Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 314159

Development of a FEM code for fluid-structure coupling

Stephan Herb

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Miriam Mehl
Betreuer/in:	Dipl.-Ing. Florian Lindner
Beginn am:	2015-06-08
Beendet am:	2015-12-08
CR-Nummer:	

Contents

1 Introduction **5**

2 Framework Evaluation **6**
2.1 General Aspects . . . 6
2.2 Frameworks Overview . . . 7
2.2.1 Feel++ . . . 7
2.2.2 OOFEM . . . 8
2.2.3 GetFEM++ . . . 8
2.2.4 MFEM . . . 8
2.2.5 libMesh . . . 9

3 Shell Elements **10**
3.1 Introduction to Linear Elastic Problems . . . 10
3.2 Plane Element . . . 11
3.2.1 Problem Definition . . . 11
3.2.2 Tri-3 Plane Element . . . 12
3.2.3 Quad-4 Plane Element . . . 18
3.3 Plate Bending Element . . . 22
3.3.1 Problem Definition . . . 22
3.3.2 Tri-3 Plate Element . . . 27
3.3.3 Quad-4 Plate Element . . . 31
3.4 Coordinate Transformation . . . 36
3.5 Shell Element . . . 39

4 FEM Code Implementation **43**
4.1 Introduction to libMesh . . . 43
4.2 Implementation Details . . . 45
4.2.1 Initialization . . . 45
4.2.2 Mesh file import . . . 46
4.2.3 System setup . . . 47
4.2.4 Matrix and vector assembly . . . 47
4.2.5 Solving the system . . . 48
4.2.6 Output . . . 48
4.3 Parallelization with MPI . . . 48
4.3.1 libMesh requirements . . . 48
4.3.2 Partitioning the mesh . . . 48
4.3.3 Local elements . . . 48
4.3.4 Assembly changes . . . 48

5 Coupling with preCICE **49**
5.1 Introduction to coupling . . . 49
5.2 Introduction to preCICE . . . 49

5.3 Coupling methods 49
5.4 Implementation 49

6 Validation **50**
6.1 Test A: Membrane Displacement with Tri-3 50
6.2 Test B: Membrane Displacement with Quad-4 50
6.3 Test C: Plate Displacement with Tri-3 50
6.4 Test D: Plate Displacement with Quad-4 50
6.5 Test E: Shell Displacement with Tri-3 50
6.6 Test F: Shell Displacement with Quad-4 50
6.7 Test G: Convergence (increasing number of elements) 51
6.8 Test H: MPI (increasing number of processes) 51
6.9 Test I: Coupling with preCICE 51

7 Conclusion **52**

1 Introduction

Here comes the introduction. And before that the abstract (that needs to be put into LaTeX as special paragraph)

2 Framework Evaluation

Part of the thesis was to find several frameworks which ease the work with the finite element method. An evaluation of frameworks was done to select a suitable one for the given task. The evaluation's criteria are presented as well as a short description of the studied frameworks.

2.1 General Aspects

In preparation of evaluating the frameworks many criteria were created in order to find the most suitable library. The individual aspects are as follows:

- Open Source: All frameworks under consideration need to be published under free license that allows modification and redistribution. The implemented program should be used by anyone without requiring to purchase additional commercial software.

- Parallelization: Due to the need of accelerating calculations, the framework has to support an implementation of the widely used Message Passing Interface (MPI). The library should be able to use MPI internally for its own functions and procedures, but also support the developer with auxiliary function for communicating framework-related data.

- The programming language was chosen to be C++. This aspect is subjective and due to the individual experience of the developer that is larger than, for example, with Python. It is not required that the framework is internally written in C++, but if not, it must provide an interface to C++ to work with.

- Mesh file import: Since the program should be able to work with complex geometries, the definition of such a mesh cannot be done in source code. Therefore, the library should provide a possibility to import meshes from file. The file must contain definitions of the node positions as well as the topology of the elements. Additionally, boundary conditions must be definable through identifiers at nodes or edges. The actual formats of the mesh files supported is not important, as long it supports the addressed requirements. A framework proprietary format would also be possible if it is simply reproducible.

- Linked to the previous aspect is the variety of different finite element types the library has to provide. The current program version uses triangles and quadrilaterals with three and four nodes, respectively. To be able to expand the program's functionality in the future, the library should support elements like six node triangles, nine node quadrilaterals or three dimensional elements like tetrahedra and hexahedra.

- Built-in solvers: The framework must provide a variety of different iterative solvers that can be interchanged at runtime by the user, or at least easily in the source

code. Thus, the user has a higher flexibility when optimizing a problem or comparing different solvers in order to find the best for his or her problem.

- Accessible and detailed documentation: In order to guarantee maintainability and expandability the framework has to have a good documentation. This includes a complete documentation of every class and function, that is actively growing with the library itself. Additional documented example codes help the developer learning how to use the library correctly and efficiently. If problems with the framework occur the developer should be able to get in contact with the framework developers via mailing-lists or forums.

- Up-to-date: The framework should be well maintained and actively supported by its developers to ensure a long term compatibility with new features of the program code.

- The framework should has been used by at least a few projects or has been part of publications. This shows the framework's importance and usability.

- Easy to learn syntax and structure: A rather subjective aspect, not less important. The limited time for this work does not allow studying complicated structures or semantics. The developer should be able to concentrate more on the mathematical/physical problem and less on programming details. This accompanies the choice of the programming language as well as the documentation aspect.

2.2 Frameworks Overview

The following list contains FEM libraries which were evaluated in detail.

2.2.1 Feel++

Feel++ stands for "'Finite Element Embedded Language in C++"' and is a unified framework for finite and spectral Galerkin methods in 1D, 2D and 3D to solve partial differential equations [PCD+12]. It was created in 2005 and still actively maintained with daily commits and the last release version from February, 2015. One main aspect of Feel++ is to have the syntax, semantics and pragmatics close to the mathematics. It allows creation of versatile mathematical kernels for testing and comparing different techniques and methods in solving problems. A second aim is to have a small library that is good manageable but makes use wherever possible of established libraries, for example to solve linear systems. It interfaces seamlessly with MPI simplifying the work's program parallelization. Different mesh file formats are available for import, like the GMSH format and a collection of different finite element type are supported. It is currently used in projects at Cemosis (Center for Modeling and Simulation in Strasbourg, France) including fluid structure interactions, high field magnets simulation and optical tomography [Con].

2.2.2 OOFEM

[Pat09] The "‘Object Oriented Finite Element Method"’ framework is an multi-physics finite element code with object oriented architecture [PB01]. The aim of is to provide an efficient and robust tool for FEM computations as well as to offer highly modular and extensible environment for development [Pat09]; extensible in terms of new element types, boundary conditions or numerical algorithms that can be created by the user. OOFEM focuses on efficient and robust solution to mechanical, transport and fluid problems providing corresponding modules for it. It is written in C++ with focus on high portability between platforms and interfaces various external software libraries like PETSc, ParMETIS, and ParaView. The last stable release were published on February, 2014 but it is still actively developed. The framework is used in several publications [Pat].

2.2.3 GetFEM++

GetFEM++ is a generic finite element C++ library. It aims at providing finite element methods and elementary matrix computations for solving (non-)linear problems numerically. The user describes a model by gathering the variables, data and terms of a problem and some predefined bricks representing classical models. It allows easy switching from one method to another due to separation of geometric transformation or integration methods, for instance. It uses MPI for parallelization, though it is stated that "‘a certain number of procedures are still remaining sequential"’ [SPR] at the time this work were written. While the framework is implemented in C++, it provides interfaces to languages like Python and Matlab. Thus, the framework can be handled by scripts written in non-C++ languages. The latest release is dated from July, 2015 with daily commits from the developers. GetFEM++ is used in project like IceTools [Jar] (open source model for glaciers), EChem++ [BLSS] (Problem Solving Environment for Electochemistry) and SimNIBS [Thi] (software for Simulation of Non-invasive Brain Stimulation) and is part of some publications [PR].

2.2.4 MFEM

The "‘Modular Finite Element Method"’ library acts as a toolbox that provides the building blocks for developing finite element algorithms. MFEM aims to enable research and development of scalable finite element discretization and solver algorithms through general finite element abstractions. It has a wide variety of 2D and 3D finite element types, for example triangular and quadrilateral 2D elements, curved boundary elements or topologically periodic meshes. In addition to Galerkin methods, MFEM supports mixed finite elements, discontinuous Galerkin (DG) methods, or isogeometric analysis methods, for instance. The framework supports MPI-based parallelism throughout the library with minimal changes to the code to make the serial code parallel. A variety of solvers are available for the resulting linear algebra systems, including serial and parallel smoothers, solvers such as PCG, MINRES and GMRES, high-performance preconditioners from the hypre library, to name a few [mfeb]. The latest release of MFEM is dated

January, 2015 but it is still under active development. The library is used in several publications [mfea].

2.2.5 libMesh

[KPSC06] The libMesh library provides a framework for the numerical simulation of partial differential equations using arbitrary unstructured discretizations on serial and parallel platforms. A major goal of the library is to provide support for adaptive mesh refinement (AMR) computations in parallel while allowing a research scientist to focus on the physics they are modeling. It makes use of existing software whenever possible. PETSc can be used for the solution of linear systems on both serial and parallel platforms, and LASPack is included with the library to provide linear solver support on serial machines, for instance. LibMesh supports a variety of 1D, 2D, and 3D geometric and finite element types and seamlessly integrates parallel functionality with MPI throughout the whole library [KPSC06]. The framework is actively developed with daily commits and the latest release is dated from February, 2015. It is part of many publications [lib].

LibMesh was chosen to be the most suitable framework for this thesis. Its seamlessly integration of MPI reduces the effort of parallelization for the user, the required finite element types are supported and a range of many more are available for future expansions of the program's code. Complex geometries can be defined in several different mesh format as well as a simple libMesh particular format. The integration of external libraries like PETSc gives a high flexibility for users to change solvers and/or preconditioners. The library's classes and functions are well documented and a large collection of example codes are available. It is actively maintained by fixing bugs and adding more features. Although the structure and components of the framework are intuitive to understand, guided by the documentation, support by the developers are provided through mailing-lists.

3 Shell Elements

Mathematical fundamentals of shell elements divided into the two parts and the coordinate transformation

3.1 Introduction to Linear Elastic Problems

In the following the fundamental equations of linear elasticity will be considered. Here, the spatial case is used for demonstration, but every lower dimensional problem can easily be derived from it. The following definitions will be used in this thesis:

$$\vec{u}^T = (u\ v\ w)\ \text{displacement vector} \tag{1}$$

$$\vec{f}^T = (f_x\ f_y\ f_z)\ \text{external force vector} \tag{2}$$

The strains and stresses can either be described in form of tensors $\underline{\epsilon}$ and $\underline{\sigma}$, or as vectors $\vec{\epsilon}$ and $\vec{\sigma}$:

$$\underline{\epsilon} = \begin{pmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{pmatrix}; \underline{\sigma} = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{pmatrix} \tag{3}$$

$$\vec{\epsilon}^T = \begin{pmatrix} \epsilon_{xx} & \epsilon_{yy} & \epsilon_{zz} & 2\epsilon_{xy} & 2\epsilon_{yz} & 2\epsilon_{zx} \end{pmatrix}; \vec{\sigma}^T = \begin{pmatrix} \sigma_{xx} & \sigma_{yy} & \sigma_{zz} & \sigma_{xy} & \sigma_{yz} & \sigma_{zx} \end{pmatrix} \tag{4}$$

As stated in [Ste15] the relation between displacements and strains is as follows:

$$\underline{\epsilon} = \frac{1}{2}\left(\nabla \vec{u} + \vec{u}\,\nabla\right); \quad \vec{\epsilon} = \underline{L}\vec{u} \tag{5}$$

Equation 5 relates the displacement vector field $\vec{u}$ with the strain field $\underline{\epsilon}$, or $\vec{\epsilon}$ respectively. Here, $\underline{L}$ is a differential operator. This strain-displacement relation is also called *kinematic relationship* [Ste15].
In general initial strains can exist inside the material for example due to temperature changes or shrinkage. Such initial strains are denoted $\vec{\epsilon_0}$ and the stresses will be influenced by the difference between the actual and initial strains. Additionally one could imagine initial residual stresses $\vec{\sigma_0}$ that can be added to the general equation:

$$\vec{\sigma} = \underline{D}\left(\vec{\epsilon} - \vec{\epsilon_0}\right) + \vec{\sigma_0}, \tag{6}$$

where $\underline{D}$ is the material matrix. In the simplest case of linear elasticity with isotropy, $\underline{D}$ only contains two parameters, namely the elastic modulus E (also known as the Young's modulus) and the Poisson's ratio ν. The former one defines the relationship between the stress and strain in a material, the latter one results as the quotient of the fraction of expansion and the fraction of compression for small changes. In the following the initial conditions are ignored, resulting in the a simpler form of equation 6:

$$\vec{\sigma} = \underline{D}\ \vec{\epsilon} \tag{7}$$

For the said isotropic case $\underline{D}$ results in [ZT00]:

$$\underline{D} = \frac{E}{1-\nu^2}\begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix} \tag{8}$$

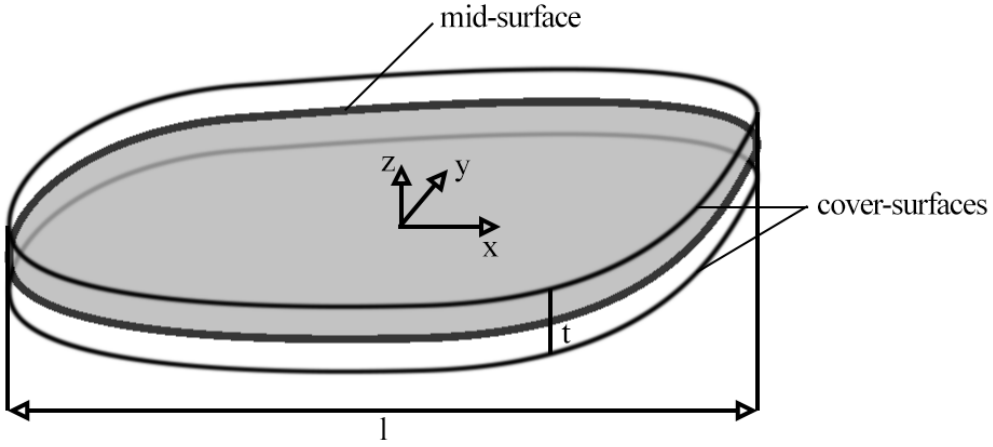

Figure 1: Schematic view of a plane object

3.2 Plane Element

First part of shell element: plane part. derivation of this part with two exemplary finite element types

3.2.1 Problem Definition

In Figure 1 an object is shown which extends to the x and y axis as its primary direction. The extend in z-direction is smaller and denoted by thickness t. The mid-surface located in between the top and bottom surface areas has the coordinate $z = 0$. Its local z-axis equals the normal vector of the mid place. Such an object is called *plane* in the following.

There are two different problem definitions regarding plane elements: Plane stress and plane strain. The directions of displacements u and v along the orthogonal local x and y axis defining its displacement field is a common feature of both problems. Also, both have in common, that only strains and stresses in the xy plane have to be considered: Instead of nine, only three components remain. While in the case of plane stress all other stress components are zero, in plane strain the stress in direction perpendicular to the xy plane is non-zero. In this thesis only plane stress will be discussed in further detail. More information about plane strain is given in [ZT00]. The following conditions must be satisfied such that a plane can be in *plane stress* [Ste15]:

- The thickness t varies only slightly and it must hold: $t/l \ll 1$, with l the extent of the larger side of the plane element.
- The load is applied to the mid place.
- Displacements, strains and stresses are constant across the thickness.

The stress components $\sigma_{xz}, \sigma_{yz}, \sigma_{zz}$ normal to the surface areas with $z \pm t/2$ vanish (equals zero). Therefore only the two normal stress components σ_{xx} and σ_{yy} and the transverse stress component σ_{xy} are left non-zero.

