z-direction	21.0990	94.4030	20.2260	17.5230	11.8540	13.8690	10.7320	56.5220	16.5620	59.8600	34.8290

Table 1.6: Absolute/relative deviation between XFEM and FEM remeshing

In Table 1.6: the absolute and relative deviation of the comparison between the XFEM and FEM remeshing is performed. The highlighting system is used as above. Here the y- and z-direction are out of place. All deformation in y- and z-direction mismatches.

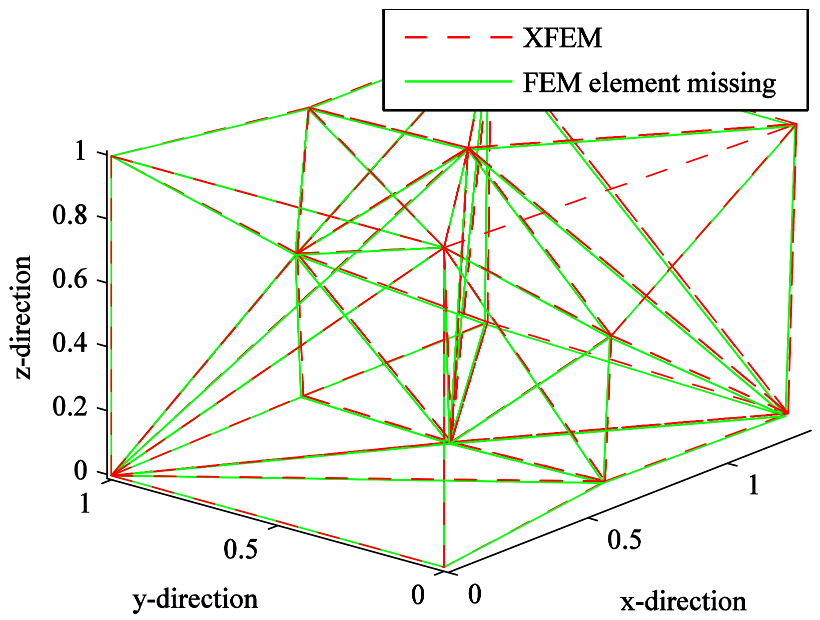


Figure 1.20: Comparison XFEM cut to FEM element missing

Absolut Error [mm]											
Node	2	3	4	5	8	9	10	12	13	14	15
x-direction	0,0006	0,0000	0,0000	-0,0007	0,0000	0,0000	0,0009	0,0033	0,0022	-0,0003	0,0002
y-direction	0,0021	0,0047	0,0044	0,0023	0,0080	0,0052	0,0022	0,0012	0,0027	0,0024	0,0021
z-direction	-0,0026	-0,0076	-0,0029	-0,0018	-0,0100	-0,0029	-0,0021	-0,0036	-0,0023	-0,0018	-0,0024
Relativ Error to displacement at the node [%]											
Node	2	3	4	5	8	9	10	12	13	14	15
x-direction	0,5024	0,0000	0,0000	0,4595	0,0000	0,0000	0,5998	2,2386	1,3375	0,2161	0,1161
y-direction	4,7245	7,6259	12,5450	7,5469	20,2920	11,5220	4,9596	3,1297	15,9440	6,1372	17,9650
z-direction	6,5737	53,9880	6,3636	4,9248	13,2630	5,0954	3,4907	33,3770	4,3218	31,6030	9,0357

Table 1.7: Absolute/relative deviation between XFEM and FEM element missing

A better conformity is achieved in the last case when the XFEM method is compared to the standard FEM with the cut element missing.

1.4 Discussion

The goal of the work is the real-time simulation of a cut procedure. Additionally, the visual effects of the deformation should be obtained. The big advantage of the XFEM is that there is no need to mesh and remesh the discontinuity surfaces. Comparing XFEM and FEM, the computational cost of the enrichment is lower than the cost for remeshing due to the stiffness matrix. Using the XFEM, the visualisation of the cut edges are missing or have to be done in a post processing algorithm, because the discontinuity is restricted to the edges of the mesh. The cutting force is predicted by a different model which is decoupled from the FEM calculation. It means that there are two models, one for visualisation and one for predicting the cut forces. There could be a possibility to get the two models together in the XFEM. The origin of the XFEM is the propagation of various discontinuities through a material. The amount of propagation is defined by the load at the break point of the discontinuity. This is formulated in an equation. Instead of defining the load as propagation, the velocity of the knife of the body can be taken, so the force model for the cut and the deformation model can be integrated.

As the remeshing can be really time consuming, it can be considered to delete the cut element out of the mesh instead of remeshing around the edges of the singularity. It can be a fair approximation, considering that small chippings are generated by a cut. The elements must have the same size as the absorbed material, so the mesh has to be fine around the cut. In medical surgeries, the position of the cut is normally known before the operation. By refining the area of the cut, an easy cut algorithm can be implemented. There are two ways of cutting the element. First, if the cut is near the edges of the element, it takes place between the elements by decoupling the nodes. Second, if the cut is in the middle, the entries of the stiffness matrix are deleted, so no connection between the nodes exists.

2 Experimental set-up

In this chapter there is given a description of the hardware components of the experimental set-up. The tension platform fulfils the task to set the test object under a certain pre-tension before the cut. The platform is also used for determine the material properties of the silicone. The scanning device scanCONTROL 2700-100 from the company Micro Epsilon allows measuring the real deformation of the test object around the cut edges and comparing it in a post-processing algorithm with the results of the simulation. In a previous project at the department of applied mechanics at the Technical University of Munich, a modular robot has been developed by Mr. Dr. Robert Engelke. The robot is called the blue robot due to his painting. A scalpel from the company Sollingen is used to perform the cut. It is mounted to the end effector of the blue robot.

2.1 Tension platform

To get a predefined tension in the test object, a tension platform has been developed. Using an adjustment mechanism, a predefined tension can be exposed to the object. In Figure 2.1, a CAD drawing of the tension platform is shown.

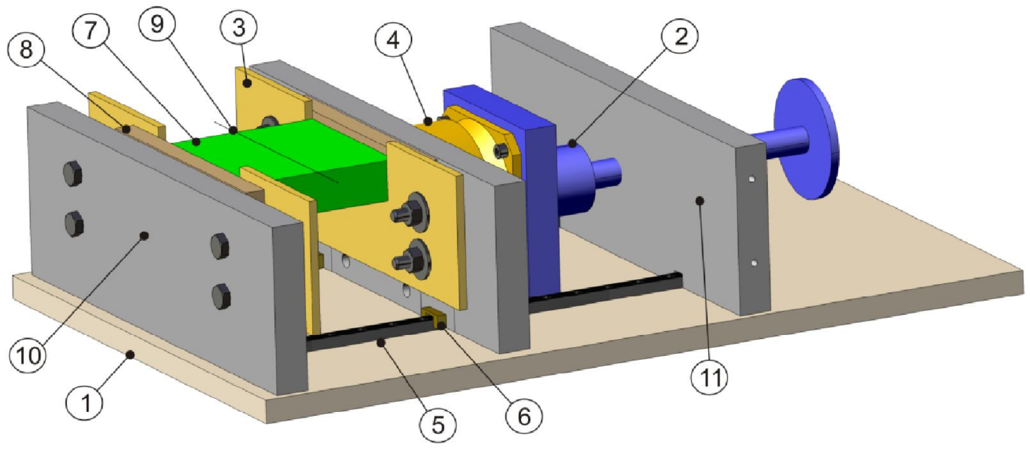


Figure 2.1: Tension platform

The tension platform consists of the following components:

1. Ground plate
2. Adjusting mechanism
3. Mounting of test object
4. Force sensor
5. Profile rail
6. Profile trolley
7. Test object
8. Hard silicon for mounting

9. Cut line of scalpel
10. Stretching block
11. Fixing of adjustment mechanism

In the following a detail description of the shape, material type and mounting of the test object will given.

2.1.1 Shape of test object

The tension platform should be suitable for different kinds of test objects. Within of the collaborative research project SFB 453 an abdominal surgery should be imitated by a test object. The platform can be used for two applications. In Figure 2.1, the first application is shown, where the fixing block of the adjustment mechanism is mounted at a distance of 300 mm to the stretching block. In this application, test objects of the size of 150x300 mm can be mounted. In the second application, the fixing block can be disarrange to the end of the ground plate, so test objects of the size 300x300mm fit in. The height of the stretching block is 100 mm. An average human being has a maximal abdominal of 200x200 mm, so platform can be used for later experiments on more complex test objects.
First, a rectangle block is chosen for the shape of the test object. The block is easy to mesh for the later FEM simulation and the mould will not be of a complicated shape.

2.1.2 Mounting of the test object

To ensure a proper mounting of the test object, different techniques have been investigated. Silicone possesses the property not to stick to anything, so it cannot be just glued to a plate. The first idea was to clip the edges of the silicone block between two metal plates. There occur two problems. At first it is hard to define the boundary condition for the later simulation and secondly the silicone is a soft tissue material which will slip through the metal plates after a certain tension.
Another idea was to mould the fixing into the test object, see Figure 2.2. As fixing an aluminium bar was chosen. Now the bar can be mounted over screws to the stretching block. The same is done for the fixing at the adjustment mechanism. By pulling the two ends apart, tension is applied to the test object. Where the edges of the moulded bar touch with the silicone block, a tension peak will occur, so a cracks occurs already after a small amount of load and will lead to a break of the mounting.

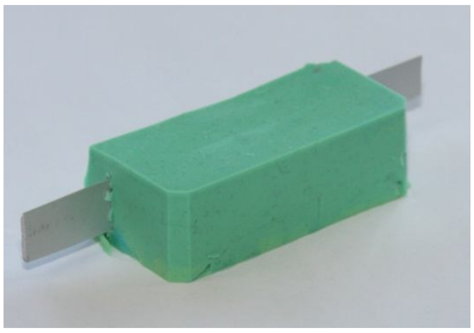


Figure 2.2: Mounting moulded into the test object

The developed solution is to use two different kinds of silicones and glue them together by themselves, see Figure 2.3. The middle part has much lower elasticity than the two outer parts. The test object can be mounted to the tension platform by squashing the stiffer silicon. The hard silicone block for fixing also deforms at a certain loading and will influence the deformation of the soft part. So, there should be drawn attention that there is no significant deformation of the hard silicon part.

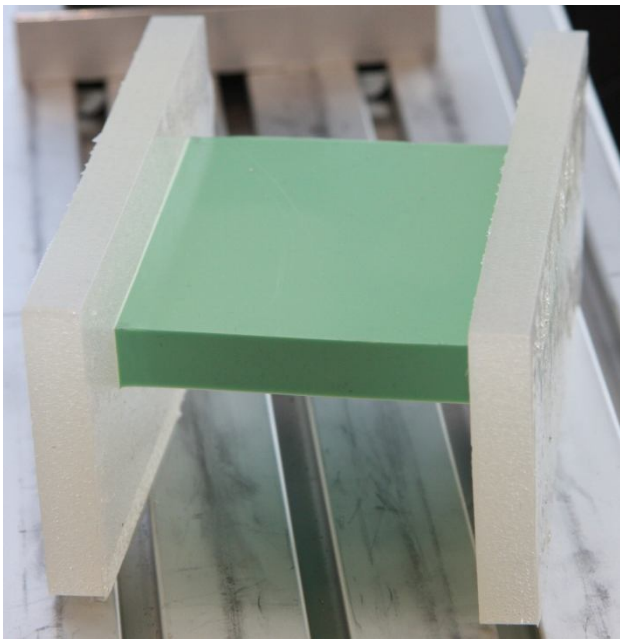


Figure 2.3: Mounting moulded into the test object

The procedure of creating the silicone blocks is to first mould the two stiffer silicone blocks. Afterwards a mould with two open ends is used. At the open ends the two stiffer silicone blocks will be placed, so during the moulding process the middle part can glue to the outer parts.

2.1.3 Material properties of test object

The material properties have to fulfil certain requirements, as well being similar to the human skin as being suitable for the FEM simulation. As described in the previous chapter the material law is linear elastic, which means that the chosen material has to satisfy this condition. The material should also have the behaviour of a soft tissue material, to deform like a human skin. This is a trade off, because most soft tissues have nonlinear material behaviour. As a solution soft silicone rubber is chosen. It is not so soft as skin, but as it can be seen in the chapter “Result”, silicone rubber deforms linear to certain loadings. There exist many types of different silicones rubber. As another requirement is to create different kind of shapes, a moulding material is used to create the wanted shapes by our self. The chosen moulding material comes from the company Breddermannm Kunstharze, and is named silicone-moulding material, shortening SI6GB. The first two letters of the shortening stand for silicone and the number stands for the hardness of the material. It is two component combinations of silicone rubber and hardener with a fast processing time.

In our application the test object is a block consisting of silicone. In the adjusting mechanism, the vertical displacement of the test object can be adjusted by a jack screw over the round grip at the end. The profile rail prevents movement the in the y-axis and ensure that the force sensor measure only forces in x-direction. The friction of the profile rail is small compared to the forces in x-direction.

To determine the material properties of the silicone which are not given by the producer, the force sensor from the company Schunk is used. The displacement between the two tension plates is measured and plotted against the corresponding tension. The elasticity modulus can be found with the help of the Hooks law. For Poisson ratio the displacement is compared to the side contraction of the test object. A more detail description will be given in the chapter five “Result”.

2.2 Description of the “Blue robot”

The cut is performed by the blue robot (see (1) in Figure 2.4), which has been developed in a previous project at the Institute of Applied Mechanics. There exists different configuration for the robot. The chosen configuration has four joints, see Figure 2.4. For the first step the cut is of a straight line, so the first joint is locked. For more information about the set up of the robot, see [10]. For a detail description of the inverse kinematic of the robot configuration and of the trajectory, it is referred to [11].

