Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

Markus Aleksander Wulff

Implementation of Tetrahedral Elements in a Finite Element Method Plug-in for Grasshopper

Master's thesis in Civil and Environmental Engineering
Supervisor: Anders Rønnquist
Co-supervisor: Sverre Magnus Haakonsen
June 2022

NTNU
Norwegian University of
Science and Technology

Markus Aleksander Wulff

Implementation of Tetrahedral Elements in a Finite Element Method Plug-in for Grasshopper

Master's thesis in Civil and Environmental Engineering
Supervisor: Anders Rønnquist
Co-supervisor: Sverre Magnus Haakonsen
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

NTNU
Norwegian University of
Science and Technology

Department of Structural Engineering
Faculty of Engineering
NTNU – Norwegian University of Science and Technology

ACCESSIBILITY

OPEN

MASTER THESIS 2022

SUBJECT AREA:	DATE:	NO. OF PAGES:
Structural Engineering	10/06/2022	Vii + 48 + 9

TITLE:

Implementation of Tetrahedral Elements in a Finite Element Method Plug-in for Grasshopper

Implementering av tetraeder-elementer i en elementmetode-programvareutvidelse for Grasshopper.

BY:

Markus Aleksander Wulff

SUMMARY:
In recent years, the algorithms-aided design environment Grasshopper has allowed architects and structural engineers to design complex gridshells with the help of form-finding methods. However, the nodes have a big impact on the structural performance and visual appearance of gridshells, but there is currently no way to analyse the nodes inside Grasshopper. This thesis aims to find out if a finite element method plug-in for Grasshopper can be used to analyse gridshell nodes with sufficient speed, accuracy, and ease of use. In this context, algorithms-aided design is defined as a design method where the geometry is generated by algorithms in a visual programming environment.

To test if gridshell nodes can be analysed inside Grasshopper, tetrahedral solid elements were implemented in an existing finite element method plug-in for Grasshopper, called Solid FEM, which, until now, only accepted hexahedral solid elements. The motivation behind this was that generation of hexahedral meshes is a highly complicated task, while an algorithm for generation of tetrahedral meshes already exist inside Grasshopper. Solid FEM was investigated through two case studies. In the first case study, an analysis of a cantilevered beam was compared to beam theory and the finite element method software, Ansys. In the second case study, a gridshell node was analysed with Solid FEM and compared to Ansys.

Solid FEM proved to be sufficiently accurate for early-stage designs as a basis for decision making and design exploration, but not for documentation of structural reliability. The solver has a maximum capacity regarding number of degrees of freedom which indirectly limits the accuracy. Correct application of loads and boundary conditions is a prerequisite for obtaining a reliable result, and the user should be aware of the impact on accuracy and computation time caused by element type and mesh size. The solver is slow, but still saves you time and work compared to exporting the geometry to other software. In combination with other useful tools in Grasshopper, Solid FEM can become a useful tool for designing complex gridshell nodes.

RESPONSIBLE TEACHER: Professor Anders Rønnquist

SUPERVISOR(S): Sverre Magnus Haakonsen

CARRIED OUT AT: Department of Structural Engineering, Norwegian University of Science and Technology

Preface

This thesis concludes my Master of Science degree in structural engineering at the Department of Structural Engineering at the Norwegian University of Science and Technology.

I would like to thank my supervisor Sverre Magnus Haakonsen for his investment in this thesis. The meetings, discussions and guidance during the course of this thesis has been invaluable. I would also like to thank Silje Knutsvik Kalleberg, Magnus Kunnas and Hilde Iden Nedland for their great work with their Master's thesis in 2021, which formed the base for my work on this thesis. I would like to thank architect and PhD candidate at the Institute of Architecture and Technology, Steinar Hillersøy Dyvik for the discussions regarding gridshell nodes and for providing finished geometry of gridshell nodes used in this thesis. Finally, I would like to thank my girlfriend, family and friends for their continuous support throughout my studies.

Markus Aleksander Wulff

Abstract

In recent years, the algorithms-aided design environment Grasshopper has allowed architects and structural engineers to design complex gridshells with the help of form-finding methods. However, the nodes have a big impact on the structural performance and visual appearance of gridshells, but there is currently no way to analyse the nodes inside Grasshopper. This thesis aims to find out if a finite element method plug-in for Grasshopper can be used to analyse gridshell nodes with sufficient speed, accuracy, and ease of use. In this context, algorithms-aided design is defined as a design method where the geometry is generated by algorithms in a visual programming environment.

To test if gridshell nodes can be analysed inside Grasshopper, tetrahedral solid elements were implemented in an existing finite element method plug-in for Grasshopper, called Solid FEM, which, until now, only accepted hexahedral solid elements. The motivation behind this was that generation of hexahedral meshes is a highly complicated task, while an algorithm for generation of tetrahedral meshes already exist inside Grasshopper. Solid FEM was investigated through two case studies. In the first case study, an analysis of a cantilevered beam was compared to beam theory and the finite element method software, Ansys. In the second case study, a gridshell node was analysed with Solid FEM and compared to Ansys.

Solid FEM proved to be sufficiently accurate for early-stage designs as a basis for decision making and design exploration, but not for documentation of structural reliability. The solver has a maximum capacity regarding number of degrees of freedom which may limit the accuracy. Correct application of loads and boundary conditions is a prerequisite for obtaining a reliable result, and the user should be aware of the impact on accuracy and computation time caused by element type and mesh size. The solver is slow, but still saves you time and work compared to exporting the geometry to other software. In combination with other useful tools in Grasshopper, Solid FEM can become a useful tool for designing complex gridshell nodes.

Sammendrag

I nyere tid har arkitekter og bygningsingeniører designet komplekse gitterskall ved hjelp av formfinningsmetoder i det algoritmiske modelleringsverktøyet Grasshopper. Knutepunktene i gitterskall påvirker i stor grad den strukturelle oppførselen og det visuelle uttrykket, men for øyeblikket er det ikke mulig å analysere knutepunktene i Grasshopper. Denne oppgaven undersøker om en programvareutvidelse til Grasshopper kan brukes til å analysere knutepunkter i gitterskall ved hjelp av elementmetoden, hvor hastighet, nøyaktighet og brukervennlighet skal vurderes. I denne sammenhengen defineres algoritmisk modellering som en design-metode hvor geometrien genereres av algoritmer i et visuelt programmeringsverktøy.

For å undersøke om slike knutepunkter kan analyseres i Grasshopper ble volumetriske tetraederelementer implementert i en eksisterende programvareutvidelse kalt Solid FEM, som utførte elementmetodeanalyser med heksaeder-elementer. Motivasjonen bak dette var at heksaeder-mesh er veldig komplisert å generere, mens det allerede eksisterer en komponent i Grasshopper som kan generere tetraeder-mesh. Solid FEM ble undersøkt gjennom to eksempelstudier. I den første ble en utkragerbjelke analysert, og sammenlignet med bjelketeori og elementmetode-programvaren Ansys. I den andre eksempelstudien ble et gitterskallknutepunkt analysert i Solid FEM og sammenlignet med resultater fra Ansys.

Solid FEM viste seg å være tilstrekkelig nøyaktig for tidlig-fase design, som et grunnlag for beslutningstaking og utforskning av designalternativer, men ikke nøyaktig nok for dokumentasjon av pålitelighet. Programvaren har en makskapasitet på antall frihetsgrader, noe som kan påvirke nøyaktigheten. Korrekt påføring av laster og grensebetingelser er en forutsetning for å oppnå et pålitelig resultat, og brukeren burde være bevisst på påvirkningen elementtypen og størrelsen på meshet har på nøyaktigheten og beregningstiden. Programvaren er treg, men den sparer brukeren likevel for tid og arbeid, sammenlignet med å eksportere geometri til en annen programvare. I kombinasjon med andre nyttige verktøy i Grasshopper kan Solid FEM bli et nyttig verktøy for design av knutepunkter i gitterskall.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

In recent years the use of algorithms-aided design (AAD), also known as parametric design or algorithmic modelling, has gained popularity among both structural engineers and architects. It is called algorithms-aided design due to the fact that the geometry is generated by algorithms in a visual programming environment. When creating an algorithmic model, there is established a relation between all the components of the geometry. For example, if a sphere is placed at the midpoint of a line, it would automatically move if the line is changed. The relation between the line and the sphere stays constant due to the definition of the algorithm. This method enables designers to quickly explore different design options, which makes it a useful tool in early-stage designs. The AAD environment Grasshopper also contains additional tools for doing optimisation, form-finding and structural analyses, among other things, which has made it an attractive software for structural engineers. Creating complex free-form structures has become more normalized due to having such user-friendly tools easily accessible. For example, gridshells are shell-like structures composed of linear members formed in a grid, rather than a continuous surface. Gridshells are usually doubly-curved, which is also the source of their strength, and can quickly be designed with form-finding algorithms like dynamic relaxation or the force density method. With integrated structural analysis tools, the reliability of the structural model can be verified and improved as well. The nodes, which connect the members, are essential parts of gridshells. They influence the structural performance and assembly, while also impacting the visual appearance. The existing structural analysis software in Grasshopper can only analyse members of the structure which can be simplified to lines or shells. The nodes are usually complex shapes which cannot be simplified to lines or shells, so there are currently no way to analyse them inside Grasshopper.

In addition to influencing the structural behaviour of the gridshell, the nodes are often the weakest point, especially in timber gridshells. This means that there is a need to analyse the performance of the nodes inside Grasshopper. Their complex geometry and three dimensional stress state cause a need for a finite element analysis (FEA) using solid elements. Such analyses can be performed in finite element method (FEM) software like Ansys or ABAQUS, but there does not currently exist any tools in Grasshopper which can perform such an analysis. In order to analyse nodes designed in Grasshopper, the geometry needs to be exported and imported in another software. This is a cumbersome and time demanding process which also eliminates the possibility to automatically combine the FEA with other useful tools inside Grasshopper, like optimisation. This is the motivation behind this thesis.

In 2021, for their masters thesis at NTNU, Hilde Iden Nedland, Magnus Kunnas and Silje Knutsvik Kalleberg developed a plug-in for Grasshopper called Solid FEM, which could perform FEA on solid **hexahedral** elements. For this thesis, Solid FEM has been further developed in order to be able to perform FEA on **tetrahedral** elements. The motivation for this is that generation of hexahedral meshes of complex geometries is a difficult task, and there does not currently exist any components in Grasshopper which can do this. However, there does exist a plug-in that can generate tetrahedral meshes of solid geometries, called Tetrino.

1.2 Research question

Through two case studies this thesis will investigate how well Solid FEM works for early-stage design of gridshell nodes. The first case study is a verification of Solid FEM where a cantilevered beam is analysed, and the results are compared to Euler-Bernoulli beam theory and results from the equivalent analysis in Ansys. The second case study is an analysis of a gridshell node with complex geometry. These two case studies will try to answer whether or not Solid FEM works according to its intentions. Namely that it

- is easy to use with little knowledge about FEM.
- is accurate enough.
- is fast enough.

1.3 Layout of this thesis

The layout is as follows: first relevant theory about the finite element method will be presented. Then the different software used for the development of Solid FEM is presented. Section 4 will present and explain Solid FEM and how to use it. Section 5 contains the two case studies with separate discussions. Finally, in Section 6, Solid FEM along with the results from the case studies will be discussed which leads to a conclusion and a part regarding relevant further development of Solid FEM.

2 Theory: The Finite Element Method

2.1 Background

The finite element method (FEM) is a numerical method used within different fields of engineering and physics to approximate the solution of differential equations where an analytical solution is difficult or practically impossible to obtain. Heat transfer and fluid flow are typical problems solved by using FEM, but the method was primarily developed for structural analysis within civil engineering (Bathe, 2014), and is today used by a lot of different structural analysis software.

In structural analysis, FEM is used in cases with large or complex structures with many unknowns. The essence of the method is to subdivide the structure into a finite number of elements and nodes, and assign degrees of freedom (DOFs) to each node. The DOFs can be both translations and rotations of the nodes, and are the unknowns of the system. Each element has a stiffness matrix which dependent on the chosen DOFs, which tells us how strongly the element will resist movements in the nodes for specific load cases.

FEM is based on the displacement method, which can be illustrated by a spring: the force on the spring is equal to the spring constant multiplied by the elongation of the spring: $F = k \cdot \Delta x$. But in structures we can have displacements and rotations in different different directions, and we generally have many nodes, so the simple spring equation turns into a matrix equation: $\mathbf{S} = \mathbf{kv}$, where $\mathbf{S}$ is the element load vector, $\mathbf{k}$ is the element stiffness matrix and $\mathbf{v}$ are the element DOFs:

$$\mathbf{S} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_n \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & \cdots & k_{1n} \\ k_{21} & k_{22} & \cdots & k_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{n1} & k_{n2} & \cdots & k_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \mathbf{kv} \tag{2.1}$$

In equation 2.1, n is the number of element DOFs. To analyse the system of elements we need to assemble the element matrices in system matrices. Different elements contributes to stiffness in the same global DOFs, and the contributions need to be added up. This is done with a connectivity matrix, $\mathbf{a}$, which says which global DOFs the local element DOFs correspond to:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nN} \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix} = \mathbf{ar} \tag{2.2}$$

where n is the number of element DOFs, N is the number of global DOFs, $\mathbf{a}$ is the connectivity matrix and $\mathbf{r}$ are the global DOFs. It can be shown that the global stiffness matrix $\mathbf{K}$ is obtained from equation 2.3:

$$\mathbf{K} = \sum_{i=1}^{N} \mathbf{a}_i^\intercal \mathbf{k}_i \mathbf{a}_i \tag{2.3}$$

where the contribution from each element is assigned to the correct global DOFs and summed up. Then the system stiffness relation is established with the global load vector **R**, global stiffness matrix **K** and the global DOFs **u** and solved by inverting the stiffness matrix as described in equation 2.4:

$$\mathbf{R} = \mathbf{K}\mathbf{u} \;\Rightarrow\; \mathbf{u} = \mathbf{K}^{-1}\mathbf{R} \tag{2.4}$$

When the system equation is solved and the displacements at the nodes have been calculated, the displacements within each element can be obtained by the chosen set of shape functions. Each node has a specific shape function which equals one at the node, and zero at all other nodes. This enables us to obtain the displacements within each element by summing up the contributions from all the nodal displacements. For example for a three dimensional element with n nodes the displacements can be obtained from equation 2.5:

$$\mathbf{u} = \begin{bmatrix} u(x,y,z) \\ v(x,y,z) \\ w(x,y,z) \end{bmatrix} = \begin{bmatrix} \mathbf{N}_0 & 0 & 0 \\ 0 & \mathbf{N}_0 & 0 \\ 0 & 0 & \mathbf{N}_0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{bmatrix} \tag{2.5}$$

where:

$$\mathbf{N}_0 = \begin{bmatrix} N_1 & N_2 & \cdots & N_n \end{bmatrix},\ \mathbf{v}_x = \begin{bmatrix} v_{x1} \\ v_{x2} \\ \vdots \\ v_{xn} \end{bmatrix},\ \mathbf{v}_y = \begin{bmatrix} v_{y1} \\ v_{y2} \\ \vdots \\ v_{yn} \end{bmatrix},\ \mathbf{v}_z = \begin{bmatrix} v_{z1} \\ v_{z2} \\ \vdots \\ v_{zn} \end{bmatrix} \tag{2.6}$$

In equation 2.6, $N_1,\ N_2 \cdots N_n$ are the shape functions related to nodes $1, 2 \cdots n$, and are functions of x, y and z. $\mathbf{v}_x,\ \mathbf{v}_y$ and $\mathbf{v}_z$ are the nodal displacements in direction $x,\ y,\ z$, respectively. The choice of shape functions and their complexity are dependent on the number of nodes and DOFs. For example if you want a one dimensional element to be able to display a state of pure bending, its not enough to have two DOFs and a linear shape function. Displaying pure bending requires at least a 2^{nd} degree polynomial which requires 3 unknowns (DOFs). This implies that the number of nodes and DOFs have a big influence on an elements capability to display different displacement modes, which will affect the elements ability to display strains and stresses. Examples of shape functions and their abilities and limitations are presented in sections 2.4.1 and 2.4.2.

2.2 Strains and stresses

A big part of the FEA is to obtain the strains and the stresses from the nodal displacements. The strains are calculated for each element using the derivatives of the displacements (Bell, 2014):

$$\varepsilon = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{xz} \end{bmatrix} = \begin{bmatrix} \frac{\delta u}{\delta x} \\ \frac{\delta v}{\delta y} \\ \frac{\delta w}{\delta z} \\ \frac{\delta u}{\delta y} + \frac{\delta v}{\delta x} \\ \frac{\delta v}{\delta z} + \frac{\delta w}{\delta y} \\ \frac{\delta u}{\delta z} + \frac{\delta w}{\delta x} \end{bmatrix} = \Delta \mathbf{u} = \Delta \mathbf{N} \mathbf{v} = \mathbf{B} \mathbf{v} \qquad \Longrightarrow \qquad \mathbf{B} = \Delta \mathbf{N} \tag{2.7}$$

where **B** is the strain-displacement matrix and:

$$\Delta = \begin{bmatrix} \frac{\delta}{\delta x} & 0 & 0 \\ 0 & \frac{\delta}{\delta y} & 0 \\ 0 & 0 & \frac{\delta}{\delta z} \\ \frac{\delta}{\delta y} & \frac{\delta}{\delta x} & 0 \\ 0 & \frac{\delta}{\delta z} & \frac{\delta}{\delta y} \\ \frac{\delta}{\delta z} & 0 & \frac{\delta}{\delta x} \end{bmatrix} \tag{2.8}$$

The stresses are then calculated from the strains (Bell, 2014):

$$\sigma = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{xz} \end{bmatrix} = \begin{bmatrix} \lambda + 2G & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2G & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2G & 0 & 0 & 0 \\ 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 \\ 0 & 0 & 0 & 0 & 0 & G \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{xz} \end{bmatrix} = \mathbf{C} \varepsilon \tag{2.9}$$

where

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \tag{2.10}$$

In equations 2.9 and 2.10, G is the shear modulus, **C** is the material dependent elasticity matrix, E is the elasticity modulus and ν is the Poisson's ratio. Both E, G, **C** and ν are constant and equal for all elements of the same material.