Displacements can only occur in x and y direction. u will be the displacement along x and v along y. The displacement field $\vec{u}$ is as follows:

$$\vec{u} = \begin{pmatrix} u(x,y) & v(x,y) \end{pmatrix}^T \tag{9}$$

The vector for the strain components:

$$\vec{\epsilon} = \begin{pmatrix} \epsilon_{xx} & \epsilon_{yy} & 2\epsilon_{xy} \end{pmatrix}^T \tag{10}$$

Sometimes $2\epsilon_{xy}$ is shortened to γ_{xy} [Ste15]. The vector holding the stress components is similar to that of the strain's vector:

$$\vec{\sigma} = \begin{pmatrix} \sigma_{xx} & \sigma_{yy} & \sigma_{xy} \end{pmatrix}^T \tag{11}$$

The kinematic relationship $\vec{\epsilon} = \underline{L}\vec{u}$ 5 linking the displacements $\vec{u}$ with the strains $\vec{\epsilon}$ at full length:

$$\vec{\epsilon} = \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ 2\epsilon_{xy} \end{pmatrix} = \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \underline{L}\vec{u} \tag{12}$$

With the strains known and considering equation 5 one can calculate the stresses $\vec{\sigma}$:

$$\vec{\sigma} = \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix} \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ 2\epsilon_{xy} \end{pmatrix} = \underline{D}\vec{\epsilon} = \underline{D}\ \underline{L}\vec{u} \tag{13}$$

natural boundary conditions in form of nodal forces: $\vec{F} = \begin{pmatrix} F_x & F_y \end{pmatrix}^T$

The total potential of the plane element problem looks as follows:

$$\Pi = 1/2 \int_V \vec{\epsilon}^T \vec{\sigma} dV - \vec{u}^T \vec{F}, \tag{14}$$

with the first term being the elastic strain energy and the second term the single forces.

3.2.2 Tri-3 Plane Element

In section 3.2.1 the plane's functional was derived. Now the focus is on the functional's discretization. Figure 2 shows a general, planar object defined to be placed in the xy-plane. The first discretization step is to divide the object into single triangles approximating the shape of it. This process is called triangulation. Every one of these triangles then represents a finite element with one node at every corner. The finer the triangulation is done the better the object and its boundary are matched by its discrete

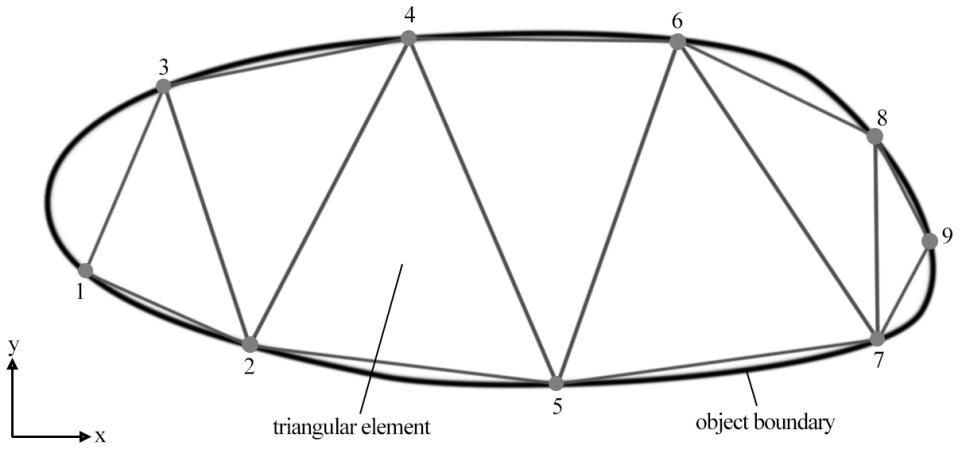

Figure 2: Subdividing plane object with triangular elements

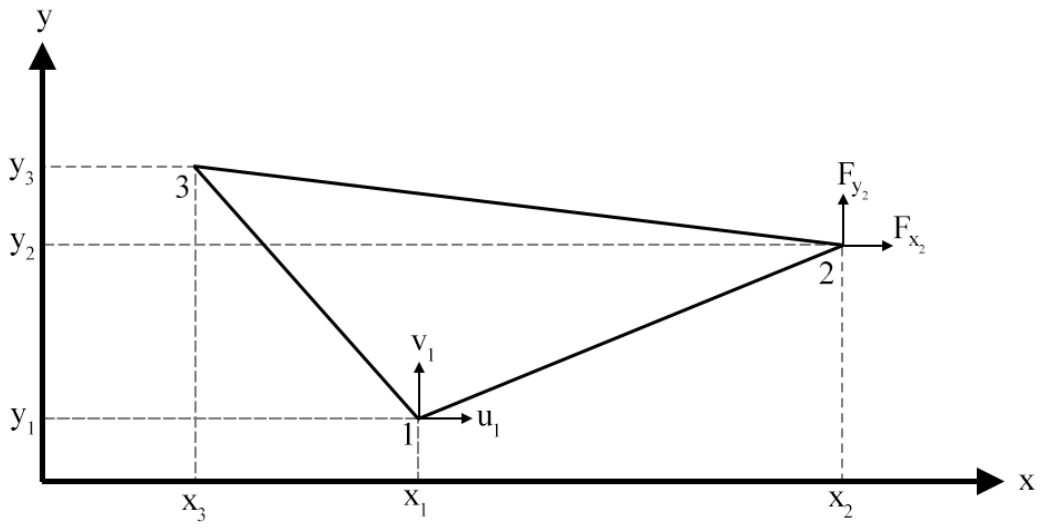

Figure 3: Three node triangular element with its coordinates, nodal displacements and nodal forces

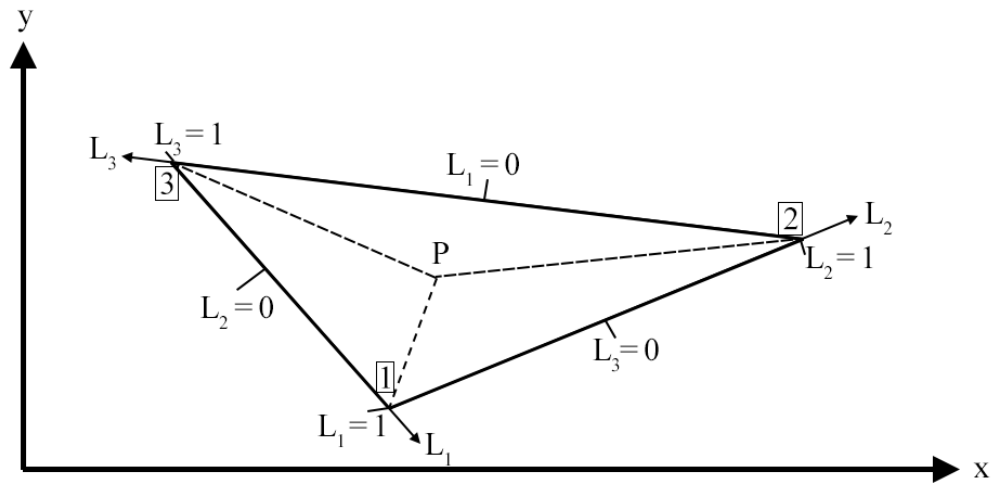

Figure 4: Triangular coordinates L_1, L_2, L_3 and sampling point P within the triangular element

complement, but also the more finite elements have to be considered in later calculations. One triangular finite element is shown in Figure 3. It is defined by the coordinates (x_i, y_i) of its three nodes. Since the element is located in the xy-plane, the z-coordinate is of no interest and will be ignored. At every node, forces can be applied denoted with F_{x_i} and F_{y_i}. Accordingly, every node can be displaced. The movement along the x-axis is denoted with u_i, or with v_i along the y-axis respectively. Note, that the node numbering is in counter-clockwise direction. This definition will be kept throughout the thesis, and is important to remember when implementing the FEM-code in order to reduce errors. In this thesis only triangles defined by three nodes are discussed. There are many more finite elements forming triangles, such as six node triangles or even seven node triangles. The main difference between these types of elements are the order of shape functions. More details about higher order triangular finite elements can be found in

In the case of a three node triangle the basis functions for the two displacements u and v are as follows [Ste15]:

$$u(x,y) = a_0 + a_1 L_1 + a_2 L_2 = \begin{pmatrix} 1 & L_1 & L_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \vec{x}^T \vec{a}, \tag{15}$$

$$v(x,y) = a_0 + a_1 L_1 + a_2 L_2 = \begin{pmatrix} 1 & L_1 & L_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \vec{x}^T \vec{a}, \tag{16}$$

both defined in triangular coordinates (see figure 4).

To get the unknown coefficients a_i, values for the triangular coordinates are set. This

creates a system of linear equations:

$$\begin{aligned} u(L_1 = 1, L_2 = 0) = u_1 \rightarrow u_1 = a_0 + a_1 \\ u(L_1 = 0, L_2 = 1) = u_2 \rightarrow u_2 = a_0 + a_2 \\ u(L_1 = 0, L_2 = 0) = u_3 \rightarrow u_3 = a_0 \end{aligned} \tag{17}$$

Alternatively, this could be also done with v. Written as matrix and vector:

$$\begin{aligned} \underline{A}\vec{a} &= \vec{u} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} &= \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \end{aligned} \tag{18}$$

Now, inverting matrix A the coefficients can be found:

$$\vec{a} = \underline{A}^{-1}\vec{u} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \tag{19}$$

If one put equation 19 into 15, or the analogon into 16, the shape functions for the three node triangular finite element will be derived, as described in [Ste15]:

$$\begin{aligned} u &= \vec{x}^T\vec{a} = \vec{x}^T\underline{A}^{-1}\vec{u} = \vec{N}^T\vec{u} \\ \vec{N}^T &= \vec{x}^T\underline{A}^{-1} = \begin{pmatrix} 1 & L_1 & L_2 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix} \\ &= \begin{pmatrix} L_1 & L_2 & 1 - L_1 - L_2 \end{pmatrix} = \begin{pmatrix} N_1 & N_2 & N_3 \end{pmatrix} \end{aligned} \tag{20}$$

Characteristically for the shape function, as stated in [Ste15], is, that shape function N_i gets the value 1 at node i and 0 at the two other nodes. The functions are linear with respect to L_1 and L_2 which can be noticed in equation 20. As stated before, these shape functions are the same for displacement u and v. With the knowledge of the displacement values of the element's nodes one can formulate the displacement functions in triangular coordinate notation as follows:

$$\begin{aligned} u &= N_1u_1 + N_2u_2 + N_3u_3 \\ v &= N_1v_1 + N_2v_2 + N_3v_3 \end{aligned} \tag{21}$$

Or in matrix form:

$$\begin{aligned} \vec{\tilde{u}} &= \underline{N}\vec{u} \\ \begin{pmatrix} u \\ v \end{pmatrix} &= \begin{pmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{pmatrix} \end{aligned} \tag{22}$$

The vector $\vec{\tilde{u}}$ describes the element's displacements as product of matrix $\underline{N}$ containing the shape functions and vector $\vec{u}$ containing the displacements of the single triangle's nodes. Now, one can put equation 22 into 12:

$$\vec{\epsilon} = \underline{L}\vec{\tilde{u}} = \underline{L}\ \underline{N}\vec{u} = \underline{B}\vec{u} \tag{23}$$

The product of $\underline{L}$ and $\underline{N}$ is called *strain-displacement matrix* $\underline{B}$. In order to calculate the strain-displacement matrix, one has to assemble the $\underline{L}$ matrix containing the first partial derivatives of the triangular element. With the chain rule applied, the partial derivatives look as follows:

$$\begin{aligned} \frac{\partial}{\partial L_1} &= \frac{\partial x}{\partial L_1}\frac{\partial}{\partial x} + \frac{\partial y}{\partial L_1}\frac{\partial}{\partial y} \\ \frac{\partial}{\partial L_2} &= \frac{\partial x}{\partial L_2}\frac{\partial}{\partial x} + \frac{\partial y}{\partial L_2}\frac{\partial}{\partial y} \end{aligned} \tag{24}$$

or in matrix notation:

$$\begin{aligned} \tilde{\nabla} &= \underline{J}\nabla \\ \begin{pmatrix} \frac{\partial}{\partial L_1} \\ \frac{\partial}{\partial L_2} \end{pmatrix} &= \begin{pmatrix} \frac{\partial x}{\partial L_1} & \frac{\partial y}{\partial L_1} \\ \frac{\partial x}{\partial L_2} & \frac{\partial y}{\partial L_2} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}, \end{aligned} \tag{25}$$

where $\underline{J}$ represents the Jacobian matrix, ∇ the partial derivatives in Cartesian coordinates and $\tilde{\nabla}$ the partial derivatives in triangular coordinates. To get the derivatives in Cartesian form the upper equation must be multiplied with the inverse Jacobian matrix $\underline{J}^{-1}$:

$$\underline{J}^{-1} = \frac{1}{|\underline{J}|}\begin{pmatrix} \frac{\partial y}{\partial L_2} & -\frac{\partial y}{\partial L_1} \\ \frac{-\partial x}{\partial L_2} & \frac{\partial x}{\partial L_1} \end{pmatrix} \tag{26}$$

The conversion between triangular and Cartesian coordinates can be summarized as follows (see Figure 4 and [Ste15]):

$$\begin{aligned} L_1 + L_2 + L_3 = 1 \rightarrow L_3 &= 1 - L_1 - L_2 \\ x &= x_1L_1 + x_2L_2 + x_3L_3 = (x_1 - x_3)L_1 + (x_2 - x_3)L2 + x_3 \\ y &= y_1L_1 + y_2L_2 + y_3L_3 = (y_1 - y_3)L_1 + (y_2 - y_3)L2 + y_3 \end{aligned} \tag{27}$$

Considering equation 27 the Jacobian matrix can now be calculated:

$$J = \begin{pmatrix} \frac{\partial x}{\partial L_1} = x_1 - x_3 = x_{13} & \frac{\partial y}{\partial L_1} = y_1 - y_3 = y_{13} \\ \frac{\partial x}{\partial L_2} = x_2 - x_3 = x_{23} & \frac{\partial y}{\partial L_2} = y_2 - y_3 = y_{23} \end{pmatrix} = \begin{pmatrix} x_{13} & y_{13} \\ x_{23} & y_{23} \end{pmatrix} \tag{28}$$

and hence the inverse Jacobian matrix:

$$\underline{J}^{-1} = \frac{1}{2A_\triangle}\begin{pmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{pmatrix} \tag{29}$$

The determinant of the Jacobian matrix is two times the area of the triangle. With the help of equation 29, 25 can be reorganized

$$\nabla = \underline{J}^{-1}\tilde{\nabla} \tag{30}$$

and this finally yields to the new version of the differential operator $\underline{L}$ [Ste15]:

$$\underline{L} = \frac{1}{2A_\triangle}\begin{pmatrix} y_{23}\frac{\partial}{\partial L_1} - y_{13}\frac{\partial}{\partial L_2} & 0 \\ 0 & -x_{23}\frac{\partial}{\partial L_1} + x_{13}\frac{\partial}{\partial L_2} \\ -x_{23}\frac{\partial}{\partial L_1} + x_{13}\frac{\partial}{\partial L_2} & y_{23}\frac{\partial}{\partial L_1} - y_{13}\frac{\partial}{\partial L_2} \end{pmatrix} \tag{31}$$

Next, the strain-displacement matrix $\underline{B}$ can be calculated:

$$\begin{aligned} \underline{B} &= \underline{L}\,\underline{N} \\ &= \frac{1}{2A_\triangle}\begin{pmatrix} y_{23}\frac{\partial}{\partial L_1} - y_{13}\frac{\partial}{\partial L_2} & 0 \\ 0 & -x_{23}\frac{\partial}{\partial L_1} + x_{13}\frac{\partial}{\partial L_2} \\ -x_{23}\frac{\partial}{\partial L_1} + x_{13}\frac{\partial}{\partial L_2} & y_{23}\frac{\partial}{\partial L_1} - y_{13}\frac{\partial}{\partial L_2} \end{pmatrix} \\ &\quad \begin{pmatrix} L_1 & 0 & L_2 & 0 & 1-L_1-L_2 & 0 \\ 0 & L_1 & 0 & L_2 & 0 & 1-L_1-L_2 \end{pmatrix} \\ &= \frac{1}{2A_\triangle}\begin{pmatrix} y_{23} & 0 & -y_{13} & 0 & y_{12} & 0 \\ 0 & -x_{23} & 0 & x_{13} & 0 & -x_{12} \\ -x_{23} & y_{23} & x_{13} & -y_{13} & -x_{12} & y_{12} \end{pmatrix} \end{aligned} \tag{32}$$

With $\underline{B}$ known, one can insert equation 23 into 13 to get the stresses:

$$\vec{\sigma} = \underline{D}\,\underline{B}\vec{u} \tag{33}$$

Finally, every term of the plane element's functional 14 can be filled with the above discretized terms:

$$\begin{aligned} \Pi &= \frac{1}{2}\int_V \vec{\epsilon}^T\vec{\sigma}\,dV - \vec{u}^T\vec{F} \\ &= \frac{1}{2}\int_V \vec{u}^T\underline{B}^T\underline{D}\,\underline{B}\vec{u}\,dV - \vec{u}^T\vec{F} \\ &= \frac{1}{2}\vec{u}^T\int_V \underline{B}^T\underline{D}\,\underline{B}\,dV\,\vec{u} - \vec{u}^T\vec{F} \\ &= \frac{1}{2}\vec{u}^T\underline{K}\vec{u} - \vec{u}^T\vec{F} \end{aligned} \tag{34}$$

with $\underline{K}$ the stiffness matrix and $\vec{F}$ the nodal force vector.