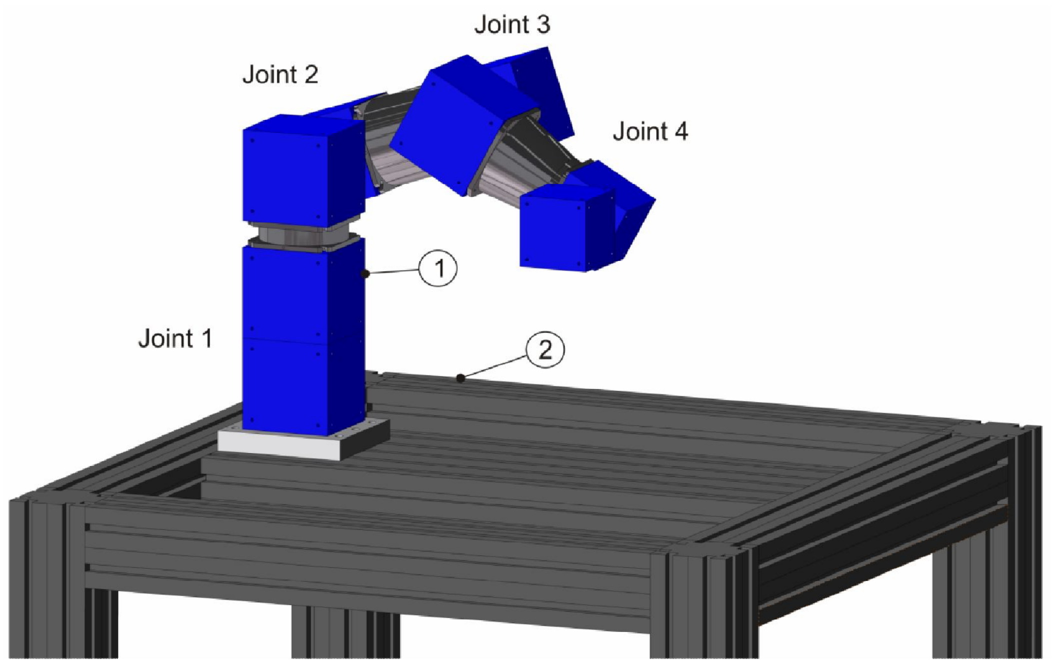


Figure 2.4: Test bench with blue robot

2.3 2D Scanning of test object

A rectangular solid, which is used in this part of the project, can be described by straight lines. In later projects, more complex shapes will be used. For this purpose, the exact dimension of the test object has to be determined. After obtaining the outer dimension of the test object, a mesh generator helps to fill the object with finite elements. The device scanCONTROL is mounted to the end effector of the blue robot to scan the outer surface of the test object. There are two different configurations to mount the 2D scanner on the end effector, see Figure 2.5.

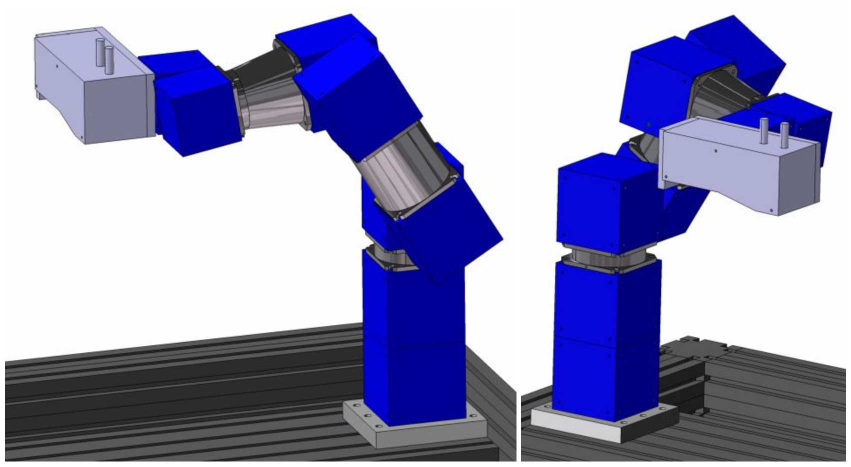


Figure 2.5: Different configuration for mounting the scanCONTROL

In Figure 2.6, the scanning of a test object is shown. In this work the scanning is not done for the purpose of generating a mesh, but the results of the simulation will be compared to the real deformation by the scanCONTROL.

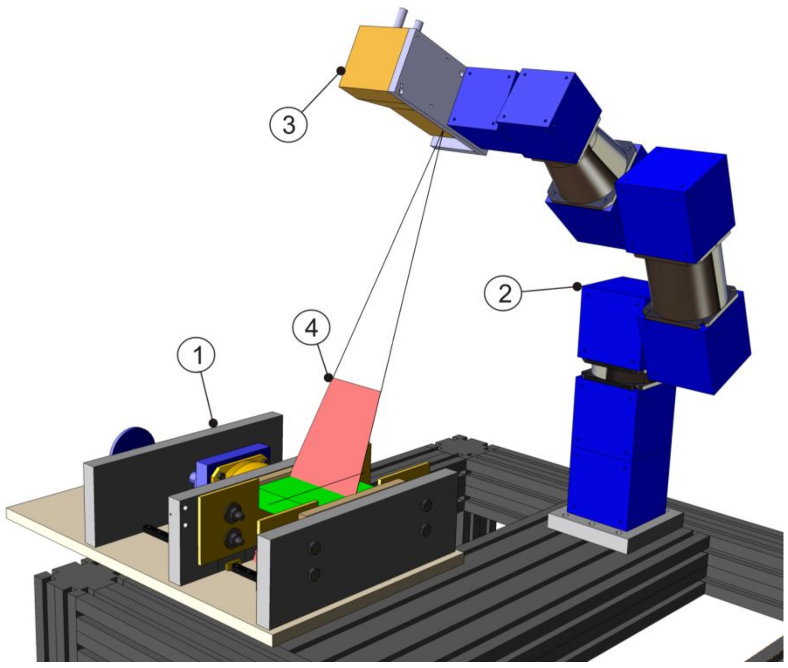


Figure 2.6: Scanning of the test object

The Scanning configuration consists of the following components:

1. Tension platform
2. Blue Robot
3. scanCONTROL
4. Effective range

Two routines are written for the scanning the object. At first the scanning coordinates are transformed via the direct kinematic into the absolute coordinate system. In this approach the scaling of the robot is crucial. If the dimension of the robot is not exact determined, the different measurements from different position of the blue robot cannot be compared to each other.

In the second routine the edges of the test object are detected by the scanCONTROL and taken origin for the absolute coordinate system. It means that the absolute coordinate system lies on the top of the silicone block. A more detailed description of the two routines can be found in chapter 4 “Software Implementation”.

2.4 Fixing of the scalpel at the robot

The scalpel is mounted to the end effector of the blue robot. For the configuration of the blue robot, the scalpel is mounted to modul 70 of the robot, see Figure 2.7.

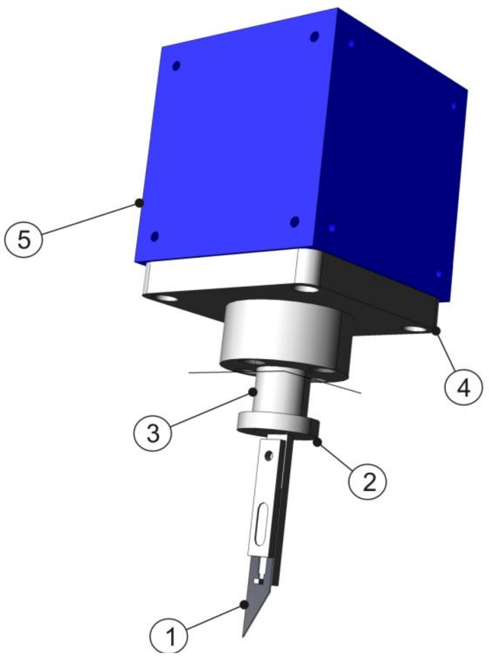


Figure 2.7: Fixing of the scalpel to the end effector

The mounting of the scalpel consists of the following components:

1. Scalpel
2. Attachment of the scalpel
3. Six axis force sensor form the company Automation Instrument
4. Mounting plate to modul 70mm of the blue robot
5. Modul 70mm

A force model for the cut process will be developed. To get experimental data, the scalpel is not mounted directly to the robot, but there is a 3D force-momentum sensor attach between. Using the measured position and velocity of the scalpel a force model for the cut process can be developed.

2.5 Experimental setup online calculation

In the experimental setup the FEM calculation is controlled online, i.e. during the cut measurements with the scanCONTROL take place and are compared online to the simulation. The scanCONTROL device is not mounted to the blue robot for this application. A mounting device has been developed to fix the scanCONTROL at a certain hight and at a certain angle, see Figure 2.8. The mounting device consists of the following components:

1. Base plate
2. Shaft
3. Fixing plate for scanCONTROL
4. Twist mechanism
5. Effective range

The shaft can be fixed in different heights. The twist mechanism makes it possible to adjust certain angles. Particular attention should be paid to the effective range of the scanCONTROL. As it is not possible to measure straight over the test object due to the movements of the blue robot, the scanning takes place from the side. There occurs certain measurement shadows when scanning from the side, i.e. some regions cannot be scanned.

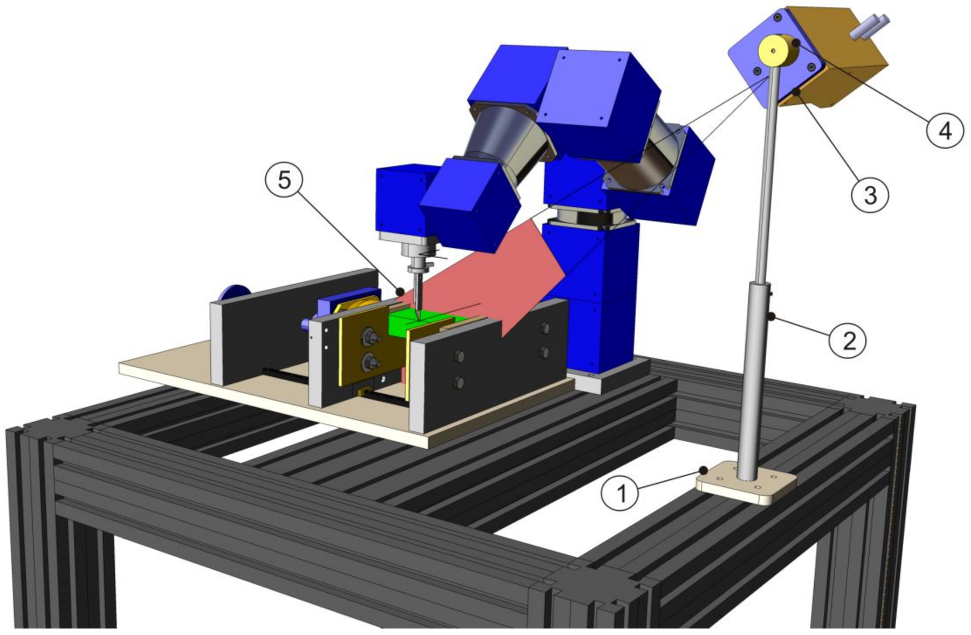


Figure 2.8: Experimental set-up online calculation

3 Software implementation

In this chapter, the programming structures for different applications are explained. Before starting with a FEM simulation, a mesh must be created. To keep track of the numbering system in our mesh, a self-written Matlab routine is designed, introduced in section 4.1. The rest of the written routines are implemented in programming language C++ on the platform Microsoft Visual Studio. In section 2.2, there is given a short introduction to C++. Afterwards the implementations of FEM programmes with their function are presented. When the incision is performed in the FEM mesh, a remeshing method is suggested to solve the problem. The device scanCONTROL 2700-100 creates a profile of the deformed test body. A algorithm is written to transform the local coordinates of the scan device into the simulation, presented in section 4.5. For the visualisation, the toolkit OpenGL is chosen for creating images on the computer screen.

In the following section, two types of variables are present. The variables presented in the theory part, do not automatically coincide with the variables used in the programme. In order to differ the variables, the theoretical ones are written with fat letter and the programming variables with cursive letters. Names of programme functions are in cursive letters with quotations mark.

3.1 Create Mesh with Matlab

At first a mesh is used which is created by a self-written Matlab routine. It is easier to keep track of the numbering system when it is known how the mesh was created. This will help later by the development of a remeshing method, because the ordering of the nodes will be known. In later project, an external programme will be used to mesh more complex geometries. For instance the programme Tetgen can be used to perform this task. As elements types tetrahedral are chosen, because on one hand they preserve good approximation results due to the shape of the element and on the other hand the linear shape functions are easy to derive.

In the Matlab routine "*Create_Mesh*" the whole algorithm is implemented to create a mesh with tetrahedral for the later calculations in the FEM calculation in C++. The routine needs three different types of inputs. As mention before the shape of the test object is a rectangular bock. The size of it is defined by the three parameters *scx*, *scy* and *scz* which are the length of the corresponding sides. The number of cubes per side is the second input and represented by the variables *lxc*, *lyc* and *lzc* for every direction. For the parameters *lxc* and *lyc*, only even positive numbers can be chosen due to the mirroring at the coordinate axis, which can be seen later in this chapter. But the variable *ly_plus* also accepts uneven numbers. The last input of the mesh creation is the feature show or hide. The variable *zeig* suppresses the plot if it is not one, if not it will be hidden. If the figure is plotted, it can be chosen, if the number of nodes or number of elements will be shown in the plot. For these parameters the same rule is applied as for *zeig*, if it is one it will be shown, otherwise it will be suppressed.

The basic idea of the mesh creation is that first a basic cube with the length of one is filled with tetrahedral and then mirroring it until a block is created. Then this block is duplicated until the wanted number of block is reached for every coordinate axis. In the end, the block is scaled according to the input parameters.

The function "*geom_tet*" creates out of the described input the block filled with tetrahedral. The function is divided into 8 different steps. First there is a basic cube defined, as it can be seen in Figure 3.1.