2.3 Stiffness matrix K

The stiffness matrix is an essential part FEAs. It contains information about how the structure reacts to different loads and is the most important part for establishing the system of equations needed to be solved in order to obtain the nodal displacements. The global stiffness matrix **K** is assembled from the element stiffness matrices $\mathbf{k}_i$ where $i = 1,2,3...N_{els}$ as seen in equation 2.3. The local stiffness matrices are established from equation 2.11:

$$\mathbf{k} = \iiint_V \mathbf{B}^T \mathbf{C} \mathbf{B} \, dV \tag{2.11}$$

where **B** is the strain-displacement matrix and **C** is the elasticity matrix as described in equations 2.7 and 2.9. But in most cases the shape functions are formulated in terms of natural or dimensionless coordinates: $\mathbf{N} = \mathbf{N}(r,s,t)$, where $r,\ s$, and t are the natural coordinates. This means that we need to transform the shape functions to the Cartesian coordinate system when evaluating the partial derivatives and when performing the integration of the stiffness matrix. This is done with a Jacobian matrix **J** which relates the natural coordinate derivatives to the Cartesian coordinate derivatives (Bathe, 2014):

$$\frac{\delta}{\delta \mathbf{r}} = \begin{bmatrix} \frac{\delta}{\delta r} \\ \frac{\delta}{\delta s} \\ \frac{\delta}{\delta t} \end{bmatrix} = \begin{bmatrix} \frac{\delta x}{\delta r} & \frac{\delta y}{\delta r} & \frac{\delta z}{\delta r} \\ \frac{\delta x}{\delta s} & \frac{\delta y}{\delta s} & \frac{\delta z}{\delta s} \\ \frac{\delta x}{\delta t} & \frac{\delta y}{\delta t} & \frac{\delta z}{\delta t} \end{bmatrix} \begin{bmatrix} \frac{\delta}{\delta x} \\ \frac{\delta}{\delta y} \\ \frac{\delta}{\delta z} \end{bmatrix} = \mathbf{J} \frac{\delta}{\delta \mathbf{x}} \tag{2.12}$$

where:

$$\mathbf{J} = \begin{bmatrix} \frac{\delta x}{\delta r} & \frac{\delta y}{\delta r} & \frac{\delta z}{\delta r} \\ \frac{\delta x}{\delta s} & \frac{\delta y}{\delta s} & \frac{\delta z}{\delta s} \\ \frac{\delta x}{\delta t} & \frac{\delta y}{\delta t} & \frac{\delta z}{\delta t} \end{bmatrix} = \sum_{i=1}^{n} \begin{bmatrix} \frac{\delta N_i}{\delta r} \\ \frac{\delta N_i}{\delta s} \\ \frac{\delta N_i}{\delta t} \end{bmatrix} \begin{bmatrix} x_i & y_i & z_i \end{bmatrix} \tag{2.13}$$

and $x_i,\ y_i$, and z_i are the Cartesian coordinates at node "i" and n is the number of element nodes. From equation 2.12, you see that in order to obtain the partial derivatives of the shape functions with respect to the Cartesian coordinates we need to invert the Jacobian:

$$\begin{bmatrix} \frac{\delta N_i}{\delta x} \\ \frac{\delta N_i}{\delta y} \\ \frac{\delta N_i}{\delta z} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \frac{\delta N_i}{\delta r} \\ \frac{\delta N_i}{\delta s} \\ \frac{\delta N_i}{\delta t} \end{bmatrix} \tag{2.14}$$

When all the terms of the **B**-matrix are calculated, and before we perform the integration of the stiffness matrix, we need to transform the volume differential dV to natural coordinates:

$$dV = \det(\mathbf{J}) \, dr \, ds \, dt \tag{2.15}$$

where $\det(\mathbf{J}) = J$ is the determinant of the Jacobian matrix.

Now the element stiffness matrix can be obtained from equation 2.16:

$$\mathbf{k} = \iiint_V \mathbf{B}^T \mathbf{C} \mathbf{B} \, J \, dr \, ds \, dt \tag{2.16}$$

According to Bathe, 2014, an analytical evaluation of the integral in equation 2.16 is, in general, not effective. Instead, numerical integration (or quadrature rule) is employed. In numerical integration the analytical integral is approximated by the formula:

$$I = \int_a^b f(x)dx \approx \sum_{k=1}^{n} w_k f(x_k) \tag{2.17}$$

where x_k are the integration points, w_k are the *weights* for each integration point and n is the number of integration points. There exists different quadrature rules that, for n integration points, can integrate a polynomial of order p exactly. For example, Zienkiewicz et al., 2013 lists a quadrature rule for integrating over tetrahedrons which is presented in Table 2.1.

Order	**Point**	**Coordinates**				**Weights**
Linear	a	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	1
Quadratic	a	α	β	β	β	$\frac{1}{4}$
	b	β	α	β	β	$\frac{1}{4}$
	c	β	β	α	β	$\frac{1}{4}$
	d	β	β	β	α	$\frac{1}{4}$
		$\alpha = 0.58541020$				
		$\beta = 0.13819660$				

Table 2.1: Integration points in natural coordinates and weights for numerical integration over tetrahedrons.

With the use of numerical integration, the final expression for the local stiffness matrices becomes:

$$\mathbf{k} = \iiint_V \mathbf{B}^T \mathbf{C} \mathbf{B} \, J \, dr \, ds \, dt = \sum_{k=1}^{n} \mathbf{B}_k^T \mathbf{C} \mathbf{B}_k \, J_k \tag{2.18}$$

where $\mathbf{B}_k$ and J_k are the strain-displacement matrix and the Jacobian determinant evaluated at the integration points.

2.4 Solid elements

In structural problems where the geometry can not be represented by beam-, plate- or shell-elements, solid elements are needed to obtain reliable results. Connections between structural members are typical candidates for solid elements. The formulation of the elasticity problem is actually simpler in three dimensions than in one and two dimensions, as described in (Bell, 2014), due to the fact that there is no need to make any assumptions in order to simplify the problem. However, solving the resulting problem demands substantially more computational power. This is best visualized by an example. Figure 2.1 shows three different ways to model a beam with finite elements. Table 2.2 shows how the number of DOFs grows from 63 for the one dimensional case, to 945 for the three dimensional case. This means that solving the system of equations involves inverting a 945x945 matrix, which is substantially more demanding than inverting a 63x63 matrix. This illustrates that one should only use solid elements in cases where the geometry can not be simplified to one or two dimensional elements.

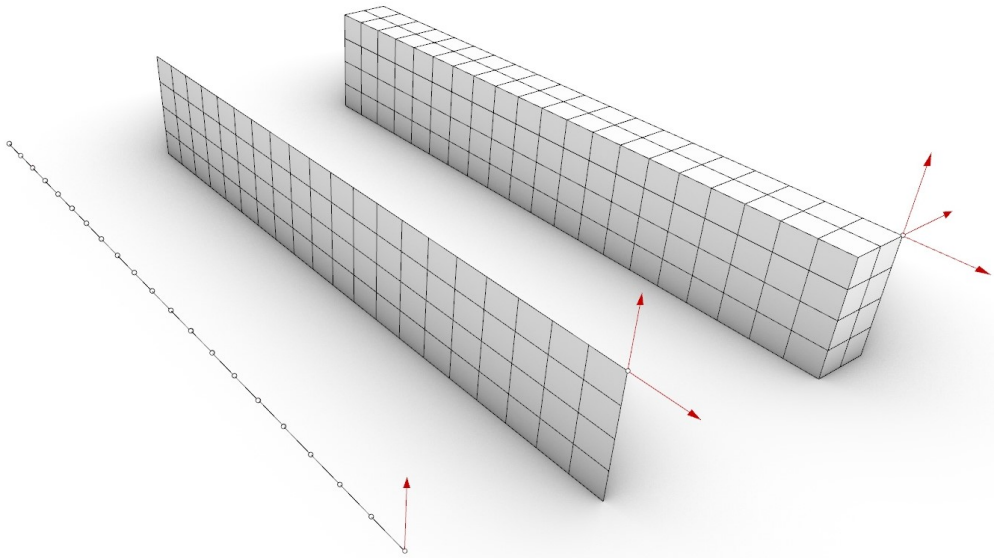


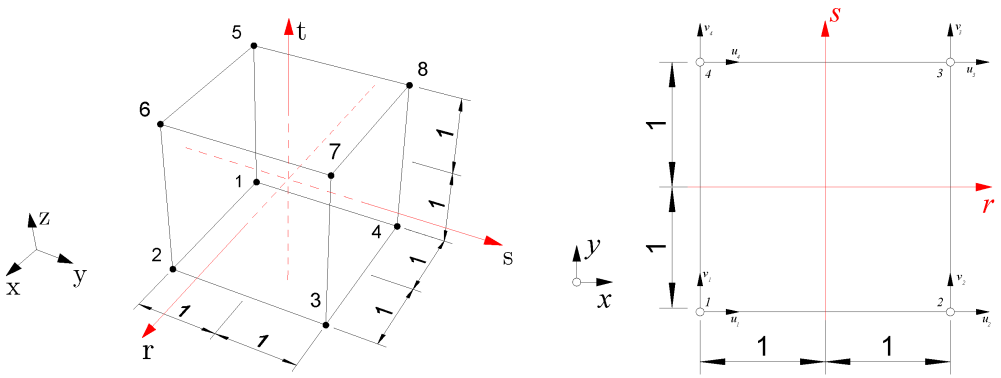
Figure 2.1: 1D, 2D and 3D modelling of a beam.

Dimensions	**1D**	**2D**	**3D**
Number of elements	20	80	160
Number of nodes	21	105	315
Number of DOFs	63	315	945

Table 2.2: Complexity of modelling a beam in one two and three dimensions.

2.4.1 Hexahedral elements

Hexahedral solid elements are rectangular prisms with six faces and 8 corners. The simplest hexahedral element is the HEX8 element, seen in Figure 2.2a, often denoted "brick element", which has one node in each of its 8 corners. Each node has three DOFs: translation in x-, y- and z-direction, denoted u, v and w. The brick elements capabilities are more conveniently illustrated by the equivalent two-dimensional element, the plane stress Q4 element with two DOFs in each node, as seen in Figure 2.2b.



(a) HEX8 element. (b) Q4 element.

Figure 2.2: Solid HEX8 element and plane stress Q4 element.

The Q4 element has two translational DOFs in each of its four corner nodes. This enables the element to display a bi-linear displacement pattern with shape functions given in equation 2.19 (Bathe, 2014).

$$\begin{aligned} N_1 &= \frac{1}{4}(1+r)(1+s) \\ N_2 &= \frac{1}{4}(1-r)(1+s) \\ N_3 &= \frac{1}{4}(1-r)(1-s) \\ N_4 &= \frac{1}{4}(1+r)(1-s) \end{aligned} \tag{2.19}$$

The shape functions are given in natural coordinates (r, s) which is a non-dimensional coordinate system where the corner nodes have coordinates $r,s = \pm 1$. This simplifies the shape functions, making it much easier to obtain the strain-displacement matrix **B** and thereby also the stiffness matrix **k**. Figure 2.3 shows the eight different displacement modes of the Q4 element, and the deformed element can have any combination of these displacement modes.

The HEX8 element has more or less the same capabilities and limitations as the Q4, but with an additional dimension. The shape functions for the HEX8 are given in equation 2.20 (Bathe, 2014):

$$\begin{aligned} N_i &= \frac{1}{8}(1+r_i r)(1+s_i s)(1+t_i t) \\ r_i, s_i, t_i &= \pm 1 \end{aligned} \tag{2.20}$$

where n_i is the shape function corresponding to node "i" and r_i, s_i, t_i are the point coordinates of node "i" given in the natural coordinate system *(r, s, t)*. This enables the HEX8 element to display a tri-linear displacement field, with the equivalent modes of the Q4, only with an additional dimension.

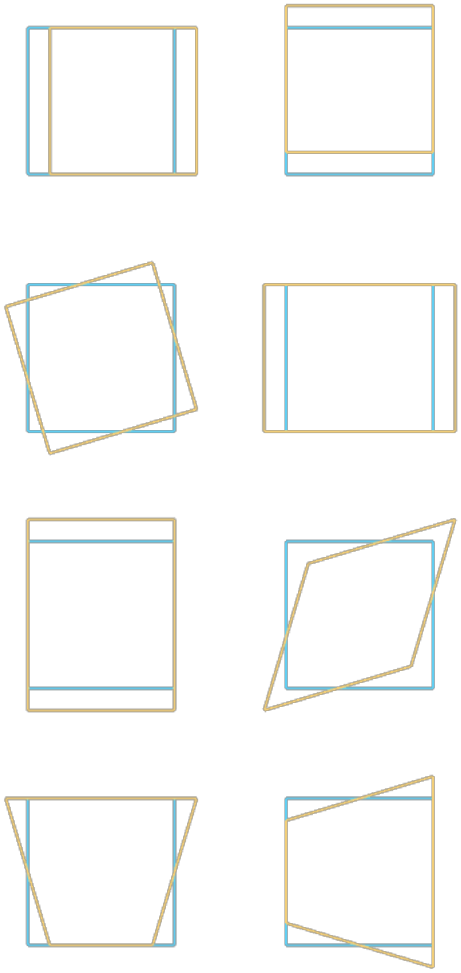


Figure 2.3: Displacement modes of the Q4 element.

HEX8 elements have some advantages and some disadvantages. The advantages are:

- They have a very simple geometry which makes it easy to mesh simple geometry which is composed of straight lines and plane surfaces, for example cubes, cuboids or prisms. If formulated as *isoparametric* elements, HEX8 elements can also have arbitrary shape which enables them to better represent geometry which not only consists of right angles and parallel lines and surfaces.
- You would need fewer hexahedral elements compared to tetrahedral elements for the same geometry and the same number for nodes/DOFs. This is because you need at least six tetrahedral elements to make up one hexahedral element (Bell, 2014). This makes it computationally more efficient to assemble the stiffness matrix as there are fewer elements that we need to add the contribution from (See equation 2.3).
- The formulation of the HEX8 element shape functions is very simple, as seen in equation

2.20. All eight nodes have very similar and clean shape functions which makes it easier to implement in a FEM-solver, as you need to calculate the partial derivatives in order to establish the stiffness matrix, as seen in 2.3.

Some disadvantages of HEX8 elements are:

- They exhibit something called shear locking behaviour which happens when the element is trying to display pure bending. The same thing happens with the plane stress Q4 element, and Figure 2.4 illustrates the phenomena. What happens is that when a Q4 (or HEX8) element is bent, the top and bottom edges remain straight, and each node only has a horizontal displacement. However, for the analytical case, the top and bottom edges become curved, and each node as a horizontal as well as a vertical displacement. This means that the element experiences so-called parasitic shear strains in addition to the pure bending strains. These parasitic shear strains absorbs strain energy so that the bending strains are smaller than they should be for the given load case, which makes the element too stiff.
- A distinct disadvantage of the HEX8 element is that they cannot easily mesh complex geometry. While they are very useful for meshing geometry with straight edges, and plane surfaces, they run into trouble when trying to mesh curved geometry, especially when it curves in multiple directions and with high curvature. If the element is upgraded to the 20 node HEX20 element, it can represent curved geometry, but generating such a mesh is still a highly complicated task.

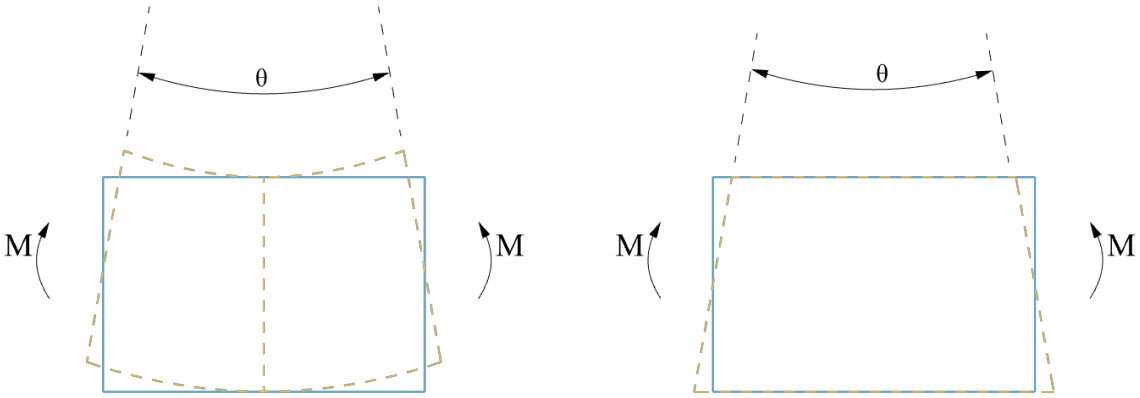


Figure 2.4: Shear locking in Q4 element. Analytical case to the left and Q4 to the right.

2.4.2 Tetrahedral elements

Tetrahedral elements are elements with four corners and four faces. The simplest tetrahedral element is the 4 noded TET4 element, also known as the constant strain tetrahedron, with four corner nodes. As the HEX8 element, the TET4 element has three DOFs in each node (translation in each direction), see Figure 2.5. The TET4 element is the three dimensional equivalent to the plane stress triangular CST element. It can display a tri-linear displacement field, and thus it can only represent a constant strain field. If you take the TET4 element and add nodes on each

element edge, you get the ten noded TET10 element, also known as the linear strain tetrahedron. This element can represent a quadratic displacement field, and thus can display a linear strain field. This enables the TET10 element to display a state of pure bending **exactly**.

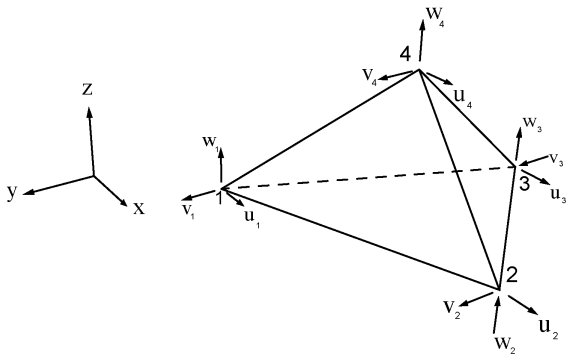


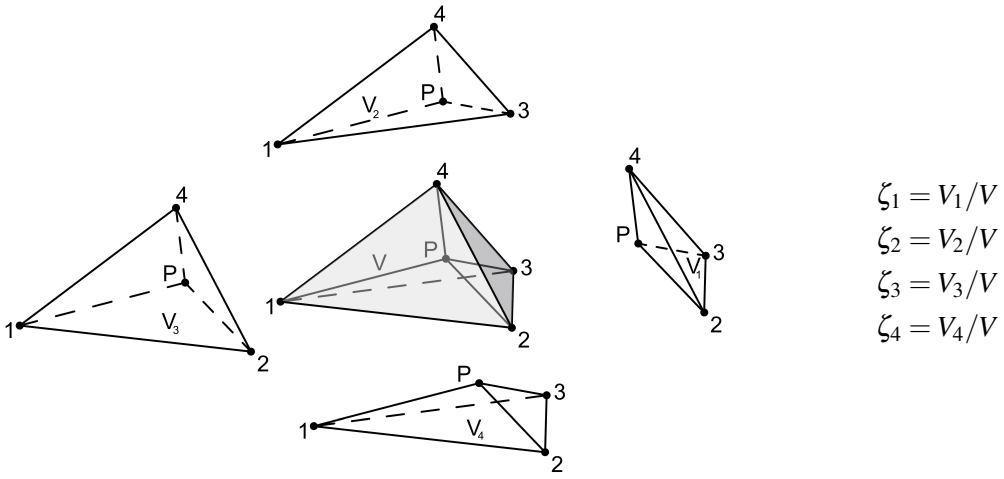
Figure 2.5: TET4 element.