The variation of the functional 34 is as follows [Ste15]:

$$\begin{aligned} \delta\Pi &= \frac{\partial\Pi}{\partial\vec{u}}\delta\vec{u} = 0 \\ &= \frac{1}{2}\delta\vec{u}^T\frac{\partial\vec{u}^T}{\partial\vec{u}^T}\underline{K}\vec{u} + \frac{1}{2}\vec{u}^T\underline{K}\frac{\partial\vec{u}}{\partial\vec{u}}\delta\vec{u} - \delta\vec{u}\frac{\partial\vec{u}^T}{\partial\vec{u}^T}\vec{F} \\ &= \delta\vec{u}^T\left(\underline{K}\vec{u} - \vec{F}\right) = 0 \end{aligned} \tag{35}$$

In order to satisfy this equation, the term in between the parenthesis must be zero ($\delta\vec{u}^T$ can have arbitrary values). This leads to the equilibrium equation of the triangular plane element as described in [Ste15]:

$$\underline{K}\vec{u} = \vec{F} \tag{36}$$

Since the thickness t of the element is constant per definition, it is $dV = t\ dA$ and therefore the integral of the stiffness matrix changes to:

$$\underline{K} = t \int_A \underline{B}^T \underline{D}\ \underline{B}\ dA = tA_{\triangle} \underline{B}^T \underline{D}\ \underline{B} \tag{37}$$

3.2.3 Quad-4 Plane Element

Sometimes it can be beneficial to use quadrilateral elements when describing a mesh. In contrast to triangles which always lie, due to their simple shape, in a plane, quadrilateral can have more complex forms. Such cases include for example: The fourth nodes does not lie in the plane defined by the other three or the shape is not convex. It is difficult to deal with such forms and one could be tempted to restrict the element to have rectangular shapes only, because these are easy to formulate and work with. But they are impractical when complicated geometry is to be modeled, especially if details should be emphasized in fine graduation.

One solution to this problem is the use of isoparametric elements. They can be non-rectangular. The trick is to use reference coordinates which map the physical element into a reference element that is a square. Thus, the physical element can have a more general shape, but a coordinate transformation and numerical integration is needed which brings in more mathematical complexity [CMPW02].

In this section quadrilateral isoparametric elements consisting of four nodes are described, but one can expand it to eight or nine node elements. Figure 5 shows the two abstraction layers: On the left side the original element is shown in physical space, on the right side the reference element is shown. The square has a side length of 2. The coordinate system with the ξ and η axis has its origin in the center of the square. Also note the numbering of the nodes is again counter-clockwise.

Similarly to the triangular element, interpolating the displacement field as well as the element geometry is done by shape functions. They are defined in reference coordinates. The displacement of a point within the element can be expressed by the displacements at the nodes and shape functions $\underline{N}$. Also, the position of that point can be expressed in terms of the (global) nodal positions and shape functions $\underline{\tilde{N}}$. The element is called *isoparametric* if $\underline{N}$ is identical to $\underline{\tilde{N}}$. If $\underline{\tilde{N}}$ is of lower degree than $\underline{N}$, the element is called *subparametric* and *superparametric* if it is the other way around [CMPW02].

Every node has two degrees of freedom: A displacement u along the x-axis and a displacement v along the y-axis. To find the shape functions it does not matter which variable to choose, so the following basis function was used for ϕ which can either

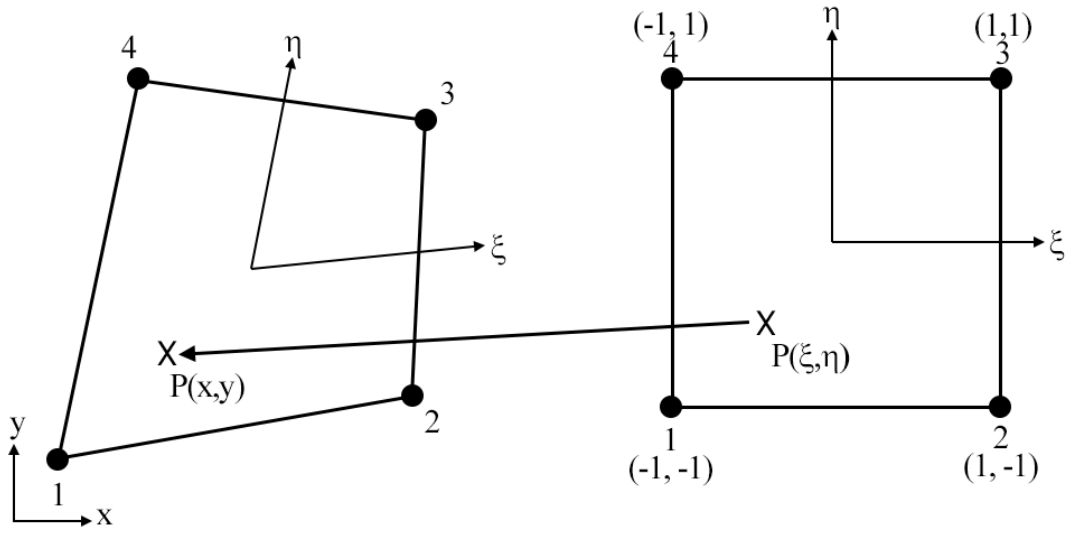

Figure 5: Coordinate transformation of four node quadrilateral element with physical space on the left side and reference space on the right side

represent u or v [Ste15]:

$$\phi(\xi,\eta) = a_0 + a_1\xi + a_2\eta + a_3\xi\eta = \begin{pmatrix} 1 & \xi & \eta & \xi\eta \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \vec{x}^T\vec{a} \tag{38}$$

The interpolation conditions at the nodes are as follows:

$$\begin{aligned} \phi(-1,-1) = \phi_1 \rightarrow \phi_1 &= a_0 - a_1 - a_2 + a_3 \\ \phi(1,-1) = \phi_2 \rightarrow \phi_2 &= a_0 + a_1 - a_2 - a_3 \\ \phi(1,1) = \phi_3 \rightarrow \phi_3 &= a_0 + a_1 + a_2 + a_3 \\ \phi(-1,1) = \phi_4 \rightarrow \phi_4 &= a_0 - a_1 + a_2 - a_3 \end{aligned} \tag{39}$$

or in matrix notation:

$$\begin{gathered} \underline{A}\vec{a} = \vec{\phi} \\ \begin{pmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix} \end{gathered} \tag{40}$$

Inversion of $\underline{A}$ yields the coefficients a_i:

$$\vec{a} = \underline{A}^{-1}\vec{\phi} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix} \tag{41}$$

If the last equation is inserted into 38 one gets the shape functions $\vec{N}$ for the quadrilateral element:

$$\begin{aligned}
\phi &= \vec{x}^T \underline{A}^{-1} \vec{\phi} \\
&= \vec{N}^T \vec{\phi} \\
&= \frac{1}{4} \begin{pmatrix} 1 & \xi & \eta & \xi\eta \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \vec{\phi} \\
&= \begin{pmatrix} \frac{1}{4}(1-\xi)(1-\eta) & \frac{1}{4}(1+\xi)(1-\eta) & \frac{1}{4}(1+\xi)(1+\eta) & \frac{1}{4}(1-\xi)(1+\eta) \end{pmatrix} \vec{\phi} \qquad (42)
\end{aligned}$$

To check the correctness of the four shape function one can evaluate shape function i with $\xi\eta$-coordinates of node i. It must evaluate to 1 while at any other node coordinate it must be zero. Now, the displacements can be expressed as follows:

$$\vec{\tilde{u}} = \begin{pmatrix} u \\ v \end{pmatrix} = \underline{N}\vec{u} = \begin{pmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{pmatrix} \qquad (43)$$

with $\underline{N}$ being the matrix containing the shape functions and $\vec{u}$ being the vector of the nodal displacements.

The assembly of the strain-displacement matrix $\underline{B}$ is more complicated with isoparametric elements. Due to the $\xi\eta$-coordinates one cannot easily describe an operator such as $\partial/\partial x$. The first step to achieve it, is to formulate a function $\phi = \phi(\xi, \eta)$. Like in the derivation on the shape functions ϕ can represent u or v. Derivatives with respect to ξ and η are as follows [CMPW02]:

$$\begin{aligned}
\frac{\partial \phi}{\partial \xi} &= \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial \xi} \\
\frac{\partial \phi}{\partial \eta} &= \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial \eta}
\end{aligned} \qquad (44)$$

or in matrix notation:

$$\vec{\tilde{\phi}} = \underline{J}\vec{\phi} \qquad (45)$$

where $\underline{J}$ is the Jacobian matrix:

$$\underline{J} = \begin{pmatrix} \sum N_{i,\xi} x_i & \sum N_{i,\xi} y_i \\ \sum N_{i,\eta} x_i & \sum N_{i,\eta} y_i \end{pmatrix} \qquad (46)$$

where $N_{i,j}$ denotes the derivation of the i-th shape function with respect to j and x_i the i-th component of the $\vec{x}$ vector. The Jacobian matrix can be written out as follows:

$$\begin{aligned}\underline{J} &= \frac{1}{4}\begin{pmatrix}-(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta)\\ -(1-\xi) & -(1+\xi) & (1+\xi) & (1-\xi)\end{pmatrix}\begin{pmatrix}x_1 & y_1\\ x_2 & y_2\\ x_3 & y_3\\ x_4 & y_4\end{pmatrix}\\ &= \begin{pmatrix}(x_{12}+x_{34})\eta - x_{12} + x_{34} & (y_{12}+y_{34})\eta - x_{12} + y_{34}\\ (x_{12}+x_{34})\xi - x_{13} - x_{24} & (y_{12}+y_{34})\xi - y_{13} + y_{24}\end{pmatrix}\end{aligned} \tag{47}$$

Next, equation 45 can be rearranged to get the derivatives with respect to x and y:

$$\vec{\phi} = \underline{J}^{-1}\vec{\tilde{\phi}} \tag{48}$$

With the derivatives calculated, the strain-displacement relation 5 can be obtained [CMPW02]:

$$\vec{\epsilon} = \begin{pmatrix}\frac{\partial u}{\partial x}\\ \frac{\partial v}{\partial y}\\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\end{pmatrix} = \underbrace{\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 1 & 1 & 0\end{pmatrix}}_{\underline{L}}\begin{pmatrix}\partial u/\partial x\\ \partial u/\partial y\\ \partial v/\partial x\\ \partial v/\partial y\end{pmatrix} \tag{49}$$

$$\begin{pmatrix}\partial u/\partial x\\ \partial u/\partial y\\ \partial v/\partial x\\ \partial v/\partial y\end{pmatrix} = \underbrace{\begin{pmatrix}j_{11} & j_{12} & 0 & 0\\ j_{21} & j_{22} & 0 & 0\\ 0 & 0 & j_{11} & j_{12}\\ 0 & 0 & j_{21} & j_{22}\end{pmatrix}}_{\hat{\underline{J}}}\begin{pmatrix}\partial u/\partial \xi\\ \partial u/\partial \eta\\ \partial v/\partial \xi\\ \partial v/\partial \eta\end{pmatrix} \tag{50}$$

$$\begin{pmatrix}\partial u/\partial \xi\\ \partial u/\partial \eta\\ \partial v/\partial \xi\\ \partial v/\partial \eta\end{pmatrix} = \underbrace{\begin{pmatrix}N_{1,\xi} & 0N_{2,\xi} & 0 & N_{3,\xi} & 0 & N_{4,\xi} & 0\\ N_{1,\eta} & 0N_{2,\eta} & 0 & N_{3,\eta} & 0 & N_{4,\eta} & 0\\ 0 & N_{1,\xi} & 0N_{2,\xi} & 0 & N_{3,\xi} & 0 & N_{4,\xi}\\ 0 & N_{1,\eta} & 0N_{2,\eta} & 0 & N_{3,\eta} & 0 & N_{4,\eta}\end{pmatrix}}_{\hat{\underline{N}}}\begin{pmatrix}u_1\\ v_1\\ u_2\\ v_2\\ u_3\\ v_3\\ u_4\\ v_4\end{pmatrix} \tag{51}$$

$$\underline{B} = \underline{L}\,\hat{\underline{J}}\,\hat{\underline{N}} \tag{52}$$

where j_i denotes the i-th entry in the inverse Jacobian matrix.

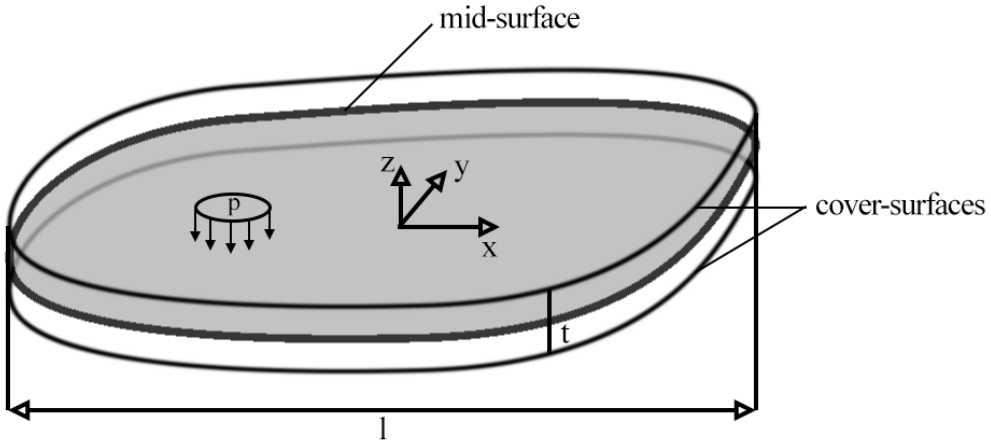

Figure 6: Schematic drawing of a Kirchhoff plate with its main dimension l, thickness t and loading p normal to the mid-surface

Together with the functional equation 34 and the material matrix $\underline{D}$ 13, the stiffness matrix for the quadrilateral isoparametric element can be written as:

$$\underline{K} = \int_V \underline{B}^T \underline{D}\, \underline{B}\, dV = t \int_A \underline{B}^T \underline{D}\, \underline{B}\, dA = t \int_{-1}^{1} \int_{-1}^{1} \underline{B}^T \underline{D}\, \underline{B}\, |\underline{J}| d\xi d\eta \tag{53}$$

For the four node element a Gaussian quadrature needs 2x2 Gauss integration points to satisfy the above equation [Ste15]. These four points are located at $\xi_i = \pm\frac{\sqrt{3}}{3}$ and $\eta_i = \pm\frac{\sqrt{3}}{3}$ with weight factors $\omega_i = 1$. The equation for the stiffness matrix can then be written in discretized form as follows:

$$\underline{K} = t \sum_{i=1}^{2} \sum_{j=1}^{2} \omega_i \omega_j \underline{B}(\xi_i, \eta_j)^T \underline{D}\, \underline{B}(\xi_i, \eta_j) |\underline{J}(\xi_i, \eta_j)| \tag{54}$$

3.3 Plate Bending Element

Second part of shell element: plate part. derivation of this part with two exemplary finite element types

3.3.1 Problem Definition

In contrast to a plane, where the load is located planar with respect to the plane, the load is perpendicular to the mid place at a plate. Therefore plate element problems are important for supporting structures of bridges or ceilings and floors in buildings, for example. In Figure 6 one can see a generalized plate object. It has a main dimension of l and a constant thickness t. With the assumption that $t \ll l$, the problem becomes two dimensional and, instead of the whole object, only the middle plane between the two surface areas will be considered. The object has a local coordinate system with its

xy-plane the mid plane and its z-axis perpendicular to this plane. The surface areas are located at $z = \pm t/2$. As stated in the beginning, the load is applied in z-direction, i.e. normal to the mid-surface.