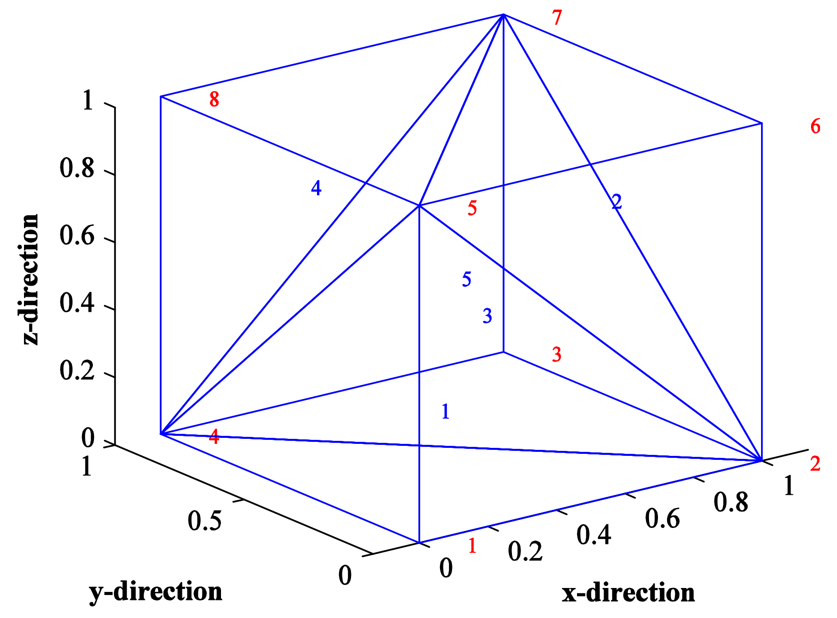


Figure 3.1: Standard cube

One cube cannot be set one to another by changing the coordinates of the edges in each corresponding direction, because the diagonals are not covering on the outer surfaces. This would violate the compatibility requirement or also sometimes called the conforming requirement. It states that the approximation of the unknown variable must be continuous over the element boundaries. If the cubes are connected one to another, the diagonals of the touching surfaces will cross each other and the continuous boundary condition for these elements will be violated.

In order to avoid this violation, the basic cube is mirrored in the second step on x-axis at the surface with the node numbers (1,4,5,8),see Figure 3.2.

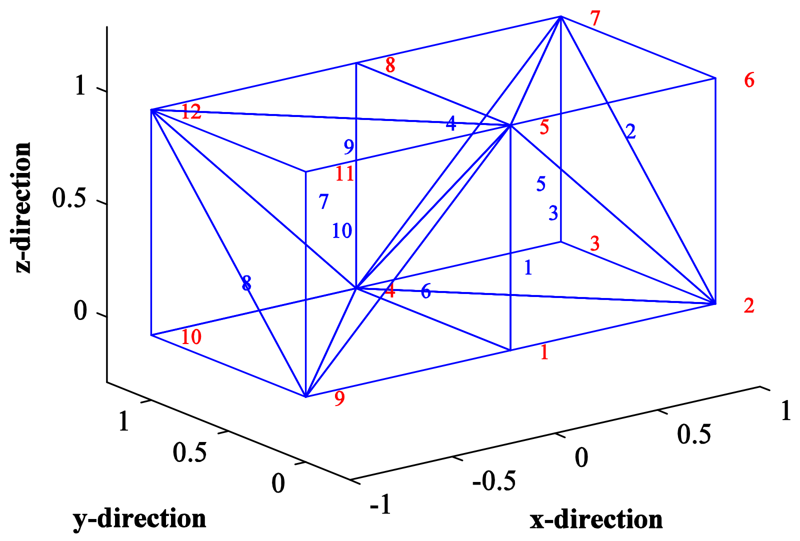


Figure 3.2: Basic cube mirror at the x-axis

As it is seen now, the contact surfaces have a common diagonal which allows continuity over the boundaries of the two elements. The same thing is done by mirroring the body above at the y-axis, see Figure 3.3.

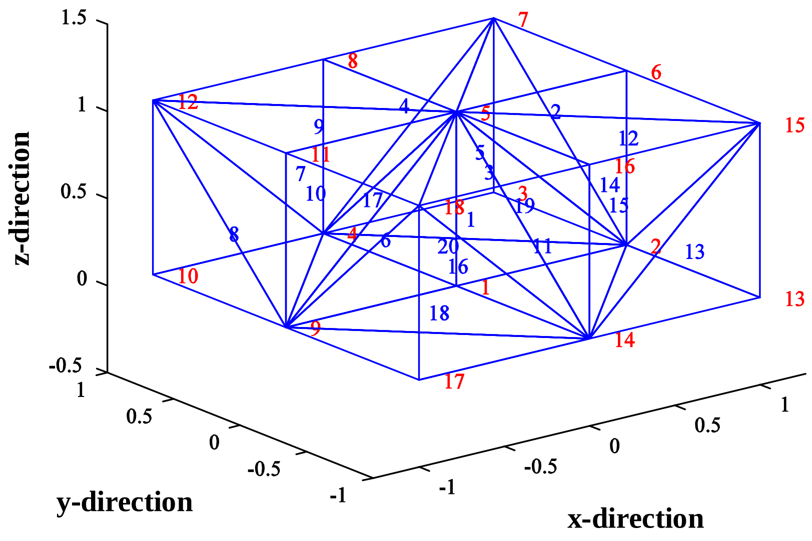


Figure 3.3: Cube mirrored at x- and y-axis

In the last step the four cubes filled with tetrahedral are mirrored at the z-axis, see Figure 3.4

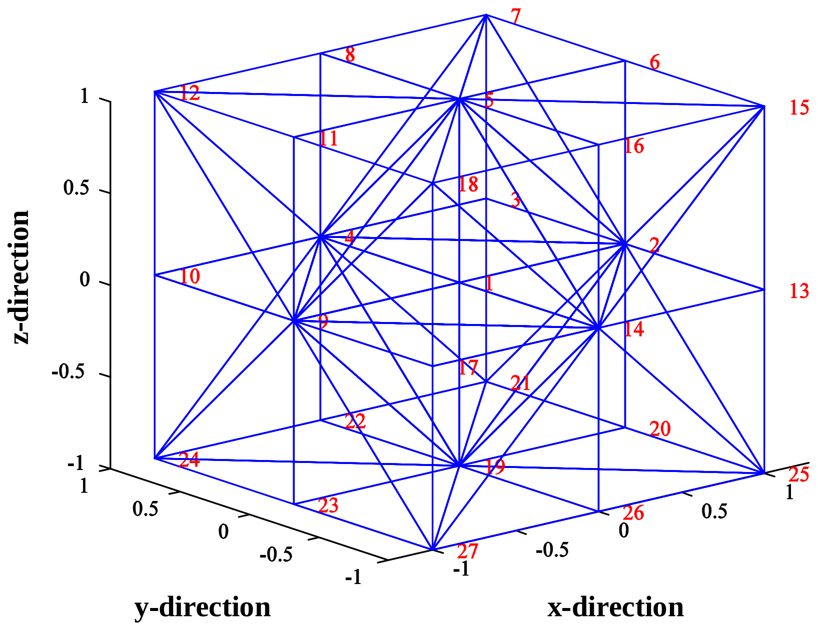


Figure 3.4: Uniform cube

The principle described above, is performed by the function *"init_Edof"*. This function needs no input and gives as a result the matrix **node** and **Edof**. The matrix **node** contains the coordinates of the nodes. The matrix **Edof** which is called the Topology matrix contains the numbering of each element. This means that the row represents the number of the element and in the column stands the edge numbers of the element.

Out of the basic cube, a new cube is obtained which is called uniform cube. This cube has a conforming outer surface when placing the cubes in a row. With the function *"cal_Edof_node_lx"*, the initial cube has been added in the x-direction by another by the variable *lx*. For instance, this variable is one, one uniform cube is been added and so on. The functions *"cal_Edof_node_ly"* and *"cal_Edof_node_lz"* are doing the same thing, but instead of a cube it is the whole row of the previous created object.

In step 6, the scaling the matrix node takes place, that one edge lies at the point of origin. The length is scaled by the parameters *scx*, *scy* and *scz* for every coordinate.

As output the matrices *node* and *Edof* are obtained in step 7. In the matrix *node* the coordinates of every nodal point are save. The matrix is organized as follows,

$$node = \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots & \vdots \\ nnod & x_n & x_n & x_n \end{bmatrix}. \tag{3.1}$$

The subscript *n* stands for the total number of elements nodes. Therefore the row represents the numbering of the nodes. For instance, for node number one, we have the coordinates in row one.

Another important matrix is the *Edof* called the Topology matrix. In this matrix, the numbering of the elements is saved, so the information is kept which node belongs to which element. The *Edof* matrix is implemented in the program as follows:

$$Edof = \begin{bmatrix} 1 & node\_nr1 & node\_nr2 & node\_nr3 & node\_nr4 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ nele & node\_nr1 & node\_nr2 & node\_nr3 & node\_nr4 \end{bmatrix} \tag{3.2}$$

The FEM algorithm presented in this thesis is written without the use of a seperated degree of freedom matrix, so with the two matrices *Edof* and *Node*, the calculation of a FEM deformation can determine. In the next section follows a short introduction to the C++ programming language, especially dealing with matrices.

3.2 Introduction of the C++

The programming languages C++ is a standard high-level programming language and has gone through a huge development since the first appearance on the market. It was created since 1979 from Bjarne Stroustrup from the AT&T as an extension from the programming language C. C++ is designed for different programming paradigms, i.e. object orientated, generic and procedural programming. C++ allows as well an efficient and machine-oriented programming as a programming with a high degree of abstraction.[12]

In the FEM calculation, the stiffness matrix becomes high order due to the degree of freedoms at the nodes. Dealing with matrices in C++ is involved with some restrictions. For every entries of a matrix a certain place in the memory has to be reserved. For dynamic matrices, i.e. change of size, the memory space has to be adapt to the situation. There exist many different ways to do that. It is possible instead of using an own written library the C++ Standard Library [13] vector<> template class. The disadvantage of this class is that some compiler vendors do not get the best performance out of elementary operations, because it is so feature-rich. Also included in the C++ Standard Library is the class valarray<>. At one time, this was supposed to be a vector-like class that was optimized for numerical computation, including some features associated with matrices and multidimensional arrays. However, the valarray classes were not designed very well. The class was introduced by a committee, but they could not come to a consensus for a final definition of the class, so the testing phase of the class is missing. Unwanted failure can occur during implementing an algorithm. The result of this history is that C++, at least now, has a good class for vectors and no dependable class at all for matrices or higher-dimensional arrays. These reasons lead to provide an own written and basic library for vectors and matrices.[14]

The ideas of the memory allocation for the elements of the matrices are copied from the book “Numerical Recipes”[14] for this part of the project and adapt to our problems. The final implementation the matrix library is done in the header “matrix_fct.h”. For a detail description of functions, it referred to the appendix D.

3.3 Static FEM calculation

In the finite element calculations, the differential equations, which describe the physical problem, are assumed to be held over a certain region. This region can be 1D, 2D or 3D. The characteristic feature of the finite element method is that instead of seeking after approximations directly over the entire region, the region is divided into smaller parts, the so called finite elements and the approximation is then carried out over each element.

In this work the problem is stated in a 3D manner. In the mesh generation, tetrahedral were chosen for the element type. The FEM calculation is performed in 5 basic steps:

1. Input parameters
2. Initialisation of matrices and loading from TXT-file
3. Calculation of stiffness matrix
4. Applying boundary conditions
5. Solving equation

3.3.1 Input parameters

At first the define directories have to be filled in for the corresponding mesh parameters in the header “stdafx.h”. These parameters obtain from the mesh generator are subdivided into three different groups boundary, cut and TXT-file parameters. The boundary conditions are different for every load situation. In our project, the load situation is equal for every experimental set-up. The silicone block is fixed on one end and pulled on the opposite site with the distance of the parameter *DIST*. The parameter *NNOD_CUT* saves the number of cut node in the block. This number changes the size of the global stiffness matrix as a new node is added to the system. Secondly, the dimension of the cut surface is saved in the cut properties *CUT_SX*, *CUT_SY* and *CUT_SZ*. At last the text file names has to be saved into the define directories *EDOF_TXT*, *NODE_TXT* and *BC_TXT*, respectively.

3.3.2 Initialisation of matrices and loading from TXT-file

Before working with the matrices, memory space has to be reserved for every matrix needed in the programme. These procedures are done with the help of the library “matrix_fct.h”.

The needed matrices are the topology matrix *Edof*, the coordinate matrix of the nodes *Node* and the boundary matrices *Bc_0* and *Bc_100*. These five matrices are loaded out of the three corresponding TXT-files defined in the header “stdafx.h”. The matrices *Edof* and *Node* are directly loaded out of the TXT-file, i.e. if a number stands for instance in column three and row two it is loaded in the element of the matrix with the indices three and two. The silicone block is locked on one side for every degree of freedom. The corresponding numbers of nodes are saved in the variable *Bc_0*. The opposite side is pulled in y-direction and the need nodes for that deformation are saved in the variable *Bc_100*.

By the calculation of the stiffness matrix an element-wise assembling takes place which means that the global stiffness matrix for the whole system is predefined in the variable *K*.

3.3.3 Calculation of stiffness matrix

The calculation of the stiffness matrix takes place in a “for” loop for every element and six steps have to be performed for every element after equation (1.20):

1. Derivates of Local Shape Functions $\frac{\partial \boldsymbol{\Phi}_i}{\partial \xi_j}$ along the local coordinates
2. Coordinates of the current element $\boldsymbol{X}^{(e)}$
3. Jacobi matrix **J,** Transpose of Jacobi matrix $\boldsymbol{J}^T$ and Inverse of Jacobi matrix$(\boldsymbol{J})^{-1}$
4. Derivative of Global Shape Function $\frac{\partial \boldsymbol{\Phi}_i}{\partial \mathrm{x}_j}$ along the global coordinates
5. Material Properties: constitutive **D**
6. Entity-wise calculation of the element matrix **Ke**

These six steps are executed in the functions “*cal_Ke*”. The function can be found in the header “fem_fect.h”. In this header all functions need for the calculation are collected. After obtaining the local element stiffness matrix of an element, it has to be assembled to the global stiffness matrix via the function “*assem*” with the help of the topology matrix *Edof*.

3.3.4 Applying boundary conditions

The Dirichlet and Neumann boundary condition have to be applied to the mesh. In the Neumann boundary conditions the known displacements at the end of the test object are inserted after equation,

$$f(Bc_{100}, 1) = f(Bc_{100}, 1) - K(Bc_{100}, Bc_{100}) \cdot u(Bc_{100})\ with\ Bc_{100} = \begin{bmatrix} nod_1 \\ \vdots \\ nod_2 \end{bmatrix}. \quad (3.3)$$

The external force vector f is filled with the product of stiffness matrix with displacement at the corresponding nodes. The vector $f(Bc_{100}, 1)$ on the right hand side of equation (3.3) represents previous values added to the force vector.

As no other boundary condition are taken into account, the Dirichlet boundary condition are zero of all the rest of the nodes,

$$f(Bc_{rest}, 1) = 0\ with\ Bc_{rest} = Bc_{all}\ witout\ Bc_{100} \quad (3.4)$$

In the programme the force vector is first filled with zeros on then with the Neumann boundary conditions.

3.3.5 Solving equation

In the static case of the standard FEM, the equation (1.7) has to be solved. On the left side, the known outer forces are described. On the left side there is the product of the stiffness matrix *K* with the displacement vector *u*. As the displacement is unknown, the inverse of the stiffness matrix has to be calculated.

3.3.5.1 Inverse of the global stiffness matrix

For the calculation of the inverse of the stiffness matrix certain goals have to be fulfilled. The chosen algorithm must be accurate, robust and asymptotically optimal in run time and memory usage. Two algorithms the Cholesky factorization and the QR factorization are chosen. As the system matrix is sparse, the concept of Csparse is used for the calculation of the algorithm. In the following subchapter, the handling of sparse matrices is described.

3.3.5.2 Sparse matrix data structure

The simplest sparse matrix data structure is a list of the nonzero entries in arbitrary order. The list consists of two integer arrays i and j and on real array x of the length equal to the number of entries in the matrix, for instance

$$\mathbf{A} = \begin{bmatrix} 4.5 & 0 & 3.2 & 0 \\ 3.1 & 2.9 & 0 & 0.9 \\ 0 & 1.7 & 3.0 & 0 \\ 3.5 & 0.4 & 0 & 1.0 \end{bmatrix} \tag{3.5}$$

A zero-based data structure for an m-by-n matrix contains row and column indices in the range 0 to m-1 and n-1, respectively. In the C++ programming language all code is zero-based, whereas in Matlab the matrices are one-based. When transforming the mesh parameters from the Matlab code to the C++ environment, should be done with caution due to the different based data structure.

In the Csparse function, to different sparse matrixes are present. At first, the triplet form is simple to create but difficult to use in most sparse matrix algorithms. An example is given in equation (3.6).

$$\begin{aligned} &\text{int}\ i[\ ] = \{2, 1, 3, 0, 1, 3, 3, 1, 0, 2\}; \\ &\text{int}\ j[\ ] = \{2, 0, 3, 2, 1, 0, 1, 3, 0, 1\}; \\ &\text{double}\ x[\ ] = \{3.0, 3.1, 1.0, 3.2, 2.9, 3.5, 0.4, 0.9, 4.5, 1.7\}; \end{aligned} \tag{3.6}$$

The compressed-colmn form is more useful and is used in almost all functions in the CSparse library. A m-by-n sparse matrix that can contain up to *nzmax* entries is represented with an integer array *p* of length *n+1*, an integer array *i* of length *nzmax*, and a real array *x* of length *nzmax*. Row indices of entries in column *j* are stored in *i*[*p*[*j*]] through *i*[*p*[*j*+1]-1], and the corresponding numerical values are stored in the location in x. The first p[0] is always zero, and *p*[*n*]≤*nzmax* is the number of actual entries in the matrix. The example matrix of equation is represented as,

```
int p [  ] =      {0            ,3            ,6      ,8      ,10 };
int i [  ] =      {0  ,1  ,3  ,1  ,2  ,3  ,0  ,2  ,1  ,3  };
double x [  ] = {4.5 ,3.1  ,3.5  ,2.9 ,1.7  ,0.4  ,3.2 ,3.0 ,0.9  ,1.0};
```
(3.7)

Both matrix types can be saved in the class *cs_entry*.

3.3.5.3 Cholesky Factorization

Classically the Gauss-Jordan algorithm is used for solving linear systems, but it is an inefficient algorithm for sparse matrix, so different method has to be chosen. The stiffness matrix of a stable FEM calculation is symmetric and positive definite matrix. This property allows the decomposition of the matrix K into a lower triangle matrix L with strictly positive diagonal entries and a conjugate transpose of the matrix L according to

$$K = LL^* \tag{3.8}$$

This is called the Cholesky decomposition. As the stiffness matrix K is positive-definite, there is only one lower triangle matrix L with strictly positive diagonal entries. It follows that the Cholesky decomposition is unique.

After obtaining the Cholesky decomposition, the sparse triangle system $Ly = b$ is first solved. This is done with the help of the elimination tree of the toolbox Csparse. After obtaining the result of the elimination tree the postordering is computed and then the column counts, which are the number of nonzero in each column of L. When the vector y is known, the equation $L^T u = b$ can be solved again with the help of the elimination tree. Now the displacement for every node is obtained.

The Cholesky factorisation is implemented in function "*cs_cholsol*". The function overwrites the input vector f with the solution of $Ku = f$. The input parameter order determines the input ordering used, zero stands for $P = I$ of the fill-reducung permutation, $LL^T = PAP^T$ or one for a minimum degree ordering of K. The functions returns one if it was successful with the calculation, zero if the matrix is not positive definite or if the method ran out of memory. The forward/backsolve steps cannot fail because they do not allocate memory.

The goal of the Cholesky algorithm, implemented in the Csparse matrix, is to keep the numeric factorization as simple as possible in terms of time complexity, memory usage and clarity of code. As it can be seen from the name, the code of the toolbox was written in C.

As for our project C++ is used on a Microsoft Visual Studio Platform, the code has to be adapted to our programming environment. It means, code bit had to be change and do not coincide totally with the original code. As not all functions of the code are used for the calculation of the Cholesky factorization, only the needed ones are added to the Header file “Csparse_fct.h”.

3.3.5.4 QR factorization

The least squares problem is to find the displacement u that minimizes the 2-norm of the residual, $\|r\|_2$, where $r = b - Ku$ and K is m-by-n with $m \geq n$. Multiplaying a vector by an orthogonal matrix Q does not change its 2-norm. If K is factorized into the product $A = QR$, then

$$\|r\|_2 = \|b - Ku\|_2 = \|Q^T b - Ru\|_2 = \left\| \begin{bmatrix} Q_1^T b - R_1 u \\ Q_2^T b \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \right\|_2, \tag{3.9}$$

where Q is m-by-m, R_1 is n-by-n, and Q_1 is m-by-n. Assuming A has full rank, R_1 is non-singular and so the upper triangle system $Q_1^T b = R_1 u$ can be solved, which makes $r_1 = 0$ and minimizes $\|r\|_2$.

In the function “*cs_qrsol*”, $Q = H_1 H_1 \dots H_n$ is represented implicitly as a product of Householder reflections, and the permuted matrix $PK\bar{Q}$ is factorized instead of K, where $\bar{Q}$ is fill-reducing column permutation. The right-hand side of the displacement vector u is overwritten with the solution of x. The input parameter order describes the kind of ordering, one stands for natural ordering and three for a minimum degree ordering of $K^T K$

3.4 Remesh algorithm

For the first implementation, an easy remeshing algorithm is chosen. It is only allowed to cut between the elements, not through the elements. The scalpel edge has to point in the direction where it moves. It means that with the constellation FP_1W_2F [10] of the blue robot the scalpel can only move in a x,z plane. For this reason the assumption, it is a good approximation of our situation.

The silicone block will be placed vertically to the cut plane. The mesh of test object is created so that a straight cut can be made parallel to the sides only between elements, i.e. no cut through an element. The depth of the cut can be influenced by the number of elements in the z-coordinate and the scaling factor of node matrix. When cutting between two elements, new nodes have to be defined at the cut corners. To find these nodes the function “*get_Node_cut*” is written. It finds the cut nodes out of the coordinate matrix *Node* and creates a new node at the same place of the old node. In the end the new nodes are appended to the matrix *Node*.

After extending the coordinate matrix with the new nodes, the topology matrix has to be changed, too. The separated element has to be connected to degree of freedom of the new created nodes, see Figure 3.5. This is done by the function "*get_Edof_cut*". The cut surface is defined by the Hesse normal form n_H. A search gives the elements on the negative direction of the Hesse normal vector and connects them with the new created nodes.

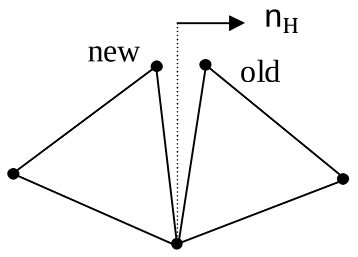


Figure 3.5: Adding new nodes to mesh

After obtaining the new topology and new coordinate matrix, the standard FEM calculation can be performed as described in the previous subchapters.

3.5 Scanning object

The device scanCONTROL 2700-100 from the company Micro-Epsilon [15] with an integrated controller is used for scanning the test object. The laser line scanner uses the triangulation principle for a 2D acquisition of a height profile of various target surfaces. A laser line is generated with the help of special lenses and projected onto the target surface. The optical system projects the diffusely reflected light back onto a sensor matrix. In the sensor head, the distance information is calibrated by a controller and the sensor matrix is used to position along the laser line are calculated. This generated calibrated matched measurement values which are than output as a precise line profile. Regardless of the position or angle the profile data are absolute calibrated data sets in a 2D coordinate system that is fixed in respect to the sensor.

The scanner is used for two purposes. First to scan the surface of a complex geometry and secondly to compare the simulation results with the reality. In the following the installation and the commissioning of the sensor is outlined.

3.5.1 Reading out data from the scanCONTROL

The scanning algorithm is performed in four basic steps, see Figure 3.6. It starts with the Initialisation of the scanCONTROL device. A LLT-object has to be created and a firwire connection has to be set up.

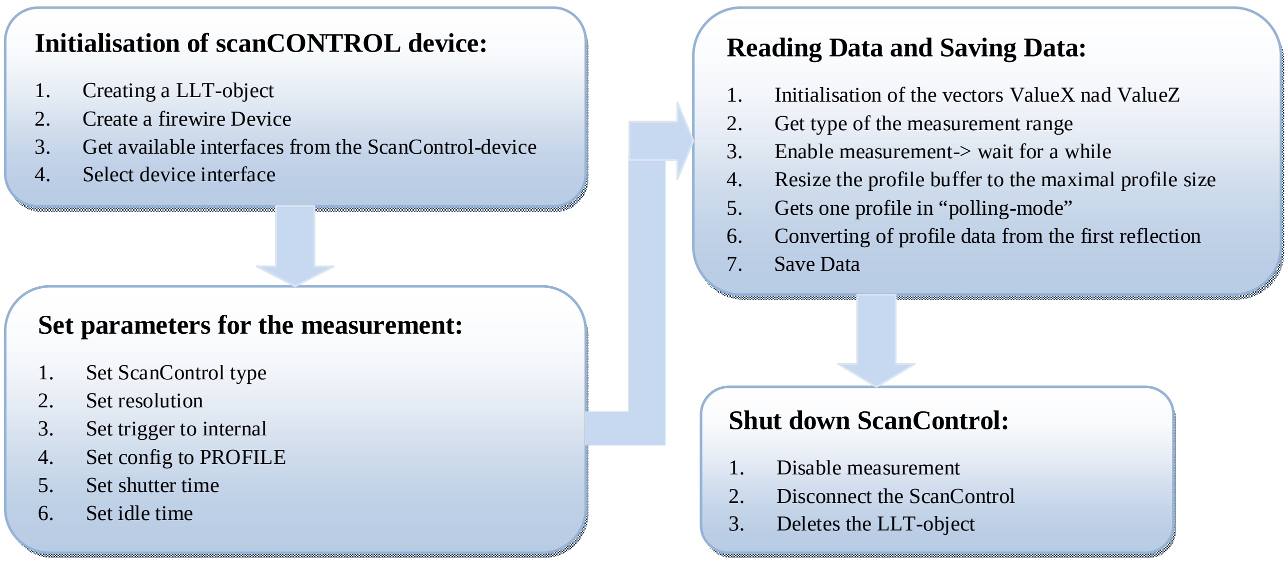

Initialisation of scanCONTROL device:
1. Creating a LLT-object
2. Create a firewire Device
3. Get available interfaces from the ScanControl-device
4. Select device interface
Set parameters for the measurement:
1. Set ScanControl type
2. Set resolution
3. Set trigger to internal
4. Set config to PROFILE
5. Set shutter time
6. Set idle time
Reading Data and Saving Data:
1. Initialisation of the vectors ValueX nad ValueZ
2. Get type of the measurement range
3. Enable measurement-> wait for a while
4. Resize the profile buffer to the maximal profile size
5. Gets one profile in “polling-mode”
6. Converting of profile data from the first reflection
7. Save Data
Shut down ScanControl:
1. Disable measurement
2. Disconnect the ScanControl
3. Deletes the LLT-object

Figure 3.6: Scanning algorithm

In the second step the parameter for the configuration has to be set. All parameters are listed on the down left box of Figure 3.6. It depends on the situation how to set the values of the parameters. For more detailed information, it is referred to [15].

The third reads the Data from the device and saves it either in a TXT-file or in a variable or both if demanded. The scanned profile points can also be showed on the console to make an online check on it. In the last the step the scanCONTROL LLT-object is deleted and the sensor is disconnected.

3.5.2 Scanning of profile

3.5.2.1 Transforming Scanned points into an absolute coordinate system

The scanCONTROL is fixed at the end effector of the blue robot. The 2D coordinate system of the scanCONTROL has to be transformed into the absolute coordinate system of the blue robot, see Figure 3.7 first constellation.

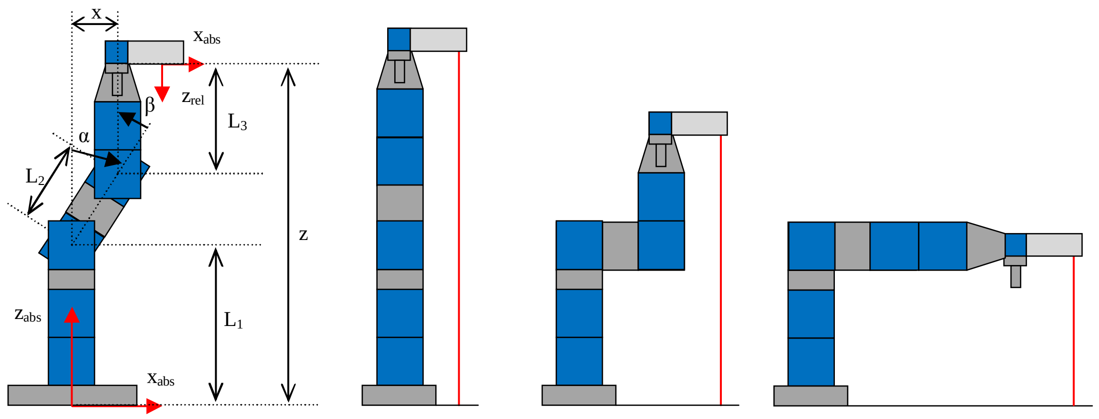


Figure 3.7: Defining length of blue robot

To obtain the exact position of the scanCONTROL, the trigonometric functions are used,

$$x = \sin(\alpha) \cdot L_2 + \cos(90 + \alpha - \beta) \cdot L_3, \tag{3.10}$$

$$z = L_1 + \cos(\alpha) \cdot L_2 + \sin(90 + \alpha - \beta) \cdot L_3.$$

Another way of transforming the scanned data is given in the next subchapter.

3.5.2.2 Reference for the transformation at the test object

Here the outer corners of the test object are taken as reference for the transformation of the profile data points. First all the gradients between the neighbour points are calculated. In the second step the gradient lying between the positive or negative value of the define directory *MAXGRAD* are sorted out, because the goal is to find edges. The parameter *MAXGRAD* has normally the value 5. It can differ from time to time. For instance, the test body is not as smooth the silicon, higher gradient can occur due to the rough surface. When a set of data points is higher than the variable *MAXGRAD*, than the end point of the first gradient is taken and the start point of the last gradient. The difference between the two points represents the distance. Now there can be calculate every distance between different corners. The algorithm is visualized in Figure 3.8.

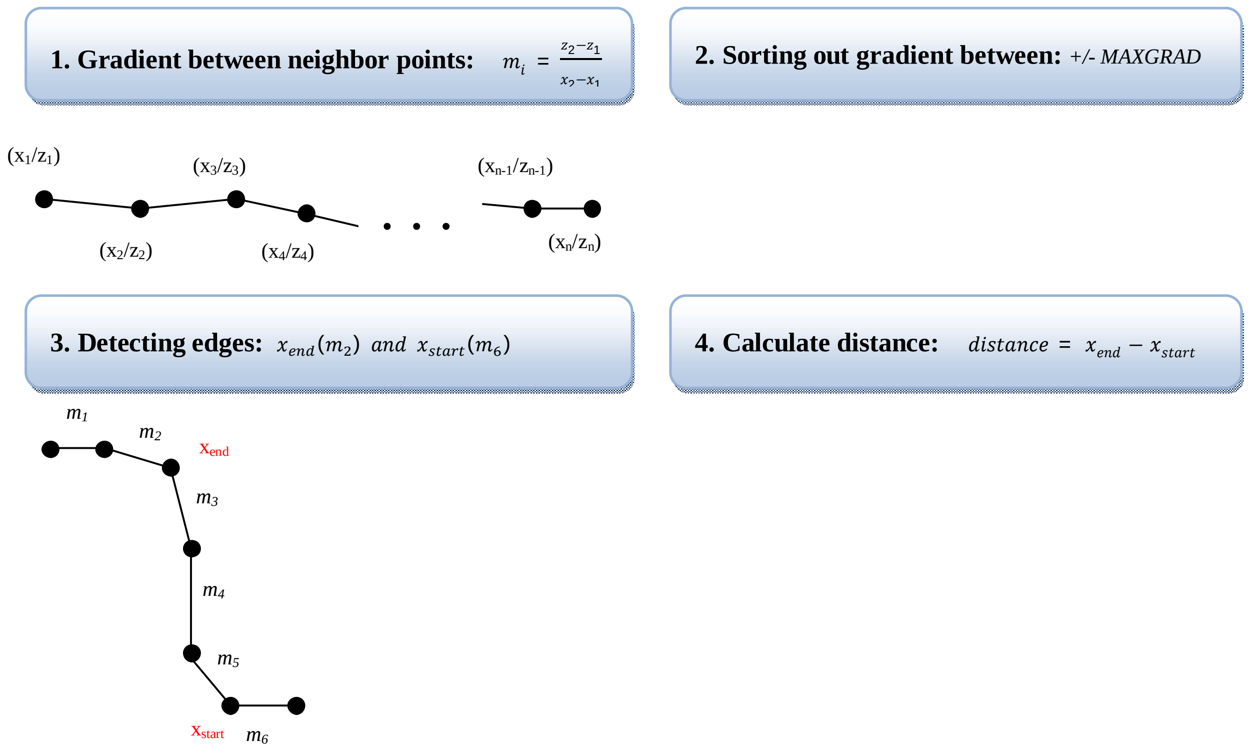

1. Gradient between neighbor points: $m_i = \frac{z_2 - z_1}{x_2 - x_1}$
2. Sorting out gradient between: +/- MAXGRAD
(x_1/z_1)
(x_3/z_3)
(x_{n-1}/z_{n-1})
(x_2/z_2)
(x_4/z_4)
(x_n/z_n)
3. Detecting edges: $x_{end}(m_2)$ and $x_{start}(m_6)$
4. Calculate distance: $distance = x_{end} - x_{start}$
m_1
m_2
x_{end}
m_3
m_4
m_5
x_{start}
m_6

Figure 3.8: Find the length between two edges

It is a straight forward method to find fast the edges of the test object. Now the edge is defined as the zero level of the test object and the simulation data points can be plotted to the corresponding profile data points of the scanCONTROL.

3.6 Visualisation

For the visualisation, the OpenGL library is chosen to create images on the windows screen. OpenGL is a powerful tool for producing high-quality, computer-generated images and interactive applications using 2D and 3D objects, bitmaps, and colour images. The basic structure is to initialize certain states that control how OpenGL renders and to specify objects to be rendered.

OpenGL is a state machine. For instance the colour of an object represents a state which is initialized with a certain colour. This colour remains unchanged until the state is changed. Some other examples for states can be the current view and projection transformations, line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn. Many states variables refer to modes that are enabled or disabled.

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. The diagram shows a assembly line approach, which OpenGL takes to processing data, see in Figure 3.9. There two main approaches to create an image on the windows screen, through vertex data or pixel data. As in our FEM calculation the mesh is already provided, the vertex data is used for the visualisation. In the following, the path of the vertex data will be explain, but not the pixel data path, because it is not used in our programmes. For a more detail description, it is referred to [16]. Geometric data follow the path through the row of boxes that includes evaluators and per-vertex operations. In the final steps it undergoes rasterization and per-fragment operations before the final pixel data is written into the frambuffer.

In the display list all data is saved for current use or later use. When a display list is executed, the retained data is sent from the display list in an immediate mode. All geometric primitives are described by vertices. When in the graphic a parametric curve is present, it initialized by control points and polynomial functions called basic functions. In the evaluators a method is provided for deriving the vertices used to represent the parametric curve from the control points. This type of conversation is not needed in our programme structure, because there are no parametric curves present in our FEM mesh. In the next stage, the vertex data is converted into primitives. Also if there is a lightning enabled, the lightning calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lightning information to produce the colour value.

In the primitive Assembly, the clipping which is a major part of primitive assembly is the elimination of portions of geometry that fall outside a half space, defined by a plane. Point clipping simply passes or rejects vertices, whereas line clipping can add additional vertices depending on how the line is clipped. The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related colour, depth, and sometimes texture-coordinates values.

In the rasterization, the conversion form geometric data into fragments is done. Each fragment square corresponds to a pixel in the framebuffer. Before values are actually stored in the framebuffer, a series of operation are performed that may alter or even throw out fragments. All these operation can be enabled or disabled, for instance texturing. Finally, the thoroughly processed fragment is drawn into the appropriate buffer, where it becomes a pixel.

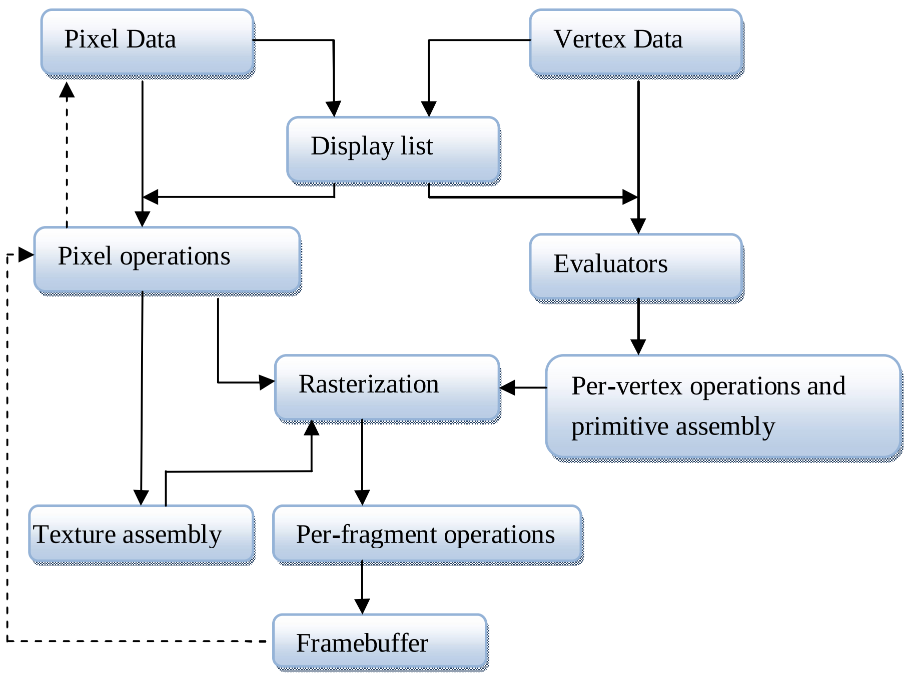


Figure 3.9: Order of operations

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanism of the windowing system. Several libraries enable to simplify your programming tasks, including the following:

1. OpenGL Utility Library(GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces.
2. The OpenGL Utility Toolkit(GLUT) is a window-system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window system APIs.

OpenGL contains no commands for opening windows or reading events from the keyboard or mouse. The GLUT is used for opening windows and detecting inputs. The window management in the GLUT toolkit are seven routines necessary for initializing a window:

1. "*glutInit*" initializes GLUT and processes any command line arguments and it should be called before any other routine.
2. "*glutInitDisplayMode*" specifies whether to use an RGBA or colour-index model. It also defines whether there should be used a single- or double-buffered windows.
3. "*glutInitWindowPosition*" specifies the screen location for the wpper-left corner of your window.
4. "*glutInitWindowSize*" specifies the size in pixels of your window.
5. "*glutCreateWindow*" creates a window with an OpenGL context. It returns a unique identifier for the new window.
6. "*glutDisplayFunc*" is the callback function in the routine. Whenever GLUT determines that the contents of the window need to be redisplayed, the callback

function is executed. It means that all redraw scenes are putted into the display callback function.

7. "*glutMainLoop*" all windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display callback is triggered. Once the loop is entered, it never comes back to the "*main*" function

4 Results

The results of the executed experiments are presented here. First, the material properties have to be found for the later FEM calculations. Afterwards a cut is performed with the robot control by a human operator over the Haptic device. The cut surface is scanned by the device scanControl and the comparison between the simulation and the reality of the deformation is made. The tension in test object can give valuable information about the growing of the cut, which will be shown in a later plot.

4.1 Identification material properties

For the material properties, Young’s modulus and Poisson’s ratio have to be defined. Young’s modulus is a measure of the stiffness of an isotropic elastic material. Poisson’s ration is the ratio of the contraction to the extension, when a sample object is stretched.

As mention in chapter three, silicone is taken as the material for the test object. Producers are normally not defining neither Young’s modulus nor Poisson’s ratio for silicone. In the ISO 527-1 “Plastics – Determination of tensile properties”, it is written down the definition of how to determine Young’s modulus and Poisson’s ratio. Special shapes of the silicone are needed and special devices are need to hold and measure the test object. The determination of the material properties are not done after the ISO norm, instead a different method is used due to available equipment.

As Young’s modulus has a bigger influence on the deformation as the Possion’s ratio, the elasticity modulus is first determined. Two methods were pursued to find Young’s modulus for our simulation. In the first method the parameter is found by an own experiment. With the tension platform the external forces and the displacement can be measured.

The vertical forces are read out of the ATI Industrial Automation sensor 330N over the serial port after expanding the test object. The extension is measured over the scanCONTROL device, describe in chapter 4.5.2.2. By plotting the displacement over the force, Young’s modulus can be obtained out of Hook’s law, according to (1.16).

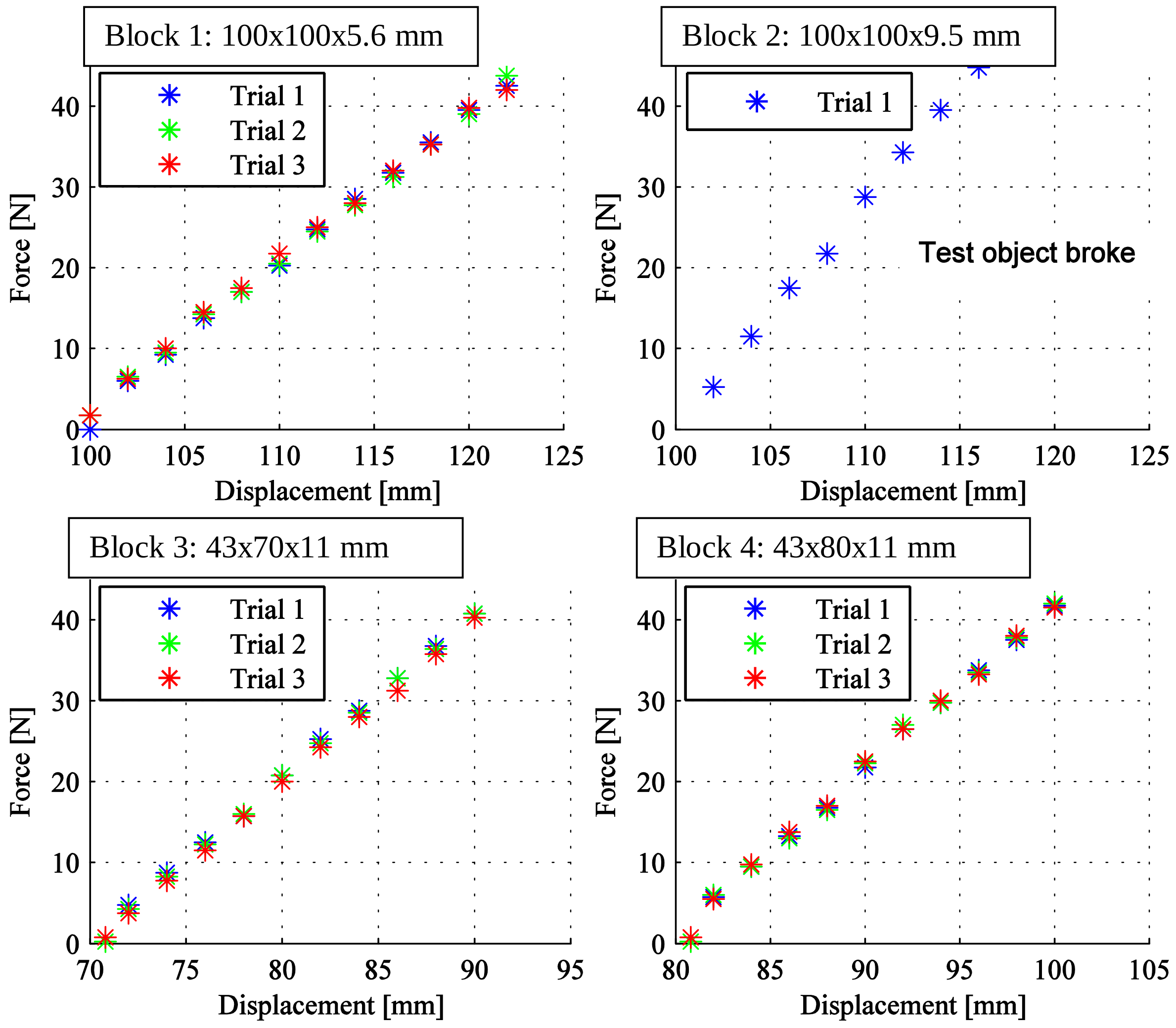


Figure 4.1: Plotting force against displacement for the block 1, 2, 3 and 4

In Figure 4.1 , there are the results of four different blocks plotted, for the force and displacement ratio. To see how the material behaves after some extension and compression, there are made three trails per block. During the experiment of block 2, top right on Figure 4.1, the mounting from the test object to the stiffer silicone broke, so there is only one trail. As it can be seen, there is a linear behaviour of the material between the force and the displacement. To define now Young's modulus the following equation is used,

$$E = \frac{f_2 - f_1}{l_2 - l_1} \cdot \frac{1}{A \cdot l_0} . \tag{4.1}$$

The parameter f_2 and l_2 are the outer force and the length of the block, respectively, at the second step. The variables A and l_0 are the surface and the starting length. Young's modulus is calculated between every measurement step of every trial and an average over the obtained Young's modulus is built. The value of Young's modulus obtained from the four blocks is 0.0385 [N/mm²]. It is nearly the same value as for rubber which lies in the range of

0.0100 to 0.1000 [N/mm²]. Poisson's ratio is obtained over the lateral strain. For this purpose, the width of the body is taken before every trial and at the end. After the following equation the ratio is calculated,

$$\nu = \frac{b_0 - b_1}{l_0 - l_1} \cdot \frac{l_0}{b_1}. \tag{4.2}$$

The average value for all four experiments of the Possion's ratio is 0.4300. As next, four different meshes are build up each corresponding to one block. As our maximal range for the later predefinition of the sensors will be 10mm, the meshes of the blocks will be expanded by the maximum value of 10mm, see Figure 4.2.

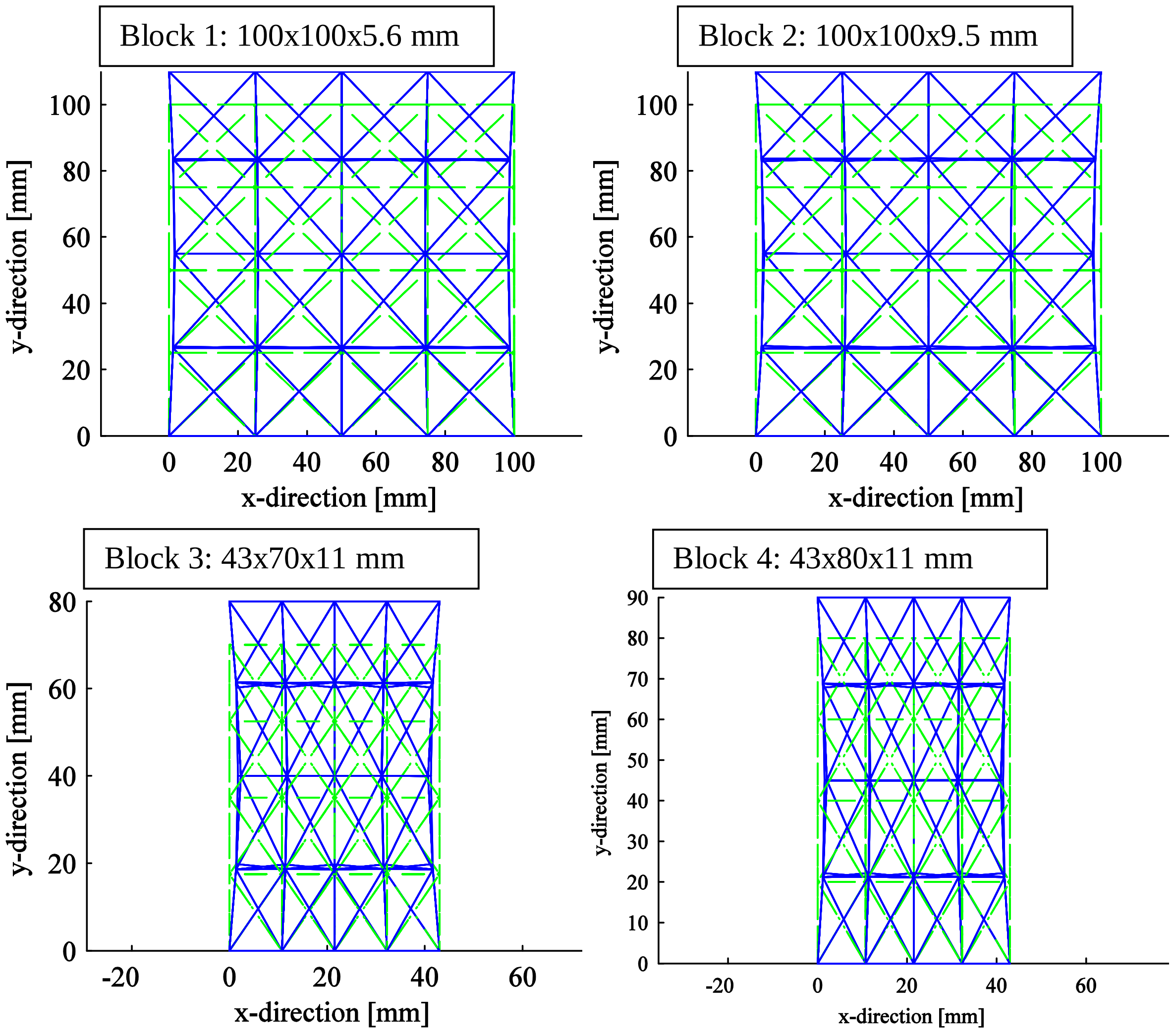


Figure 4.2: The deformed meshes of the different test objects

In Table 4.1, the forces at the deformed end are filled in for all four blocks.

Test setup				Test object - Block number:			
Nr.	E-modul	Poisson's ratio	Type of Calculation	1	2	3	4
1	?	?	Force Experiment [N]	21.00	29.50	20.75	17.25
2	0.0385	0.4300	Force Simulation [N]	15.5559	23.9595	16.2461	14.2198
3	0.0385	0.4739	Force Simulation [N]	20.9888	23.9595	16.2461	14.2198
4	0.1233	0.4900	Force Simulation [N]	21.0051	29.7196	18.2675	16.5000

Table 4.1: Comparison forces between simulation and experiment

The blocks hold the same numbering system as in Figure 4.1. There are made four different test setups. In all of the tests, the test object is hold at one end and deformed by 10mm in y-direction at the opposite end. Number one of the test setups is the measured values of the force sensor, already plotted in Figure 4.1 on the top left. In number two the E-modulus and Poisson's ratio of the experimental determination of the material properties are given as input the simulation and the forces at the end are determined and filled in Table 4.1. The values are for every block around 20 % too less. So the found values do not coincide with the reality, but what it is interesting that all values are equally off. Now there are two possibilities of adapting our material properties to the real ones. Either Young's modulus is kept at the value or Poisson ratio. In our case the E-modul is kept the same and the Possion's ratio is changed until the forces of the simulation coincide with the measured force for the first block. If now for the rest of the blocks, the same forces with the optimized values are obtained, the material properties are found, see number three in Table 4.1. Another way to gain the material properties is to look at one block and take one measurement which seems to represent an average of the other measurements. Block one seems to have the most linear behaviour compared to the others, so the measurements of it are taken for the next determination of the material parameters. Our range of extension will not be higher as 10mm, so the forces at 10mm are taken. As the forces and the displacement are known for the FEM equations, the two material parameters are the only unknown. As our silicone rubber seems to have similar behaviour to standard rubber, the value of 0.5 for Possion's ratio is taken. Now Young's modulus can be adapted by a nested iterative manner until the force of the simulation coincide with the measured one, see number four in Table 4.1.

4.2 Deformation around the cut

In the following chapter, the different types of plots are present. The same block is used in all figures and has the dimension 100x80x17mm. The cut is not performed in the middle of the block in y-direction, it is at value of 45mm. In the x-direction the block is subdivided into 6 standard cubes, 16 in y-direction and 14 in z-direction. The dimension of the block is 16.6667x5x1.2143mm. In x-direction, the subdivision is kept bigger as compared to the other two directions as it mainly interesting the deformation in y- and z-direction.

The visualisation takes place in OpenGL as described in chapter 4 "Software Implementation. The created image can be seen in.

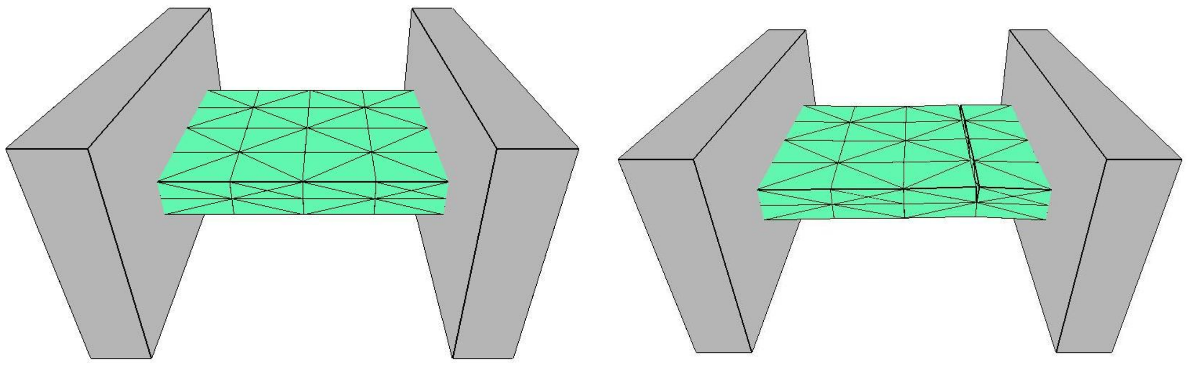


Figure 4.3: Image shown during simulation

As in OpenGL, it extensive to build up a GUI for instance, extracting points or drawing an coordinate system the post processing takes place in Matlab. First the cut block is plotted, see Figure 4.4. There is a uniform deformation of the block in x-direction. The cut means a weakness for the structure, so the bottom at the cut lifts up by the value of 2.0960 mm. The cut is of the depth of 8.5001. The cut edges gape to the opposite directions. The horizontal deformation at the end of the x-direction is 6.6 mm.

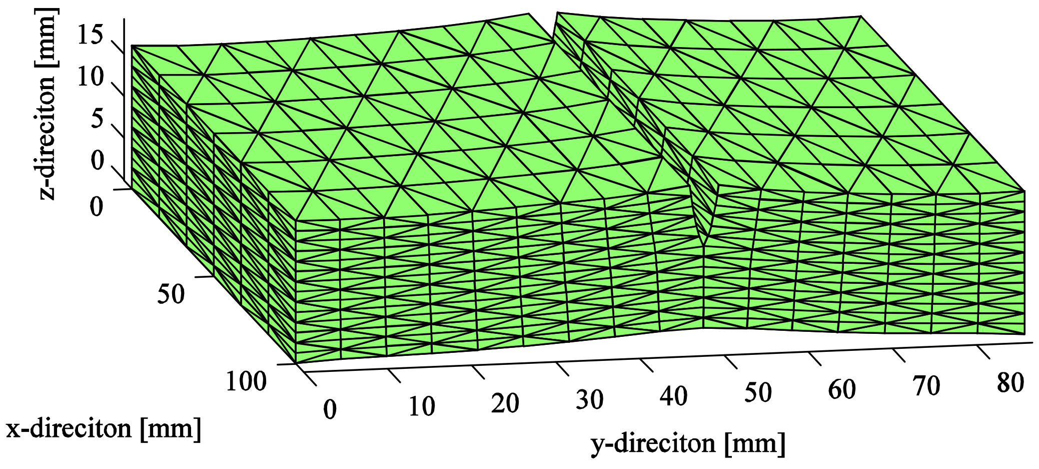


Figure 4.4: Mesh

In Figure 4.5, the same plot is used as in the previous, but the scanned points of the scanControl devise are shown too. There are made three different scans at x equals 25mm, 50mm and 75mm during the experiment. The scanned points coincide well with the simulation.

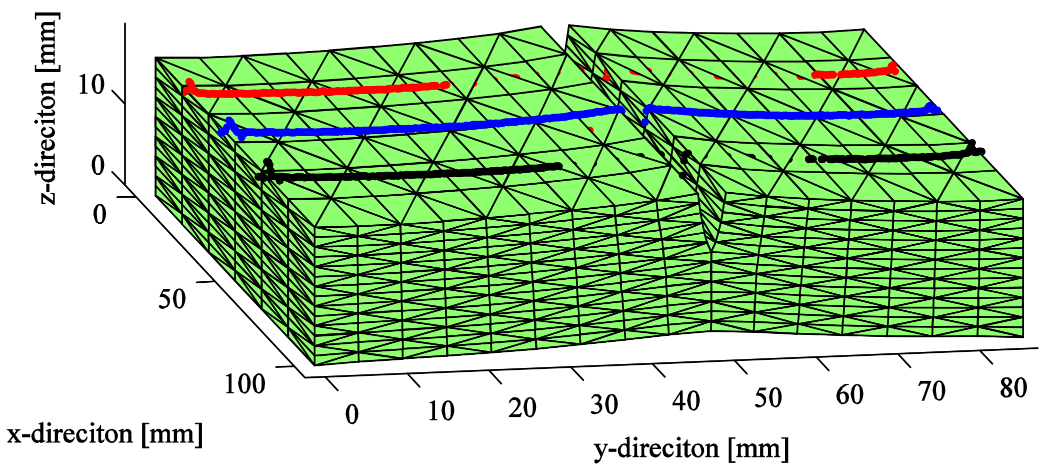


Figure 4.5: The deformed meshes of the different test objects

When a FEM calculation is performed, it is of interest to see the tension in the body.

The tension in an element are obtained out of the deformation of the nodal points after the following equation,

$$\sigma = \boldsymbol{D} \cdot \boldsymbol{B} \cdot u. \tag{4.3}$$

The matrices $\boldsymbol{D}$ and $\boldsymbol{B}$ are already defined in equation (1.18) and (1.10), respectively. The stress vector is written for every element as $\sigma = [\sigma_{xx}\ \sigma_{\ yy}\ \sigma_{zz}\ \sigma_{xy}\ \sigma_{yz}\ \sigma_{zx}]$. In Figure 4.6, the average of the von Mises stress are calculated for every node after equation,

$$\sigma_{i,m} = [\sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2 - \sigma_{xx}\sigma_{yy} - \sigma_{zz}\sigma_{yy} - \sigma_{zz}\sigma_{yy} + 3\sigma_{xy}^2 + 3\sigma_{yz}^2 + 3\sigma_{xz}^2]. \tag{4.4}$$

The stress $\sigma_{i,m}$ is the von Misses stress in node i. All node stress are compared over a colour map rating.

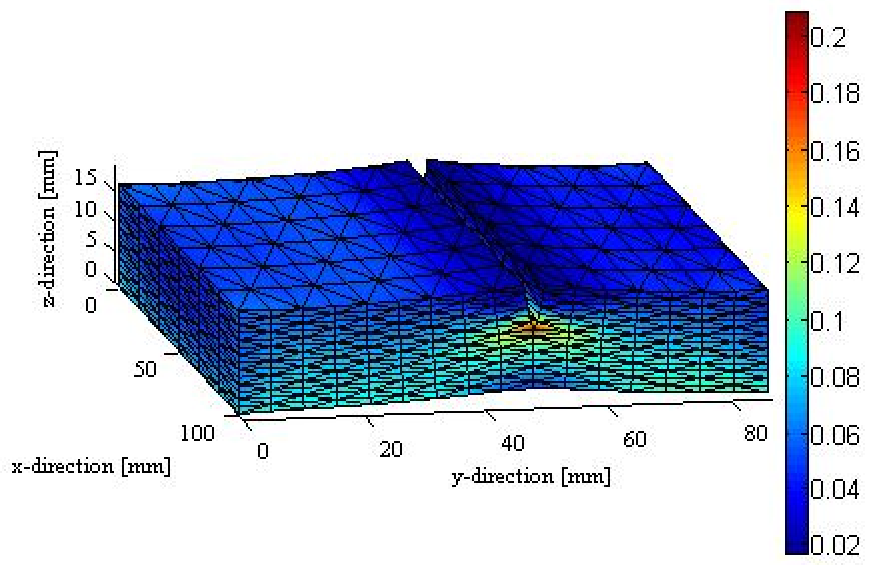


Figure 4.6: The deformed meshes of the different test objects

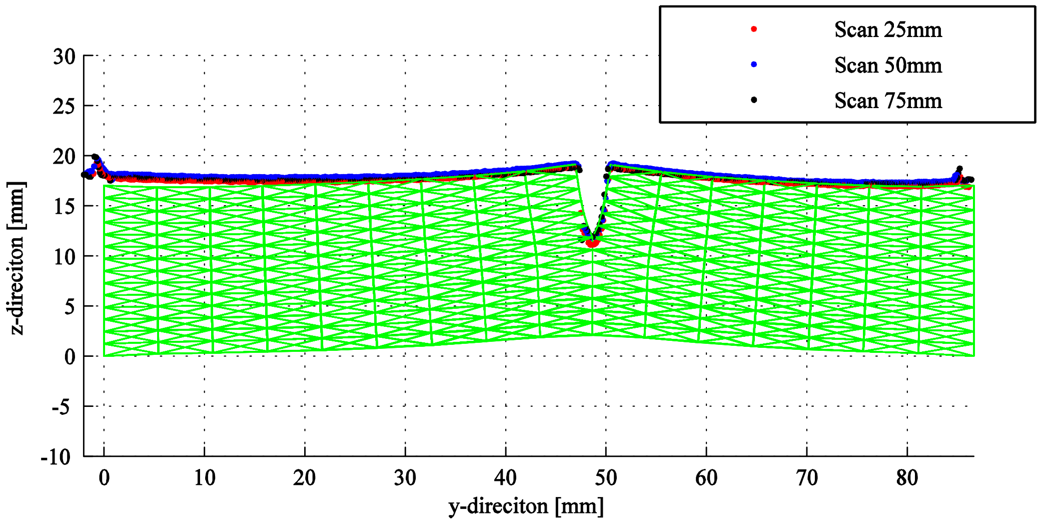


Figure 4.7: Comparison of measurement and simulation

In Figure 4.7, the measured profiles of the scanCONTROL device are plotted againt the deformation obtained from the simulation. There are scanned three different profiles at x equals 25mm(black), 50mm(green) and 75mm(red). The red line is in the cut lower than the other two lines. The robot moves on a trajectory of a straight line. There is no control loop for the movement of the robot. Single points are given to a function and the robot moves to this position. But there is no check on if the robot reaches the wanted place. It means that the robot has a small vibration during the movement. Consequently the incision is not straight.

5 Summary

An interactive simulator of a Finite Element (FE) calculation has been developed in this Diploma thesis, consisting of a mesh generator for a block filled up with tetrahedral, static FE calculation and visualisation. The computational environment takes place on a Windows system in the programming language C++. The simulator is aimed to be used in a Telepresence application where a robot is controlled by a haptik device over a long distance. The feedback information of the deformed body comes from the simulation instead of from real sensor at place of distance due to the time delay of the network connection.

A theoretical survey of an incision process for a Finite Element Method (FEM) was examined in this thesis. The incision process was assumed with a completely sharp scalpel and no friction force was considered. For FEM the wanted quantity varies steady over the body. An incision means to have to deal with a strong discontinuity in the FE mesh. The eXtended-Finite-Element-Method (XFEM) can cope with discontinuities in an element. Another way of dealing with the discontinuity is to remesh around it and then perform a standard FEM calculation with the new obtained mesh. The deformation of the XFEM calculation was compared to the remeshing FEM. The advantage of the XFEM is that there is no need of remeshing the edges of the discontinuity. But the location of the edges are not visualized directly, so they have to be found in a post processing algorithm which leads into a remesh. Due to the Telepresence operation, the calculation has to be in real-time. The longest time for a static FE calculation stays by the calculation of the inverse of the stiffness matrix. As the stiffness matrix is changed in the XFEM and the remeshing FEM, the remeshing FEM was chosen, because a direct visualisation of the incision edges is obtained. In the remeshing FEM, a function finds the nodes lying in the incision surface and decouples the nodes when the cut occurs at them, so with new added nodes a free movement between the elements are possible. An incision can only happen between the elements.

An experiment set up has been developed to evaluate the results of the simulation with the reality. The experiment aims to imitate the incision of a human skin. Silicone was used as material for this purpose, since it is also used as implants for different body parts. The texture of silicone resembles human skin in sense of touch and visual effects. The tension platform gives a certain predefined tension to the object, so the edges of the incision are realistically posed. After the incision, the surface of the test object was scanned by a 2D laser profile-scanner. It measured the vertical and horizontal displacement from the laser profile-scanner. The data was transformed into the simulation and the accuracy of the simulator can be defined.

The shape of test object was a quadrangle with the dimension 100x80x17 mm^3. The deviation of the simulation was defined lower than 1% of the maximal dimension of the object compared to reality. For our silicone block, the absolute deviation must be lower than 1mm. The deviation criterion was maintained over the whole surface of the test object.

Until now, only a visualisation of the deformation of the cut has been added to the simulator. In a previous project, a force model of the incision process has been defined. The force model has to be adapted to the silicone material and committed to the haptic device as a feedback signal. The cut is only done in a straight line, so the collision detection is done for the 1D case. Here the orientation and the direction of the scalpel are not considered, only the location is of interest. When adding more degrees of freedom to the robot, so the incision can be done in any 3D trajectory, an efficient collision model has to be found for defining the orientation, direction and location of the scalpel and more complicated remeshing method has to be defined. This should be done in future work.

A Matlab: Create mesh toolbox

1. [Edof,node]=*cal_Edof_node_lx* (lx,Edof_standard,node)

The function *cal_Edof_node_lx* add to a uniform cube filled with tetrahedral in x direction depending on the variable *lx* more cubes. As output it returns the new topology matrix **Edof** and the new coordinates of the nodes, stored in the matrix **node**, of the mesh.

2. [Edof,node]=*cal_Edof_node_ly* (ly,Edof_standard,node)

The function *cal_Edof_node_ly* does the same as the *cal_Edof_node_lx,* but only add new cubes in the y-direction.

3. [Edof,node]=*cal_Edof_node_ly_plus1* (ly,Edof_standard,node)

The function *cal_Edof_node_ly_plus1* does the same as the *cal_Edof_node_ly*, but only at the last added cube, there will be half of the uniform cube be added.

4. [Edof,node]=*cal_Edof_node_lz* (lz,Edof_standard,node)

The function *cal_Edof_node_lz* does the same as the *cal_Edof_node_lx*, but only add new cubes in the z-direction.

5. [Ex,Ey,Ez]=*coordxtr*(Edof,Coord,Dof,nen)

The function *coordxtr* extracts element nodal coordinates from the global coordinate matrix Coord for elements with equal numbers of element nodes and dof's.

6. [Ex,Ey,Ez,Edof_with,Edof_neu,node,Dof,ndof,nele]=*geom_tet*(lxc,lyc,lzc,scx, scy, scz,zeig,num_node,num_el,y_plus1)

The function *geom_tet* creates 3d block filled with cubes which consists of tetrahedral. This is done with the help of the functions *cal_Edof_node_lx*, *cal_Edof_node_ly* and *cal_Edof_node_lz.* Then the block is scaled and afterwards it is scaled and plotted.

7. [Ex_z,Ey_z,Ez_z]=*get_Exyz*(Edof,node)

The function *get_Exyz* extracts element nodal coordinates from the topology matrix **Edof** and the node coordinates matrix **node**.

8. [Edof,node]=*init_Edof*

The function *init_Edof* creates a uniform cube filled with tetrahedral, to add easier the cubes in the corresponding coordinate axis.

9. *Plot_def_Mesh_tet*(Ex,Ey,Ez,Edof_deg,node,s1,numbers,u,n)

The function *Plot_def_Mesh_tet* displays the deformed mesh by adding the displacement vector *u* to the element coordinate matrixes **Ex**, **Ey** and **Ez**. If the variable *numbers* is 1 the element numbers will be shown. The parameter *s1* defines the line type and the colour of the plot.

10. *Plot_Mesh_tet*(Ex,Ey,Ez,node, s1,num_node,num_el)

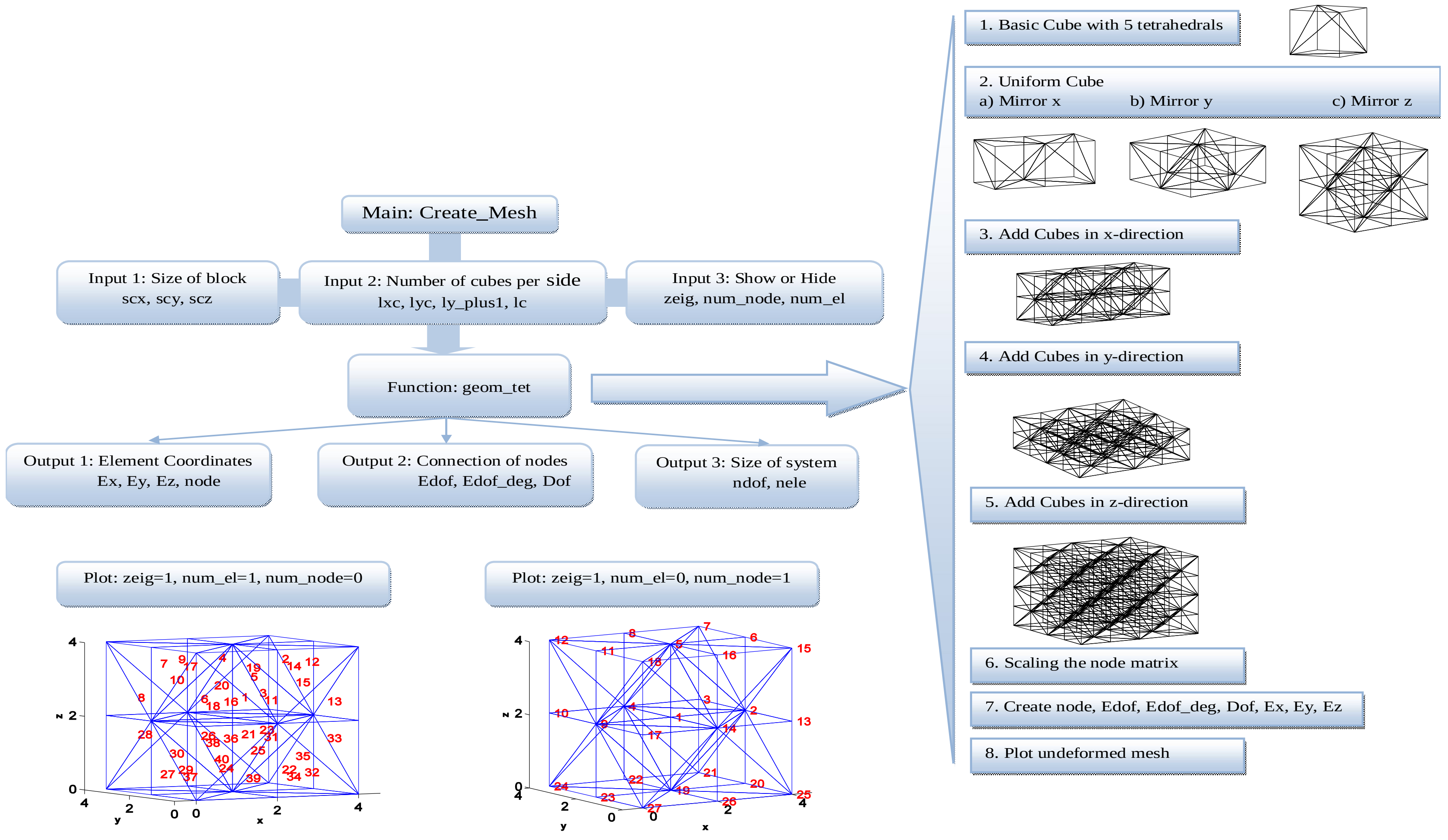
The function *Plot_Mesh_tet* plots every tetrahedral of the given mesh through the element coordinate matrixes **Ex**, **Ey** and **Ez**. The variable *s1* defines the line type and the colour of the plotted figure. If one of the variables *num_node* and *num_el* equals one the node numbers respectively the element numbers are displayed.

B List of tables

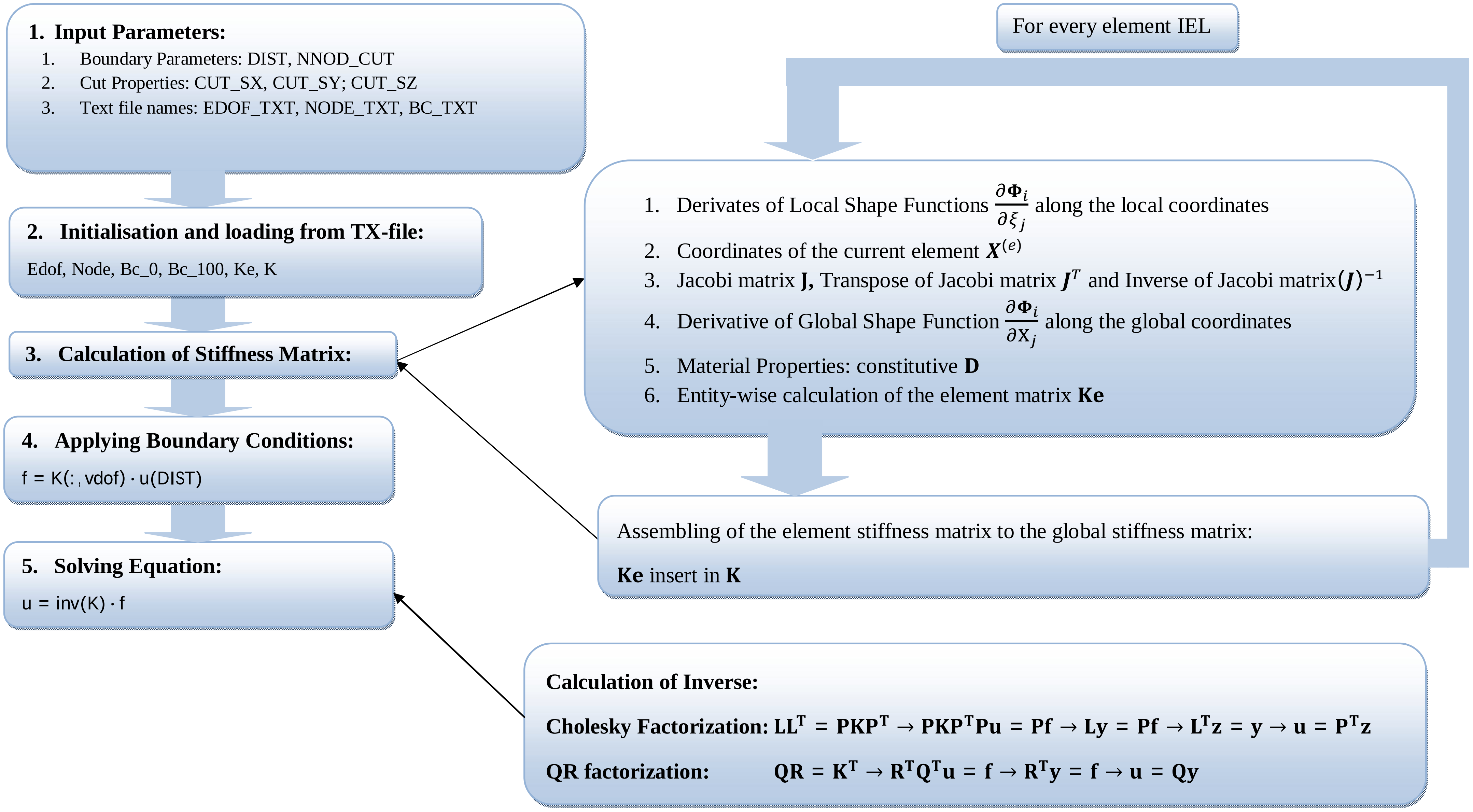
C List of figures

D Programme Structure

1. Mesh Creation:


Main: Create_Mesh
Input 1: Size of block scx, scy, scz
Input 2: Number of cubes per side lxc, lyc, ly_plus1, lc
Input 3: Show or Hide zeig, num_node, num_el
Function: geom_tet
Output 1: Element Coordinates Ex, Ey, Ez, node
Output 2: Connection of nodes Edof, Edof_deg, Dof
Output 3: Size of system ndof, nele
Plot: zeig=1, num_el=1, num_node=0
Plot: zeig=1, num_el=0, num_node=1
1. Basic Cube with 5 tetrahedrals
2. Uniform Cube
a) Mirror x
b) Mirror y
c) Mirror z
3. Add Cubes in x-direction
4. Add Cubes in y-direction
5. Add Cubes in z-direction
6. Scaling the node matrix
7. Create node, Edof, Edof_deg, Dof, Ex, Ey, Ez
8. Plot undeformed mesh

2. Standard FEM calculation



1. Input Parameters:

1. Boundary Parameters: DIST, NNOD_CUT
2. Cut Properties: CUT_SX, CUT_SY; CUT_SZ
3. Text file names: EDOF_TXT, NODE_TXT, BC_TXT

2. Initialisation and loading from TX-file:

Edof, Node, Bc_0, Bc_100, Ke, K

3. Calculation of Stiffness Matrix:

4. Applying Boundary Conditions:

$f = K(:, vdof) \cdot u(DIST)$

5. Solving Equation:

$u = inv(K) \cdot f$

For every element IEL

1. Derivates of Local Shape Functions $\frac{\partial \boldsymbol{\Phi}_i}{\partial \xi_j}$ along the local coordinates
2. Coordinates of the current element $\boldsymbol{X}^{(e)}$
3. Jacobi matrix **J**, Transpose of Jacobi matrix $\boldsymbol{J}^T$ and Inverse of Jacobi matrix $(\boldsymbol{J})^{-1}$
4. Derivative of Global Shape Function $\frac{\partial \boldsymbol{\Phi}_i}{\partial \mathrm{x}_j}$ along the global coordinates
5. Material Properties: constitutive **D**
6. Entity-wise calculation of the element matrix **Ke**

Assembling of the element stiffness matrix to the global stiffness matrix:

Ke insert in **K**

Calculation of Inverse:

Cholesky Factorization: $\mathbf{LL^T = PKP^T \rightarrow PKP^TPu = Pf \rightarrow Ly = Pf \rightarrow L^Tz = y \rightarrow u = P^Tz}$

QR factorization: $\mathbf{QR = K^T \rightarrow R^TQ^Tu = f \rightarrow R^Ty = f \rightarrow u = Qy}$

E List of abbreviations

$\mathbf{t}$	traction vector [N/mm^2]
$\mathbf{P}$	force vector [N]
A	surface area [mm^2]
x_i	Cartesian coordinate system i = 1,2,3
σ_{ij}	stress components i, j = 1,2,3 [N/mm^2]
$\mathbf{S}$	stress tensor [N/mm^2]
$\mathbf{n}$	surface normal
V	arbitrary volume [mm^3]
$\mathbf{b}$	body force vector [N/mm^3]
v_x	arbitrary function x = 1,2,3
$\mathbf{u}$	displacement vector [mm]
ε_{ii}	strain tensor i, j = 1,2,3
θ	angle [rad]
γ_{ij}	shear strain $i \neq j$; i,j can be 1,2,3
E	Young's modulus [N/mm^2]
$\mathbf{D}$	constitutive matrix [N/mm^2]
$\boldsymbol{\phi}$	shape functions (depends on the element type)
$\mathbf{B}$	derivatives of the shape functions
ξ_i	local Cartesian coordinate system in an element
$\mathbf{J}$	Jacobian matrix
a_i	displacement of enriched nodes
ψ_i	enrichment function
H	Heaviside function
$\mathbf{f}$	force vector [N]
$\mathbf{K}$	stiffness matrix[N/mm]
$\mathbf{M}$	mass matrix [g]
ρ	density [g/mm^2]
$\mathbf{D}$	damping matrix [N s/mm]

Reference

[1] *Virtual reality training imporoves operating room performance: Results of a randomized, double-blinded study.* N. E. Seymour, A. G. Gallagher, S.A. Roman, M. K. O'Brien, V. K. Banasal, D. K. Andersen, and R. M. Satava. s.l. : Annals of Surgery, 2002, Vols. 236(4):458-464.

[2] PETERSON, H. and OTTOSEN, N.: *Introduciton to the finite element method.* London : Prentice Hall, 1992.

[3] HUGHES, T.J.R.: *The finite Element Method. Linear Static and Dynamic Finite Element Analysis.* Englewood Cliffs,NJ : Prentice Hall, 1987.

[4] GALLAGHER, R.H.: *Finite Element Analysis: Fundamentals.* Englewood Cliffs,NJ : Prentice Hall, 1975.

[5] ZIENKIEWICZ, O.C., TAYLOR, R.L. and ZHU, J.Z.: *The Finite Element Method.* New York, McGraw-Hill, 1989.

[6] TAIG, I.: *Structural analysis by the matrix displacement method.* s.l. : Enlgish Electric Aviation Report no. So17, 1961.

[7] FRIES, T.P.: *The Extended Finite Element Method(XEFEM).* Braunschweig : TU Braunschweig, 2009.

[8] JERÁBKIVÁ, L.: *Interactive Cutting of Finite Elements based Deformable Objects in Virtual Environments.* Aachen : TU Aachen, 2007.

[9] GRAVOUIL, A., ELGUEDJ, T. and MAIGRE, H.: *An explicit dynamics extended finite element method.* Lyon : Université de Lyon, 2008.

[10] ENGELKE, R.: *Modellierung und Optimierung von Robotern mit einseitigen Bindungen und lokalen Verspannungen.* TU München : Lehrstuhl für Angewandte Mechanik, 2008.

[11] MANZAN, S.: *Entwicklung eines Trajektoriengenerators für den AM-Teleroboter.* München : Lehrstuhl für angewandte Mechanik, TU München, 2010.

[12] LIBERTY, J.: *C++ in 21 Tagen.* München : Markt&Technik Verlag, 1999.

[13] JOSUTTIS, N.: *The C++ Standard Library: A Tutorial and Reference.* Boston : Addison, 1999.

[14] PRESS, W.H.: *Numerical Recipes in C.* Cambridge : CAMBRIDGE UNIVERSITY PRESS, 1992.

[15] EPSILON, MICRON: *Instruction Manual.* Ortenburg : s.n., 2000.

[16] SHREINER, D., et al. *OpenGL Programming Guide.* Boston : Pearson Education, Inc., 2008.

[17] DAHLBLOM, A.: *CALEM a Finite Element Toolbox.* Sweden : KFS AB, 2004.

[18] ZIENKIEWICZ, O.C. *The Finite Element Method: The Basis.* Oxford : Butterworth-Heinemann, 2000.

[19] SCHILLHUBER, G. and ULBRICH, H.: *Real-time FEM for haptic applications under consideration of human perception.* Zürich : s.n., 2007.

[20] SCHILLHUBER, G, ZAEH, M.F. and ULBRICH, H.: *Presence Teleoperators and Virtual Enviroments.* London : The MIT Press, 2007.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form - auch nicht auszugsweise - noch nicht im Rahmen einer anderen Prüfung vorgelegt worden.

München, 19.04.10