Natural coordinates are also used for tetrahedrons, but in a slightly different manner than for hexahedral elements. Something called volume coordinates are used. There is one coordinate related to each corner node, denoted ζ_i, and they are defined as follows for an internal point P:

- The point P divides the tetrahedron into 4 different volumes/tetrahedrons
- V_i is the volume of the tetrahedron where node i is swapped with the point P, as illustrated in Figure 2.6.
- The coordinate ζ_i is then defined as V_i/V where V is the volume of the original tetrahedron such that:

$$\sum_{i=1}^{4} \zeta_i = \frac{V_1 + V_2 + V_3 + V_4}{V} = 1 \tag{2.21}$$

A property of volume coordinates is that they have the value of either 1 or 0 at the nodes, and they vary linearly between nodes. For example, ζ_1 has the value of 1 at node 1 and the value 0 at all



$$\zeta_1 = V_1/V$$
$$\zeta_2 = V_2/V$$
$$\zeta_3 = V_3/V$$
$$\zeta_4 = V_4/V$$

Figure 2.6: Volume coordinates for Tetrahedrons.

other nodes. This makes volume coordinates suitable shape functions for TET4 elements:

$$\mathbf{u} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \mathbf{N}_0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{N}_0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{bmatrix} = \mathbf{N}\mathbf{v} \tag{2.22}$$

where

$$\mathbf{N}_0 = \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix} = \begin{bmatrix} \zeta_1 & \zeta_2 & \zeta_3 & \zeta_4 \end{bmatrix} \tag{2.23}$$

and

$$\zeta_i = \frac{V_i}{V} \tag{2.24}$$

However, to uniquely describe a point in space you only need three coordinates/dimensions. And as we see in equation 2.21, the four volume coordinates are linearly dependent. For this reason one has to choose three of the four volume coordinates to represent the natural coordinate system (r,s,t). For example:

$$\begin{aligned} r &= \zeta_1 \\ s &= \zeta_2 \\ t &= \zeta_3 \end{aligned} \tag{2.25}$$

This can then easily be substituted into the shape functions:

$$\mathbf{N}_0 = \begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix} = \begin{bmatrix} r & s & t & (1-r-s-t) \end{bmatrix} \tag{2.26}$$

Volume coordinates are used for TET10 element shape functions, although not as simple and elegant as for TET4 elements. (Bathe, 2014) lists the ten shape functions as:

$$\begin{aligned} N_1 &= 1-r-s-t-\tfrac{1}{2}N_5-\tfrac{1}{2}N_7-\tfrac{1}{2}N_{10} & N_6 &= 4rs \\ N_2 &= r-\tfrac{1}{2}N_5-\tfrac{1}{2}N_6-\tfrac{1}{2}N_8 & N_7 &= 4s(1-r-s-t) \\ N_3 &= s-\tfrac{1}{2}N_6-\tfrac{1}{2}N_7-\tfrac{1}{2}N_9 & N_8 &= 4rt \\ N_4 &= t-\tfrac{1}{2}N_8-\tfrac{1}{2}N_9-\tfrac{1}{2}N_{10} & N_9 &= 4st \\ N_5 &= 4r(1-r-s-t) & N_{10} &= 4t(1-r-s-t) \end{aligned} \tag{2.27}$$

There are advantages and disadvantages of using tetrahedral elements. Advantages:

- Tetrahedrons are much better for meshing complex geometry with surfaces curved in multiple directions, as the element edges can remain straight, while still providing an approximation of the geometry close to the original shape. There exists algorithms that generates such tetrahedral meshes efficiently based on a surface mesh or a solid geometry. This is a huge advantage over hexahedral meshes which are much more difficult to generate for complex geometries.
- In the case of the TET4 element, it is remarkably easy to implement in a FEM-solver, as volume coordinates make the shape functions very simple expressions. This in turn elim-

inates the need for integration when establishing the stiffness matrix because the strain-displacement matrix **B** and the jacobian matrix **J** are constant for TET4. This also makes assembly of the global stiffness matrix computationally efficient as there is no need for numerical integration.

- TET10 elements are very good for displaying states of pure bending without the need for a very fine mesh. This can reduce the number of elements needed to obtain results with sufficient accuracy.

The disadvantages:

- As mentioned in Section 2.4.1 you need at least six tetrahedral elements to make up one hexahedral element. This means that for establishing the global stiffness matrix **K** you need to add the contribution from six times as many elements compared to hexahedral elements. However, this is more of a disadvantage for TET10 elements, as TET4 elements does not need gauss integration and the resulting TET4-mesh would contain the same number of nodes (and DOFs) as a HEX8 mesh. TET10 elements however would add many more nodes to the global mesh which increases the size of the stiffness matrix and thus the number of system equations.
- TET4 elements are poor at representing states of bending or twisting because the strains (and thus the stresses) are constant over the span of an element. This means that for TET4 elements to be sufficiently accurate you either need a situation where the stresses are almost constant, or you need a **very** fine mesh as the convergence is slow.

Comment:

- Even though you need six times more TET4 elements compared to HEX8 elements, the number of global nodes is approximately the same for an equivalent mesh. This means that in both cases you have an equal number of DOFs and elements in the stiffness matrix, so solving the resulting system of equations, which is the most time consuming and CPU-demanding task in an FEA, would be equally demanding and time consuming for TET4 and HEX8 elements.

3 Software

This section describes the different software and computer programs used for the work with this thesis. This includes some computer-aided design (CAD) software, FEM software, mesh generation software and programming software.

3.1 CAD software

3.1.1 Rhinoceros 3D

Rhinoceros 3D (McNeel et al., 2022), more commonly referred to as Rhino or Rhino 7, is a 3D CAD software developed by Robert McNeel & Associates. Rhino is a versatile CAD software which can be used for 3D modelling, analyses and presentation, among other things. By using NURBS curves and surfaces it represents geometry with high level of accuracy (Robert McNeel & Associates, 2022b), which is necessary for both architectural and engineering projects. Rhino also includes a rendering tool which can generate reality-like visualizations for presentations. For this thesis, Rhino has been used for the add-on software Grasshopper, explained in Section 3.1.2, and for visualizing the results from the FEA with coloured meshes.

3.1.2 Grasshopper

Grasshopper (Robert McNeel & Associates, 2022a) is an add-on for Rhino 7, and is an AAD environment. In Grasshopper, like other AAD environments, instead of drawing points, lines and surfaces manually, these are programmed in a visual programming interface called canvas. On the canvas there is established a relation between the different parts. For example, one can define two points by their coordinates, and define a line between these two points. If the point coordinates are changed, the line changes with the points. These relations make the model more flexible regarding changes in the design.

Another thing making AAD a useful tool is setting different parameters as variable values by the use of *sliders*. For example by letting the height, width and length of a building be variable parameters, one can easily and quickly explore many different design options. This is one of the reasons why Grasshopper is a useful tool in early design phase. Often at this stage, the design is far from determined and undergoes many changes before arriving at the final design. An algorithmic model helps to streamline this process, as one doesn't have to redraw the entire model manually every time changes occur.

There also exist numerous plug-ins for Grasshopper that are free to download. These allow the user to do much more than just make a 3D model. For example there are plug-ins that can perform environmental simulations (Ladybug), structural analysis (Karamba3D), optimisation (Galapagos) and mesh generation. Food4rhino.com contains a huge library with different plug-ins free to download. The mesh generation plug-in, Tetrino, is described in Section 3.3.2.

3.2 FEM software

3.2.1 Ansys Mechanical

Ansys Mechanical is a structural FEA software that can perform a variety of analyses. This includes linear and non-linear analyses, dynamic and modal analyses, and thermal analysis for simulating heat conduction (ANSYS Inc, n.d.). It includes a range of material models to accurately represent your structure. Ansys Mechanical includes a meshing engine producing path-conforming meshes, and lets the user select what type of elements to be used. This is the software used for comparisons in the study cases in Section 5.

3.2.2 Solid FEM

Solid FEM is a plug-in for Grasshopper and the research topic for this thesis. It is a linear FEM solver which accepts tetrahedral and hexahedral solid elements. It is described in detail in Section 4.

3.3 Mesh generation

A big challenge when it comes to FEA of complex geometry is concerning mesh generation. The major FEM software includes powerful mesh generation engines which can produce such meshes with both hexahedral and tetrahedral elements. However, generating hexahedral elements is very complicated, and there does not exist a simple algorithm that has been implemented in Grasshopper for generating such meshes. But, there exists an algorithm for generating tetrahedral meshes called TetGen which is implemented in the Grasshopper component Tetrino. These are described in sections 3.3.1 and 3.3.2.

3.3.1 TetGen

TetGen is an algorithm for generation of tetrahedral meshes developed by the research group *Numerical Mathematics and Scientific Computing* at the Weierstrass Institute for Applied Analysis and Stochastics (WIAS). TetGen is an open source code written in C++ and is highly portable - in the sense that it should run and compile on all major computer systems (Weierstrass Institute for Applied Analysis and Stochastics (WIAS), n.d.). The algorithm is both fast, memory efficient and robust. It produces high quality and adaptive tetrahedral meshes suitable for numerical methods, such as the finite element method.

3.3.2 Tetrino

Tetrino is a plug-in for Grasshopper which contains a tetrahedral mesh generator based on the TetGen algorithm (Username: tomsvilans, 2017). It can generate tetrahedral meshes from either a surface mesh or a boundary representation (BREP). This component was a prerequisite for the work related to this thesis. It enabled the author of the thesis to mainly focus his work on implementing tetrahedral elements in a FEM plug-in for Grasshopper. Tetrino, combined with Solid FEM eliminates the need to export geometry from Grasshopper to an external FEM-program, like Ansys or ABAQUS. This saves time and work, and allows you to combine the FEA with other Grasshopper-components, like Galapagos optimisation. Figure 3.1 shows an example of a geometry meshed with Tetrino.

The Tetrino component is quite simple to use and is displayed in Figure 3.2. The *mesh*-input can be either a surface mesh or a BREP. The input *Flags* are integers depending on what kind of output you want:

- 0 returns a list of tetrahedrons
- 1 returns a triangulated mesh
- 2 returns tetra indices
- 3 returns edge indices

MinRatio is the desired tetrahedron ratio.

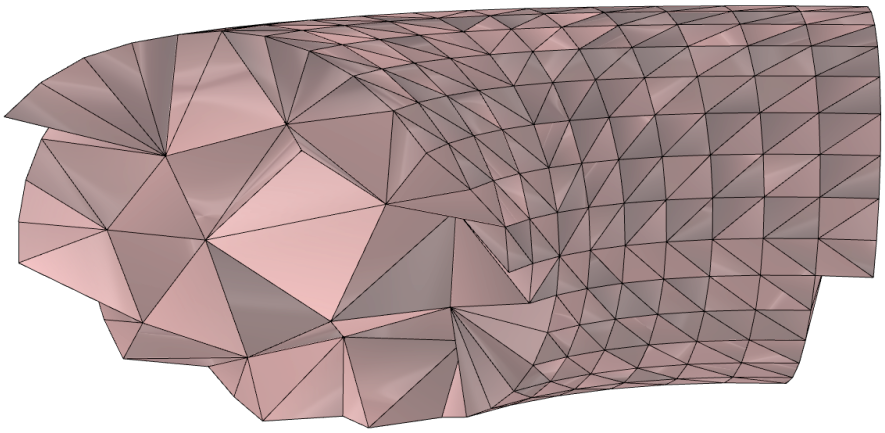


Figure 3.1: Example of geometry meshed with Tetrino.

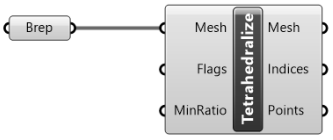


Figure 3.2: Tetrino component in Grasshopper.

3.4 Programming software

3.4.1 C# in Visual Studios

Solid FEM is written in C# using Visual Studios 2019. C# is an object oriented programming language which makes use of classes and objects. This gives the code extended functionality, and helps to keep the code clean and clear. Robert McNeel & Associates, 2022c contains the application programming interface (API) documentation of all the different classes in Rhino geometry which was used for programming the plug-in.

3.4.2 Python for plotting

Python, with the Matplotlib library has been used to create plots and graphs for presenting the results from different analyses in the case studies.

4 Solid FEM

The majority of the work related to this thesis has been dedicated to implementing tetrahedral elements in a FEM solver in Grasshopper. This is based on the work done by Hilde Iden Nedland, Magnus Kunnas and Silje Knutsvik Kalleberg for their masters thesis in 2021. They developed a plug-in for Grasshopper, called Solid FEM, which performed an FEA on eight noded hexahedral elements (HEX8). For this thesis, the plug-in has been further developed and it is now capable of performing an FEA on both linear and quadratic tetrahedral elements, namely TET4 and TET10 elements, as described in Section 2.4.2.

4.1 Classes

The plug-in is written in C# which is an object oriented programming language. This enables the use of classes and objects which provides the code with extended functionality and helps keeping the code clean and readable. The classes used in Solid FEM can be divided into two categories: **physical** and **functional** classes, which are described in the following sections.

4.1.1 Physical classes

The physical classes are classes from which objects with certain attributes are created and used throughout the code. The physical classes and the relation between them are illustrated in Figure 4.1.

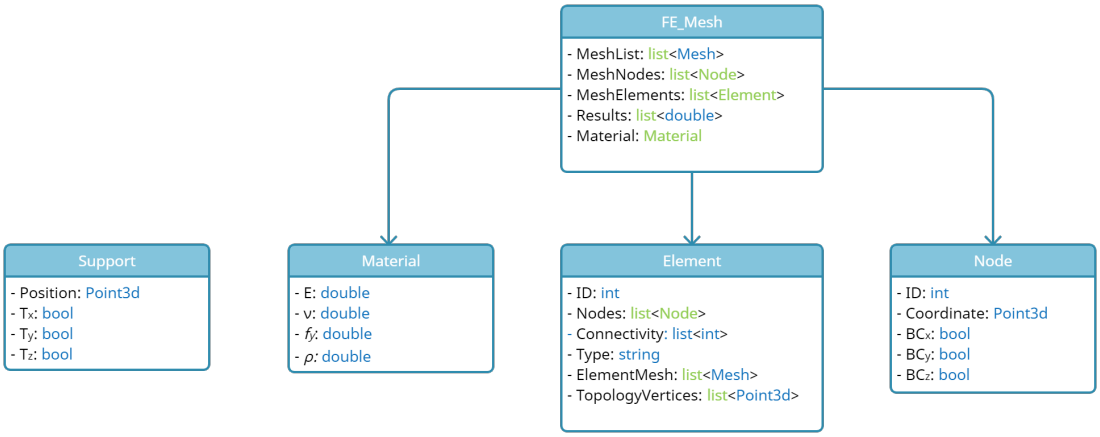


Figure 4.1: Diagram showing the different classes and their relations.

FE_Mesh:
This class contains information regarding the global mesh that is analysed. An FE_Mesh object stores the element meshes, the global nodes, material and all the elements. In addition it contains all the nodal results, like displacements and stresses.

Element:
This class contains information about each element in the FE_Mesh class. The information stored

for each element is the global ID, list of element nodes, the nodal connectivity, element type, element mesh and the list of vertices.

Node:
This class contains information about the nodes in the FE_Mesh class and the element class. Each node has a global ID, point coordinates and boundary conditions in three directions.

Material:
The material class contains the Young's modulus, Poisson's ratio, yielding stress and density of an FE_Mesh object.

Support:
The support class contains the point coordinates of constrained nodes, and in which directions they are constrained.

4.1.2 Functional classes

The functional classes are only made for containing methods (functions) for performing different operations throughout the code. This helps keeping the main code clean, compared to writing every method inside the main code. They do not have any attributes and there are not created any objects from these classes.

FEM_Utility:
Contains methods for performing necessary operations for FEA. For example obtaining the shape functions or the integration points of an element, or calculating displacements and stresses.

FEM_Matrices:
Contains all the methods for calculating different matrices. For example getting the stiffness matrix or the strain-displacement matrix.

FEM_Logger:
This is a class for debugging and storing information about the analysis. For example time consumption of each component, and warning about a negative Jacobian.

4.2 Components

The actual components that are used in Grasshopper are illustrated in Figure 4.2, and can be divided into three categories: **FEM components**, **deconstructors** and **preview compoents**.

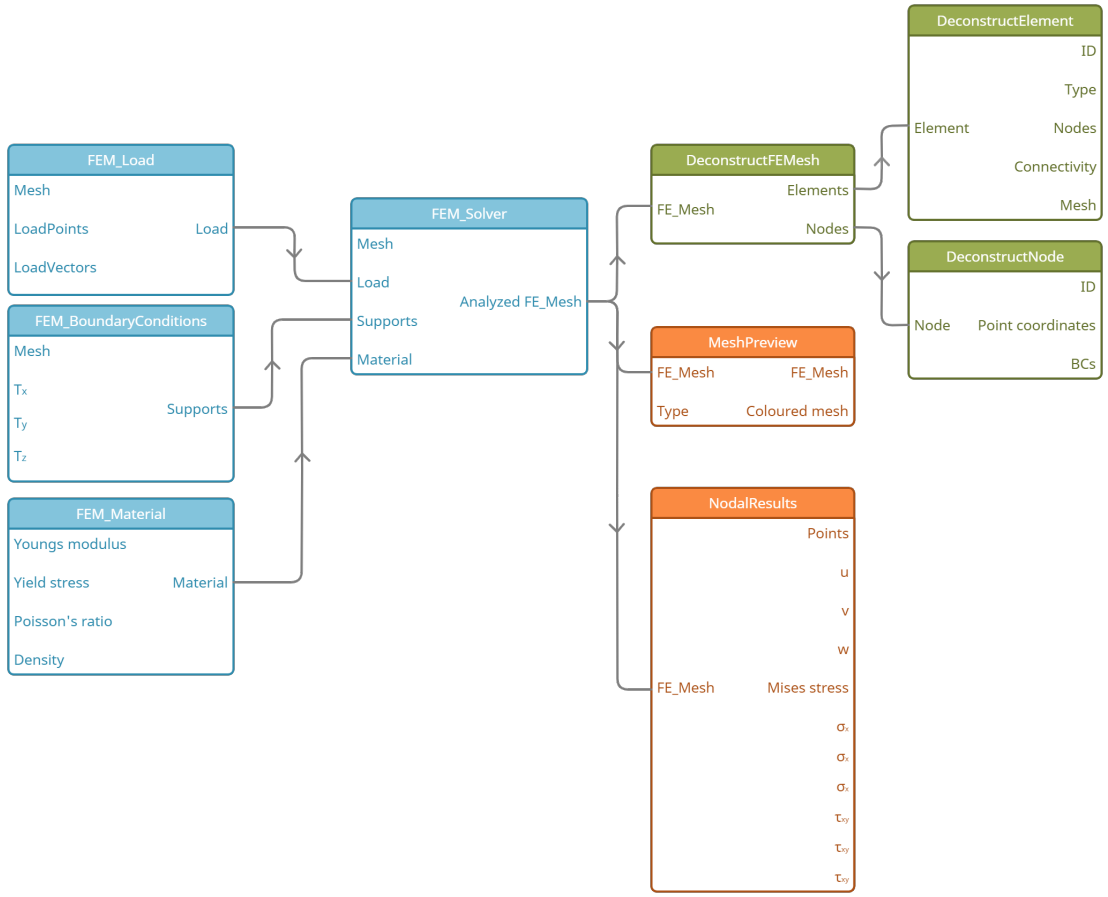


Figure 4.2: The components and how they are connected. The blue are FEM components, the green are deconstructors and the orange are preview components.

4.2.1 FEM components

FEM Boundary Condition

This component applies boundary conditions to mesh points. It takes the input points and constrains them in the directions specified. The output is a list of Support objects. The algorithm is illustrated in Figure 4.3.