In this work Kirchhoff's theory of thin plates is used. For thick plates or laminated plates, the theory of Reissner-Mindlin is more applicable. The main difference is that with Reissner-Mindlin plates one takes the shear deformations into account. Thus, the normal to the mid-surface remains straight but not necessarily perpendicular to it; instead of a Kirchhoff plate: Here, the normal remains normal to the mid-surface even after deformation.

As a short summarize the following conditions must be satisfied for a Kirchhoff plate [Ste15]:

- The thickness t must be much smaller than the main dimension l: $t \ll l$.
- Straight lines normal to the mid-surface remain straight after deformation.
- Straight lines normal to the mid-surface remain normal to the mid-surface after deformation.
- There is only a small amount of deformation w, i.e. $w < t$ and it holds $w \neq w(z)$.
- The plate is symmetrical to the mid-surface and changes in thickness must be very small.
- Normal stresses in z-direction σ_{zz} will be neglected.

With [Kle13] and [Ste15] the following displacement terms can be formulated:

$$w = w(x, y) \tag{55}$$

$$u = -z\frac{\partial w}{\partial x} \tag{56}$$

$$v = -z\frac{\partial w}{\partial y} \tag{57}$$

The deformation w suffices to explain the whole displacement vector. The two derivatives in the equations above describe the torsions around the x- and y-axis.

Similar to the plane element, the Kirchhoff plate element can have a plane strain or

plane stress, respectively [Ste15], i.e. equation 5 can be applied here, too:

$$\begin{aligned}
\vec{\hat{u}} &= \begin{pmatrix} u \\ v \end{pmatrix} = -z \begin{pmatrix} \frac{\partial w}{\partial x} \\ \frac{\partial w}{\partial y} \end{pmatrix} = -z\nabla w \\
\vec{\epsilon} &= \underline{L}\vec{\hat{u}} = -z\underline{L}\nabla w = -z\vec{\Delta}w = -z\vec{\kappa} \qquad (58) \\
\vec{\Delta} &= \underline{L}\nabla = \begin{pmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{pmatrix} = \begin{pmatrix} \frac{\partial^2}{\partial x^2} \\ \frac{\partial^2}{\partial y^2} \\ 2\frac{\partial^2}{\partial x \partial y} \end{pmatrix} \\
\vec{\kappa} &= \vec{\Delta}w = \begin{pmatrix} \frac{\partial^2 w}{\partial x^2} \\ \frac{\partial^2 w}{\partial y^2} \\ 2\frac{\partial^2 w}{\partial x \partial y} \end{pmatrix} \qquad (59)
\end{aligned}$$

Referring [Kle13] ($\sigma_{zz} = 0, \tau_{xz} = \tau_{yz} = 0$), equation 6 can be filled with the above information:

$$\begin{aligned}
\vec{\sigma} &= \underline{D}\vec{\epsilon} = -z\underline{D}\vec{\kappa} \\
&= -\frac{Ez}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix} \begin{pmatrix} \kappa_x \\ \kappa_y \\ 2\kappa_{xy} \end{pmatrix} \qquad (60)
\end{aligned}$$

The integration of the stresses $\vec{\sigma}$ over the thickness results in the vector of moments $\vec{M}^T = (M_{xx} \quad M_{yy} \quad M_{xy})$ [Ste15]:

$$\vec{M} = \int_{-t/2}^{t/2} z\vec{\sigma}dz = -\int_{-t/2}^{t/2} z^2 \underline{D}\vec{\kappa}dz = -\underline{D}\vec{\kappa}\int_{-t/2}^{t/2} z^2 dz = -\frac{t^3}{12}\underline{D}\vec{\kappa} = -\underline{D}_p\vec{\kappa} \qquad (61)$$

The above equation relates the moments with the curvatures of the plate. The integrals over the transverse stresses σ_{xz} and σ_{yz} lead to the following shear forces, as described in [Ste15]:

$$\begin{aligned}
Q_x &= \int_{-t/2}^{t/2} \sigma_{xz}dz = \int_{-t/2}^{t/2} \sigma_{xz}^{\max} \left(1 - 4\left(\frac{z}{t}\right)^2\right) dz = \frac{2}{3}\sigma_{xz}^{\max} t \\
&= \frac{2}{3}\sigma_{xz}(z=0)t \qquad (62)
\end{aligned}$$

$$\begin{aligned}
Q_y &= \int_{-t/2}^{t/2} \sigma_{yz}dz = \int_{-t/2}^{t/2} \sigma_{yz}^{\max} \left(1 - 4\left(\frac{z}{t}\right)^2\right) dz = \frac{2}{3}\sigma_{yz}^{\max} t \\
&= \frac{2}{3}\sigma_{yz}(z=0)t \qquad (63)
\end{aligned}$$

The transverse stress is distributed quadratically over the thickness t, i.e. they have their maximum at $z = 0$ and vanish at $z = \pm t/2$. The equilibrium of forces in z-direction leads to:

$$\frac{\partial Q_x}{\partial x} + \frac{\partial Q_y}{\partial y} + p = 0 \qquad (64)$$

with p the load applied perpendicular to the mid-surface. Additionally the equilibrium of moments around the x- and y-axis:

$$\begin{aligned} \frac{\partial M_{xx}}{\partial x} + \frac{\partial M_{xy}}{\partial y} + Q_x = 0 \\ \frac{\partial M_{yy}}{\partial y} + \frac{\partial M_{xy}}{\partial x} + Q_y = 0 \end{aligned} \tag{65}$$

Putting equation 65 into 64 results in:

$$\frac{\partial^2 M_{xx}}{\partial x^2} + \frac{\partial^2 M_{yy}}{\partial y^2} + 2\frac{\partial^2 M_{xy}}{\partial x \partial y} = \vec{\Delta}^T \vec{M} = p \tag{66}$$

Now, one can insert the kinematic equation 59 into equation 61 and then into the equilibrium relation 66:

$$\begin{aligned} \vec{\kappa} &= \vec{\Delta} w \\ \vec{M} &= -\underline{D}_p \vec{\kappa} = -\underline{D}_p \vec{\Delta} w \\ \vec{\Delta}^T \vec{M} &= -\vec{\Delta}^T \underline{D}_p \vec{\Delta} w = p \end{aligned} \tag{67}$$

The last equation leads to the partial differential equation of the plate bending [Kle13]:

$$\frac{\partial^4 w}{\partial x^4} + \frac{\partial^4 w}{\partial y^4} + 2\frac{\partial^4 w}{\partial x^2 \partial y^2} = -\frac{12(1-\nu^2)}{Et^3} p = \frac{p}{k} \tag{68}$$

with k denoted as *plate stiffness*.

Let P be a point on the continuous boundary of the plate with a local Cartesian coordinate system as described in [Ste15] (see Figure 7): The n coordinate is perpendicular to the boundary surface, the s axis tangential to it. The third axis equals the global z-axis of the plate. There are three essential and three natural boundary conditions defined for P: The displacement w, the drills $\theta_n = \partial w / \partial s, \theta_s = -\partial w / \partial n$, the shear force Q_n and the moments M_{ns} and M_{nn}. Since this leads to an inconsistency with the differential equation above, [Ste15] stated that Kirchhoff introduced new forces:

$$V_n = Q_n - \frac{\partial M_{ns}}{\partial s} \tag{69}$$

With them, only the four conditions for w, θ_s, V_n and M_{nn} occur. The plate can be mounted in different ways:

- clamped: $w = 0, \theta_s = -\partial w / \partial n = 0$
- simple supported: $w = 0, M_{nn} = 0$
- symmetrical edge: $\theta_s = -\partial w / \partial n = 0, V_n = 0$

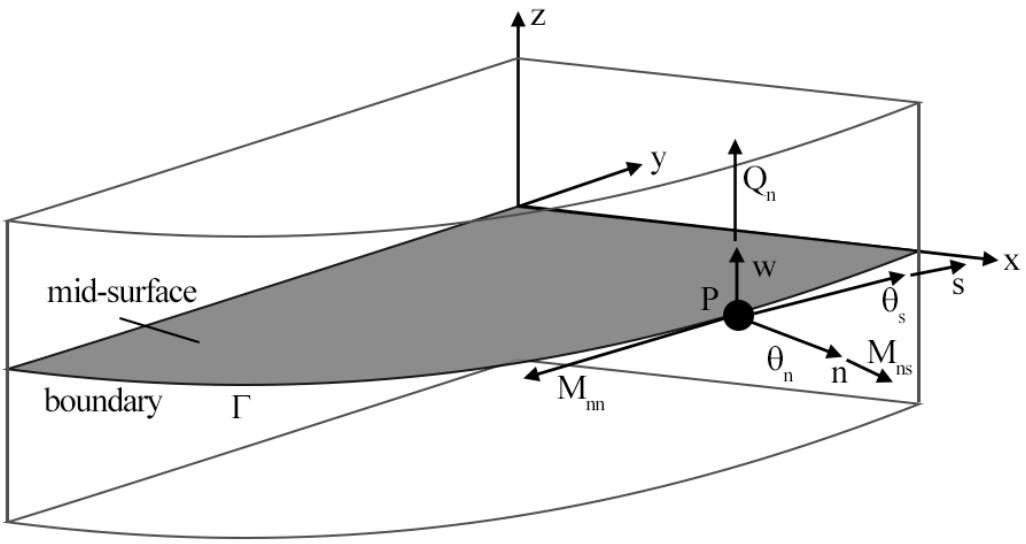

Figure 7: Part of a plate's boundary with its essential and natural boundary conditions

The plate's functional as described in [Ste15] is given below:

$$\frac{1}{2}\int_V \vec{\epsilon}^T \vec{\sigma}\, dV \tag{70}$$

One can insert equation 58 and 60 into the functional:

$$\frac{1}{2}\int_V \vec{\epsilon}^T \vec{\sigma}\, dV = \frac{1}{2}\int_V \vec{\kappa}^T \underline{D}\vec{\kappa} z^2\, dV = \frac{1}{2}\int_A \vec{\kappa}^T \underline{D}_p \vec{\kappa}\, dA \tag{71}$$

Together with the potential of the external forces the overall potential of the Kirchhoff plate is:

$$\Pi = \frac{1}{2}\int_A \vec{\kappa}^T \underline{D}_p \vec{\kappa}\, dA - \int_A p\ w\, dA - \int_\Gamma \left(V_n w - M_{nn}\theta_s\right)\, d\Gamma \tag{72}$$

Klein [Kle13] states that for the plate element discretization additional conditions must be satisfied. They are: The bending $w(x, y)$ as well as the normal derivative $\partial w/\partial n$ at the element's boundary must be continuous to the neighboring elements. This would be the case if the bending and the normal derivative are explicitly determined by the nodal parameters at the border. Further, Klein lists requirements for a plate element ansatz:

- Totality of the displacement approach in order to guarantee good convergence.
- The terms $1, x, y, x^2, xy, y^2$ should be included to get variable strains, curvatures and rigid body motion.

Steinke [Ste15] expands the requirements as follows:

- Compatibility of the displacement variable at the element's boundary (conformity condition): If the steadiness of the deformation w and its first derivatives is not satisfied the bending surface between two elements can have a sharp bend at which the elements are overlapping at one side and diverge on the opposite side. If such a behavior is shown, the element is called *non-conforming*.

- Rigid body motions must not create strains and stresses in the element. This requires a constant term in the basis function for the translative part of the motion and a linear term for the rotatory.

- The basis function must provide constant plain strain and plain stress: If the element converges in its size until it becomes a point, a constant state of bending must be describable in this situation. Since the bending is described as second order derivatives of w, the basis function must include quadratic terms.

The following sections show details of two discretizations of plate elements: A triangular element with three nodes and a quadrilateral element with four nodes.

3.3.2 Tri-3 Plate Element

There exists many different types of triangular plate elements, for example Batoz et al. [BBH80], Tocher [Toc63] or Specht [Spe88]. The three node triangular element from [Toc63] has three degrees of freedom (d.ȯ.f̈) (w, θ_x, θ_y) per node. His basis function was a complete cubic polynomial. The term xy was left out, because the polynomial has one coefficient more than the element has d.ȯ.f̈. This leads to the problem that no constant state of bending can be described (non-conforming element) and this leads to wrong results at convergence [Ste15]. Therefore, Steinke challenges the practical use of this element. A possible way to use a complete cubic polynomial would be to add another node in the center of mass of the triangle and assign the only degree of freedom w to it [Ste15]. But the problem of non-conformity persist, as the nodal twists don't suffice to describe the twists along the element's edges, which are quadric. Here, additional nodes on the edges would be needed. To get a conforming element one can choose a basis function with a complete polynomial of order five. It has 21 coefficients and d.o.f. They are distributed as follows: Every node has six d.o.f $(w, \partial w/\partial x, \partial w/\partial y, \partial^2 w/\partial x^2, \partial^2 w/\partial y^2, \partial^2 w/\partial x \partial y)$ and the mid node of every edge gets the degree of freedom $\partial w/\partial n$. This results in continuous element edge twists, conformity and convergence. The problem: 21 d.o.f. per element leads to high computational effort and second order derivatives at the boundaries are needed. Hence, Steinke advices against using it in practice [Ste15].

In this work an element from [Spe88] were implemented which is also described in [Ste15]. It has three nodes and also three d.o.f. per node: The deformation w and the

two twists θ_x and θ_y. The basis function for the deformation w is as follows:

$$\begin{aligned} w =& a_0 L_1 + a_1 L_2 + a_2 L_3 + a_3 L_1 L_2 + a_4 L_2 L_3 + a_5 L_3 L_1 \\ &+ a_6 \left(L_2 L_1^2 + \frac{1}{2} L_1 L_2 L_3 \left(3(1-\mu_3) L_1 - (1+3\mu_3) L_2 + (1+3\mu_3) L_3\right)\right) \\ &+ a_7 \left(L_3 L_2^2 + \frac{1}{2} L_1 L_2 L_3 \left(3(1-\mu_1) L_2 - (1+3\mu_1) L_3 + (1+3\mu_1) L_1\right)\right) \\ &+ a_8 \left(L_1 L_3^2 + \frac{1}{2} L_1 L_2 L_3 \left(3(1-\mu_2) L_3 - (1+3\mu_2) L_1 + (1+3\mu_2) L_2\right)\right) \end{aligned} \tag{73}$$

with

$$\begin{aligned} \mu_1 &= \frac{S_{21} - S_{31}}{S_{32}} \\ \mu_2 &= \frac{S_{32} - S_{21}}{S_{31}} \\ \mu_3 &= \frac{S_{31} - S_{32}}{S_{21}} \end{aligned} \tag{74}$$

$$\begin{aligned} S_{32} &= x_{32}^2 + y_{32}^2 \\ S_{31} &= x_{31}^2 + y_{31}^2 \\ S_{21} &= x_{21}^2 + y_{21}^2 \end{aligned} \tag{75}$$

S_{ij} denoted the square of the length of the edge between node i and j. This can be written is vector form:

$$\begin{aligned} w &= \vec{x}^T \vec{a} \\ \vec{x} &= \begin{pmatrix} L_1 \\ L_2 \\ L_3 \\ L_1 L_2 \\ L_2 L_3 \\ L_3 L_1 \\ \left(L_2 L_1^2 + \frac{1}{2} L_1 L_2 L_3 \left(3(1-\mu_3) L_1 - (1+3\mu_3) L_2 + (1+3\mu_3) L_3\right)\right) \\ \left(L_3 L_2^2 + \frac{1}{2} L_1 L_2 L_3 \left(3(1-\mu_1) L_2 - (1+3\mu_1) L_3 + (1+3\mu_1) L_1\right)\right) \\ \left(L_1 L_3^2 + \frac{1}{2} L_1 L_2 L_3 \left(3(1-\mu_2) L_3 - (1+3\mu_2) L_1 + (1+3\mu_2) L_2\right)\right) \end{pmatrix} \\ \vec{a}^T &= \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \end{pmatrix} \end{aligned} \tag{76}$$

The twists θ_x and θ_y are yet described in Cartesian coordinates. They must be transformed into triangular coordinates with the help of equation 25:

$$\vec{\theta} = \begin{pmatrix} \theta_x \\ \theta_y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \nabla w = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \underline{J}^{-1} \tilde{\nabla} \vec{x}^T \vec{a} = \underline{G} \vec{a} \tag{77}$$

with $\underline{J}^{-1}$ the inverse Jacobian matrix and $\tilde{\nabla}$ the nabla operator in triangular coordinates. The matrix $\underline{G}$:

$$\underline{G} = \frac{1}{2A_{\triangle}} \begin{pmatrix} x_{32} & x_{13} \\ y_{32} & y_{13} \end{pmatrix} \begin{pmatrix} \frac{\partial x_1}{\partial L_1} & \frac{\partial x_2}{\partial L_1} & \cdots & \frac{\partial x_9}{\partial L_1} \\ \frac{\partial x_1}{\partial L_2} & \frac{\partial x_2}{\partial L_2} & \cdots & \frac{\partial x_9}{\partial L_2} \end{pmatrix} \tag{78}$$

Next, the interpolation conditions at the three nodes for the three unknowns can be set (cf. Figure 4). Following the notation of [Ste15]:

$$\underbrace{\begin{pmatrix} \vec{x}^T(1,0) \\ G_1(1,0) \\ G_2(1,0) \\ \vec{x}^T(0,1) \\ G_1(0,1) \\ G_2(0,1) \\ \vec{x}^T(0,0) \\ G_1(0,0) \\ G_2(0,0) \end{pmatrix}}_{\underline{A}} \vec{a} = \underbrace{\begin{pmatrix} w_1 \\ \theta_{x_1} \\ \theta_{y_1} \\ w_2 \\ \theta_{x_2} \\ \theta_{y_2} \\ w_3 \\ \theta_{x_3} \\ \theta_{y_3} \end{pmatrix}}_{\vec{w}} \tag{79}$$

$$\tag{80}$$

where $\underline{G}_i$ is the i-th row of matrix $\underline{G}$. The unknown coefficients a_i can be computed by inverting the matrix $\underline{A}$: $\vec{a} = \underline{A}^{-1}\vec{w}$:

$$\underline{A}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & y_{12} & x_{21} & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & y_{23} & x_{32} \\ 1 & y_{31} & x_{13} & 0 & 0 & 0 & -1 & 0 & 0 \\ 2 & y_{21} & x_{12} & -2 & y_{21} & x_{12} & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & y_{32} & x_{23} & -2 & y_{32} & x_{23} \\ -2 & y_{13} & x_{31} & 0 & 0 & 0 & 2 & y_{13} & x_{31} \end{pmatrix} \tag{81}$$

where x_{ij} and y_{ij} denotes the differences of the node's coordinates $x_i - x_j$ and $y_i - y_j$. Now, the coefficients can be inserted into equation 76:

$$w = \vec{x}^T\vec{a} = \vec{x}^T\underline{A}^{-1}\vec{w} = \vec{N}^T\vec{w} \tag{82}$$

The vector $\vec{N}$ containing the shape functions N_i can then be calculated as follows:

$$\vec{N} = \left(\underline{A}^{-1}\right)^T \vec{x} \tag{83}$$

Since the shape functions following a pattern, due to the regular order in the matrix $\underline{A}^{-1}$, one can summarize the nine shape functions into three groups; one for every node:

$$N_i = \begin{cases} \chi_i - \chi_{i+3} + \chi_{k+3} + 2\left(\chi_{i+6} - \chi_{k+6}\right) & \text{for d.o.f. } w \\ -y_{ki}\left(\chi_{k+6} - \chi_{k+3}\right) + y_{ji}\chi_{i+6} & \text{for d.o.f. } \theta_x \\ x_{ki}\left(\chi_{k+6} - \chi_{k+3}\right) - x_{ji}\chi_{i+6} & \text{for d.o.f. } \theta_y \end{cases} \tag{84}$$

The variables χ_i denotes the i-th component of the vector $\vec{x}$, the indexes i, j, k under χ are cyclic permutations of $1, 2, 3$. x_{ij} and y_{ij} denote the coordinate differences $x_i - x_j$ and $y_i - y_j$. The index under N is incremented in such a way, that N_1, N_4, N_7 describes the d.o.f. w, N_2, N_5, N_8 describes the d.o.f. θ_x and N_3, N_6, N_9 describes the d.o.f. θ_y. Similar to the plane elements, one can check the correctness of the shape functions by evaluating them at the triangular coordinates of the three triangle's nodes. For example, shape function N_7 will evaluate to 1 for the coordinates $(L_1 = 0, L_2 = 0)$ (node 3) and will be zero for $(L_1 = 1, L_2 = 0)$ (node 1) and $(L_1 = 0, L_2 = 1)$ (node 2).

The displacement-strain relation 58 introduced for the plate element contains an operator living in the Cartesian space. It has to be converted into triangular coordinates. With equation 30 ($\nabla = \underline{J}^{-1}\tilde{\nabla}$) one can describe a second order derivative operator Δ:

$$\Delta = \nabla\nabla^T = \underline{J}^{-1}\tilde{\nabla}\left(\underline{J}^{-1}\tilde{\nabla}\right)^T = \underline{J}^{-1}\tilde{\nabla}\tilde{\nabla}^T\left(\underline{J}^{-1}\right)^T = \underline{J}^{-1}\tilde{\Delta}\left(\underline{J}^{-1}\right)^T \tag{85}$$

$$\Delta = \begin{pmatrix} \frac{\partial^2}{\partial x^2} & \frac{\partial^2}{\partial x \partial y} \\ \frac{\partial^2}{\partial y \partial x} & \frac{\partial^2}{\partial y^2} \end{pmatrix} \rightarrow \vec{\Delta} = \begin{pmatrix} \frac{\partial^2}{\partial x^2} \\ \frac{\partial^2}{\partial y^2} \\ 2\frac{\partial^2}{\partial x \partial y} \end{pmatrix}$$

$$\vec{\Delta} = \frac{1}{4A_\triangle^2}\begin{pmatrix} y_{32}^2 & y_{31}^2 & y_{23}y_{31} \\ x_{32}^2 & x_{31}^2 & x_{13}x_{32} \\ 2x_{32}y_{23} & 2x_{13}y_{31} & x_{32}y_{31} + x_{31}y_{32} \end{pmatrix}\begin{pmatrix} \frac{\partial^2}{\partial L_1^2} \\ \frac{\partial^2}{\partial L_2^2} \\ 2\frac{\partial^2}{\partial L_1 \partial L_2} \end{pmatrix}$$

$$\vec{\Delta} = \underline{Y}\vec{\tilde{\Delta}} \tag{86}$$

Next, equation 58 can be rewritten for triangular coordinates:

$$\vec{\epsilon} = -z\vec{\Delta}w = -z\underline{Y}\vec{\tilde{\Delta}}w = -z\vec{\kappa} \tag{87}$$

And additionally, with the help of equation 82, this yields a new version of equation 59:

$$\vec{\kappa} = \vec{\Delta}w = \underline{Y}\vec{\tilde{\Delta}}\vec{N}^T\vec{w} = \underline{Y}\,\underline{\tilde{B}}\vec{w} = \underline{B}\vec{w} \tag{88}$$

$$\underline{\tilde{B}} = \vec{\tilde{\Delta}}\vec{N}^T = \begin{pmatrix} \frac{\partial^2 N_1}{\partial L_1^2} & \frac{\partial^2 N_2}{\partial L_1^2} & \cdots & \frac{\partial^2 N_9}{\partial L_1^2} \\ \frac{\partial^2 N_1}{\partial L_2^2} & \frac{\partial^2 N_2}{\partial L_2^2} & \cdots & \frac{\partial^2 N_9}{\partial L_2^2} \\ 2\,\frac{\partial^2 N_1}{\partial L_1 \partial L_2} & 2\,\frac{\partial^2 N_2}{\partial L_1 \partial L_2} & \cdots & 2\,\frac{\partial^2 N_9}{\partial L_1 \partial L_2} \end{pmatrix} \tag{89}$$

With the help of equation 88, the first term (denoted as Π_1) of the plate element's functional 72 can be written out:

$$\begin{aligned} \Pi_1 &= \frac{1}{2}\int_A \vec{\kappa}^T \underline{D}_p \vec{\kappa}\; dA \\ &= \frac{1}{2}\vec{w}^T \int_A \underline{B}^T \underline{D}_p \underline{B}\; dA\vec{w} \\ &= \frac{1}{2}\vec{w}^T \underline{K}\vec{w} \end{aligned} \tag{90}$$

where $\underline{K}$ describes the stiffness matrix for the three node triangular plate element. The stiffness matrix must be integrated in triangular coordinates. This will be done by a Gaussian quadrature with the Gauss points located at: $(L_{1_1} = 1/6, L_{2_1} = 1/6), (L_{1_2} = 2/3, L_{2_2} = 1/6), (L_{1_3} = 1/6, L_{2_3} = 2/3)$ and weights $\omega_i = 1/6$ for all three points. For an exact integration one would accumulate four sampling points, but [Ste15] states that this leads to an element, that is too stiff; with only three samplings a more natural element results.

$$\underline{K} = 2A_{\triangle} \sum_{i=1}^{3} \omega_i \underline{B}^T(L_{1_i}, L_{2_i}) \underline{D}_p \underline{B}(L_{1_i}, L_{2_i}) \tag{91}$$

The plate's functional 72 has two more terms including the surface load p and edge loads V_n. These two can now be written as follows (see also [Ste15]):

$$\int_A p\ w\ dA = \vec{w}^T \vec{F}_p = \vec{w}^T p \int_A \vec{N}\ dA = 2\vec{w}^T A_{\triangle} p \int_0^1 \left(\int_0^{1-L_1} \vec{N}\ dL_2 \right)\ dL_1 \tag{92}$$

where $\vec{F}_p$ is a 1×9 vector containing forces and moments emerging from the surface load p. As an example, an edge load V_n is applied to edge S_{13}. This can be described as follows [Ste15]:

$$\int_{\Gamma_V} V_n\ w\ d\Gamma = \int_{\Gamma_V} V_n \vec{w}^T \vec{N}(L_2 = 0)\ d\Gamma \tag{93}$$

with $d\Gamma = S_{13} dL_1$. With V_n being constant all over the edge:

$$\int_{\Gamma_V} V_n \vec{w}^T \vec{N}(L_2 = 0)\ d\Gamma = \vec{w}^T S_{13} V_n \int_0^1 \vec{N}\ dL_1 = \vec{w}^T \vec{F}_v \tag{94}$$

The edge load applies forces and moments contained in $\vec{F}_v$ to the nodes forming that edge. The above equation can be applied to every other edge.

3.3.3 Quad-4 Plate Element

The element implemented in this work is the so-called *DKQ* element, introduced by Batoz et al. [BT82]. It is a four node, 12 degrees-of-freedom quadrilateral element for thin plates. It is based on a generalization of the *Discrete Kirchhoff Triangular* (DKT) element which is a three node, 9 d.o.f. triangular element. Like the triangular element of the previous section, the DKQ element's nodes have all three degrees of freedom: The displacement w and the rotations θ_x and θ_y around the element's local x- and y-axis. Figure 8 shows an example of such an element.

The formulation of the DKQ element by Batoz et al. [BT82] uses the discrete Kirchhoff technique. It is based on the discretization of the strain energy and neglects the transverse shear energy. This results in the following functional:

$$\Pi = \frac{1}{2} \int_A \vec{\kappa}^T \underline{D}_p \vec{\kappa}\ dA \tag{95}$$

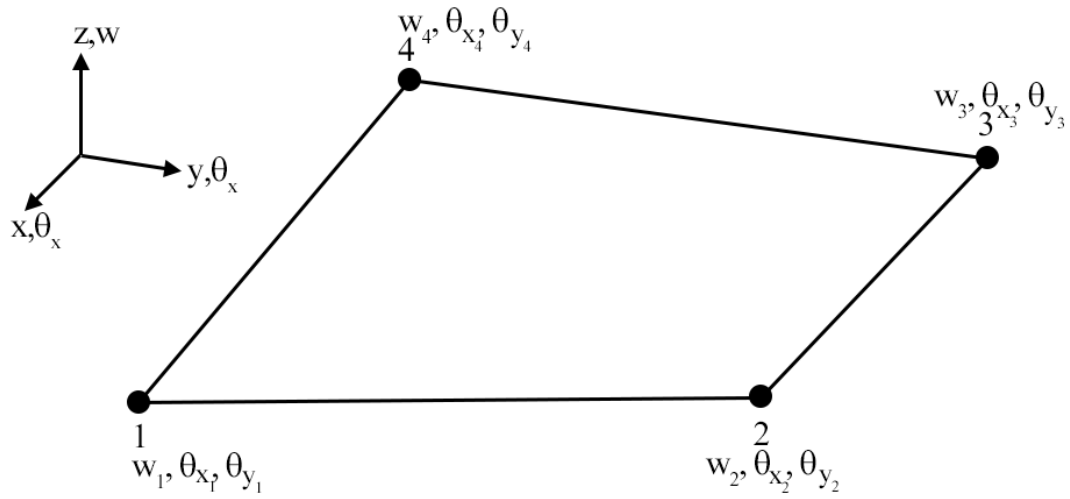

Figure 8: 12 node quadrilateral plate element DQK

where $\underline{D_p}$ is the material matrix as defined previously (equation 61) and $\vec{\kappa}$ denotes:

$$\vec{\kappa} = \begin{pmatrix} \frac{\partial \beta_x}{\partial x} \\ \frac{\partial \beta y}{\partial y} \\ \frac{\partial \beta_x}{\partial y} + \frac{\partial \beta_y}{\partial x} \end{pmatrix} \tag{96}$$

β_i is the rotation of the normal to the undeformed mid-surface in x-z-plane and y-z-plane, respectively. For Π only C^0 continuity is required [BT82]. Further, Batoz et al. states that β_x and $beta_y$ must be related to w in such a way, that the final element satisfies the following requirements:

- The nodal variables must be w, θ_x and θ_y with respect to x and y at the four element's nodes ($\theta_x = \partial w/\partial y, \theta_y = -\partial w/\partial x$)
- The Kirchhoff boundary conditions must be satisfied.