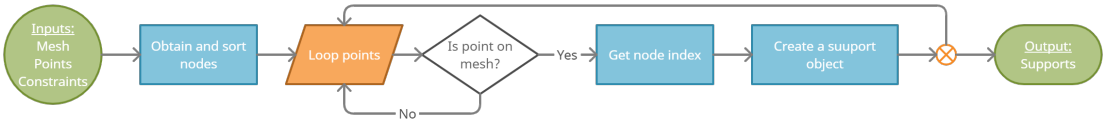


Figure 4.3: Flowchart describing the algorithm of the FEM Boundary Condition component.

FEM Load

This is the component for applying loads. It accepts point loads and the inputs are a list of points along with a list of load vectors and the tetrahedral mesh. The output is a list of n_{dof} values, where the first three values correspond to the force components in x-, y- and z-direction applied to the first global node, the next three values correspond to the second node and so on. The algorithm is illustrated in Figure 4.4.

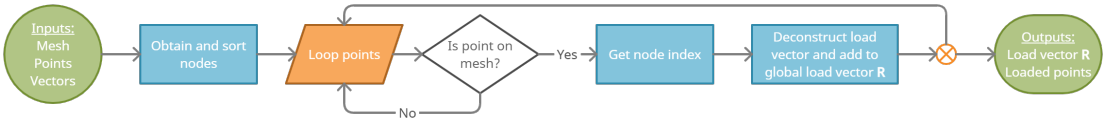


Figure 4.4: Flowchart describing the algorithm of the FEM Load component.

FEM Material

FEM Material creates an object of the material class where the input variables are Young's modulus, Poisson's ratio, yield stress and density. As default these variables correspond to S355 steel.

Add mid-edge nodes

This component transforms TET4 elements to TET10 elements. It takes a tetrahedral mesh with four corner nodes, and returns a tetrahedral mesh with ten nodes, where the additional six nodes are placed on the midpoint of the mesh edges.

FEM Solver

This is the main component which performs the FEA based on mesh, load, boundary conditions and material. The output is an FE_Mesh object which contains all the results from the analysis. The algorithm is illustrated in Figure 4.5.

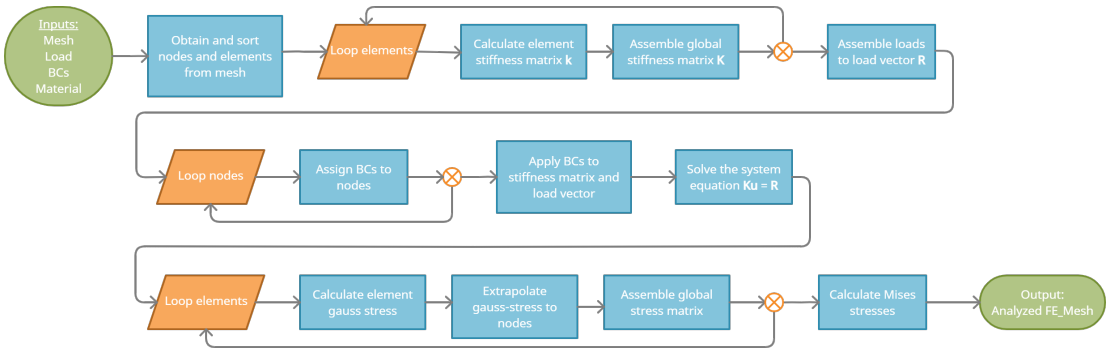


Figure 4.5: Flowchart describing the algorithm of the FEM Solver component.

4.2.2 Deconstructors

Deconstructors are components which allows the user to extract the properties of a class.

Deconstruct FE_Mesh
This components takes an FE_Mesh object and returns its nodes and elements (objects of node class and element class).

Deconstruct Element
Deconstruct Element takes an element object and returns its nodes, connectivity, type, ID and mesh.

Deconstruct Node
Deconstruct Node takes a node object and returns its ID, point coordinates and boundary conditions in x-, y- and z-directions.

4.2.3 Preview components

There are two preview components for presenting and obtaining the result data from the FEA:

Mesh Preview
Mesh Preview takes the analysed FE_Mesh object and returns a coloured mesh displaying either displacements, utilization or the different stress components (x-dir, y-dir, shear stresses etc.).

Nodal Results
Nodal Results takes the analysed FE_Mesh and returns a list of displacements and stresses in all directions for all global nodes in addition to their point coordinates.

4.3 Solid FEM workflow

The purpose of the plug-in is to provide a way to quickly analyse complex geometry with limited knowledge about FEM and structural analyses. This section describes the necessary steps for performing an FEA with the plug-in through a simple example.

The flowchart in Figure 4.6 illustrates the overall procedure which consists of: creating and meshing the geometry, obtaining the load and support points, performing the FEA and previewing the results from the analysis. The colours correspond to the background colours in the following sections describing each step in detail.

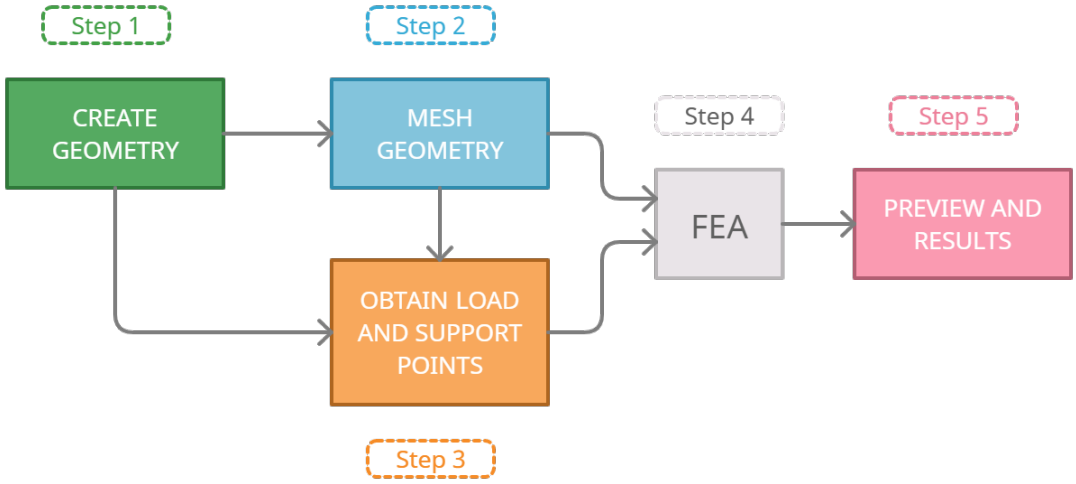
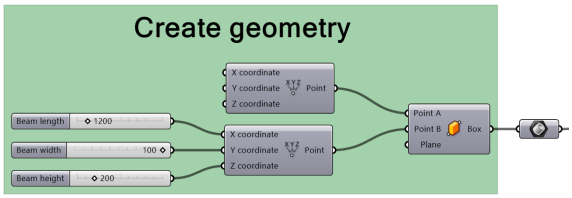


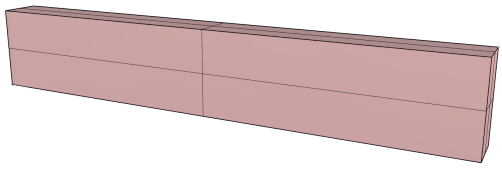
Figure 4.6: Flowchart describing the procedure for performing an FEA with Solid FEM.

4.3.1 Step 1: Create the geometry

The first step is to create the geometry for the object we want to analyse. For example creating a closed boundary representation (BREP). Figure 4.7 shows the algorithm for creating a BREP of a cantilevered beam, by making a box from two points. The dimensions of the beam are variable parameters.



(a) Create geometry algorithm.



(b) Cantilever geometry.

Figure 4.7: Creating geometry.

4.3.2 Step 2: Mesh the geometry

The second step is to mesh the geometry. In order to create the tetrahedral mesh elements, the geometry needs to be formulated as either a BREP or a surface mesh. In order to have more control over the size and quality of the mesh, two components called *Mesh Brep* and *Quad Remesh* can be used to create a surface mesh of the geometry, before being input into the *Tetrahedralize* component. The Solid FEM component *AddMidEdgeNodes* can be used to turn TET4 elements to TET10 elements. Figure 4.8 shows the algorithm for meshing the geometry and Figure 4.9 shows the meshed beam.

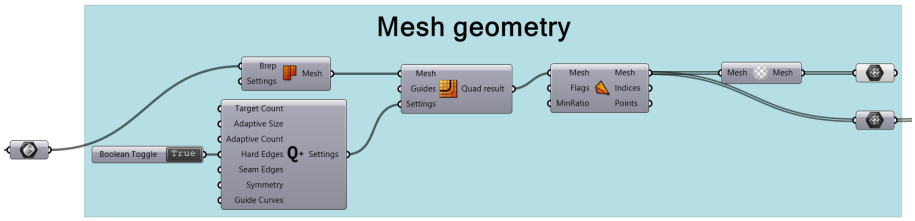


Figure 4.8: Mesh BREP algorithm.

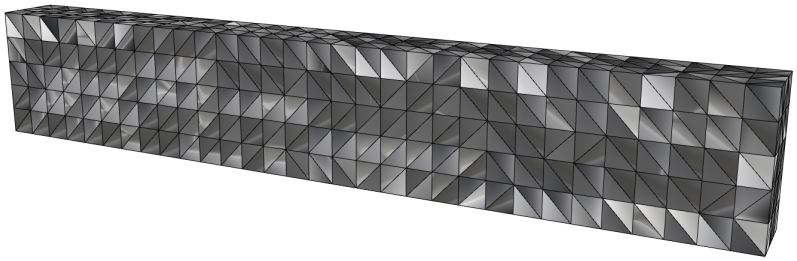


Figure 4.9: Tetrahedral mesh of the beam.

4.3.3 Step 3: Obtain load and support points

The algorithm for this step is illustrated in Figure 4.10. The important part about this step is that the load points and the support points need to be vertices in the mesh. This is why the component deconstruct mesh is used. Figure 4.11 shows the support points (green dots) to the left and the load vectors (green arrows) to the right.

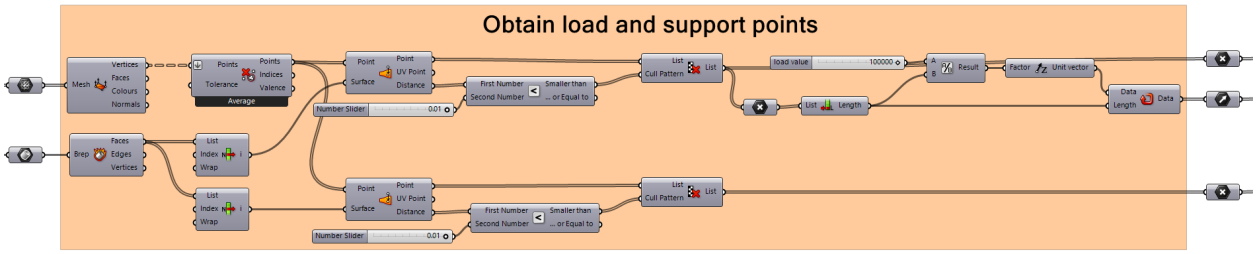


Figure 4.10: Load and support points algorithm.

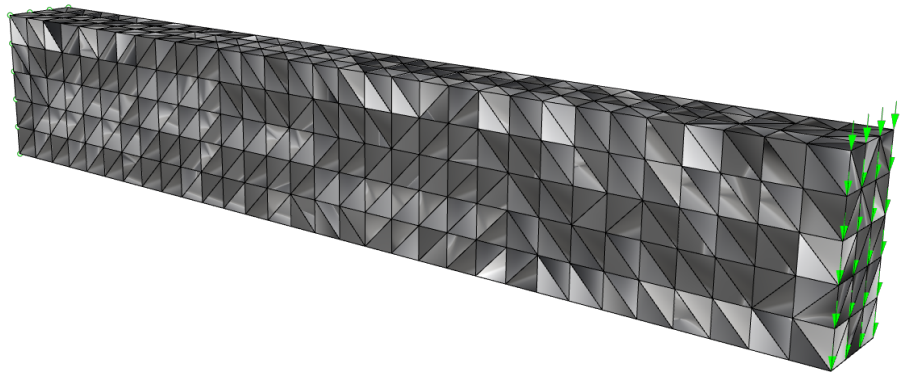


Figure 4.11: Load and support points.

4.3.4 Step 4: Performing the FEA

Step 4 is the main part which is performing the FEA. This includes turning the load points and vectors into correct format through *FEM Load*, and creating Support-objects from the support points with *FEM Boundary Condition*. The inputs Tx, Ty and Tz are boolean values (True or false) which tell in which directions the support points are constrained. *FEM Material* doesn't need any input as the default values correspond to S355 steel, but these need to be specified for any other material. The main component, *FEM Solver*, takes the mesh, loads, BCs and material, and performs the FEA. It returns a list of nodal displacements in all three directions, the elemental and nodal Mises stress, the diagnostics of the analysis, as well as the analysed FE_Mesh object. The algorithm is shown in Figure 4.12.

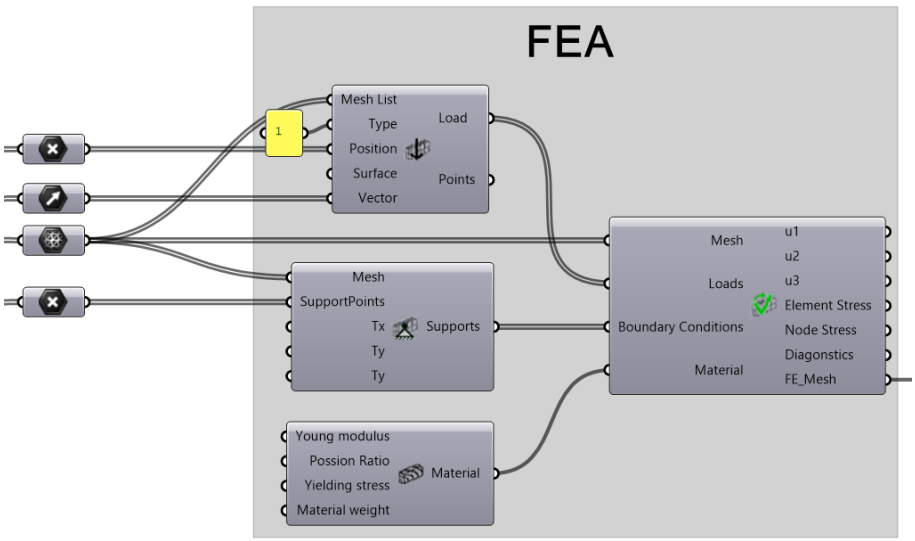


Figure 4.12: FEA algorithm.

4.3.5 Step 5: Preview and results

The final step is to obtain the results from the analysis. *Nodal Results* returns a list of the directional displacements and stresses in the global nodes. With this component the node with the largest displacement and/or stresses can easily be located. *Mesh Preview* takes the analysed

FE_Mesh and returns the deformed mesh with colours showing the distribution of stresses or displacements. The input *Type* determines what kind of results the colour map represents. The algorithm is shown in Figure 4.13. Figure 4.14 shows a colour map of displacements.

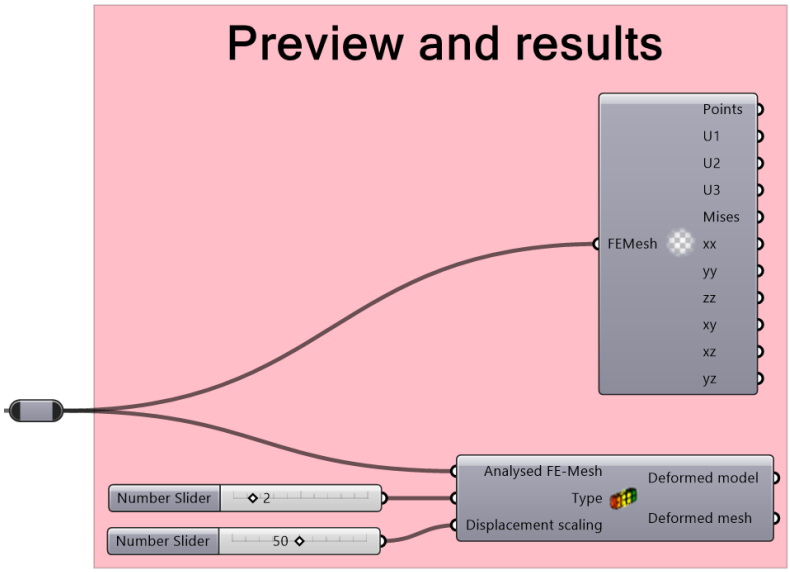


Figure 4.13: Preview and results algorithm.

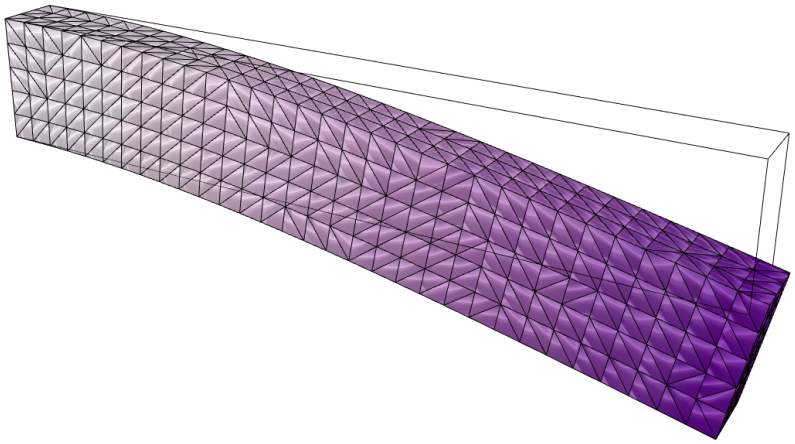


Figure 4.14: Colour map of displacements.

5 Case studies

5.1 Case study 1: Verification of Solid FEM

The first case study is a study of the accuracy and speed of Solid FEM. Through a simple example of a cantilevered beam with a point load at the tip, the accuracy and efficiency of Solid FEM will be compared to Ansys and to the analytical solution obtained using beam theory. TET4 elements will be compared and discussed first, followed by TET10 elements.

The structural problem is illustrated in Figure 5.1 and the relevant data is presented in Table 5.1 and 5.2.

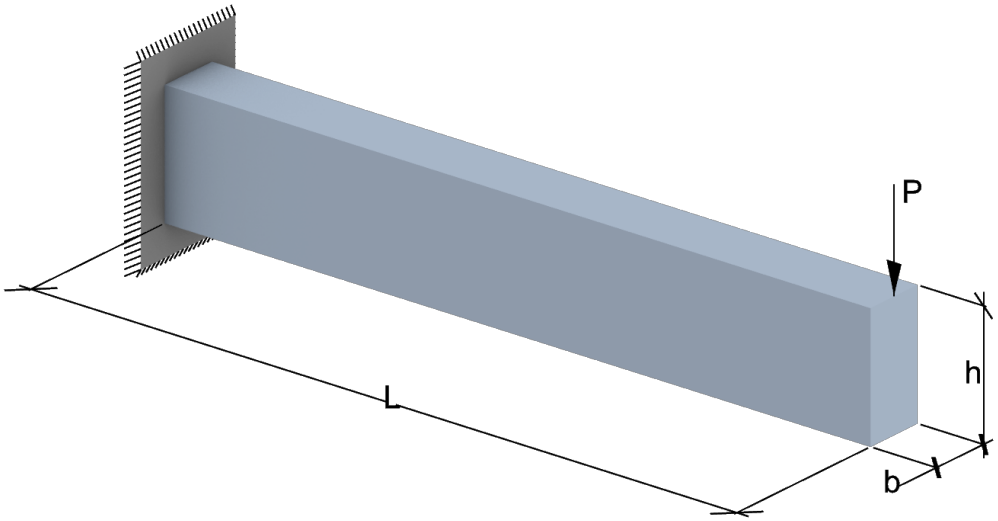


Figure 5.1: Cantilever beam with point load.

Length L	**Width b**	**Height h**	**Load P**	**Material**	w_{max}	σ_{max}
1200 mm	100 mm	200 mm	100 kN	S355 Steel	4.11 mm	180 MPa

Table 5.1: Cantilever data. w_{max} and σ_{max} are calculated from beam theory.

Density ρ	**Yield stress f_y**	**Youngs modulus E**	**Shear modulus G**	**Poisson's ratio ν**
7850 $\frac{kg}{m^3}$	355 MPa	210 GPa	81 GPa	0.3

Table 5.2: Material data.

5.1.1 TET4 elements

For the TET4 test, the cantilever mesh was created semi-manually in Grasshopper, by dividing the beam into cubes and then using the *Tetrino* component to generate tetrahedrons. By doing this, the element size and the number of elements was easier to control and the elements avoid getting stretched. To eliminate sources of error, the exact same mesh was imported into Ansys by creating

a script in Grasshopper. The script contained a list of all mesh nodes, it specified the element type (TET4) and listed all elements and their connectivity. This was created by deconstructing the analysed FE_Mesh, the nodes and elements, and extracting the nodal coordinates and element connectivity. The algorithm for creating these scripts and an example of such a script is found in Appendix A.

The structural problem was analysed for five different mesh divisions to study the convergence of the FEA. Figure 5.2 shows the different mesh divisions. In order to test the accuracy of Solid FEM, the largest displacement w_{max}, the largest stress σ_{xx} and the element Mises stress is compared to the results obtained in Ansys for the same mesh, as well as analytical solution. The results from the FEA along with the mesh divisions are presented in Table 5.3.

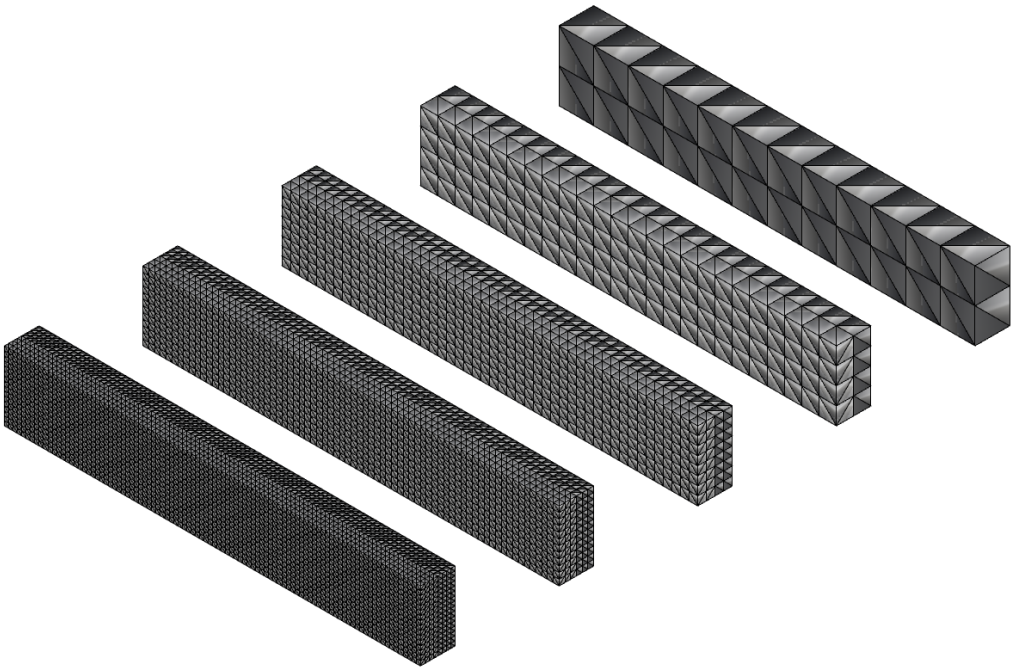


Figure 5.2: Variation of mesh division for the cantilever beam.

Result parameter	**Calculation type**	**12x1x2**	**24x2x4**	**48x4x8**	**72x6x12**	**96x8x16**
w_{max} [mm]	GH TET4	2,27	3,50	4,03	4,13	4,16
	Ansys TET4	2,17	3,50	4,03	4,13	4,16
σ_{xx} [MPa]	GH TET4	80,4	128,8	166,2	188,5	201,8
	Ansys TET4	106,3	171,6	225,8	256,4	279,6
Element mises [MPa]	GH TET4	99,8	158,7	195,8	210,3	219,9
	Ansys TET4	94,6	158,7	195,8	210,3	219,8
Computation time [ms]	GH TET4	4	127	5347	86094	492048
	Ansys TET4	1300	1500	3141	6375	14203

Table 5.3: Results from analyses with TET4 elements.

Displacements:
As can be seen in Table 5.3 the largest displacement at the tip of the beam is almost exactly the same in GH and Ansys for all the mesh divisions. Only the coarsest mesh deviates slightly from the Ansys result. This is very promising and strengthens the credibility and integrity of Solid FEM because Ansys is a reliable FEM software. That being said, in order to obtain a reliable solution, the beam needs to be divided into many elements, as seen in figures 5.3 and 5.4, which increases the computational time substantially. As described in Section 2.4.2, TET4 elements are poor at representing fields of bending, which makes it unfit to model a cantilevered beam where bending is the source of the displacements and stresses. Another remark is that when the mesh is refined, the displacement converges towards a value higher than the beam theory solution, as seen in Figure 5.3. This may be due to the effect of shear deformation. Euler-Bernoulli beam theory neglects shear deformation in the calculation of vertical displacement, but solid elements will experience some shear deformation. This will increase the vertical displacement for the FEA and is a possible cause of the difference.

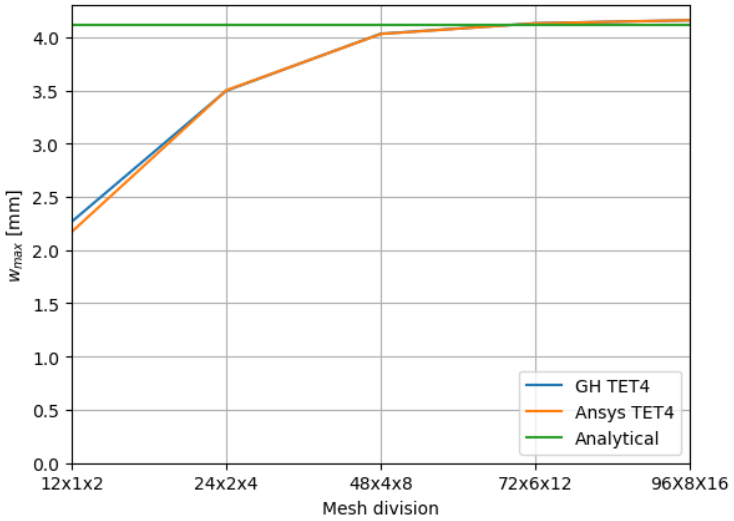


Figure 5.3: Displacement convergence for TET4 elements.

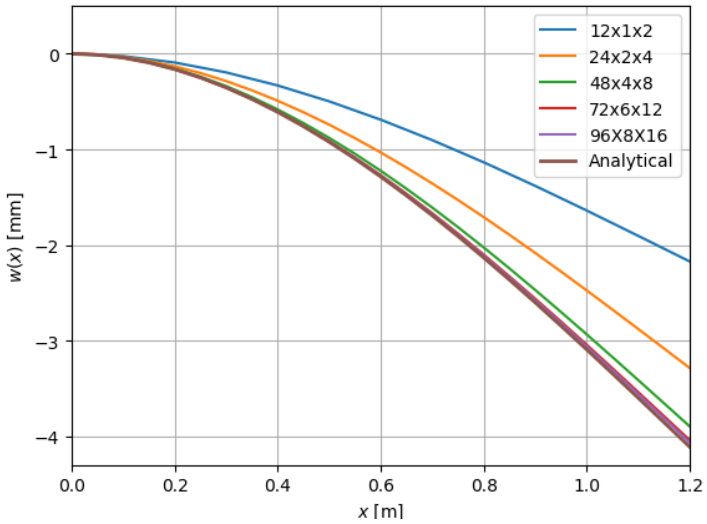


Figure 5.4: Displacements along the length of the beam for the different mesh divisions with TET4 elements and the beam theory solution.

Normal stress, σ_{xx}:
From Table 5.3 we see that the normal stress $\sigma_{xx,max}$ is approximately 73% of the Ansys solution for all the mesh divisions. This is a curious result considering the remarkably close results for displacements. Both the Solid FEM and Ansys solution converges towards a value higher than the beam theory solution, as seen in Figure 5.5. A potential cause of this is stress concentrations near the support points. However, Solid FEM is closer to the beam theory solution when the mesh is refined. The deviation of the stresses is larger compared to the deviation of displacements. This is due to the fact that stresses is derived from the displacements, which means the deviation from analytical solution increases. Also, as described in Section 2.4.2, TET4 elements are suitable in situations where stresses are constant over the span of an element. In a cantilevered beam the stress distribution is linear along the length and height of the beam. This means that the beam needs to be divided in a very fine mesh in order to obtain accurate results with TET4 elements.

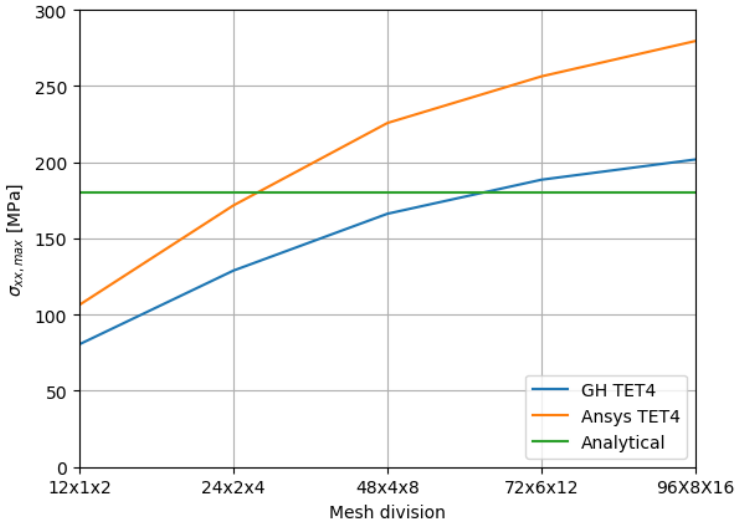


Figure 5.5: σ_{xx} convergence for TET4 elements.

Computation time:
As seen in Table 5.3, the computation time in Solid FEM increases in a much higher rate than in Ansys when the mesh is refined, which inplies that the computation time in Ansys is more stable than in Solid FEM. Solid FEM uses less time than Ansys for the two coarsest meshes, but this is less important than the computation time for the finest meshes, because the absolute difference is still small, and both will be perceived as practically momentarily. For the finest meshes, however, Ansys is much faster. For the finest mesh division, Ansys spends 14 seconds, whereas Solid FEM spends over eight minutes on the same analysis. TET4 elements usually requires a fine mesh to be accurate, which means that Solid FEM needs to improve its efficiency in order to compete with Ansys.

Mises stress:
From Table 5.3 and Figure 5.6 we see that the results for Mises stress from Solid FEM and Ansys are almost identical, like the displacements. Figure 5.7 shows the distribution of Mises stresses. This is also very promising result because in most cases when performing a solid FEA the Mises stresses are more representative for a three dimensional stress state, rather than the directional

stresses separately. This is why Mises stresses are more important for a solid FEA. Another remark is that the Mises stresses, as the normal stresses, converge toward a value higher than analytical value. However, Solid FEM converges to a value higher than the normal stress, and Ansys converges to a value lower than the normal stress. A possible cause for the difference in Solid FEM is that Mises stress accounts for all the directional stresses, like σ_{zz} and τ_{xz}. This could contribute to the difference from σ_{xx}. Regarding the difference between Mises stress and σ_{xx} in Ansys, this can be explained by the fact that the Mises stress is the average value of the Mises stress in the element's nodes, while the normal stress is the stress value in the most stressed node. The stress values in the other less strained nodes contributes to lower the average stress value of the element.

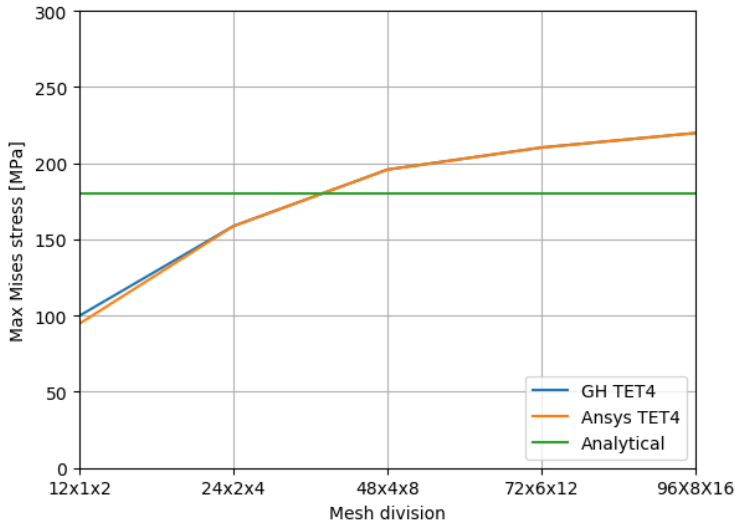


Figure 5.6: Mises stress convergence for TET4 elements.

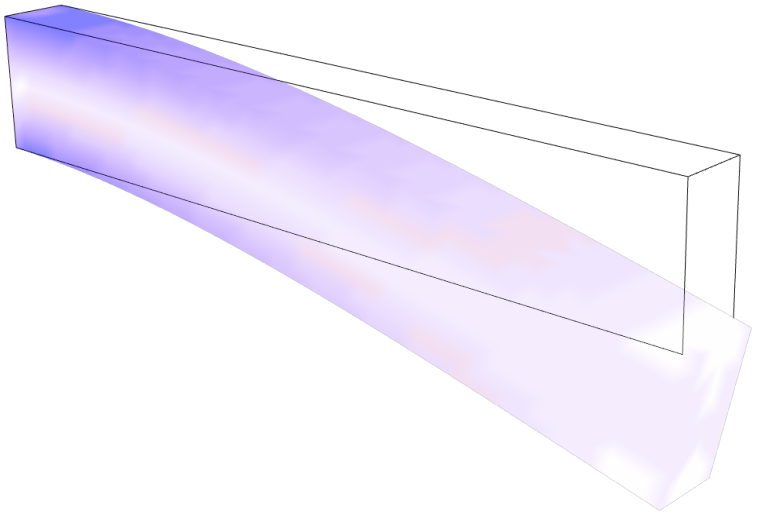
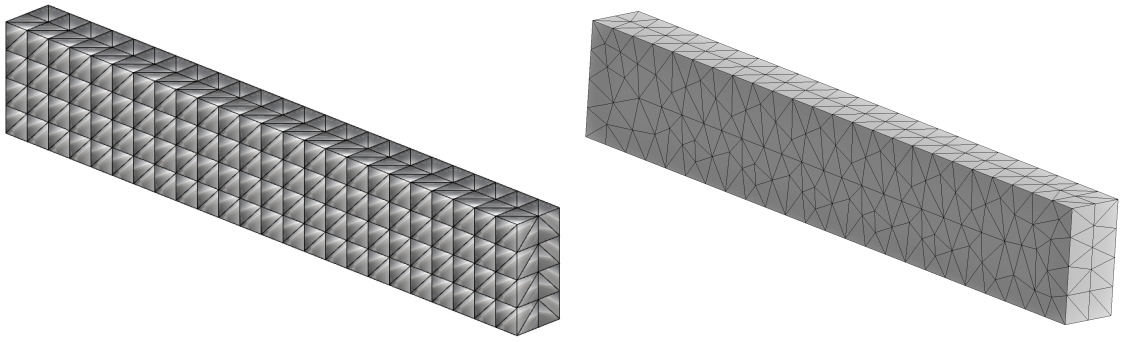


Figure 5.7: Mises stress distribution for the finest mesh with TET4 elements.

5.1.2 TET10 elements

For the TET10 test, the mesh was created in the same manner as for TET4 in Grasshopper, but Ansys used its own meshing engine. This means that there were slight differences in the analysed meshes, but the corresponding meshes in Grasshopper and Ansys consisted of approximately the same number of elements. The difference in meshes between Grasshopper and Ansys is illustrated in Figure 5.8 for the middle mesh division. For TET10 only three different mesh divisions were analysed due to the quickly increasing number of DOFs when the mesh was refined. The results from the analyses are presented in Table 5.4



(a) Meshed beam in Grasshopper.

(b) Meshed beam in Ansys.

Figure 5.8: Difference in meshing between Solid FEM and Ansys for the middle mesh division.

Result parameter	**Calculation type**	**12x1x2**	**24x2x4**	**48x4x8**
w_{max} [mm]	GH TET10	4,143	4,167	4,173
	Ansys TET10	4,150	4,168	4,173
σ_{xx} [Mpa]	GH TET10	188,1	193,7	222,0
	Ansys TET10	188,2	209,5	269,3
Element mises [Mpa]	GH TET10	129,9	154,7	179,8
	Ansys TET10	147,4	161,2	185,9
Computation time [ms]	GH TET10	188	4817	330291
	Ansys TET10	812	1375	3844

Table 5.4: Results from analyses with TET10 elements.

Displacements:
As seen in Table 5.4 the displacement results are very good for all the mesh divisions. In Figure 5.9, the displacement plots of the different mesh divisions can barely be distinguished from each other. This is expected due to the TET10 elements capabilities. As stated in Section 2.4.2, the TET10 element is capable of representing a state of pure bending exactly. Based on Table 5.4 and Figure 5.10, it looks like the largest displacement is converging towards 4,17-4,18, which is 1.5% higher than beam theory solution. TET10 elements also account for shear deformation, so it makes sense that the displacement converges to a value larger than the beam theory solution. Also, Solid FEM provides almost the exact same displacements as Ansys, even though the meshes are

not identical. This strengthens the reliability of Solid FEM.

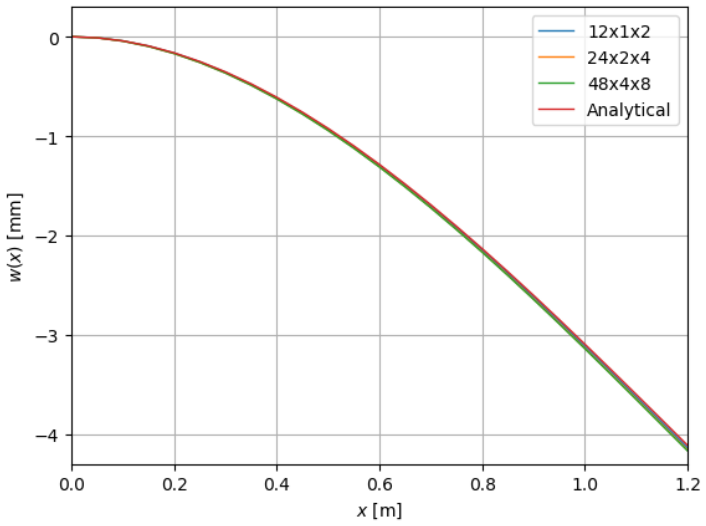


Figure 5.9: Displacements along the length of the beam for the different mesh divisions with TET10 elements and the beam theory solution.

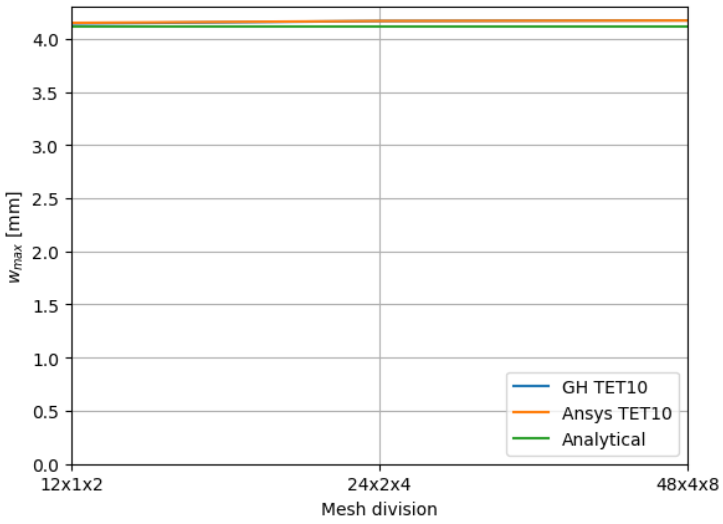


Figure 5.10: Displacement convergence for TET10 elements.

Normal stress, σ_{xx}:
As the displacements, the normal stress is also close to beam theory solution for the coarsest mesh. This is also expected, because TET4 elements can represent a quadratic displacement field and a linear stress field, because normal stress is the derivative of the displacements. By this logic you do not need a very fine mesh in order to obtain reliable results. However when the mesh is refined, the maximum stress increases, and based on Figure 5.11 it even looks like it diverges. This is probably due to stress concentrations at the support points, and may not be realistic. From Figure 5.12 we see that the stresses are larger in the supported corners, and this effect increases when the mesh is refined. Another remark is that Ansys solution is even higher than the Solid FEM solution,

which can be interpreted in two ways. On one side it is a good thing that the Solid FEM solution is closer to the beam theory solution, as this is an analytical solution. On the other hand the stresses should be the same in Solid FEM and Ansys when the displacements are so similar.

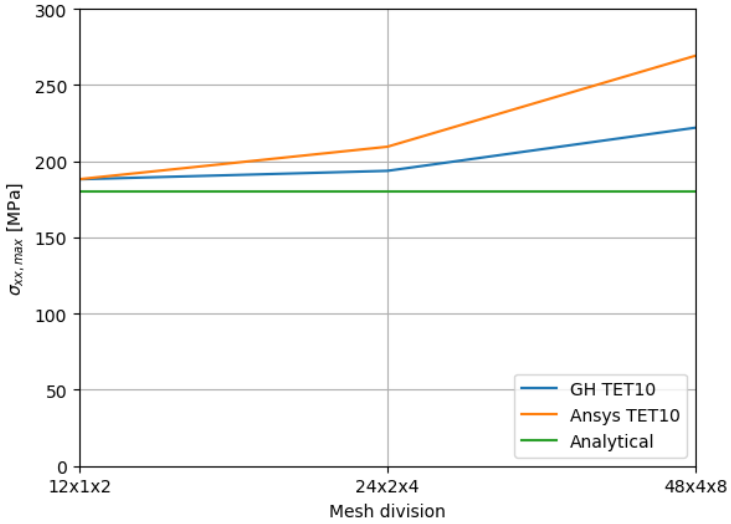


Figure 5.11: Normal stress convergence for TET10 elements.

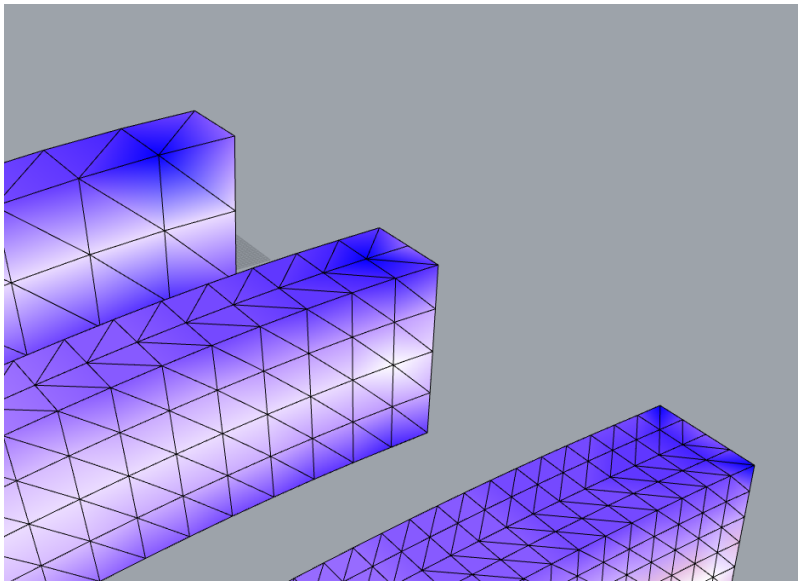


Figure 5.12: Stress concentrations for TET10 meshes.

Element Mises stress:
Table 5.4 and Figure 5.13 shows that the Mises stress is close between Solid FEM and Ansys, and the difference decreases when the mesh is refined. For the coarsest mesh we see that the Mises stress is 72% of beam theory solution, which is a larger deviation than the displacement. As described in Section 5.1.1 this is also expected due to the fact that stress is a derived result from the displacements. It is also partly due to the fact that the element Mises stress is the mean value of the Mises stresses at the element nodes. The lower stressed nodes will contribute to lower the element mean value. When the mesh is refined, the element size reduces, which leads to smaller

differences in the nodal Mises stresses within one element, which increases the element mean value.

Finally, the Mises stress is closer to the beam theory stress as the mesh is refined compared to σ_{xx}. This is a positive result because, as said in Section 5.1.1, Mises stress is more informative than directional stresses when dealing with a three dimensional stress state, which is usually the case when using solid elements in an FEA.

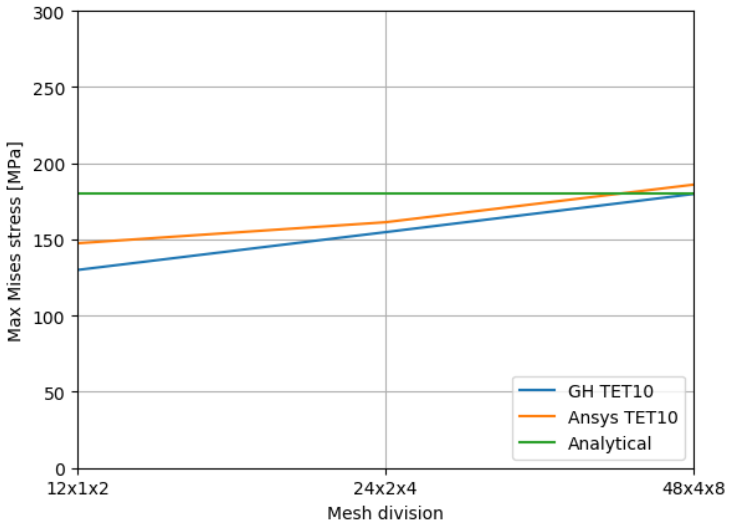


Figure 5.13: Mises stress convergence for TET10 elements.

Computation time:
Table 5.4 shows that the computation time increases even faster for TET10 elements than TET4 elements in Solid FEM. This is because when the number of elements increases, the number of nodes, and thereby also the number of DOFs, increases in a higher rate for TET10 elements than TET4 elements. When comparing the computation time with Ansys, we see the same trend as for TET4 elements: Ansys is more stable and much more efficient when the mesh is refined.

5.2 Case study 2: Analysis of gridshell node

For this case study a gridshell node was analysed in order to investigate how well Solid FEM works with regards to its intentions, namely analyzing complex geometry in an AAD environment. The gridshell node is called POLO-1, and Figure 5.14 illustrates an example where it connects six timber members. It is a typical node design and consists of a cylinder with vertical welded-on plates. The plates are slotted in the timber members, and connected with bolts. This simple design makes it adaptable to other situations. For example a variable number of connected members, or different angles of the members.

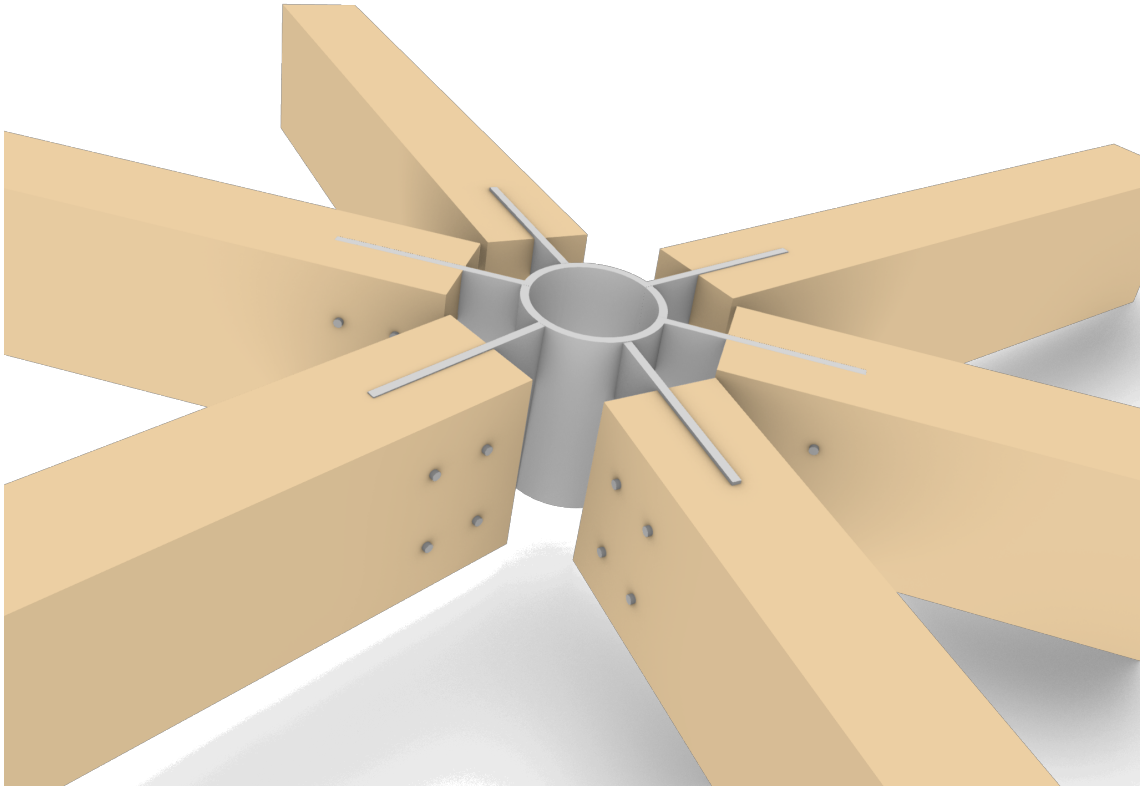


Figure 5.14: POLO-1 gridshell node.

In this case study, the POLO-1 gridshell node was analysed for a specific load case obtained from an article on the British Museum gridshell roof (Brun, 2019). The loads for each member is presented in Table 5.5. Neither the timber members nor the bolts were analysed in this example.

Member	Axial force [kN]	Shear force [kN]	Bending moment [kNm]
1	68.6	3.04	-15.79
2	-16.12	-0.35	-0.99
3	-5.52	-0.6	-2.3
4	62.99	-2.19	-14.49
5	-14.27	0.7	-3.05
6	-7.37	0.08	-0.54

Table 5.5: Loads and bending moments for each timber member.

5.2.1 Workflow

This section will present the workflow and the different steps when analyzing the gridshell node and previewing the results.

Create the geometry:
The geometry was generated with an algorithm in Grasshopper created by Steinar Hillersøy Dyvik for his research on gridshell nodes. The geometry is illustrated without the timber members and the bolts in Figure 5.15, and in order to limit the number of elements in the mesh, the holes for the bolts were removed. If the holes remained then the mesh would need to be divided into a very high number of elements in order to provide a realistic mesh close to the original geometry, and that led to the number of DOFs exceeding the capacity of Solid FEM.

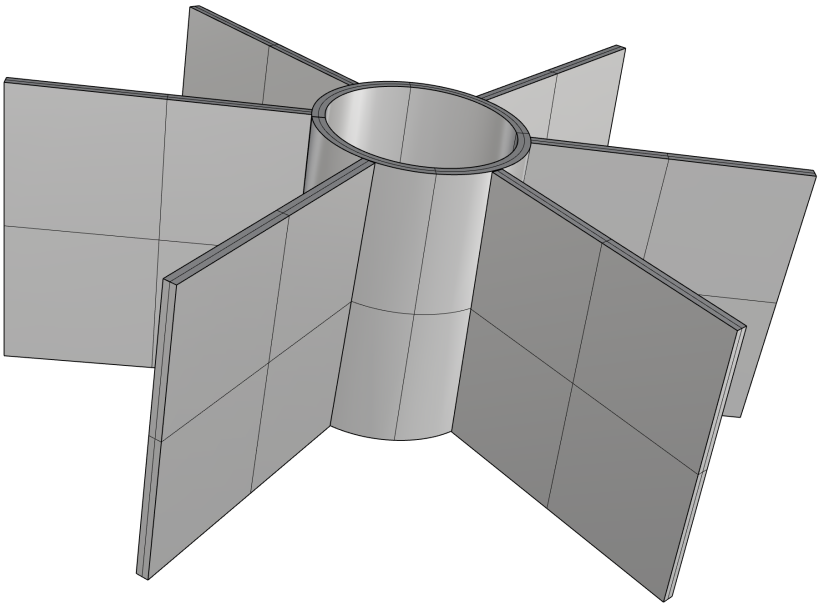


Figure 5.15: Gridshell node geometry.

Meshing the geometry:
For meshing the geometry a component called *Quad Remesh* was used. This accepts either meshes, surfaces or BREPs, and returns a new surface mesh. The user can select a number of settings, for example the target number of faces in the resulting mesh. This component provided the ability to control the number of elements in the mesh. The result from *Quad Remesh* was the input to the *Tetrino* component which returned a list of tetrahedral elements. The process is illustrated in Figure 5.16 and the meshed geometry is illustrated in Figure 5.17.

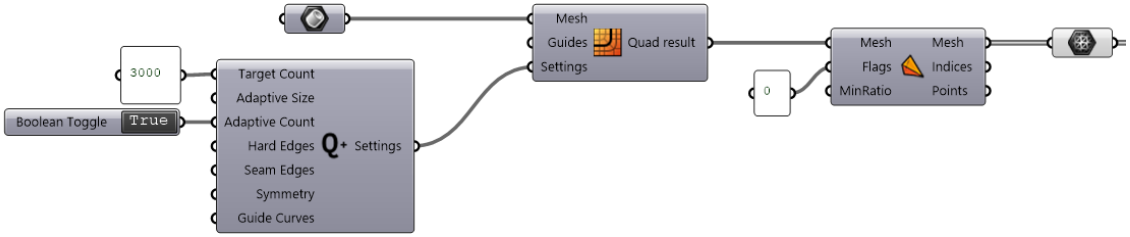
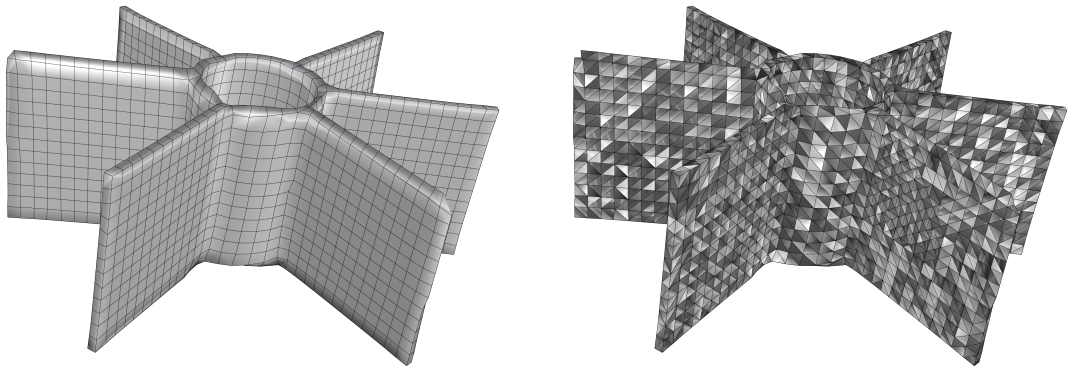


Figure 5.16: Meshing algorithm.



(a) POLO-1 quadrilateral surface mesh. (b) POLO-1 tetrahedral solid mesh.

Figure 5.17: Meshing the gridshell node geometry.

Obtaining load and support points:
Even though the geometry was simplified and the holes were removed, the loads would still be transferred to the node through these points. This was handled by obtaining the vertices in the mesh within a set distance to the original bolt holes. The algorithm is illustrated in Figure 5.18. Due to the fact that this node is in the middle of a gridshell without any fixed support points, in order to perform an analysis without the node flying away due to rigid body motion, the load points in one of the connecting plates were set as fixed supports. In Figure 5.19 the load points are marked with green and the support points are marked with yellow.

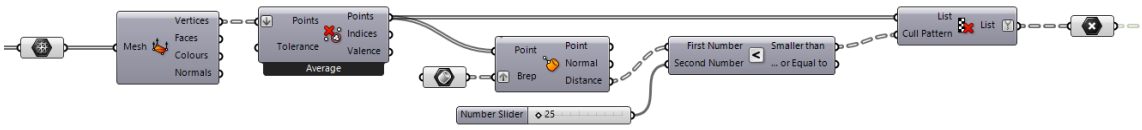


Figure 5.18: Workflow for obtaining the load and support points in the mesh.

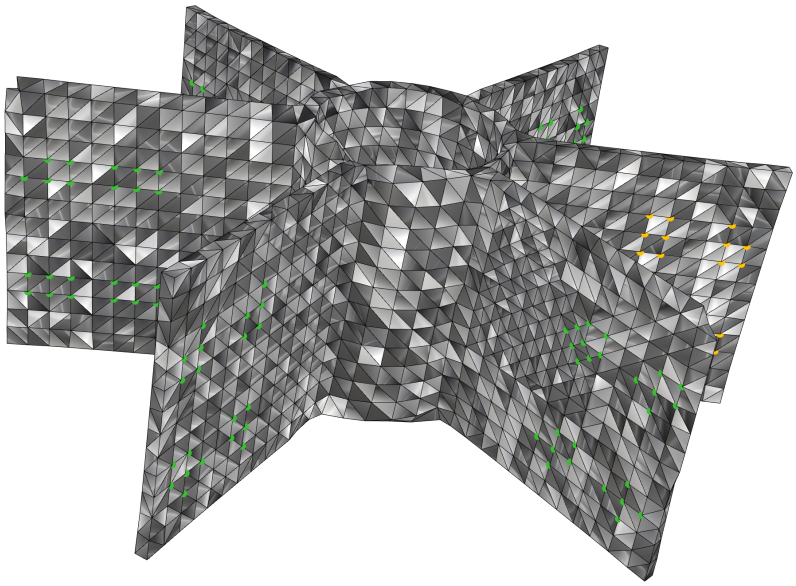


Figure 5.19: Load and support points in the mesh.

Applying loads and boundary conditions:
The main concern when applying loads and boundary conditions was to apply a different load to each of the connecting plates, as well as transforming the bending moment into a force couple. In Grasshopper it was handled by sorting the loading points in six groups, one for each connecting plate. Then the axial and shear force was applied to the correct points with their vector components. Finally, the bending moment was divided by the distance between the bolts, and the force couple was applied to the correct loading points. The algorithm is illustrated in Figure 5.20. This part of the algorithm used clusters. Clusters are very useful in Grasshopper for when performing the same algorithm many times. They allow users to create small (or large) algorithms and set inputs and outputs. The cluster becomes a kind of customized component, hiding the algorithm inside the cluster. This helps keeping the main algorithm clean and clear. In Figure 5.20, the clusters take the loads and bending moments along with the moment arm and load points, and return the load vectors distributed to the load points as well as the load points.

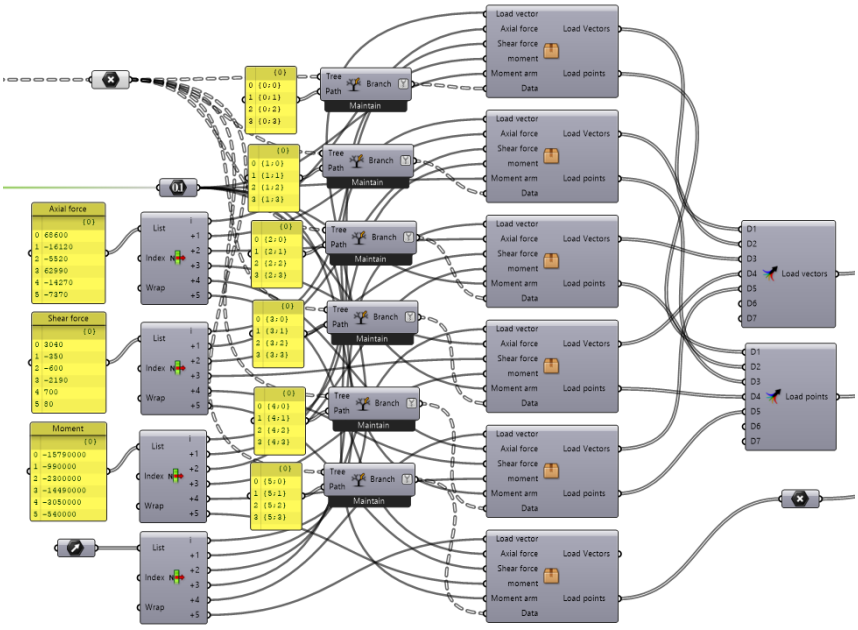


Figure 5.20: Algorithm for obtaining the load vectors.

Analysis and preview:
Finally, the mesh, load points and support points were inputs to the Solid FEM components which performed the analysis, as illustrated in Figure 5.21.

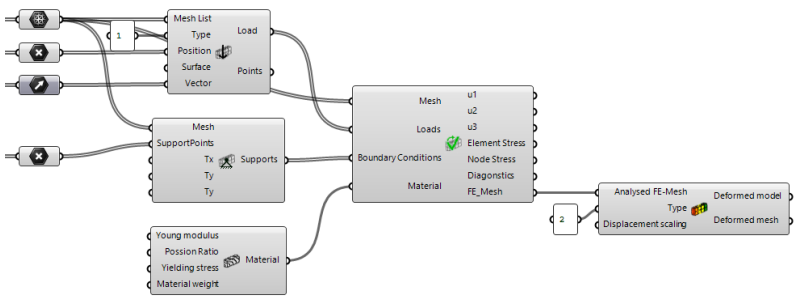


Figure 5.21: FEA workflow.

5.2.2 Results

The gridshell node was analysed in Ansys in order to compare the results from Solid FEM. For this analysis, the final geometry including the bolt holes were imported and meshed using the Ansys meshing engine. Both Solid FEM and Ansys performed the analysis with TET4 elements. The results from the analyses are presented in Table 5.6, and colour maps showing the distribution of stresses and displacements are presented in figures 5.22 to 5.25.

Calculation	n_{els}	n_n	u_{max} **[mm]**	**Mises stress [MPa]**	**Computation time [ms]**
Solid FEM	9267	3264	2,38	414	7028
Ansys	8550	3133	2,04	376	2984

Table 5.6: Results from analysis of POLO-1 gridshell node. n_{els} is the number of elements, n_n is the number of nodes and u_{max} is the largest total displacement.

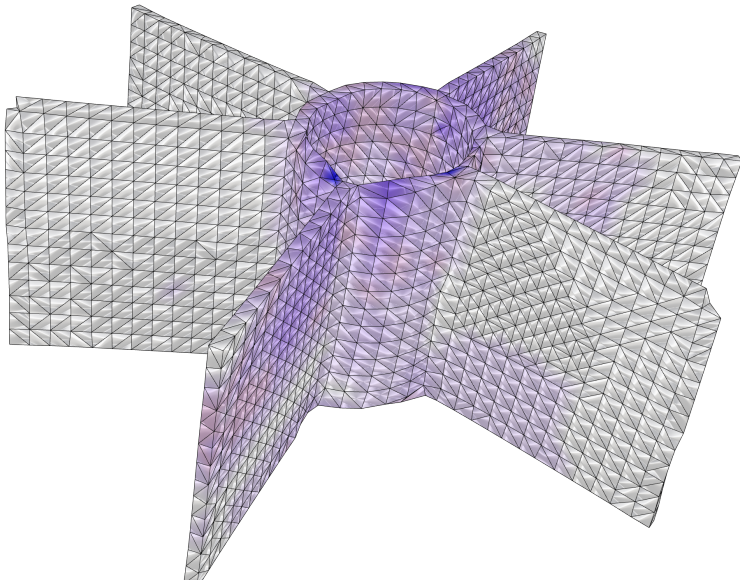


Figure 5.22: Distribution of Mises stress in the gridshell node obtained from Solid FEM.

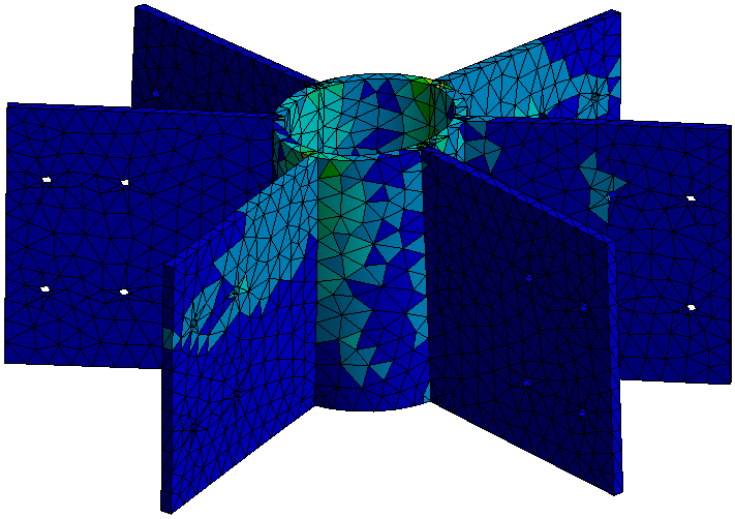


Figure 5.23: Distribution of Mises stress in the gridshell node obtained from Ansys.

From Table 5.6 we see that there are distinct differences between Solid FEM and Ansys. The number of elements and nodes are approximately the same, and so the results should be comparable. The largest displacement was 16% higher than Ansys and the Mises stress was 10% higher. When comparing the colour maps for Mises stress from Solid FEM and Ansys (figures 5.22 and 5.23), we see that the stress concentrations occur at the same places, and the general distribution of stresses looks very similar. The same applies to the displacement maps (figures 5.24 and 5.25).

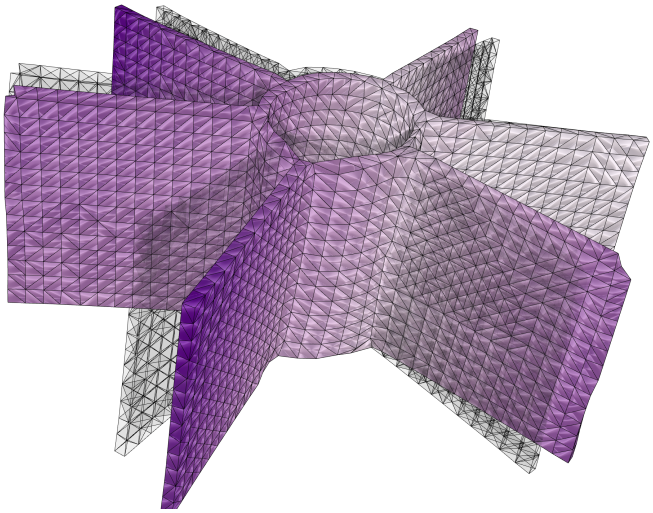


Figure 5.24: Distribution of displacements in the gridshell node obtained from Solid FEM.

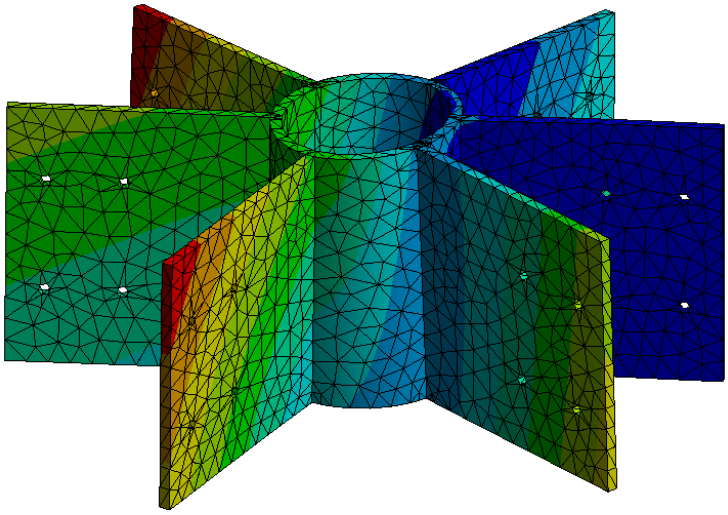


Figure 5.25: Distribution of displacements in the gridshell node obtained from Ansys.

However, in both Solid FEM and Ansys the stress concentrations exceed the yield stress which is a bad sign. We might expect that the stresses increases for a refined mesh. This means that a linear FEA might be insufficient for this geometry, or that the geometry needs changing. Ansys used under half the computation time compared to Solid FEM, but for an analysis of this scale, the additional three seconds are insignificant. The relevance of computation time increases when the mesh is refined, and the number of DOFs increase.

Solid FEM has a maximum capacity with regards to number of DOFs. The capacity depends on the

computer running Solid FEM, and it is caused by the establishing of the stiffness matrix. When the number of DOFs increase, the number of elements in the stiffness matrix increases exponentially. In the code, the stiffness matrix is established as a dense matrix, which means that all elements are established and stored. The capacity depends on the available memory on the computer running Solid FEM. This is a huge drawback because, when analyzing complex geometry, there is usually a need for a very fine mesh, especially in some sections of the geometry.

5.2.3 Discussion

Is Solid FEM easy to use?

One goal of Solid FEM is that it is supposed to be easy to use with limited knowledge about FEM. However it is absolutely a prerequisite that the users of Solid FEM have some experience or basic knowledge about structures and structural mechanics regarding load application and boundary conditions. For example they need to know where and how to apply the load, which can be challenging when dealing with bending moments. Bending moments are often the most contributing factor to strains and stresses, which makes it very important to apply them correctly. To neglect or wrongly apply bending moments can therefore in some cases be a critical mistake. Additionally, the users should be aware of the limitations of the element type. When using TET4 elements, there is a need for a very fine mesh in order to obtain accurate results, due to the fact that the stresses are constant over the span of an element. This increases the number of DOFs and it is likely that the maximum capacity of the solver is reached. TET10 elements are capable of representing a trilinear stress state, which makes them suitable for coarse meshes. However, the number of DOFs increase rapidly when refining the mesh, which means that the capacity of the solver is reached for a coarser mesh than with TET4 elements. Therefore the users should be able to choose what is more important: a fine mesh which can more accurately represent the geometry, or a coarser mesh which can represent a linear stress distribution.

Assuming the user has a good understanding of how to apply the loads, then the challenge when using Solid FEM might be to generate the algorithm in Grasshopper. However, the target users for Solid FEM are architects and structural engineers who already are familiar with Grasshopper and know how to use it. For a competent Grasshopper user, Solid FEM does not introduce any new challenges, other than load application. The most important operations you need to perform when using Solid FEM is meshing the geometry and sorting points. These operations are familiar for most experienced Grasshopper users, and the Tetrino component helps to simplify the meshing. Sorting the points in a suitable data structure is an important part in order to apply the correct loads at the correct points. This might be one of the most difficult parts of parametric design, and requires some grasshopper- or programming experience. But again, the target users of Solid FEM are architects and structural engineers with Grasshopper experience.

Is it accurate enough?

From Table 5.6, we see that there is a slight difference between the solutions in Solid FEM and Ansys for the corresponding meshes. The differences are likely to be caused by the different meshes. The impact of the holes seem to be relatively low as there are not any stress concentrations surrounding them in Ansys, which may indicate that the simplification of the geometry in

Grasshopper was appropriate. For a different load case however, there might occur stress concentrations in such locations, and simplifications should be done carefully.

However, the analysed mesh is relatively coarse, considering that TET4 elements were used. These elements can only represent a state of constant stress, which means that the mesh needs to be refined in such a way, so that stresses are approximately constant over the span of an element. With this in mind, it very well may be that the real stresses are even higher than the results from the analysis. As said in Section 5.2.2, Solid FEM has a maximum capacity when it comes to number of DOFs, and this analysis was performed close to the capacity limit on a standard laptop. This implies that Solid FEM has an upper limit with regards to accuracy as well, because it cannot analyse the mesh when it becomes too large. This is an absolute disadvantage of Solid FEM.

On the other hand, the intentions of Solid FEM is to be used for comparing of different design options and in combination with other Grasshopper tools like optimisation. For this case it might be accurate enough with the current capacity. Even though the results from the analyses deviate from the real solution, an optimisation process could still lead to a more efficient and reliable geometry with either lower stresses or displacements. For comparison of different geometries, as long as the deviation is more or less the same for the different geometries, the analysis results could still provide useful results, and be a basis for decision making in early design phase. For the detail design phase, the reliability of the geometry would still need to be documented in a different FEM software like Ansys or ABAQUS.

Is it fast enough?

As expected, Solid FEM spends more time performing the analysis than Ansys. Solid FEM was written and created by structural engineers, and not software engineers. This is why Solid FEM has great potential to improve the computation time. But even if Solid FEM is slower than Ansys, the user still saves time and work compared to exporting the data from Grasshopper and running the analysis in Ansys. For each time the geometry changes, the loads and supports would need to be reapplied in Ansys, which makes it a pretty cumbersome process. While in Grasshopper, the algorithm automatically obtains the load and support points, which eliminates the need for manual work to be done for each time the geometry changes (assuming the algorithm works as intended).

Regarding optimisation processes, the speed of Solid FEM becomes an important aspect. For example, Galapagos optimisation is an evolutionary algorithm which generates a set number of combinations of input parameters for each round, called a population. Based on the results from each round it generates a new population with new parameter combinations. This process is repeated until ended by the user or the solution converges towards a certain combination. A normal population size is around 50 combinations, which means that for each round the algorithm needs tor run 50 times. And the optimisation process usually takes several rounds to converge. This means that an optimisation process potentially can spend many hours, or even days, depending on the computer and the efficiency of the algorithm. Even though this may take a lot of time, the process can be automated, such that the users can work on other things in the mean time.

6 Discussion and conclusion

6.1 Discussion

This discussion part summarizes the most important parts from the discussions of the case studies in sections 5.1 and 5.2. Additionally, Solid FEM will be discussed in general regarding pros and cons, and whether or not it works as intended.

Regarding the verification of Solid FEM and the comparison with Ansys in case study 1, the results were satisfactory. For the TET4 case, the displacements and the Mises stresses were remarkably close to the results from Ansys, which is a very good and important result. Displacements and Mises stresses are arguably the most important result parameters in a solid FEA. This strengthens the integrity and reliability of Solid FEM. The same applies to the analyses with TET10 elements. In this test, the differences were bigger, but could be explained by the fact that the meshes in Grasshopper and Ansys were not identical. However, with this in mind the results were still very satisfactory, even for the coarse meshes.

A less positive result is regarding the directional normal stress σ_{xx}. The differences between Solid FEM and Ansys were bigger than for displacements and Mises stresses. This was a slightly surprising result, considering that the displacements and Mises stresses were almost identical for both TET4 and TET10 elements. The cause of this might be stress concentrations at the support points, and that Solid FEM and Ansys handles this in different ways. It is more understandable for the TET10 case due to the different meshes, but still a little surprising for the TET4 case. However, in solid FEA, the Mises stresses are more important than the directional stresses because it accounts for all the directional stresses, and in solid FEA, the stress situation is often three dimensional.

The results from Case study 2 didn't harm the results from Case study 1, as the results from Solid FEM and Ansys were relatively similar. In this case there were bigger differences in the meshes which were decisive for the differences in displacements and stresses.

Regarding the use and workflow of Solid FEM, and whether or not it is easy to use, is of course to some degree subjective. But as described in Section 5.2.3, Solid FEM does not introduce any new challenges concerning the workflow inside Grasshopper. The necessary steps for performing an FEA with Solid FEM only requires tools and methods which are very frequently used in algorithmic design in Grasshopper. Furthermore, the target group for Solid FEM are architects and structural engineers with Grasshopper experience. A ton of knowledge about FEM is not a requirement, but the users need to have a certain understanding of structures and correct application of loads. This is potentially the most complicated part regarding the use of Solid FEM, but structural engineers should have the necessary experience and understanding of this. Architects might not have the same understanding of this, but since it is an important part of the analysis it could be necessary to study or to confer with a structural engineer. Additionally, the users should be aware of the impact on accuracy caused by the element type and mesh size. For example, in Case study 1, TET10 elements were clearly the best option, while in Case study 2, even the coarsest mesh resulted in too many DOFs regarding the capacity of the solver when using TET10 elements. Choosing the best suitable element type and mesh size could greatly impact both accuracy and computation

time, and is something the users should know be aware of.

As mentioned in Section 5.2.3, Solid FEM has a maximum capacity when it comes to number of DOFs. The capacity depends on the computer running Solid FEM and its memory. This acts as a limiting factor regarding the accuracy of the analyses. Complex geometries require a fine mesh in order to properly represent the actual geometry and to obtain accurate results from the analyses. In some cases, however, it might not be an issue. Solid FEM is intended to be used in early design phase, where accurate results are not that necessary. The goal at this stage can be either to investigate whether or not a design proposal is a logical/possible solution, or to compare several different design options to serve as a basis for decision making. For these goals Solid FEM can be useful and provide results in order to serve as an assessment basis for choosing a design option, or to alter the existing geometry.

Concerning the use of Solid FEM in combination with other useful Grasshopper tools, like Galapagos optimisation, the results are mixed. In principle, Solid FEM is arranged in such a way so that an optimisation process is possible to perform. The output and post-processing components enable the users to easily obtain the critical result parameters, like stresses or displacements. The optimisation process could then be based on trying to minimize the displacements while still having a utilization below 100%. This is a great feature of Solid FEM. On the other hand the solver efficiency should be improved with the help of a software engineer. As it works now, the solver is slow, which would affect an optimisation process greatly. An optimisation would run the solver numerous times, which means that the computation time plays a bigger part. Regardless, it would still work as it is, but would be a time demanding process with potential to be improved.

Some final remarks/limitations of Solid FEM. The solver is only capable of analyzing geometry made of isotropic materials, like steel or aluminium. This eliminates the ability to analyse timber, concrete and other composite materials. This would also mean that for study case 2, Solid FEM does not say anything about the timber members nor the bolts. In timber structures, the connections are often the critical part when it comes to structural resistance. This may cause a situation where an optimisation process arrives at a result which is best for the node, but worse for the timber member.

6.2 Conclusion

After two case studies it is time to answer the question regarding whether or not Solid FEM can be used in the early design phase for analysing a complex geometry gridshell node. From Case study 1 (section 5.1), it became clear that Solid FEM itself is reliable and sufficiently accurate. The two result parameters, displacements and equivalent Mises stresses, was almost identical in Solid FEM and Ansys for the cantilevered beam, and converged to a value close to beam theory solution, which strengthens the solvers integrity and reliability.

Case study 2 presented an example of how Solid FEM can be used for analyzing a complex geometry gridshell node. The use of Solid FEM in Grasshopper is simple and does not introduce any new programming challenges for users who are familiar with Grasshopper. However, correct application of loads and boundary conditions is a prerequisite for obtaining a reliable result. This is probably the most complicated part regarding an analysis, and any simplifications of the load situation should be done with care. In addition, the choice of element type and mesh size impacts the accuracy and computation time, which the users should be aware of. Regardless, Solid FEM has potential to become a useful tool for architects and structural engineers.

Solid FEM does have an upper limit when it comes to number of DOFs. This depends on the computer running it, and can potentially affect the accuracy of the result due to the mesh being too coarse. In other words, the solver is accurate, the mesh may not be. However, Solid FEM is not intended to be used as documentation of structural reliability, but more as a basis for decision making and a way to efficiently explore different design options while simultaneously obtaining information about their structural performances. Even though the solver is slow, it still saves time compared to exporting the geometry to another FEM software, and in combination with other Grasshopper tools, like Galapagos optimisation, Solid FEM can become a very useful tool in designing complex geometries for gridshell nodes. It is still an early version of Solid FEM, and with further development it is likely that both the speed and limitation of DOFs can be improved significantly.

6.3 Further development of Solid FEM

Even though Solid FEM is working as intended at the moment, there are several things that could be done in order to improve its performance and efficiency. The first thing is the computation time. This solver was developed by structural engineers, and not software engineers. With the help of software engineers, the solver could probably run faster and increase the limit of DOFs so that it can analyse more refined meshes and obtain even more accurate results for complex geometries.

Another thing that can be improved is the load component. At the moment this component only accepts point loads. For the results to be more accurate, the load component could be expanded to accept surface loads and apply these as a consistent load vector. This would mean using the shape functions to distribute the loads to the mesh nodes.

A final thing that could be done is to expand Solid FEM to handle anisotropic materials. At the moment it is limited to isotropic materials like steel and aluminium. If anisotropic materials are accepted, Solid FEM could analyse timber and composite materials.

Bibliography

ANSYS Inc. (n.d.). *Ansys mechanical | structural fea analysis software*. Retrieved 16th May 2022, from https://www.ansys.com/products/structures/ansys-mechanical#tab1-1

Bathe, K.-J. (2014). *Finite element procedures*. Klaus-Jürgen Bathe.

Bell, K. (2014). *An engineering approach to finite element analysis of linear structural mechanics problems*. Fagbokforlaget.

Brun, H. J. K. (2019). British museum grid shell analysis.

McNeel, R. et al. (2022). Rhinoceros 3d, version 7.0. *Robert McNeel & Associates, Seattle, WA*.

Robert McNeel & Associates. (2022a). *Grasshopper - algorithmic modelling for rhino*. Retrieved 29th May 2022, from https://www.grasshopper3d.com/

Robert McNeel & Associates. (2022b). *Rhino - features*. Retrieved 16th May 2022, from https://www.rhino3d.com/features/

Robert McNeel & Associates. (2022c). *Rhinocommon api*. Retrieved 29th May 2022, from https://developer.rhino3d.com/api/RhinoCommon/html/R_Project_RhinoCommon.htm

Username: tomsvilans. (2017). *Tetrino*. Retrieved 6th June 2022, from https://www.food4rhino.com/en/app/tetrino

Weierstrass Institute for Applied Analysis and Stochastics (WIAS). (n.d.). *Tetgen - a quality tetrahedral mesh generator and a 3d delaunay triangulator*. Retrieved 16th May 2022, from https://wias-berlin.de/software/index.jsp?id=TetGen#Documentation

Zienkiewicz, O., Taylor, R. & Zhu, J. (2013). *The finite element method: Its basis and fundamentals*. Butterworth-Heinemann.

Appendix

A Mesh import in Ansys

Figures A1 to A3 illustrates the algorithm for creating the .txt-file for importing the exact same mesh into ansys from Grasshopper. The Grasshopper file is added to the zip file described in section D.

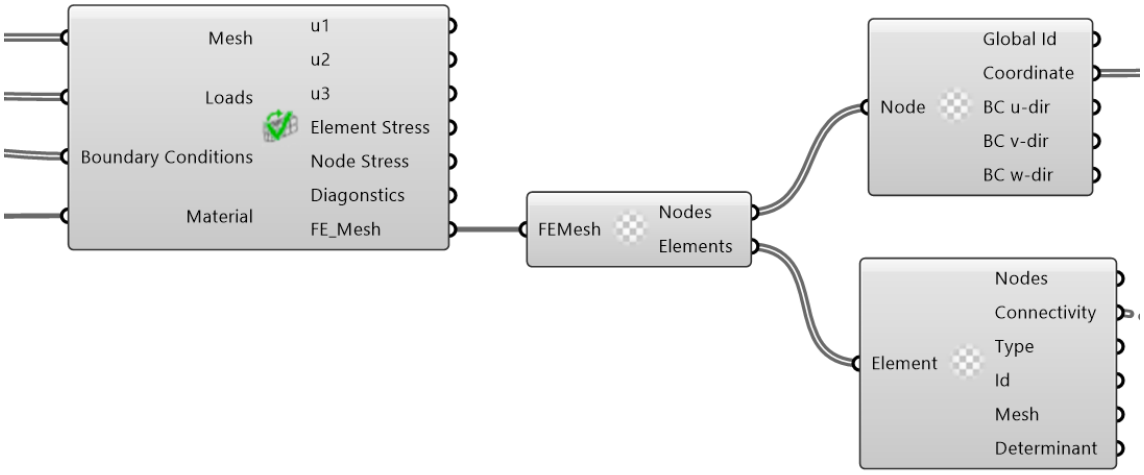


Figure A1: Deconstructing nodes and elements in order to obtain coordinates and connectivities.

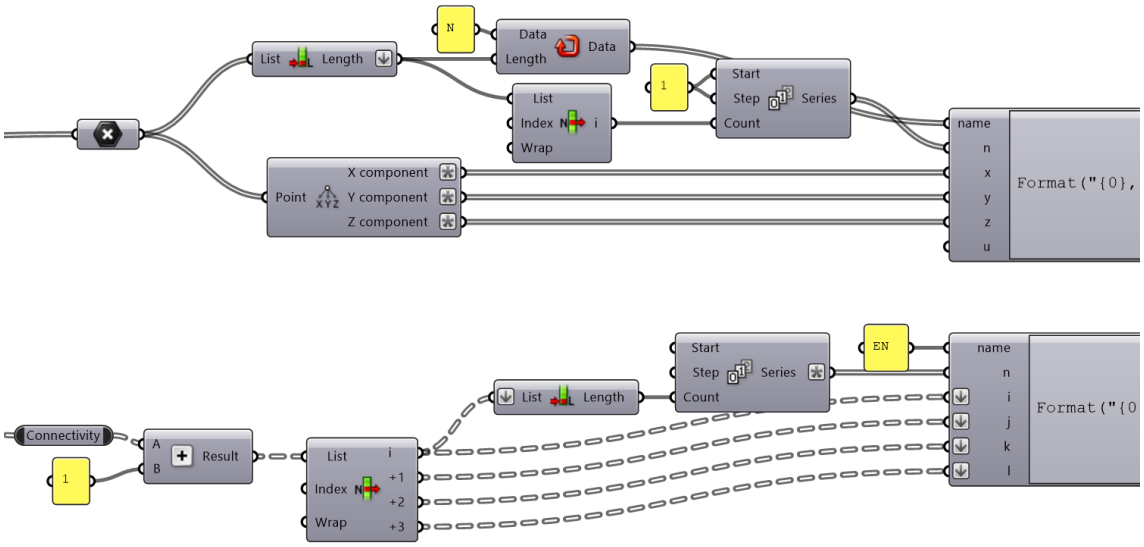


Figure A2: Sorting the coordinates and connectivities.

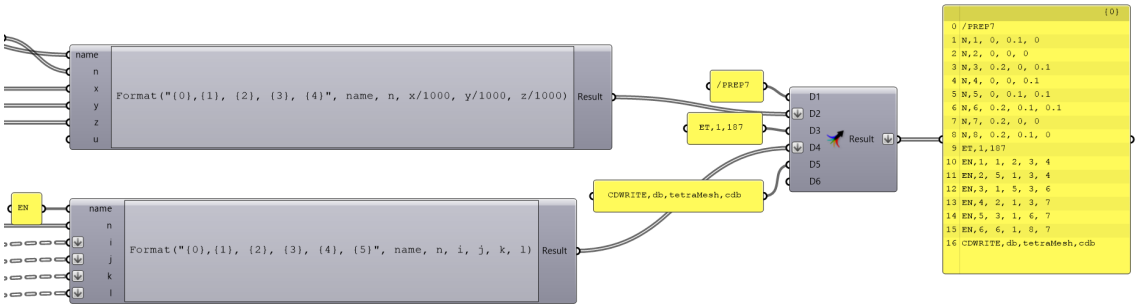


Figure A3: Assembly of the different components to make the final script.

The following script is an example of a .txt-file to import a mesh consisting of eight nodes and six four noded tetrahedral elements:

```
/PREP7
N, 1, 0, 0.1, 0
N, 2, 0, 0, 0
N, 3, 0.2, 0, 0.1
N, 4, 0, 0, 0.1
N, 5, 0, 0.1, 0.1
N, 6, 0.2, 0.1, 0.1
N, 7, 0.2, 0, 0
N, 8, 0.2, 0.1, 0
ET, 1, 187
EN, 1, 1, 2, 3, 4
EN, 2, 5, 1, 3, 4
EN, 3, 1, 5, 3, 6
EN, 4, 2, 1, 3, 7
EN, 5, 3, 1, 6, 7
EN, 6, 6, 1, 8, 7
CDWRITE,db,tetraMesh,cdb
```

Line 2-9 lists the nodes. The N stands for node, the next number is the node ID, the next three numbers are the coordinates.

Line 10 specifies the element type (ET). The element code 187 is the four noded tetrahedral element.

Line 11-16 lists the elements. EN stands for element number, the next number specifies the element ID, the next four numbers specify which nodes the element is made up of (connectivity).

B Python script: convergence plots

```
import matplotlib.pyplot as plt
import pandas as pd


### Import data from excel file ###
data1 = pd.read_excel(r'C:filePath', sheet_name='GH TET4')
GHTET4 = pd.DataFrame(data1, columns=['Max displacement', 'sigma_xx', ...
        ...'Mises element', 'Analytical w', 'Analytical sigma_xx'])
data2 = pd.read_excel(r'C:filePath', sheet_name='GH TET10')
GHTET10 = pd.DataFrame(data2, columns=['Max displacement', 'sigma_xx', ...
        ...'Mises element', 'Analytical w', 'Analytical sigma_xx'])
data3 = pd.read_excel(r'C:filePath', sheet_name='Ansys TET4')
AnsysTET4 = pd.DataFrame(data3, columns=['Max displacement', 'sigma_xx', ...
        ...'Mises element', 'Analytical w', 'Analytical sigma_xx'])
data4 = pd.read_excel(r'C:filePath', sheet_name='Ansys TET10')
AnsysTET10 = pd.DataFrame(data4, columns=['Max displacement', 'sigma_xx', ...
        ...'Mises element', 'Analytical w', 'Analytical sigma_xx'])

dataType = 'sigma_xx'    # "Max displacement", "Mises stress" or "sigma_x".
elementType = 'TET4'     # Set the element type. "TET4" or "TET10".

### Lists of all the data ####
dataListTET4 = [GHTET4, AnsysTET4]
dataListTET10 = [GHTET10, AnsysTET10]

# Mesh divisions
x1 = ["12x1x2", "24x2x4", "48x4x8", "72x6x12", "96X8X16"]    # TET4
x2 = ["12x1x2", "24x2x4", "48x4x8"]                          # TET10

# Beam theory displacement
w_analytical1 = [4.11, 4.11, 4.11, 4.11, 4.11]               # TET4
w_analytical2 = [4.11, 4.11, 4.11]                           # TET10

# Beam theory stress
sigma_xx_analytical1 = [180, 180, 180, 180, 180]             # TET4
sigma_xx_analytical2 = [180, 180, 180]                       # TET10

## Labels ##
labelsTET4 = ["GH TET4", "Ansys TET4"]
labelsTET10 = ["GH TET10", "Ansys TET10"]
```

```
### Get the correct data and labels based on element type ###
dataList = []
labels = []
x = []
w_analytical = []
sigma_xx_analytical = []
if elementType == 'TET4':
    dataList = dataListTET4
    labels = labelsTET4
    x = x1
    w_analytical = w_analytical1
    sigma_xx_analytical = sigma_xx_analytical1
elif elementType == 'TET10':
    dataList = dataListTET10
    labels = labelsTET10
    x = x2
    w_analytical = w_analytical2
    sigma_xx_analytical = sigma_xx_analytical2

### Plotting ###
plt.figure(figsize=[6.4, 4.8])
for type, label in zip(dataList, labels):    # Loop the data list and labels
    y = type[dataType]                       # Obtain correct data type

    ## Set x-limit
    if elementType == 'TET10':
        plt.xlim(0, 2)
    else:
        plt.xlim(0, 4)

    plt.plot(x, y, label=label)     # Plot

### Set y-limit and plot beam theory solution based on data type ###
if dataType == 'Max displacement':
    plt.ylim(0, 4.3)
    plt.plot(x, w_analytical, label='Analytical')

elif dataType == 'sigma_xx' or dataType == 'Mises element':
    plt.ylim(0, 300)
    plt.plot(x, sigma_xx_analytical, label='Analytical')

### Set labels and legends ###
plt.xlabel('Mesh division')
plt.ylabel(dataType)
```

```
plt.legend(loc='lower right')
plt.grid()
plt.show()
```

C Python script: displacement plots

```
# This is the script used to plot the displacements along the length
# of the beam for Case study 1.

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

### Beam data ###
E = 210000
P = 100000
L = 1200
Iy = 6.6666667*10**7
W = 20/3*10**5

### Import data from excel ###
data = pd.read_excel(r'C:filepath', sheet_name='GH TET4 Plot')
GHTET4Data = pd.DataFrame(data, columns=['Disp 1', 'Disp 2', 'Disp 3',...
            ...'Disp 4', 'Disp 5'])
data = pd.read_excel(r'C:filePath', sheet_name='GH TET10 Plot')
GHTET10Data = pd.DataFrame(data, columns=['Disp 1', 'Disp 2', 'Disp 3'])

elementType = 'TET4'    # Set the element type. 'TET4' or 'TET10'

### Beam theory solution ###
x = np.linspace(0, 1.2, 25)
w_analytical = -P*(x*1000)**2/(6*E*Iy)*(3*L-x*1000)

### Obtain correct data based on element type ###
disp = []
if elementType == 'TET10':
    disp = [GHTET10Data['Disp 1'], GHTET10Data['Disp 2'],...
            ...GHTET10Data['Disp 3'], w_analytical]
    labels = ["12x1x2", "24x2x4", "48x4x8", "Analytical"]
elif elementType == 'TET4':
    disp = [GHTET4Data['Disp 1'], GHTET4Data['Disp 2'], GHTET4Data['Disp 3'],...
            ...GHTET4Data['Disp 4'], GHTET4Data['Disp 5'], w_analytical]
    labels = ["12x1x2", "24x2x4", "48x4x8", "72x6x12", "96X8X16", "Analytical"]


### Plot ###
plt.figure(figsize=[6.4, 4.8])
plt.xlim(0, 1.2)
```

```
plt.ylim(-4.3, 0.3)
for y, label in zip(disp, labels):
    if label == "Analytical":
        plt.plot(x, y, label=label, linewidth=2)
    else:
        plt.plot(x, y, label=label)
plt.xlabel('$x$ [m]')
plt.ylabel('$w(x)$ [mm]')
plt.grid()
plt.legend(loc='upper right')
plt.show()
```

D Additional files

The plug-in, Solid FEM, can be found in its entirety in the branch **develop_tetraMesh** on GitHub: https://github.com/marcinluczkowski/SolidFEM/tree/develop_tetraMesh

Additionally, the following files are uploaded in the zip-file *Appendix*:

File name	**Description**
CantileverTestResults.xlsx	Excel file containing the results from Case study 1.
Create_meshfile_for_Ansys_Illustration.gh	Grasshopper file showing how to obtain the .txt file for importing mesh into Ansys from Grasshopper.
SolidFEM_POLO-1_Analysis.gh	Grasshopper file used to analyse the gridshell node in Case study 2.
SolidFEM_Verification_Cantilever.gh	Grasshopper file used to analyse the cantilever beam in Case study 1.
SolidFEM_Workflow_Illustration.gh	The Grasshopper file used to illustrate the workflow of Solid FEM in section 4.

Table D.1: List of additional files used for this thesis.

NTNU

Norwegian University of
Science and Technology