Two incomplete cubic polynomial expressions define β_x and β_y:

$$\beta_x = \sum_{i=1}^{8} N_i \beta_{x_i} \tag{97}$$

$$\beta_y = \sum_{i=1}^{8} N_i \beta_{y_i} \tag{98}$$

$N_i(\xi, \eta)$ are here the shape functions with isoparametric coordinates ξ and η. They are the same as of the eight node Serendipity element, described for example in [ZT00] and seen in Figure 9. The shape functions of this element are achieved by products of linear

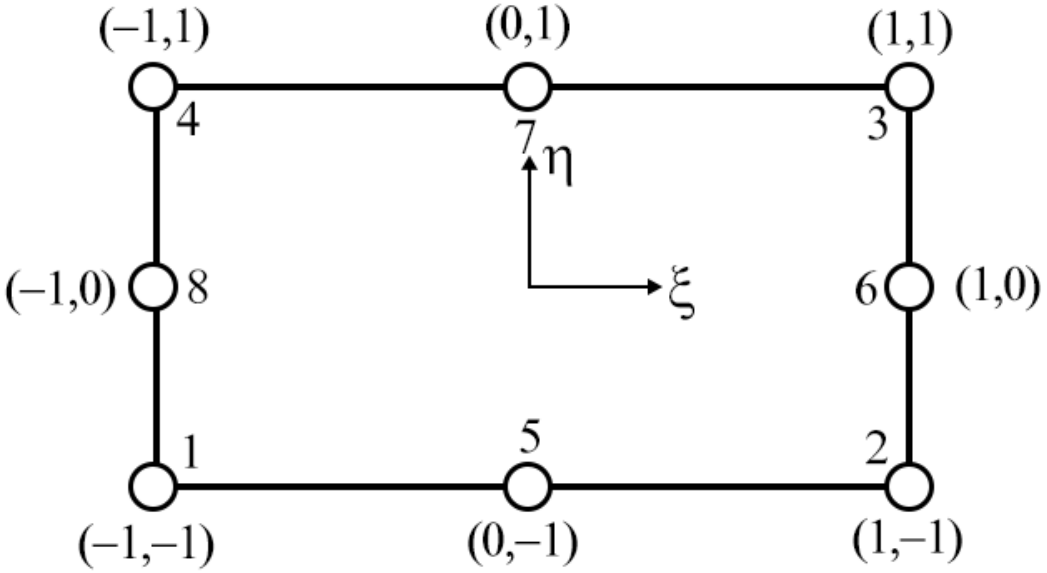

Figure 9: lange Unter-Überschrift

lagrangian polynomials of the form $\frac{1}{4}(\xi+1)(\eta+1)$. For the eight node element the following shape functions result:

$$
\begin{aligned}
N_1(\xi,\eta) &= \frac{1}{4}(1-\xi)(1-\eta)(-\xi-\eta-1)\\
N_2(\xi,\eta) &= \frac{1}{4}(1+\xi)(1-\eta)(\xi-\eta-1)\\
N_3(\xi,\eta) &= \frac{1}{4}(1+\xi)(1+\eta)(\xi+\eta-1)\\
N_4(\xi,\eta) &= \frac{1}{4}(1-\xi)(1+\eta)(-\xi+\eta-1)\\
N_5(\xi,\eta) &= \frac{1}{2}(1-\xi^2)(1-\eta)\\
N_6(\xi,\eta) &= \frac{1}{2}(1+\xi)(1-\eta^2)\\
N_7(\xi,\eta) &= \frac{1}{2}(1-\xi^2)(1+\eta)\\
N_8(\xi,\eta) &= \frac{1}{2}(1-\xi)(1-\eta^2)
\end{aligned}
$$

β_{x_i} and β_{y_i} are transitory nodal variables at the four nodes and mid-sides of the element. Next, Batoz et al. described the Kirchhoff assumptions at the corner nodes (cf. Figure 9 for the following):

$$
\begin{pmatrix} \beta_{x_i} + \partial w/\partial x_i \\ \beta_{y_i} + \partial w/\partial y_i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad i = 1,2,3,4 \tag{99}
$$

and at the mid-nodes:

$$\beta_{s_k} + \partial w/\partial s_k = 0, \qquad k = 5, 6, 7, 8 \tag{100}$$

where s denotes the coordinate along the element boundary and $\partial w/\partial s_k$ is the derivative of the displacement w with respect to the mid-node k:

$$\frac{\partial w}{\partial s_k} = -\frac{3}{2l_{ij}}(w_i - w_j) - \frac{1}{4}\left(\frac{\partial w}{\partial s_i} + \frac{\partial w}{\partial s_j}\right) \tag{101}$$

with $k = 5, 6, 7, 8$ being the mid-node of side $ij = 12, 23, 34, 41$ and l_{ij} denotes the length of side ij. β_n varies linearly along the sides:

$$\beta_{n_k} = \frac{1}{2}\left(\beta_{n_i} + \beta_{n_j}\right) = -\frac{1}{2}\left(\frac{\partial w}{\partial n_i} + \frac{\partial w}{\partial n_j}\right) \tag{102}$$

with k same as before. β_x and β_y can be rewritten as follows:

$$\beta_x = H^x(\vec{\xi}, \eta)^T \vec{w} \tag{103}$$

$$\beta_y = H^y(\vec{\xi}, \eta)^T \vec{w} \tag{104}$$

$$\vec{w}^T = \begin{pmatrix} w_1 & \theta_{x_1} & \theta_{y_1} & w_2 & \theta_{x_2} & \theta_{y_2} & w_3 & \theta_{x_3} & \theta_{y_3} \end{pmatrix}$$

with

$$\vec{H^x}^T = \begin{pmatrix} H^x_1 & \dots & H^x_{12} \end{pmatrix}$$

$$H^x_{[1,4,7,10]} = \frac{3}{2}\left(a_{[5,6,7,8]}N_{[5,6,7,8]} - a_{[8,5,6,7]}N_{[8,5,6,7]}\right)$$

$$H^x_{[2,5,8,11]} = b_{[5,6,7,8]}N_{[5,6,7,8]} + b_{[8,5,6,7]}N_{[8,5,6,7]}$$

$$H^x_{[3,6,9,12]} = N_{[1,2,3,4]} - c_{[5,6,7,8]}N_{[5,6,7,8]} - c_{[8,5,6,7]}N_{[8,5,6,7]}$$

$$\vec{H^y}^T = \begin{pmatrix} H^y_1 & \dots & H^y_{12} \end{pmatrix}$$

$$H^y_{[1,4,7,10]} = \frac{3}{2}\left(d_{[5,6,7,8]}N_{[5,6,7,8]} - d_{[8,5,6,7]}N_{[8,5,6,7]}\right)$$

$$H^y_{[2,5,8,11]} = -N_{[1,2,3,4]} + e_{[5,6,7,8]}N_{[5,6,7,8]} + e_{[8,5,6,7]}N_{[8,5,6,7]}$$

$$H^y_{[3,6,9,12]} = -b_{[5,6,7,8]}N_{[5,6,7,8]} - b_{[8,5,6,7]}N_{[8,5,6,7]}$$

The function notation $H^x_{[i,j,k,l]}$ groups four functions together. The first function of the group gets the first index of the squared brackets, the second function the second index,

and so on. The coefficients a, b, c, d and e are as follows:

$$
\begin{aligned}
a_k &= -\frac{x_{ij}}{l_{ij}^2} \\
b_k &= \frac{3}{4}\frac{x_{ij}y_{ij}}{l_{ij}^2} \\
c_k &= \frac{\frac{x_{ij}^2}{4} - \frac{y_{ij}^2}{2}}{l_{ij}^2} \\
d_k &= -\frac{y_{ij}}{l_{ij}^2} \\
e_k &= \frac{\frac{y_{ij}^2}{4} - \frac{x_{ij}^2}{2}}{l_{ij}^2}
\end{aligned}
$$

where $k = 5, 6, 7, 8$ for the sides $ij = 12, 23, 34, 41$, $x_{ij} = x_i - x_j$, $y_{ij} = y_i - y_j$ and $l_{ij}^2 = x_{ij}^2 + y_{ij}^2$. For more details about the derivation of these coefficients and functions H^x and H^y see Batoz et al. [BT82].

Next, the Jacobian matrix $\underline{J}$ can be assembled, that is:

$$
\underline{J} = \frac{1}{4}\begin{pmatrix} (x_{12}+x_{34})\eta - x_{12} + x_{34} & (y_{12}+y_{34})\eta - x_{12} + y_{34} \\ (x_{12}+x_{34})\xi - x_{13} - x_{24} & (y_{12}+y_{34})\xi - y_{13} + y_{24} \end{pmatrix} = \begin{pmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{pmatrix} \tag{105}
$$

With its determinant and inverse:

$$
|\underline{J}| = J_{11}J_{22} - J_{12}J_{21} \tag{106}
$$

$$
\underline{J}^{-1} = \frac{1}{|\underline{J}|}\begin{pmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{pmatrix} = \begin{pmatrix} j_{11} & j_{12} \\ j_{21} & j_{22} \end{pmatrix} \tag{107}
$$

The strain-displacement matrix can now be obtained:

$$
\underline{B} = \begin{pmatrix} \vec{H_x^x} \\ \vec{H_y^y} \\ \vec{H_y^x} + \vec{H_x^y} \end{pmatrix} = \begin{pmatrix} j_{11} & j_{12} & 0 & 0 \\ 0 & 0 & j_{21} & j_{22} \\ j_{21} & j_{22} & j_{11} & j_{12} \end{pmatrix} \begin{pmatrix} \vec{H_\xi^x} \\ \vec{H_\eta^x} \\ \vec{H_\xi^y} \\ \vec{H_\eta^y} \end{pmatrix} \tag{108}
$$

The expressions $vec H_\xi^x, \vec{H_\eta^x}, \vec{H_\xi^y}$ and $\vec{H_\eta^y}$ are vectors containing the derivatives of the corresponding components of the vectors $\vec{H^x}$ and $\vec{H^y}$ with respect to ξ and η, respectively. And the matrix $\underline{B}$ can then be inserted into the displacement-strain relation, like equation 88:

$$
\vec{\kappa} = \underline{B}\vec{w} \tag{109}
$$

Next, $\vec{\kappa} = \underline{B}\vec{w}$ can be used in the functional to get the first term like equation 90:

$$\begin{aligned}\Pi_1 &= \frac{1}{2}\int_A \vec{\kappa}^T \underline{D}_p \vec{\kappa}\ dA \\ &= \frac{1}{2}\vec{w}^T \int_A \underline{B}^T \underline{D}_p \underline{B}\ dA \vec{w} \\ &= \frac{1}{2}\vec{w}^T \underline{K} \vec{w}\end{aligned}$$

with the stiffness matrix of the DKQ element $\underline{K}$:

$$\begin{aligned}\underline{K} &= \int_A \underline{B}^T \underline{D}_p \underline{B}\ dA \\ &= \int_{-1}^{1}\int_{-1}^{1} \underline{B}^T \underline{D}_p \underline{B}\ \ |\underline{J}|\ \ d\xi d\eta \end{aligned} \tag{110}$$

The stiffness matrix can be numerically integration with a 2×2 Gaussian integration scheme. Batoz et al. states that four sampling points are enough, although a 3×3 point scheme would be necessary for exact integration on a rectangular element [BT82]. Those four sampling points are located at $\xi_i = \pm\frac{\sqrt{3}}{3}$ and $\eta_i = \pm\frac{\sqrt{3}}{3}$ with weight factor $\omega_i = 1$ equivalent to all four. The equation for the stiffness matrix can then be written in discretized form as follows:

$$\underline{K} = \sum_{i=1}^{2}\sum_{j=1}^{2} \omega_i \omega_j \underline{B}(\xi_i, \eta_j)^T \underline{D}_p \underline{B}(\xi_i, \eta_j)\, |\underline{J}(\xi_i, \eta_j)| \tag{111}$$

When all nodal values $\vec{w}$ are known, the moments $\vec{M}$ at point (x, y) in the element can be calculated:

$$\vec{M}(x, y) = \underline{D}_p \underline{B}(x, y)\vec{w} \tag{112}$$

with

$$\vec{M} = \begin{pmatrix} M_x \\ M_y \\ M_x y \end{pmatrix} \tag{113}$$

3.4 Coordinate Transformation

The nodes and elements in the mesh are defined in a global three dimensional coordinate system. The elements need to be transformed into a two dimensional local coordinate system in order to be able to calculate their local stiffness matrices. This local stiffness matrix must then be transformed back into the global system before adding it to the global stiffness matrix. This section describes the building of the transformation matrix, that will be used in the following section for the addressed transformation steps. First the transformation of an arbitrary triangle defined in 3D space is described.

Given a triangle with vertices $\vec{A} = (a_x, a_y, a_z)^T, \vec{B} = (b_x, b_y, b_z)^T$ and $\vec{C} = (c_x, c_y, c_z)^T$ ordered in counter-clockwise direction, as shown in Figure 10. Let $\vec{u}$ be the vector from

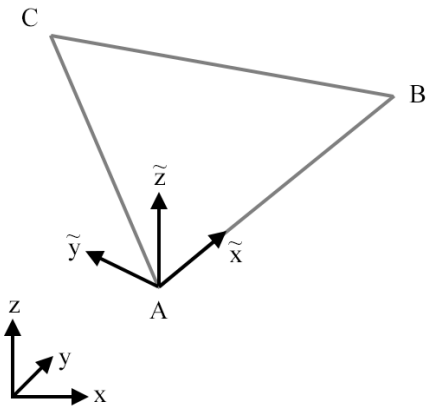

Figure 10: Arbitrary triangle with nodes A, B and C, defined in global xyz coordinate system. After transformation a new, local, $\tilde{x}\tilde{y}\tilde{z}$ coordinate system with node A in its origin, is created.

node $\vec{A}$ to $\vec{B}$ and $\vec{v}$ be the vector from node $\vec{A}$ to $\vec{C}$:

$$\vec{u} = \vec{B} - \vec{A} = \begin{pmatrix} b_x - a_x & b_y - a_y & b_z - a_z \end{pmatrix}^T$$
$$\vec{v} = \vec{C} - \vec{A} = \begin{pmatrix} c_x - a_x & c_y - a_y & c_z - a_z \end{pmatrix}^T$$

First local unit vector:

$$\vec{\tilde{x}} = \frac{1}{|\vec{u}|}\vec{u}$$

Second local unit vector:

$$\vec{\tilde{z}} = \vec{u} \times \vec{v}$$
$$\vec{\tilde{z}} \leftarrow \frac{1}{\left|\vec{\tilde{z}}\right|}\vec{\tilde{z}}$$

Third local unit vector:

$$\vec{\tilde{y}} = \vec{\tilde{z}} \times \vec{\tilde{x}}$$

Define transformation matrix $\underline{T}$ as follows:

$$\underline{T} = \begin{pmatrix} \vec{\tilde{x}}^T \\ \vec{\tilde{y}}^T \\ \vec{\tilde{z}}^T \end{pmatrix} = \begin{pmatrix} \tilde{x}_x & \tilde{x}_y & \tilde{x}_z \\ \tilde{y}_x & \tilde{y}_y & \tilde{y}_z \\ \tilde{z}_x & \tilde{z}_y & \tilde{z}_z \end{pmatrix} \tag{114}$$

Assembly of element's stiffness matrix needs partial derivatives. In order to get these derivatives with less computational effort, every triangle can be translated in such a way, that node $\vec{A}$ lies in the global origin before transforming it to local coordinates. Node $\vec{A}$

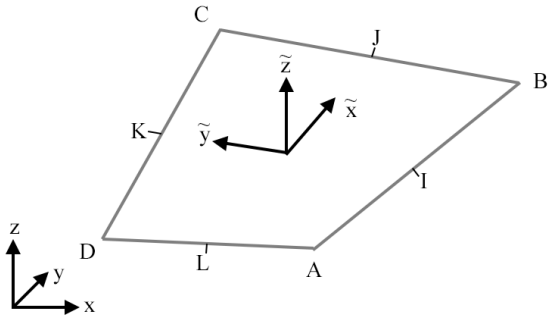

Figure 11: Quadrilateral with nodes A, B, C and D, defined in global xyz coordinate system. After transformation a new, local, $\tilde{x}\tilde{y}\tilde{z}$ coordinate system is created.

stays at (0, 0, 0) coordinates which then simplifies getting the derivatives (see section **??**). It follows:

$$\vec{\tilde{A}} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$$
$$\vec{\tilde{B}} = \begin{pmatrix} \tilde{b}_x & 0 & 0 \end{pmatrix}^T$$
$$\vec{\tilde{C}} = \begin{pmatrix} \tilde{c}_x & \tilde{c}_y & 0 \end{pmatrix}^T$$

Node $\vec{A}$ will not be changed by the transformation with $\underline{T}$, $\vec{B}$ will be projected onto the local x-axis that is defined to be the normalized vector between $\vec{A}$ and $\vec{B}$. Node $\vec{C}$ will be projected onto the local xy-plane. One can see that the z-component vanishes by transforming into local space, thus generating the wished two dimensional local space.

The other element described in this work is the quadrilateral. Let an arbitrary quadrilateral be given with vertices $\vec{A} = (a_x,\ a_y,\ a_z)^T, \vec{B} = (b_x,\ b_y,\ b_z)^T, \vec{C} = (c_x,\ c_y,\ c_z)^T, \vec{D} = (d_x,\ d_y,\ d_z)^T$ ordered in counter-clockwise direction, cf. Figure 11. Next, let $\vec{I}$ be the midpoint of edge $\overline{AB}$:

$$\vec{I} = \vec{A} + \frac{1}{2}\left(\vec{B} - \vec{A}\right)$$

Analogously let $\vec{J}, \vec{K}$ and $\vec{L}$ be the midpoints of the edges $\overline{BC}, \overline{CD}$ and $\overline{DA}$:

$$\vec{J} = \vec{B} + \frac{1}{2}\left(\vec{C} - \vec{B}\right)$$
$$\vec{K} = \vec{C} + \frac{1}{2}\left(\vec{D} - \vec{C}\right)$$
$$\vec{L} = \vec{D} + \frac{1}{2}\left(\vec{A} - \vec{D}\right)$$

Let then $\vec{u}$ be the vector from node $\vec{L}$ to $\vec{J}$ and $\vec{v}$ be the vector from node $\vec{I}$ to $\vec{K}$:

$$\vec{u} = \vec{J} - \vec{L} = \begin{pmatrix} j_x - l_x & j_y - l_y & j_z - l_z \end{pmatrix}^T$$
$$\vec{v} = \vec{K} - \vec{I} = \begin{pmatrix} k_x - i_x & k_y - i_y & k_z - i_z \end{pmatrix}^T$$

First local unit vector:

$$\vec{\tilde{x}} = \frac{1}{|\vec{u}|}\vec{u}$$

Second local unit vector:

$$\vec{\tilde{z}} = \vec{u} \times \vec{v}$$
$$\vec{\tilde{z}} \leftarrow \frac{1}{\left|\vec{\tilde{z}}\right|}\vec{\tilde{z}}$$

Third local unit vector:

$$\vec{\tilde{y}} = \vec{\tilde{z}} \times \vec{\tilde{x}}$$

Define transformation matrix T as follows:

$$\underline{T} = \begin{pmatrix} \vec{\tilde{x}}^T \\ \vec{\tilde{y}}^T \\ \vec{\tilde{z}}^T \end{pmatrix} = \begin{pmatrix} \vec{\tilde{x}}_x & \vec{\tilde{x}}_y & \vec{\tilde{x}}_z \\ \vec{\tilde{y}}_x & \vec{\tilde{y}}_y & \vec{\tilde{y}}_z \\ \vec{\tilde{z}}_x & \vec{\tilde{z}}_y & \vec{\tilde{z}}_z \end{pmatrix} \tag{115}$$

Remark: In order to transform a quadrilateral element from 3D to a local two dimensional space, the nodes of the original element must all be situated on a common plane. Otherwise such a transformation cannot be performed. Such shaped quadrilaterals cannot be used as shell elements.

3.5 Shell Element

Shell elements combine the potential of both, plane and plate elements. Every time a thin walled structure like a car body, dome structure or container with multiaxial pressures is to be simulated, shell elements provide a good solution. In this work only the so-called flat shell elements are described and used in the implementation. Details about curved shell elements, shells of revolution and general shells can be found in Cook et al. [CMPW02].

Flat shell elements have a state of bending and membrane stress that can be described by superposition of the plane and plate element [Kle13]. Figure 12 shows the superposition of plane and plate elements to a flat shell element. The degrees of freedom of plane and plate at every node are combined at the node of the shell element. Obviously the plane and plate element must be of the same finite element's type, for example three node triangular or eight node quadrilateral.

The plane has displacements u and v with dedicated forces F_x and F_y. The plate has the deformation w with assigned normal force F_z and the two twists θ_x and θ_y with assigned moments M_x and M_y. Through the linking of the elements the shell element node has now five natural degrees of freedom. An additional twist around the local z-axis θ_z will be introduced [Ste15], increasing the number to six degrees of freedom per node. In vector notation the resulting displacement vector $\vec{u}_i$ of a shell element's node

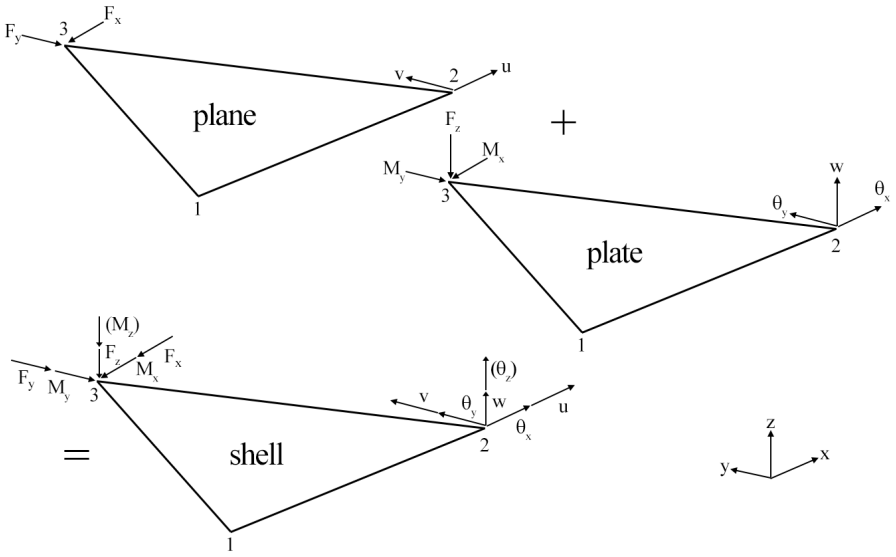

Figure 12: Creation of a triangular flat shell element by superimposing a triangular plane and a triangular plate element. The elements are described in the same local coordinate system with their degrees of freedom shown exemplary at one node per element.

i is:

$$\vec{u} = \underbrace{\begin{pmatrix} u \\ v \\ w \\ \theta_x \\ \theta_y \\ \theta_z \end{pmatrix}}_{\text{shell}} = \underbrace{\begin{pmatrix} u \\ v \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{\text{plane}} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ w \\ \theta_x \\ \theta_y \\ 0 \end{pmatrix}}_{\text{plate}} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \theta_z \end{pmatrix} \tag{116}$$

(117)

For the two different finite elements in this work - the three node triangular element and the four node quadrilateral element - the stiffness matrix for the shell element is described by a block matrix of either 3×3 submatrices for the triangular case or 4×4 submatrices for the quadrilateral. The following equation shows the latter case as example:

$$\underline{K}\vec{u} = \vec{F}$$

$$\begin{pmatrix} \underline{K}_{11} & \underline{K}_{12} & \underline{K}_{13} & \underline{K}_{14} \\ \underline{K}_{21} & \underline{K}_{22} & \underline{K}_{23} & \underline{K}_{24} \\ \underline{K}_{31} & \underline{K}_{32} & \underline{K}_{33} & \underline{K}_{34} \\ \underline{K}_{41} & \underline{K}_{42} & \underline{K}_{43} & \underline{K}_{44} \end{pmatrix} \begin{pmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vec{u}_3 \\ \vec{u}_4 \end{pmatrix} = \begin{pmatrix} \vec{F}_1 \\ \vec{F}_2 \\ \vec{F}_3 \\ \vec{F}_4 \end{pmatrix} \tag{118}$$

where the vectors $\vec{u}_i$ are the same as in equation 116. The single submatrices $\underline{K}_{ij}$ of $\underline{K}$ were created by the superposition of the stiffness matrices of the plane and the plate:

$$\underline{K}_{ij} = \left(\hat{\underline{K}}_{ij}\right)_m + \left(\hat{\underline{K}}_{ij}\right)_p \tag{119}$$

The submatrix $\underline{K}_{ij}$ has the following structure:

$$\begin{array}{c} \begin{array}{cccccc} \underline{u} & \underline{v} & \underline{w} & \underline{\theta_x} & \underline{\theta_y} & \underline{\theta_z} \end{array} \\ \begin{pmatrix} \circ & \circ & 0 & 0 & 0 & 0 \\ \circ & \circ & 0 & 0 & 0 & 0 \\ 0 & 0 & \star & \star & \star & 0 \\ 0 & 0 & \star & \star & \star & 0 \\ 0 & 0 & \star & \star & \star & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ \mid u \\ \mid v \\ \mid w \\ \mid \theta_x \\ \mid \theta_y \\ \mid \theta_z \end{array} = \begin{array}{c} \begin{array}{cccccc} \underline{u} & \underline{v} & \underline{w} & \underline{\theta_x} & \underline{\theta_y} & \underline{\theta_z} \end{array} \\ \begin{pmatrix} \circ & \circ & 0 & 0 & 0 & 0 \\ \circ & \circ & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ \mid u \\ \mid v \\ \mid w \\ \mid \theta_x \\ \mid \theta_y \\ \mid \theta_z \end{array} + \begin{array}{c} \begin{array}{cccccc} \underline{u} & \underline{v} & \underline{w} & \underline{\theta_x} & \underline{\theta_y} & \underline{\theta_z} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \star & \star & \star & 0 \\ 0 & 0 & \star & \star & \star & 0 \\ 0 & 0 & \star & \star & \star & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ \mid u \\ \mid v \\ \mid w \\ \mid \theta_x \\ \mid \theta_y \\ \mid \theta_z \end{array} \tag{120}$$

$\left(\hat{\underline{K}}_{ij}\right)_m$ describes the submatrix of the stiffness matrix of the plane element for node i and j and is marked with the $\circ$-symbol, $\left(\hat{\underline{K}}_{ij}\right)_p$ describes the corresponding matrix part of the plate element's stiffness matrix and is symbolized with a $\star$. The degree of freedom θ_z does not exists in both, the plane and the plate element, and were introduced with the shell element. Therefore $(\underline{K}_{ij})_{66}$ is zero. The sixth degree of freedom is necessary because the missing stiffness regarding a rotation around the axis normal to the element could produce singularities in the overall stiffness matrix. This happens for example, if all neighboring elements of a node lie in the same plane, i.e. they are coplanar. A singularity can lead to a non-solvable system, so this case needs be excluded [Ste15]. One way is to introduce this sixth degree of freedom and give it a value that is so small that it does not influence the displacements and stresses too much. The number of this values varies: Werkle suggests a value of 1/10000 of the smallest diagonal entry of $\underline{K}_{ij}$ [Wer95], where Ibrahimbegovic et al. suggests 1/1000 of the smallest diagonal entriy of $\underline{K}_{ij}$. The value must be small, but big enough to prevent the singularities. Since this value is an approximation, one has to modify it, if the solution is not as expected or one cannot get a solution at all due to the addressed singularities.

The stiffness matrix for the shell element was constructed in a local coordinate system as described in section **??**, **??** and **??**. The overall stiffness matrix containing information about all elements need to be described in a global coordinate system. Before the element stiffness matrix is added to the global stiffness matrix, it has to be transformed from local to global. This can be achieved by transforming the single blocks of $\underline{K}$ from equation 118 with the relation:

$$\check{\underline{K}}_{ij}\vec{\check{u}}_j = \vec{\check{F}}_i \tag{121}$$

where " $\check{}$ " denotes that the matrix and vector are represented in local coordinates. With the help of the transformation matrix $\underline{\tilde{T}}$, the globally described displacement vector $\vec{u}_j$

and load vector $\vec{F}_i$ can be represented in local coordinates:

$$\check{\vec{u}}_j = \tilde{\underline{T}}\vec{u}_j \tag{122}$$

$$\check{\vec{F}}_i = \tilde{\underline{T}}\vec{F}_i \tag{123}$$

Since the load vector is to be defined in a global coordinate system and the resulting displacements are to be defined globally, too, equation 121 will be multiplied by $\tilde{\underline{T}}^T$ from left:

$$\tilde{\underline{T}}^T \check{\underline{K}}_{ij} \tilde{\underline{T}} \vec{u}_j = \tilde{\underline{T}}^T \tilde{\underline{T}} \vec{F}_i = \vec{F}_i \tag{124}$$

Hence, the two vectors will be represented in global coordinates and only the local element stiffness matrix need to be transformed. The addressed 6×6 transformation matrix $\tilde{\underline{T}}$ is made up of the 3×3 transformation matrix $\underline{T}$ from section **??**:

$$\tilde{\underline{T}} = \begin{pmatrix} \underline{T} & 0 \\ 0 & \underline{T} \end{pmatrix} \tag{125}$$

In order to get the local element stiffness matrix $\underline{K}$ transformed to the global coordinate system, one has to transform the single submatrices $\underline{K}_{ij}$ as follows:

$$\underline{K}_{ij} = \tilde{\underline{T}}^T \check{\underline{K}}_{ij} \tilde{\underline{T}} \tag{126}$$

for $1 \leq i, j \leq 3$ in the case of the triangular element and $1 \leq i, j, \leq 4$ for the quadrilateral element, respectively.

4 FEM Code Implementation

contains development of the program code with focus on the assembly of the system and its solving, the process of parallelization

4.1 Introduction to libMesh

The libMesh finite element library was stared as part of the Ph.D. work of Benjamin Kirk [Kir07]. It is a tool for numerical simulation of partial differential equations on serial and parallel platforms and uses the finite element method. Major goals are to provide data structures and algorithms for applications that need implicit numerical methods, parallel computing, adaptive mesh refinement techniques, or, a combination of them. Further, it simplifies many programming details for the user, such as: Reading the mesh from file, initialize data structures, solving the dicretized system, and, writing out the results [KPS13].

LibMesh allows discretization of one, two and three dimensional problems using several geometric element types, including: Edges, quadrilaterals, triangles, tetrahedra, hexahedra, pyramids, prisms and some infinite elements of quadrilaterals or hexahedra. Finite elements include traditional first and second order Lagrange, as well as arbitrary order hierarchical bases, and Nédélec elements of first type.

Mesh partitioning is available in libMesh through interfaces to several external packages, but also some internal partitioning algorithms are provided: Linear and centroid partitioner as examples of internal algorithms, Metis and ParMetis [KK98] as examples for external partitioner. In addition to these two, libMesh includes interfaces to solver libraries such as PETSc [pet] and LASPack [las]. Thus, libMesh provides several linear equation solvers such as GMRES, CG, Bi-CGSTAB, QMR, and preconditioners like Jacobi, incomplete LU factorization and incomplete Cholesky factorization. The choice of an appropriate solver and preconditioner is made by the user at runtime.

A wide variety of mesh formats are supported by libMesh to facilitate use of complex geometries. The following is an incomplete list of supported input and output formats: Nemesis, TetGen, I-deas Universal UNV, AVS's ASCII UCD, Visualization Toolkit VTK, libMesh formats XDR/XDA, ExodusII, GMSH, LANL's General Mesh Viewer GMV, GnuPlot (only output), Matlab (only input) [KPS13].

An example program using the libMesh library would look like listing 1.

Listing 1: Example libMesh program

```
1 #include "libmesh/libmesh.h"
2 #include "libmesh/additional_libmesh_components"
3
4 using namespace libMesh;
5
6 void assemble_something(EquationSystems& es, const std::string& system_name);
7
8 int main (int argc, char** argv)
9 {
```

```
10      LibMeshInit (int argc, char** argv);
11
12      Mesh mesh( init.comm() );
13
14      // mesh generation via MeshTools::Generation::build_... or mesh import from file via
            mesh.read(std::string filename)
15
16      EquationSystems es(mesh);
17
18      LinearImplicitSystem& system = es.add_system<LinearImplicitSystem> ("example_system");
19
20      system.add_variable ("a", FIRST);
21      system.add_variable ("b", SECOND, LAGRANGE);
22
23      system.attach_assemble_function (assemble_something);
24
25      es.init();
26
27      system.solve();
28
29      VTKIO (mesh).write_equation_systems ("out.pvtu", es);
30
31      return 0;
32  }
```

In fact, this is the base construction of nearly every libMesh program. It starts with including libMesh components that are needed by the program, e.g. *mesh.h, equation_systems.h, fe.h.* Then, the library needs to be initialized (line 10). This is necessary because it may depend on a number of other external libraries like MPI and PETSc that require initialization before use. On the other hand, if the **LibMeshInit** object goes out of scope, the other libraries are finalized automatically by libMesh. Next, a mesh is created (lines 12-14) on the default MPI communicator (even if the program is executed single-threaded). The mesh can either be read from file or created by internal mesh generation tools. In line 16 an equation systems object is created. It can contain multiple different systems. Here, only one linear implicit system is added to the object (line 16). Each system can contain multiple variables of different approximation orders (see lines 20/21). Many systems require a user-defined function that will assemble the (linear) system (lines 6 and 23). Now, the data structures for equation system must be initialized which is done in line 25. The solving of the systems is done in line 27 of the code. This one line of code calls the assemble function defined earlier and invokes the default numerical solver. If the external library PETSc is installed, the solver can be controlled from the command line by the user. After solving the system, the solution can be written to file; here, for example, the results are written to a VTK-formatted plot file (line 29).

4.2 Implementation Details

details about the implementation with the libmesh FEM framework
- short overview: stand-alone-version: gets input parameters from user and mesh file and produces results stored in output file

4.2.1 Initialization

The program expects a few parameters set by the user through the command line at start. The ordering of these parameters are not relevant; some are optional. Here is a complete list of all parameters that can be set in the command line:

- **-nu**: The Poisson's ratio ν is required by the material matrices. A value in the range $0.0 < \nu \leq 0.5$ is recommended for most scenarios.

- **-e**: The elastic modulus or Young's modulus E is also required by the material matrices. Here, a value $E \gg 0$ is recommended.

- **-t**: The thickness t of the mesh. It is used at both, the material matrices and the strain-displacement matrices and thus a required parameter to be set by the user.

- **-d**: If set to "'1"' additional messages regarding transformation matrix entries, strain-displacement matrices, force load vectors and other internal mathematic structures are put out on the console. This parameter is optional, as it only gives the user more information in case of finding error. Since it slows down the calculation, it should only be set if needed. To turn off the messages, simply ignore the parameter or set it to 0.

- **-mesh**: The file name of the mesh file to be imported. A required parameter, because no default mesh is coded into the program to be used. The relative path to the file (+ extension) must be specified. Allowed file formats are: libMesh format *xda* (ASCII) and *xdr* (binary) as well as GMSH format *msh*. For more details, see section 4.2.2.

- **-out**: The relative path and filename (*without* extension) for the output of the resulting mesh. This parameter is optional. If not set, no output file will be created. The path to the filename must exist, otherwise no file can be created.

If the external library PETSc is installed and libMesh is configured to be able to use it, the user can set additional optional parameters [BAA+15]. In fact, if one uses PETSc, it will look though all command line arguments by itself to find those it can process. The following list is therefore limited to parameters that directly coincide with the need of this program. For more PETSc command line argument see [BAA+15].

- **-ksp_type**: Specifies the Krylov subspace method. Options are: `richardson`, `chebyshev`, `cg`, `gmres`, `tcqmr`, `bcgs`, `cgs`, `tfqmr`, `cr`, `lsqr`, `bicg`, `preonly`.

- **-pc_type**: To employ a particular preconditioning method used with the Krylov space method, the user can select one using this command line argument. Options are: `none`, `jacobi`, `bjacobi`, `sor`, `eisenstat`, `icc`, `ilu`, `asm`, `gasm`, `gamg`, `bddc`, `ksp`, `composite`, `lu`, `cholesky`, `shell`.

4.2.2 Mesh file import

The mesh geometry needs to be defined in a mesh file. LibMesh can import meshes from many different formats, including its own libMesh formats XDA and XDA, the first one stored in readable ASCII format, the latter one stored as binary code. Another one is the GMSH format *msh*. There are other formats libMesh can import, but only the three mentioned formats are currently supported by the thesis' program. A mesh file must provide the following information in order to be usable be the implementation:

- A list of vertices. Every vertex must be specified with its xyz-coordinates defined in the global coordinate system.
- A list of elements the mesh consists out of. The elements are normally defined by their type, for example three node triangle or four node quadrilateral, and a list of vertex identifiers representing the nodes of the element.
- A list of boundary conditions. The program provides two different types of boundary conditions. The type is to be specified in form of identifiers on element's nodes or edges. In the latter case the boundary condition is used on both nodes defining the edge.

Listing 2 shows a short example of a mesh defined in the xda-format. It represents the unit square with its center at the global origin, composed of two three node triangles. It has different boundary conditions on the bottom and top edge.

Listing 2: Example xda mesh file

```
1  libMesh-0.7.0+
2  2       # number of elements
3  4       # number of nodes
4  .       # boundary condition specification file
5  n/a     # subdomain id specification file
6  n/a     # processor id specification file
7  n/a     # p-level specification file
8  2       # n_elem at level 0, [ type (n0 ... nN-1) ]
9  3 0 1 2          # 3 -> triangle with 3 nodes, 0 1 2 -> vertices 0, 1 and 2
10 3 1 3 2          #                             1 3 2 -> vertices 1, 3 and 2
11 -1.0 -1.0  0.0   # x y z coordinates of vertex 0
12  1.0 -1.0  0.0   #                       vertex 1
13 -1.0  1.0  0.0   #                       vertex 2
14  1.0  1.0  0.0   #                       vertex 3
15 2                # number of boundary conditions
16 0 0 1            # 0 -> element 0, 0 -> edge 0 (between vertex 0 and 1), bc-type 1
17 1 1 0            # 1 -> element 1, 1 -> edge 1 (between vertex 3 and 2), bc-type 0
```

The program features two different types of boundary conditions whose identifier must be set in the mesh file:

- Clamped boundary has the type "‘**0**"’.
- Simply supported boundary has type "‘**1**"’.

Because the stand-alone version of the program has no coupled fluid solver which provides it with pressures/forces at the nodes, these values must be imported via file, too. The structure of such a file is fairly simple: Listing 3 shows such an example corresponding to the example mesh of listing 2. The first line defines the number n of nodes/vertices the mesh has (in this case $n = 4$). The second line holds a floating point number representing a global factor that is multiplied by every force component defined below. A value of 1.0 has no effect on the load values. Lines three to $n + 2$ are the xyz-components of the single forces put on the corresponding mesh nodes. The ordering is the same as the vertices in the mesh file. The xyz-coordinates must also be represented in the global coordinate system. In the example a load is applied on the first and third node. The first load is directed along the negative z-axis, the second load along the positive y-axis. The other two nodes have no forces applied.

Listing 3: Example force file

```
4
1.0
0.0 0.0 -0.65
0.0 0.0 0.0
0.0 2.34 0.0
0.0 0.0 0.0
```

4.2.3 System setup

- setting up libmesh
  - LinearImplicitSystem, erklärung aus doxygen
  - 6 variablen erstellen
  - set boundary condition types, DirichletBoundary class
  nochmal auf theorieteil eingehen

4.2.4 Matrix and vector assembly

- material matrices: D_m und D_p
- creation of local and global stiffness matrix
- integral with gauss-quadrature
- read corresponding entry from force array and set it in the right-hand-side vector
- storing already processed nodes with unordered_set-structure

4.2.5 Solving the system

- solving part is just one line. calls the assemble-function and the defined solver (e.g. if PETSc is installed)
- which settings are possible (error-eps, #iters)
- build_solution_vector

4.2.6 Output

- for standalone-version output as exodus2-file
- exodus2 can be read by ParaView, parallel vtk-output is not supported by libMesh
  - according to the user command line parameter, output file is written to specified file or not at all

4.3 Parallelization with MPI

additional steps to make the code ready for multi process execution with MPI

4.3.1 libMesh requirements

grundsätzlich ist zum lösen des gleichungssystem mit mehreren prozessen petsc als externe lib notwendig

4.3.2 Partitioning the mesh

am mesh muss nichts verändert werden, da libmesh automatisch eine partitionierung des meshes vornimmt (kann aber verbessert werden)

4.3.3 Local elements

grundsätzlich sorgt libMesh dafür, dass jeder prozessor nur zugriff auf "seine" element, knoten usw. hat. Das wird hier wichtig mit änderungen an den iteratoren usw.

4.3.4 Assembly changes

damit rhs korrekt gesetzt wird muss über die prozessgrenzen hinweg klar sein, ob knoten bereits bearbeitet wurde oder nicht. wie das gelöst wurde kommt hier rein

5 Coupling with preCICE

5.1 Introduction to coupling

5.2 Introduction to preCICE

short introduction what preCICE is

5.3 Coupling methods

siehe paper

5.4 Implementation

"modification" of the code to work with preCICE

- - 2: inner node, part of preCICE interface region (only used by preCICE coupled variant)
  - 20: same as 0 but additionally part of preCICE interface region (only used...)
  - 21: same as 1 but additionally part of preCICE ...

6 Validation

The code was tested with several problems to validate its correctness and state where and why there are differing results to the existing (commercial) FEM codes

6.1 Test A: Membrane Displacement with Tri-3

Sprungbrett bestehend aus 8 Elementen; links fest eingespannt, rechts Kraft in y-Richtung an beiden Randknoten
Sprungbrett bestehend aus 32 Elementen in Fischgrätmuster angeordnet; selbe BCs aber andere Kraftwerte
test_c.xda, test_d.xda - beides korrekt

6.2 Test B: Membrane Displacement with Quad-4

Sprungbrett bestehend aus 3 Elementen mit BC wie in Test A aber einzelne Kraft auf oberen rechten Knoten in neg. y-Richtung
selbes Mesh wie in test_d.xda nur eben mit 16 Elementen. Selbe BCs, selbe Kraftwerte
test_e.xda, test_f.xda - beides korrekt

6.3 Test C: Plate Displacement with Tri-3

Platte an allen 4 Seiten eingespannt. Einzelne Kraft im Zentrum in neg. z-Richtung
Alternativ mit anderen Parametern test_g
test_a_triN.xda, test_g_triAB_N.xda - korrekt, noch nicht getestet

6.4 Test D: Plate Displacement with Quad-4

Selbes mesh wie Test C nur eben mit Quadelementen
test_a_quadN.xda, test_g_quad_N.xda - korrekt, noch nicht getestet

6.5 Test E: Shell Displacement with Tri-3

Ein H-Trägerbalken. Am einen Ende fest eingespannt. Am anderen Ende wird oben eine Kraft am äußeren Knoten in den Balken hinein in flacher Ebene gegeben, gleichzeitig wird unten an der gegenüberliegenden Seite eine Kraft in entgegengesetzter Richtung gegeben
test_j_tri.xda - korrekt

6.6 Test F: Shell Displacement with Quad-4

Gleich wie Test E nur eben Quadelemente
test_j_quad.xda - korrekt

6.7 Test G: Convergence (increasing number of elements)

??? theoretisch mit Test C/D bereits durchführbar mit N=2,4,8,16,32,64,128

6.8 Test H: MPI (increasing number of processes)

??? theoretisch alle Tests, z.B. E/F mit Prozessoranzahl = 1,2,4,8,16. In dem Fall ist natürlich die Zeit interessant und ob die Ergebnisse jeweils alle gleich sind

6.9 Test I: Coupling with preCICE

???

7 Conclusion

What does my code do, what problems arose, what problems persist, what does my code cannot do, where are opportunities for extensions, etc.

References

[BAA+15] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.

[BBH80] Jean-Louis Batoz, Klaus-JÜRgen Bathe, and Lee-Wing Ho. A study of three-node triangular plate bending elements. *International Journal for Numerical Methods in Engineering*, 15(12):1771–1812, 1980.

[BLSS] Martin Bogdan, Kai Ludwig, Elena Sapozhnikova, and Bernd Speiser. Echem++ - a problem solving environment for electrochemistry. `http://www.echem.uni-tuebingen.de/echem/software/EChem++/echem++.shtml/`.

[BT82] Jean-Louis Batoz and Mabrouk Ben Tahar. Evaluation of a new quadrilateral thin plate bending element. *International Journal for Numerical Methods in Engineering*, 18(11):1655–1677, 1982.

[CMPW02] Robert D. Cook, David S. Malkus, Michael E. Plesha, and Robert J. Witt. *Concepts and Applications of Finite Element Analysis*. J. Wiley, 2002.

[Con] Feel++ Consortium. Feel++ - finite element embedded library in c++. `http://www.feelpp.org`.

[Jar] Alexander H. Jarosch. icetools: a numerical ice flow model download. `http://sourceforge.net/projects/icetools/`.

[Kir07] Benjamin Shelton Kirk. *Adaptive Finite Element Simulation of Flow and Transport Applications on Parallel Computers*. PhD thesis, Austin, TX, USA, 2007.

[KK98] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[Kle13] Bernd Klein. *FEM: Grundlagen und Anwendungen der Finite-Elemente-Methode*. Springer-Verlag, 2013.

[KPS13] B. S. Kirk, J. W. Peterson, and R. H. Stogner. The libMesh finite element library: A case for object-oriented high-performance computing. In *PRACE Summer School 2013 - Frameworks for Scientific Computing on Supercomputers; Ostrava, Czech Republic*, June 17–21, 2013. `http://ntrs.nasa.gov/search.jsp?R=20130013759`.

[KPSC06] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3–4):237–254, 2006. http://dx.doi.org/10.1007/s00366-006-0049-3.

[las] LASpack download page. http://www.netlib.org/linalg/.

[lib] libmesh publications listing. http://libmesh.github.io/publications.html.

[mfea] MFEM - finite element discretization library. http://mfem.org/publications/.

[mfeb] MFEM: Modular finite element methods. http://www.mfem.org.

[Pat] Bořek Patzák. publications [oofem wiki]. http://www.oofem.org/wiki/doku.php?id=publications.

[Pat09] Bořek Patzák. Oofem project home page, 2009. http://www.oofem.org.

[PB01] B. Patzák and Z. Bittnar. Design of object oriented finite element code. *Advances in Engineering Software*, 32(10):759–767, 2001.

[PCD+12] Christophe Prud'Homme, Vincent Chabannes, Vincent Doyeux, Mourad Ismail, Abdoulaye Samaké, and Gonçalo Pena. Feel++: A computational framework for galerkin methods and advanced numerical methods. In *ESAIM: Proceedings*, volume 38, pages 429–455. EDP Sciences, 2012.

[pet] PETSc web page. http://www.mcs.anl.gov/petsc.

[PR] Julien Pommier and Yves Renard. Getfem++ - an open source library based on collaborative development. http://download.gna.org/getfem/html/homepage/index.html.

[Spe88] Bernhard Specht. Modified shape functions for the three-node plate bending element passing the patch test. *International Journal for Numerical Methods in Engineering*, 26(3):705–715, 1988.

[SPR] Luis Saavedra, Julien Pommier, and Yves Renard. Mpi parallelization of getfem++ - getfem++. http://download.gna.org/getfem/html/homepage/userdoc/parallel.html.

[Ste15] Peter Steinke. *Finite-Elemente-Methode: Rechnergestützte Einführung*. Springer-Verlag, 2015.

[Thi] Axel Thielscher. free software package for the simulation of non-invasive brain stimulation. http://simnibs.de/.

[Toc63] James Lionel Tocher. *Analysis of plate bending using triangular elements*. PhD thesis, University of California, Berkeley, 1963.

[Wer95] Horst Werkle. *Finite Elemente in der Baustatik*. Springer, 1995.

[ZT00] Olgierd Cecil Zienkiewicz and Robert Leroy Taylor. *The finite element method: Solid mechanics*, volume 2. Butterworth-heinemann, 2000.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift