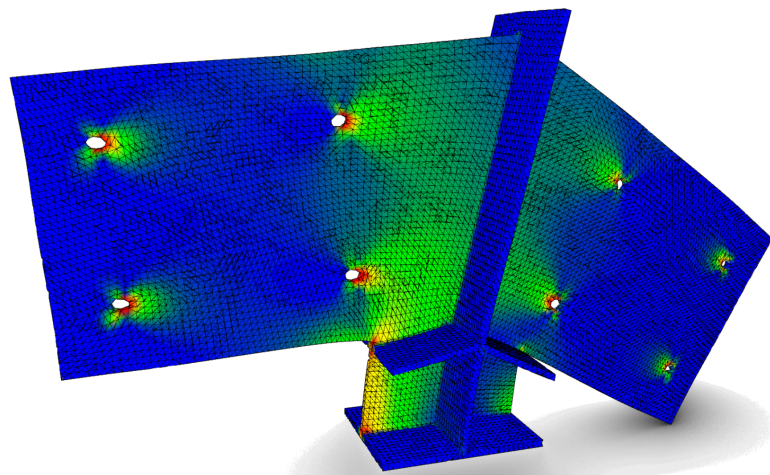


Vegard Øyre
Lars Olav S. Toppe

Integrating Finite Element Method with Solid Elements in Algorithms-Aided Design for the Conceptual Phase of Projects

Master's thesis in Civil and Environmental Engineering
Supervisor: Associate Professor Marcin Luczkowski
Co-supervisor: PhD Candidate Sverre Magnus Haakonsen and
Research Scientist August Johansson
June 2023



Vegard Øyre
Lars Olav S. Toppe

Integrating Finite Element Method with Solid Elements in Algorithms- Aided Design for the Conceptual Phase of Projects

Master's thesis in Civil and Environmental Engineering
Supervisor: Associate Professor Marcin Luczkowski
Co-supervisor: PhD Candidate Sverre Magnus Haakonsen and
Research Scientist August Johansson
June 2023

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





MASTER THESIS 2023

| | | |
|---|---------------------|----------------------------|
| SUBJECT AREA: Structural Engineering | DATE: 09/06/2023 | NO. OF PAGES: viii + 90 |
|---|---------------------|----------------------------|

TITLE:

Integrating Finite Element Method with Solid Elements in Algorithms-Aided Design for the Conceptual Phase of Projects

Integrering av Elementmetode med Volumelementer i Algorithms-Aided Design for Konseptfasen av Prosjekter

BY:

Lars Olav Toppe
Vegard Øyre



SUMMARY:

This thesis investigates the potential of using the Finite Element Method (FEM) with solid elements in the conceptual project phase. Through the Algorithm-Aided Design (AAD) environment Grasshopper from the Computer-Aided Design application Rhinoceros, the opportunities of achieving results comparable with the commercial FEM software Abaqus are explored.

Two main methods for solving structural FEM problems are developed in the Grasshopper environment. One method solves the problem by matrix algebra, where two different FEM plugins are developed. The first plugin is developed entirely by the authors of this thesis, while the other plugin named FERret is already constructed, and thus developed further. The expansion of the second plugin involves the implementation of orthotropic materials, as well as facilitating interaction between parts and materials in one single model. For the second method, which utilizes Partial Differential Equations (PDEs), a plugin is developed to examine if there lies a larger potential for this approach to the problem. The plugin utilizes already existing code inside the FEniCSx platform but is to be considered new in the Grasshopper environment.

Both developments of FERret and FEniCSx reveal promising results. FERret can solve Finite Element problems that combine interacting components of both isotropic and orthotropic materials. On the other hand, FEniCSx demonstrates remarkable potential with a commendably low time consumption while maintaining a high level of accuracy. For both plugins, the meshing procedure of the geometry stands out as the most challenging problem for efficiently achieving satisfactory results.

Generally, utilizing FEM within an AAD environment during a project's conceptual phase can significantly enhance the efficiency and accuracy of the process. FEM can aid in decision-making inside an environment where geometries and properties are easily and rapidly modified while providing satisfactory results for complex problems in real-time.

RESPONSIBLE TEACHER: Associate Professor Marcin Luczkowski.

SUPERVISORS: Associate Professor Marcin Luczkowski, PhD Candidate Sverre Magnus Haakonsen and Research Scientist August Johansson.

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

Preface

This thesis marks the completion of our Master of Science degree in Structural Engineering at the Department of Structural Engineering at the Norwegian University of Science and Technology.

Our thesis is motivated by the shared interest in enhancing collaboration between architects and engineers, aiming to make them work closer in the design process. Through the Minor program in Architecture at NTNU, we have both gained a deeper understanding of the importance of creating structures that unite structural functionality and visual form. We hope this thesis can contribute to designing secure and meaningful constructions in the future.

Acknowledgments

Our supervisor, Marcin Luczkowski, and co-supervisors, Sverre Magnus Haakonsen and August Johannson, deserve our sincere appreciation for their invaluable guidance throughout the completion of our master's thesis. We extend our gratitude to the Minor program team and the Conceptual Structural Design Group at NTNU for their inspiring lectures and courses, which have significantly contributed to our understanding and passion for the subject.

Lastly, we would like to express our appreciation to the teachers, fellow students, friends, and family for engaging in meaningful discussions and providing motivation throughout our years at NTNU. The exchange of ideas and perspectives within the academic community has played a crucial role in our personal and intellectual growth.

Trondheim
June 9, 2023



Lars Olav S. Toppe



Vegard Øyre

Abstract

This thesis investigates the potential of using the Finite Element Method (FEM) with solid elements in the conceptual project phase. Through the Algorithms-Aided Design (AAD) Environment Grasshopper from the Computer-Aided Design application Rhinoceros, the opportunities of achieving results comparable with the commercial FEM software Abaqus are explored. The thesis contains six case studies targeted to resolve the following research question.

What are the opportunities and challenges associated with implementing Finite Element Method analysis using solid elements during the conceptual phase of projects, through the use of an Algorithms-Aided Design environment?

To answer the research question, two main methods of solving structural FEM problems are developed and applied in the six case studies. One method solves the problem by matrix algebra, where two different FEM plugins are developed for the Grasshopper environment. The first plugin is developed entirely by the authors of this thesis, while the other plugin, FERret, is already constructed and thus developed further. The expansion of the second plugin involves the implementation of orthotropic materials, as well as facilitating interaction between parts and materials in one single model.

For the second method, which utilizes partial differential equations to solve the problem, another plugin is developed to investigate if there lies a larger potential for this approach to the problem. The plugin utilizes existing code inside the FEniCSx platform but is considered new in the Grasshopper Environment.

Both developments of FERret and FEniCSx reveal promising results. FERret can solve Finite Element problems that combine interacting components of both isotropic and orthotropic materials. On the other hand, FEniCSx demonstrates remarkable potential with a commendably low time consumption while maintaining high accuracy. For both plugins, the meshing procedure of the geometry stands out as the most challenging problem for efficiently achieving satisfactory results.

Generally, utilizing FEM within an AAD environment during a project's conceptual phase can significantly enhance the efficiency and accuracy of the process. FEM can aid decision-making in an environment where geometries and properties are easily and rapidly modified while providing satisfactory results for complex problems in real-time.

Sammendrag

Denne masteroppgaven utforsker potensialet i å bruke Elementmetoden (FEM) med volumelementer under den konseptuelle fasen i et prosjekt. Ved hjelp av Grasshopper, et Algorithms-Aided Design (AAD)-verktøy som en integrert del av det dataassisterte konstruksjonsprogrammet Rhinoceros, undersøkes mulighetene for å oppnå resultater som er sammenlignbare med den kommersielle FEM-programvaren Abaqus. Masteroppgaven inneholder seks case-studier med målsetting om å besvare følgende problemstilling.

Hva er mulighetene og utfordringene knyttet til implementering av Elementanalyse med volumelementer under konseptfasen i prosjekter, gjennom bruk av et Algorithms-Aided Design-miljø?

For å besvare problemstillingen er to hovedmetoder for å løse konstruksjonsrelaterte FEM-problem utviklet og brukt i de seks case-studiene. I den ene metoden baserer løsningen seg på matriseoperasjoner, der to programvareutvidelser er utviklet for Grasshopper-miljøet. Den første utvidelsen er utviklet av oppgavens forfattere i sin helhet, mens den andre utvidelsen, kalt FERret, er allerede etablert og derfor kun videreutviklet. Denne videreutviklingen omhandler implementering av ortotrope materialer, samt tilrettelegging for samspill mellom forskjellige deler og materialer i en og samme modell.

Den andre metoden bruker partielle differensialligninger for å løse problemet. En programvareutvidelse er utviklet for å kartlegge potensialet for denne måten å håndtere problemet på. Denne utvidelsen bruker allerede eksisterende programvarekode innen FEniCSx-plattformen, men betraktes som ny innen Grasshopper-miljøet.

Videreutviklingen av både FERret og FEniCSx viser lovende resultater. FERret er i stand til å løse FEM-problemer som kombinerer komponenter bestående av både isotrope og ortotrope materialer. På den andre siden viser FEniCSx et bemerkelsesverdig potensial med imponerende lav tidsbruk til et høyt nøyaktighetsnivå. For begge utvidelsene er prosedyren for å lage et godt mesh av geometrien den største utfordringen for å oppnå et tilfredsstillende resultat på en effektiv måte.

Bruken av FEM i et AAD-miljø under konseptfasen av et prosjekt kan utvilsomt øke effektiviteten og nøyaktigheten av prosessen. FEM kan bistå i avgjørelser gjennom et miljø der geometri og egenskaper enkelt lar seg endre samtidig som man oppnår tilfredsstillende resultater for komplekse problem i sanntid.

Glossary

| | |
|--------------|--|
| AAD | Algorithms-Aided Design, is the use of algorithms to create, modify, analyze, and optimize a design. |
| BC | Boundary Conditions are necessary constraints for solving the structural problem. |
| BREP | Boundary REPresentations represents geometry by defining the boundaries of its volume. |
| CAD | Computer-Aided Design uses computer software to create a design. |
| DOFs | Degrees Of Freedom are the independent variables that define the position and possible motion of a system. |
| FEA | Finite Element Analysis is an analysis using the Finite Element Method to obtain the results of a problem. |
| FEM | Finite Element Method is a method of dividing a structure into smaller parts and solving differential equations numerically to obtain the solution to the problem. |
| GH | Grasshopper is a visual programming language inside the Rhinoceros CAD software. |
| HEX8 | Hexehedral element with 8 nodes. |
| HEX20 | Hexehedral element with 20 nodes. |
| IDE | Integrated Development Environment is an application that provides help to the programmer to create software code efficiently. |
| PVW | Principle of Virtual Work is a method that utilizes hypothetical displacements to obtain the behavior of a structure. |
| TET4 | Tetrahedral element with 4 nodes. |
| TET10 | Tetrahedral element with 10 nodes. |

Table of Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Background | 1 |
| 1.2 | Research Question | 3 |
| 1.3 | Structure of the Thesis | 3 |
| 2 | Theory | 4 |
| 2.1 | Finite Element Method | 4 |
| 3 | Software | 15 |
| 3.1 | Rhino Grasshopper | 15 |
| 3.2 | Visual Studio | 15 |
| 3.3 | Visual Studio Code | 16 |
| 3.4 | Abaqus/CAE | 16 |
| 3.5 | Ubuntu | 16 |
| 4 | Methods | 17 |
| 4.1 | Plugin for 8-node Hexahedral Solid Elements | 17 |
| 4.2 | Plugin for Hexahedral and Tetrahedral Solid Elements | 22 |
| 4.3 | Development towards Orthotropic Materials | 28 |
| 4.4 | Development towards Combining Materials | 29 |
| 4.5 | Development of FEniCSx | 30 |
| 5 | Case Studies | 36 |
| 5.1 | Case Study 1: Verifying the PreFERret Plugin | 36 |
| 5.2 | Case Study 2: Verifying the Development towards Orthotropic Materials | 41 |
| 5.3 | Case Study 3: Verifying the Development towards Combining Materials | 48 |
| 5.4 | Case Study 4: Analyzing Timber Elements from 3D-scanned House with FERret | 56 |
| 5.5 | Case Study 5: Verifying the Implementation of FEniCSx in Grasshopper | 66 |
| 5.6 | Case Study 6: Analyzing Complex Geometry with FEniCSx | 73 |

| | |
|----------------------------------|-----------|
| 6 Discussion | 82 |
| 6.1 FERret and FEniCSx | 82 |
| 6.2 Meshing Challenges | 85 |
| 7 Conclusion | 87 |
| 8 Further work | 88 |
| Bibliography | 89 |
| Appendix | 90 |
| A GitHub Repositories | 90 |
| B Videos | 90 |

1 Introduction

This thesis is written for the Conceptual Structural Design Group at NTNU. CSDG's research is based on the collaboration and working methods between architects and structural engineers during the design process. The group highlights that conceptual structural design should do more than just safely carry loads. The structures should also be meaningful, beautiful, and engaging. The objective of this master thesis is to develop solutions that enhance the level of detail in the structural design during the conceptual phase. This could help create structures that integrate structural functionality and visual form into a meaningful, engaging whole.

1.1 Background

The traditional approach to project resolution involves the architect being responsible for design while the engineer is assigned calculations. The engineer's role is to ensure the feasibility of the architect's output and suggest modifications if necessary. Throughout the project phase, the architect sends design solutions to be verified by the engineer. In return, the engineer performs calculations and provides updated information. This iterative exchange of information continues during the project. Figure 1.1.1 illustrates the typical relationship between cost and design freedom during the design process. In the early stages, minimal effort has been invested in the project. Consequently, the costs of making changes are minimal, allowing for greater design freedom. However, as the project progresses, the cost of making changes increases, reducing the possibility of making modifications. Such a design methodology leads to projects where time and budget constraints limit the architect's and the engineer's ability to create solutions that unify architectural and structural design well.

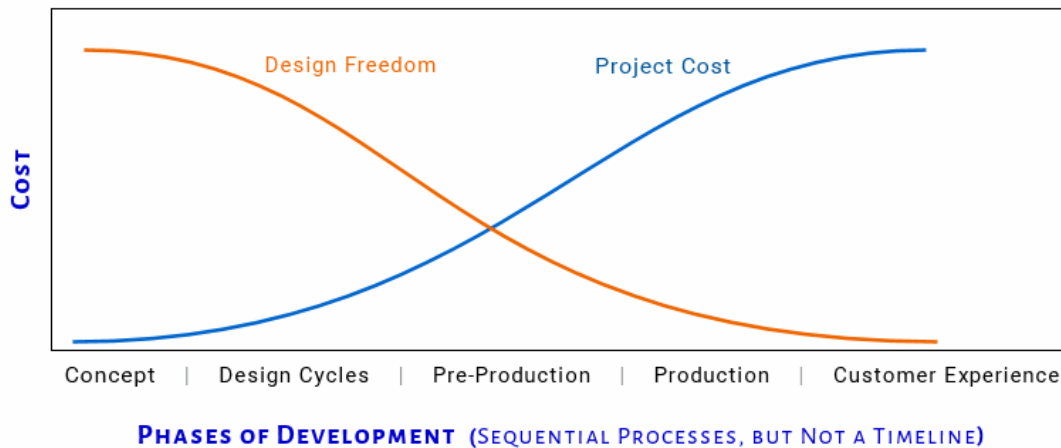


Figure 1.1.1: Relationship between cost and design freedom in a project (Synthesis, 2021).

In typical projects, the willingness to explore new ideas and make significant changes is often greater when they can be easily implemented. One solution to address this is the creation of a digital prototype. Two possible production methods for generating digital prototypes are Computer-Aided Design (CAD) and Algorithms-Aided Design (AAD). These methods are described in the following.

Computer-Aided Design

CAD is software specifically designed for drawing and documentation in both 2D and 3D design, incorporating photorealistic rendering techniques that enable the simulation of real-world functionality. This significantly enhances documentation accuracy, automates the processes, facilitates rapid modifications, handles complex geometries, and promotes easier collaboration with partners (Suzuki, 2021). In the early stages, quickly adjustable models can be produced, allowing for rapid generation of multiple design proposals. Consequently, architects and engineers can collaboratively work on the design, resulting in solutions that integrate architecture and construction better. However, a challenge with this method is that design solutions need to be transferred to Finite Element Analysis (FEA) software to conduct calculations. Constructing a FEM model can be time-consuming, making the design process somewhat cumbersome when utilizing CAD.

Algorithms-Aided Design

AAD has gained popularity as a method for producing digital prototypes over the last few years. With AAD, the traditional graphical representation of a design is transformed into an algorithm that can generate the design. Instead of manually drawing lines and objects fixed in their assigned positions, they can be created and modified in real-time by adjusting the parameters of the algorithm's input. This modeling approach enables the rapid generation of numerous design proposals. Moreover, solutions must not necessarily be finalized, as they can be easily modified later by adjusting the input parameters (Tedeschi, 2014). By transitioning from a manual design approach to an algorithmic-based approach, it is natural to consider incorporating Finite Element Analysis (FEA) into the design process. Shifting the entire design methodology to an AAD environment can enhance the efficiency of the design while also opening possibilities for improved solutions.

For AAD-based FEA, existing software is already available, primarily developed based on beam and shell elements. Beam elements are suitable for modeling columns, beams, and trusses, while shell elements work well for planar and curved surfaces such as walls, slabs, and roof constructions. However, beam and shell elements have limitations regarding highly complex geometries. If a connection or a 3D-scanned object with irregularities on its surface should be modeled correctly, solid elements are required. The challenge with solid elements is that they can be computationally costly, especially as the mesh becomes finer or when higher-order elements are used. In an AAD environment, it is crucial for information and calculations to be updated in real-time. Developing FEA software within the AAD environment or transferring data from the AAD environment to an FEA program and then back to the AAD environment are possible solutions for implementing solid elements.

1.2 Research Question

This master's thesis aims to answer the following research question:

What are the opportunities and challenges associated with implementing Finite Element Method analysis using solid elements during the conceptual phase of projects, through the use of an Algorithms-Aided Design environment?

By exploring this research question, we aim to find a solution that enhances efficiency and workflow during the conceptual phase of a project. By examining different approaches for integrating FEA and AAD environments, we seek to develop a program capable of performing detailed FEM modeling with various material properties in real-time. This advancement would improve the collaboration between architects and engineers during the design phase, allowing both parties to have a more significant stake in decision-making processes. Additionally, it would enable more accurate FEM analyses of 3D-scanned models and provide correct material behavior in the connections of a structure. Consequently, reclaimed material documentation would improve, and the possibility of building more complex geometries will be more achievable.

To address the research question, a FEM plugin for solid elements needs to be developed within an AAD system such as Grasshopper (GH). To handle various geometries, the program should be capable of managing both hexahedral and tetrahedral elements with shape functions of different orders. To account for different materials and their interactions, the program must incorporate various material properties as input, and the individual building components should be distinguishable in the analysis. Exploring different solvers, such as direct and multigrid solvers, is necessary to determine which approach can provide real-time analysis results.

1.3 Structure of the Thesis

This master's thesis is based on case studies that explore the research question. The thesis begins with an introduction to relevant theory in Finite Element Method (FEM). The subsequent section describes the various programs used to develop the solver. Section 4 offers a detailed explanation of the structure and architecture of the software created by the authors of this thesis. This includes the first version of the FEM plugin for 8-node elements called PreFERret, further development of the existing plugin FERret, and the newly implemented integration of FEniCSx as a plugin in Grasshopper. All codes can be found on GitHub through a link provided in Appendix A, while Appendix B provides links to YouTube videos illustrating the case studies. Section 5 presents six case studies using the software, including a brief discussion of each case. Finally, the discussion and conclusion section brings together all the elements of the thesis in relation to the research question. Additionally, this section features a segment where auditors share their recommendations for future work.

2 Theory

2.1 Finite Element Method

2.1.1 Introduction

FEM is a numerical method that makes it possible to approximate the load responses for different structures. By dividing the geometry into smaller pieces, it is possible to find solutions for complex geometry. These systems are often impossible to solve with analytical formulations. The smaller pieces are called elements, and by dividing the structure into such elements, it is possible to approach a good representation of the real response. Four well-established elements are commonly used today: beam, shell, plate, and solids. This thesis will focus on isotropic and orthotropic materials solved with solid elements. The theory section will explain general concepts for Finite Element Modelling. Then the theory for solid elements will be presented with the help of an eight-noded hexahedral element for both isotropic and orthotropic materials. In this section, the book about Finite Element Analysis by Kolbein Bell is used as reference (Bell, 2013).

Assumptions

Some basic assumptions are needed to describe the physical world using mathematical formulations. In FEM calculations, all materials are assumed to be elastic and homogeneous, and the structural response can be determined by linear theory with satisfying accuracy. Linear theory is based on two basic assumptions (Bell, 2013, s.17):

- "The *displacements are so small* that we can, with sufficient accuracy, base both equilibrium and kinematic compatibility on the original, undeformed geometry."
- "All materials are *linearly elastic, i.e.*, the relationship between stress and strain is linear and reversible. "

Using the *Principle of Virtual Work* (PVW) makes it possible to formulate the relationship between force, displacement, and stress. PVW states that a mechanical system is in static equilibrium if *virtual work* done by the external forces δW_{ext} are equal to the *virtual work* of the internal forces δW_{int} . A consequence of the assumptions above is that PVW has unlimited validity.

2.1.2 FEM Procedure

Discretization

The FEM method divides structures into more minor elements, as mentioned in the introduction. Each element consists of a certain number of nodes, where the number and placement of nodes depend on the type of element. The nodes work as the connection between the neighboring elements and make the model *connectivity*. This gives compatible elements and ensures the entire model is connected without gaps or overlaps.

Stiffness Relation

The relationship between loads and displacements is the basis of the FEM method. By solving Equation 2.1.1, the displacements are obtained.

$$\mathbf{K}\mathbf{r}=\mathbf{R} \quad (2.1.1)$$

The equation consists of the *global stiffness matrix* \mathbf{K} , the *nodal displacement* \mathbf{r} , and the applied *load vector* \mathbf{R} . The obtained nodal displacements can further be utilized to calculate the structure's strains and stresses.

Shape Functions

The displacement within the elements is described by interpolating the nodal displacements. The interpolation is done using *shape functions*. These functions have to fulfill certain requirements to give valid results, which are described in the book of Kolbein Bell (Bell, 2013):

- The *continuity principle* requires a continuous displacement over the elements. The highest-order differential equation in the FEM solution is denoted m , and the *shape functions* and their derivatives must be continuous functions up to and including the degree of $m-1$ along the entire boundary between neighboring elements. This requirement is called C^{m-1} continuity.
- The completeness principle requires that the *shape function* is able to describe the *rigid body movements* of the element correctly. This means that a pure rigid body movement should not cause any stress in the element and must be able to represent a state of constant stress within the element.

The mathematical notation of the shape functions is \mathbf{N} , which determines the variation of displacements within an element. The shape functions are structured in vectors and matrices as shown in Equation 2.1.2 and 2.1.3.

$$\mathbf{N}(x, y, z) = [\mathbf{N}_1(x, y, z) \quad \mathbf{N}_2(x, y, z) \quad \dots \quad \mathbf{N}_{n_d}(x, y, z)] \quad (2.1.2)$$

, where

$$N_i = \begin{bmatrix} N_{i1} & 0 & \dots & 0 \\ 0 & N_{i2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & N_{in_f} \end{bmatrix} \quad (2.1.3)$$

The total number of nodes is given by n_d , while n_f is the number of *degrees of freedom* (DOFs). In a model based on solid elements, the number of DOFs in each node will be three since these elements only account for the motion of translation in the three directions.

By utilizing the shape functions, it is possible to calculate the displacements by Equation 2.1.4.

$$\mathbf{u}(x, y, z) = \sum_{i=1}^{n_d} \mathbf{N}_i(x, y, z) \mathbf{r}_i = \mathbf{N}(x, y, z) \mathbf{r}_e \quad (2.1.4)$$

, where \mathbf{r}_i is the displacement of node i described by the three DOFs:

$$\mathbf{r}_i = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} \quad (2.1.5)$$

By multiplying $n_f \cdot n_d$, the total number of DOFs is achieved. It is worth mentioning that the shape function is equal for all the DOFs, i.e., $N_i = N_{i1} = N_{i2} = N_{i3}$.

Stiffness Matrix

From the principle of virtual displacement, the local stiffness matrix can be defined as follows:

$$\mathbf{k}_e = \int_{V_e} \mathbf{B}^T \mathbf{c} \mathbf{B} dV \quad (2.1.6)$$

, where \mathbf{B} is the strain matrix and, defined as the derivative of the shape function $\mathbf{B} = \Delta \mathbf{N}$. The material matrix is represented by \mathbf{c} .

The local stiffness matrix above is only related to one element. To achieve the global solution, it is necessary to assemble all the local stiffness matrices into one global stiffness matrix. The assembling is done by constructing a connectivity matrix \mathbf{a} for all elements. The matrix describes the relationship between the nodes' local and global positions. The global stiffness matrix is then constructed as shown in Equation 2.1.7.

$$\mathbf{K} = \sum_{k=1}^{n_e} \mathbf{a}_e^T \mathbf{k}_e \mathbf{a}_e \quad (2.1.7)$$

Support Conditions

To get a solution in the FEM analysis, at least all rigid body motions need to be prevented. This has to be ensured to avoid a singular stiffness matrix that can not be inverted. The solution is to introduce support conditions in the nodes. These conditions may be set as fixed, i.e., the displacements in the nodes are zero or given with a specific stiffness. The simplest way to include the support conditions is to predefine them before the global stiffness matrix is inverted. If the displacement of the nodal DOF is set to zero, the program can be more efficient by reducing the global stiffness matrix and the global load vector as shown in Equation 2.1.8.

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} \\ K_{21} & K_{22} & K_{23} & K_{24} \\ K_{31} & K_{32} & K_{33} & K_{34} \\ K_{41} & K_{42} & K_{43} & K_{44} \end{bmatrix} \begin{bmatrix} r_1 \\ 0 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \end{bmatrix} \rightarrow \begin{bmatrix} K_{11} & K_{13} & K_{14} \\ K_{31} & K_{33} & K_{34} \\ K_{41} & K_{43} & K_{44} \end{bmatrix} \begin{bmatrix} r_1 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} R_1 \\ R_3 \\ R_4 \end{bmatrix} \quad (2.1.8)$$

Loading Conditions

The external loading of a system can be both point loads and surface loads. The point loads are deconstructed and applied directly to the respective DOFs to build the load vector \mathbf{R} . For the surface load, the load definition can be done in two ways. The simplest way is to divide all the structural members into many small elements and *lump* the loads into equivalent forces. Here the load is applied directly to the nodes as for the point loads. The second method is based on PVW, which aims to construct a consistent nodal load vector \mathbf{S}^0 . External work \mathbf{W} is a result of nodal loads \mathbf{S}^0 moving over the nodal displacements \mathbf{v} . This is equal to the total work done by the distributed load \mathbf{F} and Φ when the system is moving through the displacement field $\mathbf{u}=\mathbf{N}\mathbf{v}$. See the following equations for a mathematical description.

$$W = \mathbf{v}^T \mathbf{S}^0 = - \int_V \mathbf{v}^T \mathbf{F} dV - \int_{S_\Phi} \mathbf{u}^T \Phi dS \quad (2.1.9)$$

$$\mathbf{S}^0 = - \int_V \mathbf{N}^T \mathbf{F} dV - \int_{S_\Phi} \mathbf{N}^T \Phi dS = \mathbf{S}_F^0 + \mathbf{S}_\Phi^0 \quad (2.1.10)$$

$$\mathbf{S} = \mathbf{k}\mathbf{v} + \mathbf{S}^0 \quad (2.1.11)$$

Equation 2.1.10 illustrates how the nodal loads are divided into the contribution from body forces, \mathbf{S}_F^0 , and the contribution from surface traction, \mathbf{S}_Φ^0 . To account for external action between the element nodes, the consistent nodal vector can be applied as shown in Equation 2.1.11.

Note that when lumping surface loads, it is necessary to use a fine mesh. This can result in longer computational time. Additionally, this method is only suitable for linear elements with corner nodes. When dealing with higher-order shape functions, it is advisable to utilize consistent nodal loads.

2.1.3 Isoparametric Mapping

A problem with hexahedral and tetrahedral elements is that they are not well-suited when the element sizes vary and have irregular boundaries. The solution to this problem is *isoparametric mapping*. For an arbitrary element, the natural coordinates (ξ, η, ζ) are generally *curvilinear* in the physical coordinate space (x, y, z) . Normalizing the curvilinear coordinates makes it possible to perform all mathematical operations on the simplest possible geometry, as shown to the right in Figure 2.1.1.

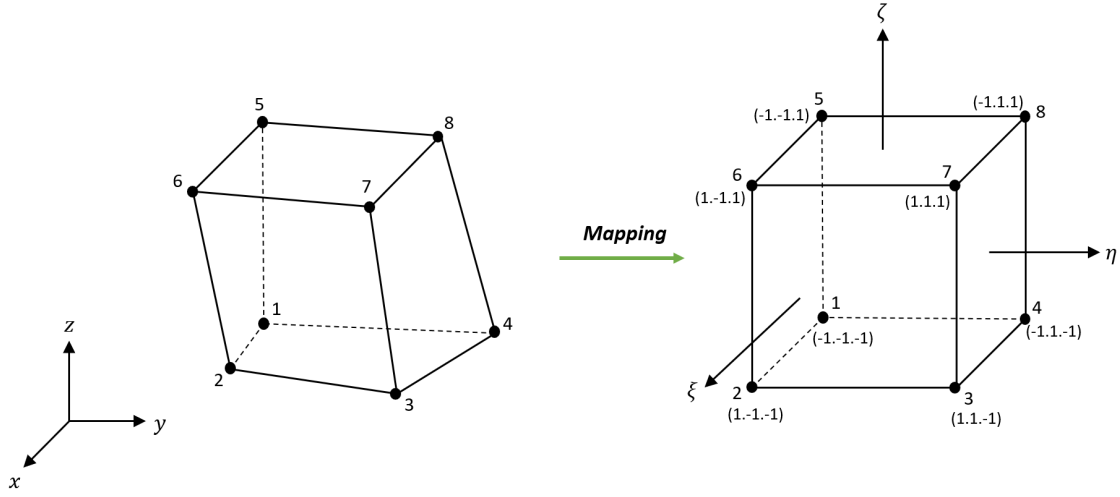


Figure 2.1.1: Isoparametric mapping of an eight-node hexahedral element.

The mapping creates an unambiguous and reversible relation from the physical coordinates (x, y, z) to the corresponding natural coordinates (ξ, η, ζ) for an arbitrary point inside the element. With this relationship established, the shape functions, as well as the derivation and integration, can be expressed with normalized natural coordinates as shown in Equation 2.1.12. This is the same shape function described in Equation 2.1.2. Thus it also has to fulfill C^{m-1} continuity (Bell, 2013).

$$\mathbf{N}_{gi} = \frac{1}{8}(1 + \xi\xi_i)(1 + \eta\eta_i)(1 + \zeta\zeta_i) \quad (2.1.12)$$

where ξ_i , η_i and ζ_i representing the natural coordinates of node i .

By using the interpolation function above for the geometry and the field variable, the element becomes *isoparametric*. This also implies that the relationship from Equation 2.1.4 still applies. However, when elements are mapped, determining the \mathbf{B} -matrix becomes less trivial. To solve the derivation of \mathbf{N} , the *chain rule* in Equation 2.1.13 has to be used.

$$\frac{\partial N_i}{\partial x} = \frac{\partial N_i \partial \xi}{\partial \xi \partial x} + \frac{\partial N_i \partial \eta}{\partial \eta \partial x} + \frac{\partial N_i \partial \zeta}{\partial \zeta \partial x} \quad (2.1.13a)$$

$$\frac{\partial N_i}{\partial y} = \frac{\partial N_i \partial \xi}{\partial \xi \partial y} + \frac{\partial N_i \partial \eta}{\partial \eta \partial y} + \frac{\partial N_i \partial \zeta}{\partial \zeta \partial y} \quad (2.1.13b)$$

$$\frac{\partial N_i}{\partial z} = \frac{\partial N_i \partial \xi}{\partial \xi \partial z} + \frac{\partial N_i \partial \eta}{\partial \eta \partial z} + \frac{\partial N_i \partial \zeta}{\partial \zeta \partial z} \quad (2.1.13c)$$

The equations above include components from the inverse *Jacobian matrix*. In FEM terminology, the Jacobian matrix is usually defined as in Equation 2.1.14.

$$\mathbf{J} = \frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \quad (2.1.14)$$

To calculate the Jacobian matrix, the explicit relationship between the physical and natural coordinates must be formulated. This is done by expressing the element geometry as an *interpolation* between the different nodal points as shown in Equation 2.1.15.

$$x = \sum_i N_{gi}x_i = \mathbf{N}_g\mathbf{x}, \quad y = \sum_i N_{gi}y_i = \mathbf{N}_g\mathbf{y}, \quad z = \sum_i N_{gi}z_i = \mathbf{N}_g\mathbf{z}, \quad (2.1.15)$$

By utilizing the inverse Jacobian matrix, the \mathbf{B} -matrix can be calculated. For a hexahedral, this matrix is a 24x6 matrix and is shown in a compact format in Equation 2.1.16.

$$\mathbf{B}_i = \begin{bmatrix} N_{i,x} & 0 & 0 \\ 0 & N_{i,y} & 0 \\ 0 & 0 & N_{i,z} \\ N_{i,y} & N_{i,x} & 0 \\ 0 & N_{i,z} & N_{i,y} \\ N_{i,z} & 0 & N_{i,x} \end{bmatrix} \quad (2.1.16)$$

In the equation above, $N_{i,x}$, $N_{i,y}$, $N_{i,z}$ corresponds to the partial derivative of N_i with respect to x , y , z , respectively. To calculate the derivatives, Equation 2.1.17 is employed.

$$\begin{bmatrix} N_{i,x} \\ N_{i,y} \\ N_{i,z} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} N_{i,\xi} \\ N_{i,\eta} \\ N_{i,\zeta} \end{bmatrix} \quad (2.1.17)$$

The relations above make calculating the local stiffness matrix possible, as defined in Equation 2.1.18.

$$\mathbf{k}_e = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 \mathbf{B}^T \mathbf{cB} |\mathbf{J}| d\xi d\eta d\zeta \quad (2.1.18)$$

2.1.4 Numerical Integration

For several elements, especially isoparametric elements, it may be difficult and sometimes impossible to calculate the analytical expressions for equations similar to Equation 2.1.18. *Numerical integration* is therefore needed to establish a solution. In FEM solvers, it is most common to use the *Gaussian quadrature rule*. The rule makes it possible to fit and integrate a polynomial of degree $2n - 1$ exactly. Here n is the number of *Gauss points* in each direction. By evaluating the weighted integrands in predefined Gauss points, the integral is determined as shown in Equation 2.1.19.

$$I = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 f(\xi, \eta, \zeta) d\xi d\eta d\zeta = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n w_i w_j w_k f(\xi, \eta, \zeta) \quad (2.1.19)$$

In Table 2.1.1, the predefined Gauss points with the corresponding weights are listed. The rule yields for all directions. Thus, the values in the table will apply for both ξ_i , η_i , and ζ_i . The eight-noded hexahedral element will need $n = 2$ Gauss points in each direction to provide the exact solution for the local stiffness matrix \mathbf{k}_e . From Table 2.1.1, $\xi = \eta = \zeta = \pm 1/\sqrt{3}$ should be chosen with the belonging weights $w_i = w_j = w_k = 1$.

| $\pm\xi_i$ | Nr. Gauss points | w_j |
|--------------|------------------|----------|
| 0 | $n = 1$ | 2.000000 |
| $1/\sqrt{3}$ | $n = 2$ | 1.000000 |
| $\sqrt{0.6}$ | $n = 3$ | 5/8 |
| 0.000000 | | 8/9 |
| 0.861136 | $n = 4$ | 0.347855 |
| 0.339981 | | 0.347855 |

Table 2.1.1: Gauss points and weights

2.1.5 Stress and Strain Relation

The strains can be calculated after the global displacement \mathbf{r} is established according to Equation 2.1.1. This is done by multiplying the \mathbf{B} -matrix with the global displacement \mathbf{r} as shown in Equation 2.1.20.

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x & \varepsilon_y & \varepsilon_z & \gamma_x & \gamma_y & \gamma_z \end{bmatrix}^T = \Delta u = \mathbf{B}\mathbf{r} \quad (2.1.20)$$

To calculate the stresses, *Hooke's law* is used, expressed in Equation 2.1.21. This law defines a linear relationship between stresses and strains for linearly elastic materials.

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x & \sigma_y & \sigma_z & \tau_{xy} & \tau_{yz} & \tau_{zx} \end{bmatrix}^T = \mathbf{C}\boldsymbol{\varepsilon} \quad (2.1.21)$$

In Hooke's law, \mathbf{C} represents the material matrix. This unique matrix will be described in Section 2.1.6.

It is worth noticing that when the stiffness matrix is calculated using numerical integration, the calculated strains and stresses are evaluated at the Gauss points. Therefore, extrapolation must be performed to obtain the strains and stresses in the nodal points as shown in Figure 2.1.2. The shape function is once more used to approximate the average values in the nodes. A new set of coordinates have to be introduced for this to be done. Specific nodal points must be utilized for the shape function to equal one of the Gauss points. Using $n = 2$ Gauss points, the equation shown in Equation 2.1.19 includes the following numerical values.

$$k = m = l = 1 \quad \text{and} \quad \xi = \eta = \zeta = 1 \rightarrow k = \xi\sqrt{3} \quad m = \eta\sqrt{3} \quad l = \zeta\sqrt{3} \quad (2.1.22)$$

The extrapolated stresses are defined in (2.1.23), where \mathbf{N}_α is the shape function that corresponds to the Gauss points in terms of the coordinates k , m and l . σ_{extr} gives the stresses in the desired places.

$$\sigma_{extr} = \sum_{i=1}^n N_\alpha \sigma_\alpha \quad (2.1.23)$$

, where

$$\mathbf{N}_\alpha = \frac{1}{8}(1 + k_\alpha k)(1 + m_\alpha m)(1 + l_\alpha l) \quad (2.1.24)$$

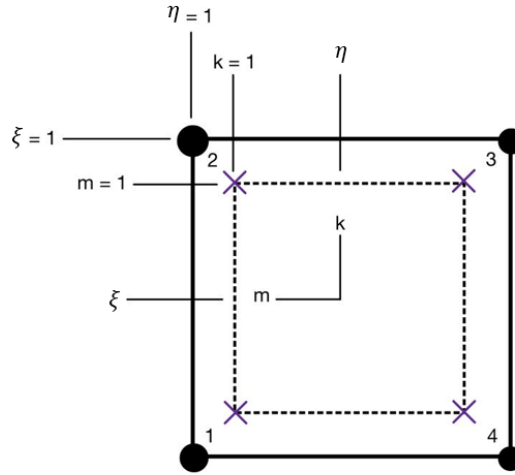


Figure 2.1.2: Extrapolation of Gauss points for an eight-node hexahedral element seen from bellow

With Figure 2.1.2 as a reference, the coordinates for, e.g., node 1, become $k = -\sqrt{3}$, $m = \sqrt{3}$ and $l = -\sqrt{3}$.

2.1.6 Materials

As mentioned in the introduction, this thesis will contain several case studies to test if the FEM solvers are working sufficiently. These studies include isotropic and orthotropic materials that differ in stiffness, strength, and general structure. This section contains essential definitions needed in the FEM solver for these different classifications of materials. Both materials are assumed to be linearly elastic. Thus, Hooke's law gives the general relation between strains and stresses. This relation is shown in Equation 2.1.25 and 2.1.26, where \mathbf{C} is the material matrix, and \mathbf{S} is the compliance matrix.

$$\sigma = \mathbf{C}\epsilon \quad (2.1.25)$$

$$\epsilon = \mathbf{S}\sigma \quad (2.1.26)$$

Isotropic Materials

Isotropic materials are materials where all planes are planes of material symmetry. By definition, isotropy means that the material properties are independent of the direction. There are only two independent constants associated with these types of materials and 12 nonzero terms in the material matrix \mathbf{C} (Staab, 2015). These constants are the Elasticity modulus \mathbf{E} and the Poisson's ratio ν . The material matrix \mathbf{C} and its compliance matrix \mathbf{S} is shown in Equation 2.1.27-2.1.28.

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ & 1-\nu & \nu & 0 & 0 & 0 \\ & & 1-\nu & 0 & 0 & 0 \\ & & & \frac{1}{2}(1-2\nu) & 0 & 0 \\ & & & & \frac{1}{2}(1-2\nu) & 0 \\ & & & & & \frac{1}{2}(1-2\nu) \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_x \\ \gamma_y \\ \gamma_z \end{bmatrix} \quad (2.1.27)$$

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_x \\ \gamma_y \\ \gamma_z \end{bmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ -\nu & 1 & -\nu & 0 & 0 & 0 \\ -\nu & -\nu & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2(1+\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & 2(1+\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(1+\nu) \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} \quad (2.1.28)$$

The most common yield criterion for metallic materials is the *von Mises* yield criterion, expressed in Equation 2.1.29. The criterion makes it possible to compare the stress in the material to the yield stress (Cook and Malkus, 2002).

$$\sigma_m = \sqrt{\frac{(\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_z - \sigma_x)^2 + 6(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2)}{2}} \quad (2.1.29)$$

Orthotropic Materials

An orthotropic material has two or three mutually orthonormal planes of symmetry (Malo, 2021). Timber is an example of a material with these properties, which differs from the isotropic material steel, whose properties are equal in all directions. As the material's properties depend on the load directions, more engineering constants must be defined to determine the response. The matrices are further derived with their components in equations (2.1.30) to (2.1.31). Several elasticity moduli E , Poisson's ratios ν , and shear moduli G must be known to determine the solution. Either destructive or non-destructive testing of the relevant material obtains these engineering constants.

$$\begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{31} \\ \sigma_{12} \end{bmatrix} = \begin{bmatrix} \frac{1-\nu_{23}\nu_{32}}{E_2 E_3 D} & \frac{\nu_{21}+\nu_{31}\nu_{23}}{E_2 E_3 D} & \frac{\nu_{31}+\nu_{21}\nu_{32}}{E_2 E_3 D} & 0 & 0 & 0 \\ & \frac{1-\nu_{13}\nu_{31}}{E_1 E_3 D} & \frac{\nu_{32}+\nu_{12}\nu_{31}}{E_1 E_3 D} & 0 & 0 & 0 \\ & & \frac{1-\nu_{12}\nu_{21}}{E_1 E_2 D} & 0 & 0 & 0 \\ & & & G_{23} & 0 & 0 \\ & Sym. & & & G_{13} & 0 \\ & & & & & G_{12} \end{bmatrix} \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ \gamma_{23} \\ \gamma_{31} \\ \gamma_{12} \end{bmatrix} \quad (2.1.30)$$

, where

$$D = \frac{1}{E_1 E_2 E_3} \begin{vmatrix} 1 & -\nu_{21} & -\nu_{31} \\ -\nu_{12} & 1 & -\nu_{32} \\ -\nu_{13} & -\nu_{23} & 1 \end{vmatrix} = \frac{1}{E_1 E_2 E_3} (1 - 2\nu_{21}\nu_{13}\nu_{32} - \nu_{23}\nu_{32} - \nu_{12}\nu_{21} - \nu_{13}\nu_{31}) \quad (2.1.31)$$

$$\begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ \epsilon_{23} \\ \epsilon_{31} \\ \epsilon_{12} \end{bmatrix} = \begin{bmatrix} \frac{1}{E_1} & \frac{-\nu_{21}}{E_2} & \frac{-\nu_{31}}{E_3} & 0 & 0 & 0 \\ \frac{-\nu_{12}}{E_1} & \frac{1}{E_2} & \frac{-\nu_{32}}{E_3} & 0 & 0 & 0 \\ \frac{-\nu_{13}}{E_1} & \frac{-\nu_{23}}{E_2} & \frac{1}{E_3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{G_{23}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{G_{31}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{G_{12}} \end{bmatrix} \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{31} \\ \sigma_{12} \end{bmatrix} \quad (2.1.32)$$

The following relation in Equation 2.1.33 is utilized to manipulate the matrices to become symmetric about the diagonal. This relation is a prerequisite for the equations above.

$$v_{ij} = v_{ji} \cdot \frac{E_{ii}}{E_{jj}} \quad (2.1.33)$$

2.1.7 Higher-Order Shape Functions

For the theory presented above, an eight-noded hexahedral element has been used to describe the procedures in FEM for solid elements. This element gives a small number of DOFs but has some limitations that can cause big errors. Two important ones for the described theory section are listed below.

1. The element cannot be used to model beam action exactly due to the linear order of the shape functions.
2. Straight element edges also introduce spurious shear strains since the element cannot describe curvature exactly. Thus, *shear locking* appears, which results in the elements acting stiffer than the actual behavior.

To overcome these problems, one solution is to reduce the element size. However, spurious strains will still occur. A better solution is introducing the 20-noded serendipity hexahedral element, shown in Figure 2.1.3. Including a mid-node enables the description of curved edges, making it more suitable for accurately representing beam action. Thus, the serendipity element has 60 DOFs in contrast to the 24 DOFs in the eight-node element. By this, converging toward a solution using fewer elements is possible. Nevertheless, it should be noticed that employing 20-noded elements also leads to a larger stiffness matrix and increased computational time (Mathisen, 2021). The modifications that have to be applied in the FEM solver to use the 20-noded serendipity hexahedra elements are described in the following.

The procedure within the FEM solver does not change when a higher-order element is applied, but to be able to describe the element edges, a new set of shape functions have to be established (Zienkiewicz and Taylor, 2013). These shape functions are shown in Equation 2.1.34.

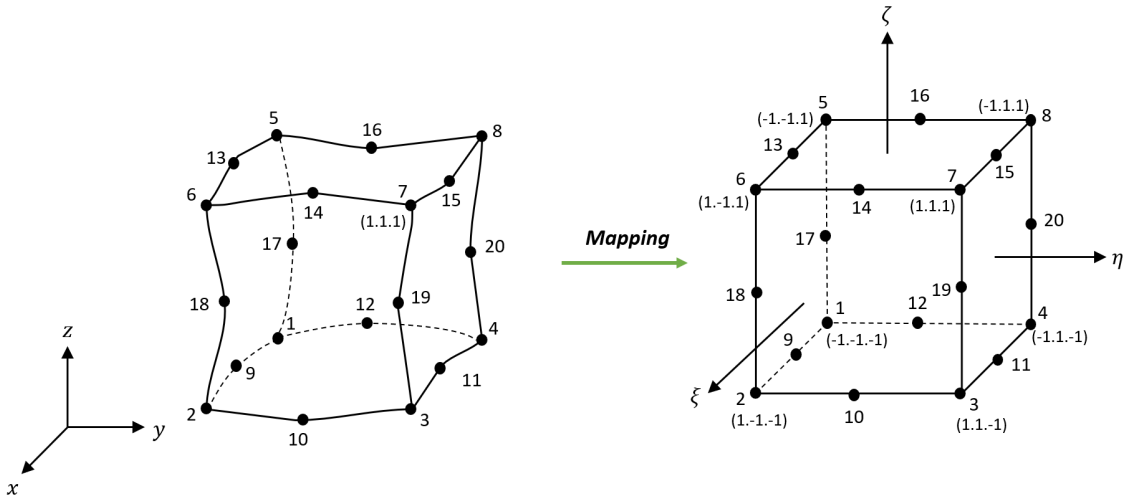


Figure 2.1.3: Isoparametric mapping of a 20-node serendipity hexahedral element

$$N_i = \frac{1}{8}(1 + \xi_i\xi)(1 + \eta_i\eta)(1 + \zeta_i\zeta)(\xi_i\xi + \eta_i\eta + \zeta_i\zeta - 2) \quad \text{for } i = 1\dots 8 \quad (2.1.34a)$$

$$N_i = \frac{1}{4}(1 - \xi^2)(1 + \eta_i\eta^2)(1 + \zeta_i\zeta^2) \quad \text{for } i = 9, 11, 13, 15 \quad (2.1.34b)$$

$$N_i = \frac{1}{4}(1 + \xi_i\xi^2)(1 - \eta^2)(1 + \zeta_i\zeta^2) \quad \text{for } i = 10, 12, 14, 16 \quad (2.1.34c)$$

$$N_i = \frac{1}{4}(1 + \xi_i\xi^2)(1 + \eta_i\eta^2)(1 - \zeta^2) \quad \text{for } i = 17\dots 20 \quad (2.1.34d)$$

When the higher-order shape functions are set, $n = 3$ Gauss points will be required in all three directions to achieve full integration. This corresponds to a total of 27 sampling points, as defined in Table 2.1.1 (Cook and Malkus, 2002).

3 Software

This thesis utilizes several software from different developers. The most dominating software is Grasshopper. It was chosen as the desired AAD environment due to the authors' experience, as it is an intuitive, robust, and powerful application. Also, the possibility of making new components was significant. To create new components the IDE Visual Studio was well suited for its well-developed and seamless process of making new components. This choice led to using Visual Studio Code as a code editor to develop code quickly. For comparing the results obtained from Grasshopper, a well-known commercial FEM software was preferable to benchmark the results against. Due to the authors' previous experiences with the software and its well-known robustness, versatility, stability, and speed, the choice landed on Abaqus. For the case studies related to the FEniCSx platform, Ubuntu was introduced, which is an operating system based on Linux. All the software mentioned above are further elaborated in the following.

3.1 Rhino Grasshopper

Grasshopper is a parametric modeling tool offering a visual programming interface inside the Rhinoceros software. This interface allows dragging wires to connect various components; the fantasy is the only limit for what one can make. The geometry scripted in Grasshopper is visualized in Rhino, and the opportunity of parameterization enables the possibility to make rapid changes with the resulting geometry updating simultaneously in Rhino. In addition, the use of algorithms helps the developers overcome the limitations of traditional CAD software and reach a level of complexity and control beyond the manual human ability (Tedeschi, 2014). However, as the script for a given geometry might be more comprehensive to establish in Grasshopper than modeling in regular CAD software, it should be considered whether parameterization is needed. If needed, the investment in time to establish a well-functioning script in Grasshopper will probably give severe benefits when modifications must be made. Only some parameters need to be changed instead of modeling the whole geometry from scratch. This benefit shows the massive advantage of utilizing parameterization in the design process. However, it should not be used regardless of the project since some problems might be solved more efficiently in other ways (SimplyRhino, 2023).

3.2 Visual Studio

To make components in Grasshopper, Microsoft's Visual Studio is well suited to use. This powerful source code editor enables building and debugging the code in Grasshopper right from the editor. So-called IDEs make it possible to write and develop software. The Visual Studio IDE provides, among other things, compilers, code completion tools, and graphical designers, making it more comprehensive than most IDEs on the market (Microsoft, 2023b). As the Grasshopper environment is based on the programming language C#, this is the language that is used for making components for the plugins. C# is an object-oriented programming language provided by Microsoft. It is based on C++ but includes elements from Visual Basic and Java (Nätt, 2023).

3.3 Visual Studio Code

For solving tasks with plain code in either Python or C#, Visual Studio Code will be used. This code editor supports operations like debugging, task running, and version control, aiming to provide a quicker editor for less complex workflows (Microsoft, 2023a).

3.4 Abaqus/CAE

The Finite Element Modeling software Abaqus is used to verify the results from the plugins in Grasshopper. With this software, one can quickly create and edit geometry and meshing with specific loads and supports and visualize the results from the finite element analysis provided by Abaqus (Simulia, 2023). By choosing the same elements with equal properties as the ones used in the plugins, the credibility of the plugins can be evaluated, as well as discovering errors along the way. Evaluating the results from Abaqus/CAE as the solution, the error of the plugins can be measured, giving a sense of the accuracy of the own-developed software. If the results from the plugins are similar to the Abaqus solution for simple geometries, there are reasons to believe the plugin will perform well for more complex cases.

3.5 Ubuntu

Ubuntu Desktop is developed by Canonical and is a so-called *Linux distribution*. It is among the most popular distributions due to its user-friendly interface and ease of use. Ubuntu is an operating system developed from the Linux kernel, and it is open-source and free to use. Ubuntu enables the execution of software developed for the Linux operating system on PCs that use the Windows operating system, utilizing a docker development environment (Ubuntu, 2023). This approach is used to execute the FEniCSx code.

4 Methods

4.1 Plugin for 8-node Hexahedral Solid Elements

The following plugin, from now on called *PreFERret*, is a FEM solver developed by the authors for the 8-node trilinear element, also called the 8-node brick element. This element designated Hex8, is formulated as an isoparametric element in the plugin, enabling the irregular shape of the Hex8, as described in 2.1.2. The plugin contains five components; *Element*, *Support*, *Load*, *Assembly* and *Solver*. These are described in the next section. Further, specific classes are defined to store the necessary information and bring it through the process to the solver. The classes may have multiple so-called constructors which take different arguments for different use of the classes. The constructors enable the user to set and fetch properties from the class. These classes are described in Table 4.1.1.

| Class | Arguments | Description |
|---------------|---|---|
| ElementClass | Point3D, MeshFace | The ElementClass stores the nodes for the given element, in addition to its faces. |
| SupportClass | Point3D | The SupportClass holds the support point with its global coordinates. |
| LoadClass | Point3D, Vector3D | The LoadClass contains information about the point where the load acts, as well as a vector describing the magnitude and direction of the load. |
| AssemblyClass | ElementClass, SupportClass, LoadClass | The AssemblyClass contains the information from the classes above, collecting all the information before the solver. |

Table 4.1.1: Custom classes made in the plugin for Hex8 elements.

4.1.1 Component: Element

The Element component takes a mesh as an input and gives an ElementClass as output. The mesh is deconstructed to fetch the vertices from the faces of the mesh. As some vertices from a face coincide with vertices from other faces, a selection process is run to achieve a list with eight unique points. To ensure that the points are numbered in the same order as the nodes for the isoparametric element, the points are sorted based on their global coordinates. The sorted list of points is then used to construct an ElementClass, which is returned as the output of the component. If there are several elements, this process is done multiple times, and a list of ElementClasses is returned.

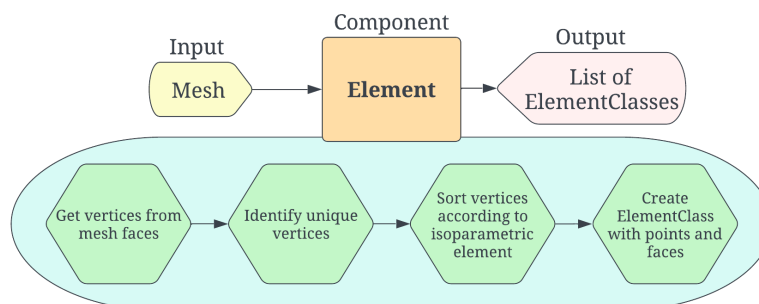


Figure 4.1.1: Workflow in the Element component.

4.1.2 Component: Support

With one or multiple points as input, the Support component returns either a single SupportClass or a list of SupportClasses as output. To ensure the correct support description, if there should be a case with no points as input, the component returns an empty list if a null check is passed. For simplicity, the support condition for this plugin is fixed in all directions in the given node if the node is chosen as a support point. In other words, the three DOFs in the node are set to zero.

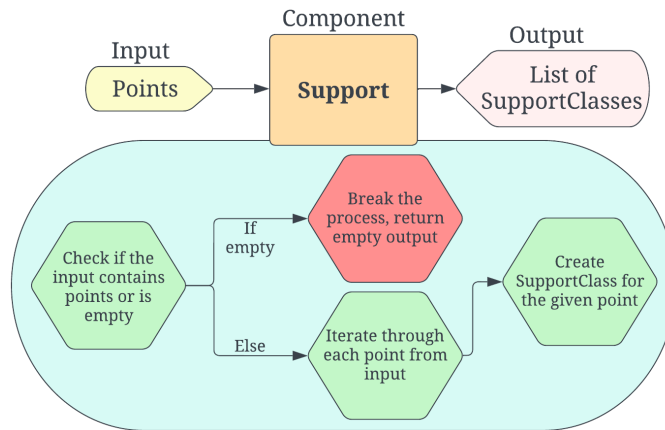


Figure 4.1.2: Workflow in the Support component.

4.1.3 Component: Load

The load component is similar to the Support component but holds one more input parameter. With both points and vectors as inputs, the component constructs LoadClasses, which are returned as outputs.

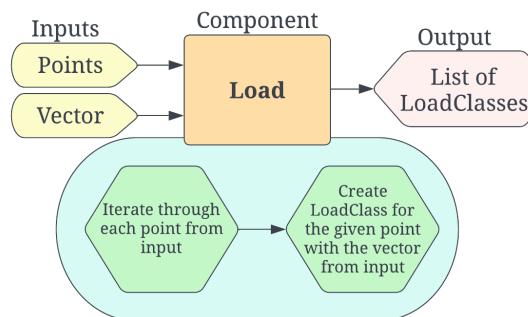


Figure 4.1.3: Workflow in the Load component.

4.1.4 Component: Assembly

The assembly component gathers all the information from the three previous components to prepare for the solver. It takes ElementClasses, SupportClasses, and LoadClasses as inputs and returns an AssemblyClass as output.

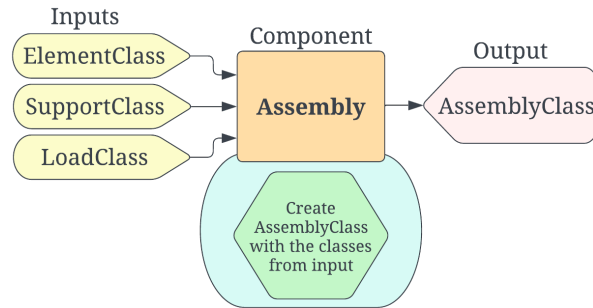


Figure 4.1.4: Workflow in the Assembly component.

4.1.5 Component: Solver

The Solver Component stands for most of the computations in this plugin. It takes an Assembly-Class from the Assembly Component as input and returns the results in terms of displacements as output. The solver is made using the material properties of steel, with Young’s modulus of 210 000 MPa and Poisson’s ratio of 0.3. A bunch of methods is made to achieve a smoother and more readable script by calling these methods from the main code. These are described in the following part, starting with a brief introduction of the methods’ arguments in Table 4.1.2. Finally, an illustration of the workflow of the component is provided.

| Argument | Description |
|--------------------|---|
| Points | Nodes in the global system. |
| PointsNat | Nodal coordinates in the natural coordinate system. Nodes of the isoparametric element. |
| ξ, η, ζ | Coordinates in the natural coordinate system. |
| Loads | LoadClasses containing placement, magnitude, and direction of the loads. |
| C-matrix | The C-matrix containing the material properties for computation of the stiffness matrix. |
| Numb | The number of unique nodes multiplied by DOFs in each node, which in this plugin is three DOFs. |
| K | The global stiffness matrix. |
| R | The global load vector. |
| SupPts | The points where support is applied. |
| Size | The number of unique nodes. |
| ndNodes | List of unique points for the system. |
| El | List of all the elements in the system, in ElementClass format. |
| PointA, PointB | Two points to be checked whether they are equal or not. |
| BREP | Boundary representation. |
| BC | Boundary Condition. |

Table 4.1.2: Arguments used in the methods.

GetJacobi(Points, ξ , η , ζ): The 3x3 Jacobi matrix is computed in this method, according to the theory in 2.1.3. Derivations were calculated by hand to reduce computational time since the derivatives are demanding to compute in coding.

$$\mathbf{Returns: J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}$$

JacobiDet(Points, ξ , η , ζ): Calculates and returns the determinant of the Jacobi Matrix. The determinant is needed for calculating the stiffness matrices. It can also be used to verify the Jacobi Matrix, as the sum of the eight determinants should be equal to the box's volume.

$$\mathbf{Returns: J} = |\mathbf{J}| = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{vmatrix}$$

NiXis(PointsNat, ξ , η , ζ): Calculates the derivatives of the shape functions with respect to ξ .

$$\mathbf{Returns: } [N_{1,\xi}, N_{2,\xi}, \dots, N_{8,\xi}]$$

NiEtas(PointsNat, ξ , η , ζ): Calculates the derivatives of the shape functions with respect to η .

$$\mathbf{Returns: } [N_{1,\eta}, N_{2,\eta}, \dots, N_{8,\eta}]$$

NiZetas(PointsNat, ξ , η , ζ): Calculates the derivatives of the shape functions with respect to ζ .

$$\mathbf{Returns: } [N_{1,\zeta}, N_{2,\zeta}, \dots, N_{8,\zeta}]$$

f_i(Points, PointsNat, ξ , η , ζ , C-matrix): The method calculates the functions which are to be summed to achieve the stiffness matrix by use of numerical integration. Firstly, the derivatives of the shape functions are calculated, now with respect to the cartesian coordinates, as described in Equation 2.1.17 by using $N_{i,\xi}$, $N_{i,\eta}$ and $N_{i,\zeta}$. Further, the 6x24 B-matrix is constructed by placing the elements from Equation 2.1.17 into the correct rows and columns, as shown in Equation 2.1.16. The B-matrix is then transposed and used to multiply the matrices $\mathbf{B}^T \mathbf{C} \mathbf{B}$. Finally, the resulting matrix from this multiplication is multiplied with the Jacobi Determinant from the **JacobiDet** method to achieve the integral described in Equation 2.1.19.

$$\mathbf{Returns: } \mathbf{f}_i(\xi, \eta, \zeta) = \mathbf{B}^T \mathbf{C} \mathbf{B} |\mathbf{J}|$$

LoadVec(Points, Loads, Numb): In this method, the load vector is constructed. By comparing the points where the load acts with the point list for the box nodes, the load magnitudes are applied to the corresponding nodes and DOFs in the load vector.

$$\mathbf{Returns: } \mathbf{R}$$

ReduceK(Points, **K**, SupPts): To account for the support points, in addition to reducing computational time, the global stiffness matrix is reduced in accordance with the global coordinates of the support points. For the points where support has been applied, the rows and columns for the corresponding DOFs are removed.

Returns: K_r

ReduceR(Points, **R**, SupPts): With the same arguments as for **ReduceK**, the load vector is also reduced similarly. As the operations are now acting on a vector, the rows for the corresponding DOFs are removed at the same indexes as in **ReduceK**.

Returns: R_r

ReducePoints(Points, SupPts): To account for the deleted rows and columns in **K** and **R**, the point list has to be reduced correspondingly to ensure that the correct rows and columns are removed.

Returns: Points_r

GetConnectivity(Size, ndNodes, el): Connectivity needs to be established between local and global coordinates to use the plugin for multiple elements. This way, the stiffness matrix can be established for the whole system by assigning every DOF to the correct place in the global system according to each element's placement. This is done by comparing the nodes in each element to the global nodes, achieving the indexes in the global system corresponding to the indexes in the local systems. The connectivity is returned and later used to construct the global stiffness matrix.

Returns: ConMatrix

DistPoint(PointA, PointB): To compare the similarity between two points, this method checks the distance between two points, A and B. Since the accuracy of coordinate placements is not exact, a small tolerance is added to ensure that similar points are found, even though the numerical description of the coordinates is not exactly equal. The method returns true if the points are equal and false if not.

Returns: True/False

Main code: From the inputs of the solver component, the properties of the elements, supports, and loads are gathered. The nodes of the elements run through a process of finding the unique points in a global sense. The natural coordinates are defined with a specific order, reflected in every element, as mentioned about the Element Component. Further, the C-matrix is defined. In this plugin, steel is used as material, and the properties are explicitly defined with Young's modulus of 210 000 MPa and a Poisson's ratio of 0.3. Preparing for the numerical integration of the stiffness matrix, the Gauss points are defined using a 2x2x2 Gaussian quadrature rule. With this rule, the weights w_i, w_j and w_k in Equation 2.1.19 are all equal to unity, as Table 2.1.1 shows.

By implementing the connectivity, the local stiffness matrices are now calculated and placed where they belong in the global stiffness matrices. Using the reduced global stiffness matrix and load vector, the displacements are calculated by solving for **r** in Equation 2.1.1: $\mathbf{r} = \mathbf{K}^{-1}\mathbf{R}$. Finally, the nodal displacements are returned as output from the Solver Component.

Returns: r

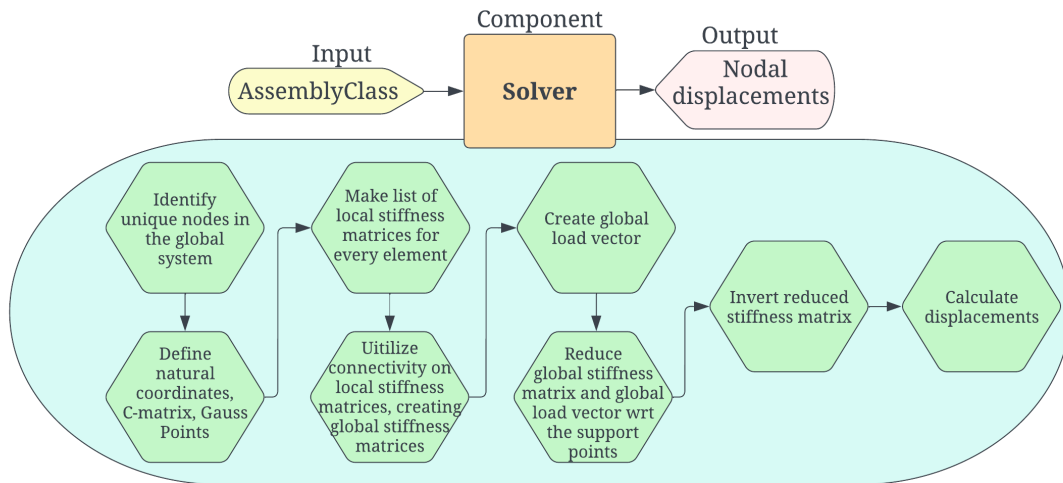


Figure 4.1.5: Workflow in the Solver component.

4.2 Plugin for Hexahedral and Tetrahedral Solid Elements

To achieve higher accuracy in the computations than the 8-node solid elements provide for complex geometries, the 4-node tetrahedral elements, abbreviated TET4, can be used. These elements are less costly in a computational sense, as the stiffness matrix will be smaller due to the lesser amount of DOFs. An already existing Grasshopper plugin for both tetrahedral and hexahedral elements was developed by Marcin Luczkowski and Sverre Magnus Haakonsen. The plugin, called FERret, will be described similarly to the plugin for 8-node solid elements. This plugin contains similar components as the PreFERret plugin, named **FEM_BoundaryCondition**, **FEMLoad**, **FEMMaterial**, **NEW_FEMSolver** and **MeshPreview**. In the same way as PreFERret, the FERret plugin contains several classes to store the relevant information for the solver. These classes are described in Table 4.2.1. In addition to the linear elements of the HEX8 and TET4, quadratic elements like the HEX20 and TET10 can be used. These higher-order elements, described in 2.1.7, are preferable when calculating shapes subjected to bending or generally when a solution with a higher level of accuracy is required. However, due to a problem in the FERret solver while this thesis is being written, these higher-order elements are not available to use for obtaining reliable results. Therefore, the work in this thesis connected to the FERret plugin is limited to the linear elements of HEX8 and TET4.

| Class | Arguments | Description |
|--------------|--|--|
| Element | Nodes, Connectivity, Type, Id, ElementMesh, TypologyVertices | Stores the important features about the elements and constructing the elements' faces and meshes. |
| Material | YoungModulus, PoissionRatio, YieldingStress, Weight | Stores the material properties, and calculates the C-matrix. |
| Node | GloablId, Coordinate, BC_U, BC_V, BC_W, Type | Categorizes the nodes into corner, edge, face, and interior nodes. Stores the information about each node. |
| Support | Position, Tx, Ty, Tz | Contains the position of the support points, and which DOFs that are restrained and not. |
| TempFE_Mesh | MeshList, MeshNodes, MeshElements, Sigma_ii, NodalMisesStresses, MisesStresses, dU, dV, dW, Material | Contains the results from the solver, preparing them for preview. |
| FEM | | Contains functions to calculate shape functions and natural coordinates. |
| FEM_Matrices | | Contains functions to calculate the Jacobi determinant and the stiffness- and element matrices, as well as reducing the matrices according to BCs. |
| FEM_Utility | | Contains functions to prepare and execute most of the calculations to be used in the solver. |
| FEMLogger | | Logging files meanwhile the process is running. |
| GrahamScan | | Contains functions to execute Graham scan based on either mesh or points. |

Table 4.2.1: Custom classes made in the FERret plugin.

4.2.1 Class: FEM_Utility

As the FEM_Utility class is such a central part of the plugin, this class is described in the following section. The class contains a bunch of methods that are doing most of the calculations and arranging the process leading to the analysis results.

AddMidEdgeNodes(Mesh): As the mesh is described similarly for the 8-node elements, the mid-side nodes have to be added for the element to become a 20-node element. This method also adds this mid-side node for tetrahedral elements. If the mesh consists of 4-node tetrahedral elements, the resulting mesh will be represented by 10-node tetrahedral elements.

Returns: MeshList

ElementsFromMeshList(MeshList, PointList, ElementList): Local and global IDs, and the element type are assigned to the elements. From these IDs, the connectivity for the nodes is developed. As the method is of type void, it returns no value but updates the elements with their properties.

GetMeshNodes(MeshList): When the plugin uses a series of points describing the mesh, there must be no point duplicates to ensure the calculations are done correctly. With a certain threshold value to retrieve duplicates even if the points are not equally described, the method returns a list of points consisting of unique points.

Returns: UniquePoints

GetBodyForceVector(Material, ElementList, NumberGlobalNodes, Logger): To include the self-weight of the model, this method calculates the force that arises due to gravity, i.e., the body force.

Returns: BodyForce

GetGaussPointMatrix(Order, ElementType): Depending on the element type and element order, this function calculates the Gauss Points in terms of local coordinates. These points are to be used in the numerical integration to achieve the displacements in the solver.

Returns: GaussNaturalCoordinates

GetShapeFunctions(ξ , η , ζ , ElementType): Depending on the element type and element order, this function calculates the shape functions based on the natural coordinates ξ , η , ζ .

Returns: [N_1, N_2, \dots, N_i]

PartialDerivativeShapeFunctions(ξ , η , ζ , ElementType): Depending on the element type and order, this function calculates the shape functions' derivatives based on the natural coordinates.

Returns: [$N_{i,\xi}, N_{i,\eta}, N_{i,\zeta}$]

DisplacementInterpolationMatrix(ShapeFunctions, dofs): This method returns a matrix containing the shape functions on the diagonal, preparing it to be multiplied with the load vector in the correct format.

Returns: InterpolationMatrix

CalculateDisplacementCSparse(K_Global, R_Global, BCs, Logger): By taking the boundary conditions into account, the displacements are calculated by manipulating the K- and R-matrices to express the restrained motion, and then solving Equation 2.1.1: $\mathbf{r} = \mathbf{K}^{-1}\mathbf{R}$.

Returns: \mathbf{r}

CalculateElementStrainStress(Element, Displacement, Material, Logger): This component calculates the stresses and strains for an element based on the material properties and displacements.

Returns: [ElementStrain, ElementStress]

CalculateGlobalStress (ElementList, Displacement, Material, Logger): Based on the element stresses and strains, this method calculates the stresses and strains in a global sense. The method returns the nodal stresses for each direction and the Mises stress for both nodes and elements.

Returns: [GlobalStress, MisesStress, ElementMisesStress]

SortedVerticesGraham(Mesh): This method sorts the mesh vertices to ensure the correct nodes are represented in calculations. It is important that the nodes in each element are sorted in the correct and consistent order. The method for sorting the vertices is the Graham Scan.

Returns: SortedVertices

4.2.2 Component: FEM_BoundaryConditions

Describing the support condition for the model, this component takes seven inputs; A mesh describes the geometry to which the boundary conditions are applied. Either a set of points or surfaces is given to define the points or area the geometry should be given boundary conditions at. An integer value is the fourth input to define whether the type of support is a point or surface. The three last inputs describe which DOFs are locked and which are not, with one input for each direction. For the given mesh, the component uses the method GetMeshNodes from the FEM_Utility class to determine the nodes that should be defined as support points. If the support type is a surface, the component considers the points included in this surface area. In the end, the component outputs a list of SupportClasses, ready to be taken into the calculations in the solver.

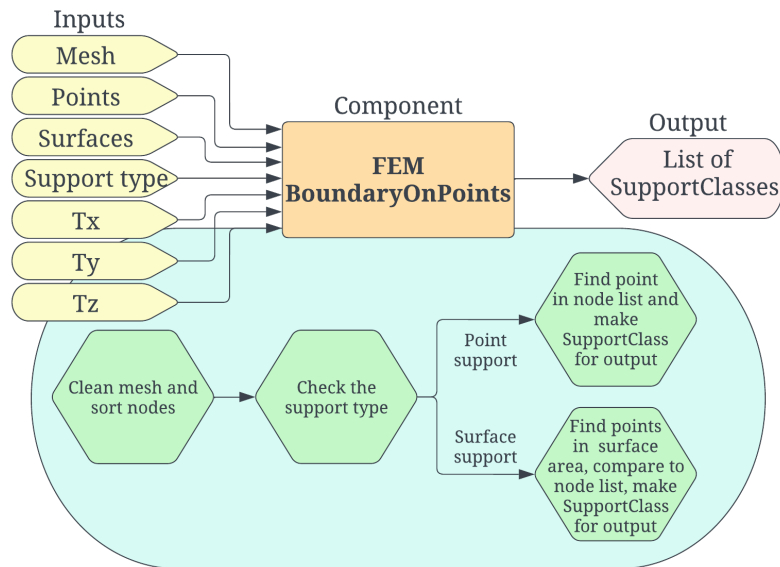


Figure 4.2.1: Workflow in the Boundary Conditions component.

4.2.3 Component: FEMLoad

The inputs for the FEMLoad component are quite similar to the ones for FEM_BoundaryConditions consisting of a list of meshes, a number defining the load type, points for point loads, surface for surface loads, and a vector describing the load's magnitude and direction. The component utilizes methods from the GrahamScan and FEM_Utility classes to define the load vector. This vector is given as an output, in addition to the list of points where the load is applied and values of the resultant load and load area.

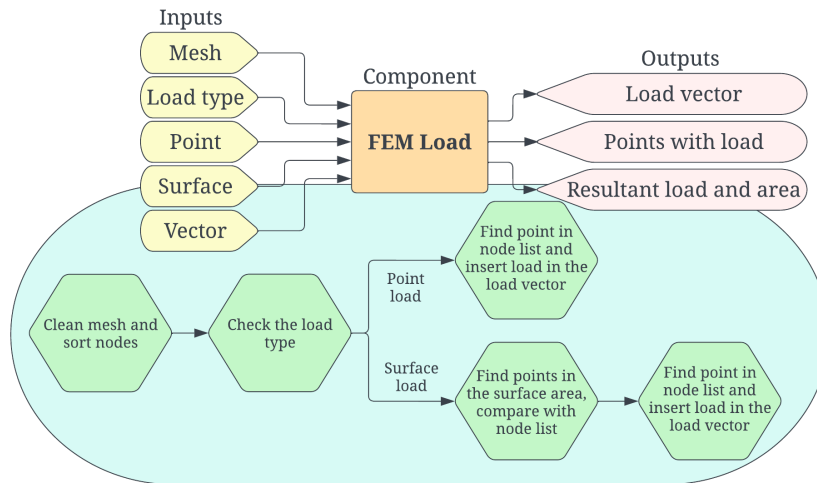


Figure 4.2.2: Workflow in the Load component.

4.2.4 Component: FEMMaterial

Young's modulus, Poisson's ratio, yielding stress, and material weight are the parameters given as input to the component to describe the material of the model. These properties are used to make a material class which is returned as the only output from the component.

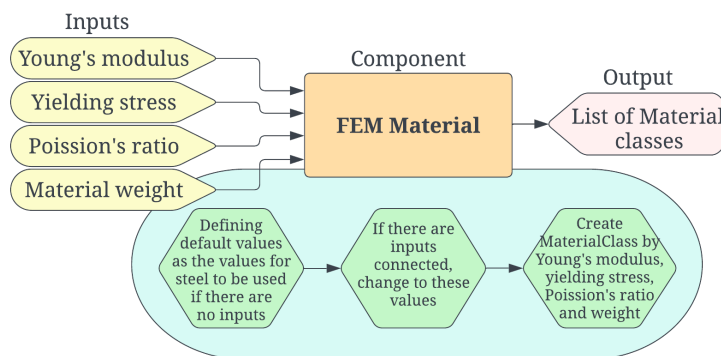


Figure 4.2.3: Workflow in the Material component.

4.2.5 Component: NEW_FEMSolver

The NEW_FEMSolver is the main component in this plugin. It takes geometry mesh, element order, loads, boundary conditions, and the material as the inputs and returns the analysis results as the output. As most of the calculations are done in methods inside the class FEM_Utility, the solver component mostly arranges the parameters used for the calculations in the methods. The results are gathered and organized, while the FEMLogger class monitors the time used on the primary operations. As outputs, the solver gives the displacements in the three cartesian axis directions, the Mises stresses in both nodes and elements, the log list from the FEMLogger class, the analyzed mesh, and the unique nodes within the mesh. From these outputs, the displacements and stresses can be evaluated at the desired node by using the values with assistance from the node list, as these contain the corresponding indexes. The results can be visualized by connecting the analyzed mesh to the last component of the plugin, the MeshPreview.

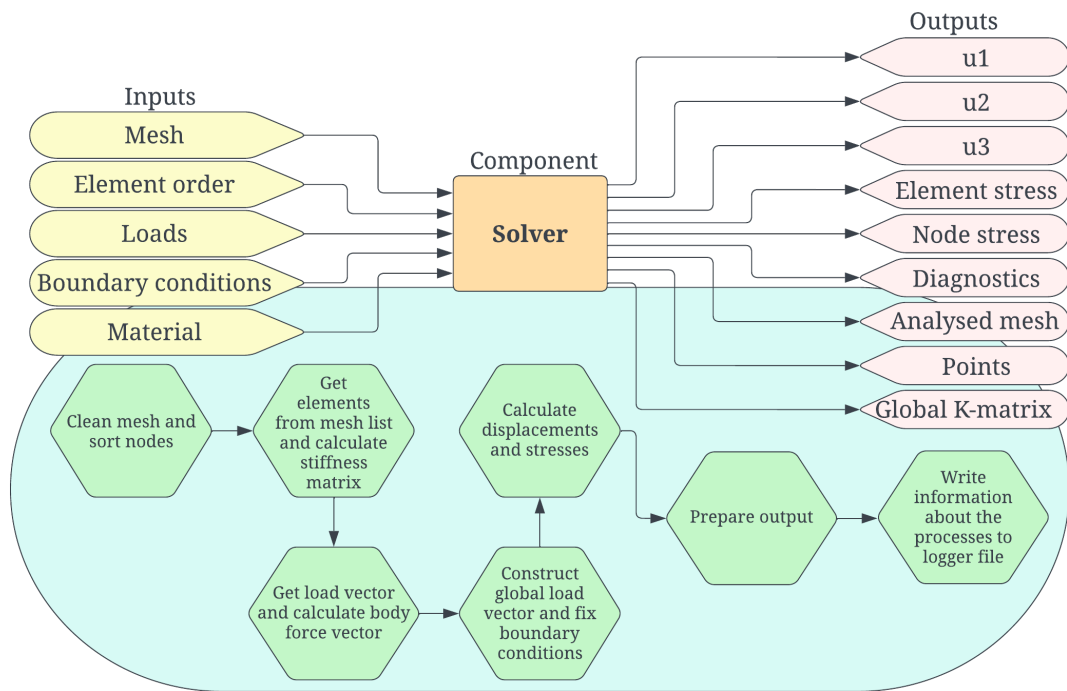


Figure 4.2.4: Workflow in the Solver component.

4.2.6 Component: MeshPreview

In addition to the analyzed mesh, the component takes in an integer for the user to specify the type of results to preview. The last of the three inputs is a scaling factor to scale the results to a more convenient size for visualization of the results. The component computes a deformed mesh, and through a coloring algorithm, the result values are represented by a colored mesh, showing how the displacements, stresses, and utilizations change throughout the model. As output, the deformed geometry and the deformed mesh are returned.

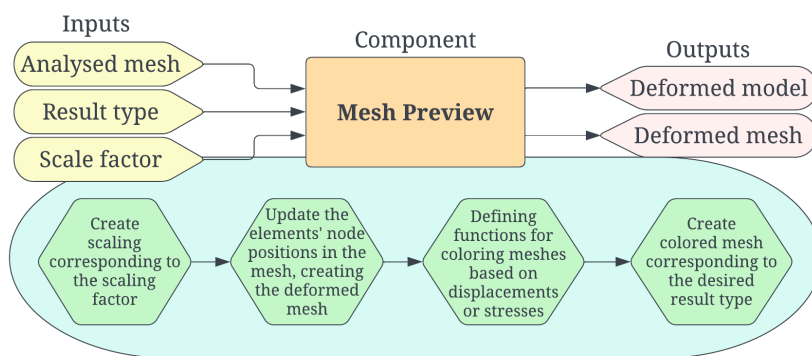


Figure 4.2.5: Workflow in the Mesh Preview component.

4.3 Development towards Orthotropic Materials

The FERret plugin is developed based on isotropic materials, i.e., the properties of the model are independent of the direction. To develop the plugin to handle orthotropic materials, the solver needs to take care of these direction-dependent properties. To achieve this, the material class and component described in 4.2 must be modified. These developments made by the authors are described in the following, summarized in the end in a flowchart illustrating the changes.

4.3.1 Developing Class: Material

At first, the authors developed a new class based on the original Material class. The new class was customized for orthotropic materials, with all the engineering constants that follow, and returning the C-matrix for this material, see Section 2.1.6. However, as the FERret plugin relies on the Material component, this new class would not be compatible with the rest of the code without manipulating several classes and components. Therefore, the original Material class was instead modified to include the orthotropic case. This was done by introducing subclasses, one for isotropic materials and one for orthotropic materials, both based on the mother class Material. By utilizing different constructors, the correct material subclass would be chosen based on the parameters in the class, as these differ for the two cases. In this way, the correct material class is defined, and the C-matrix from either Equation 2.1.27 or Equation 2.1.30 is returned for the material given as input.

4.3.2 Developing Component: FEMMaterial

To fully implement the orthotropy, a new component was developed. While the original FEMMaterial component only has one single Young's modulus and one Poisson ratio, the new component must take in parameters for three Young's moduli, three Poisson ratios, and three shear moduli. Making this component is regarded easier for both the developers and users to work with, as it will be easy to distinguish between the materials. The new component, FEMMaterialOrto, is made with the abovementioned inputs, including the inputs yielding stress and weight from the original component. The process is similar to the FEMMaterial component for making a material class, which is then returned as the output.

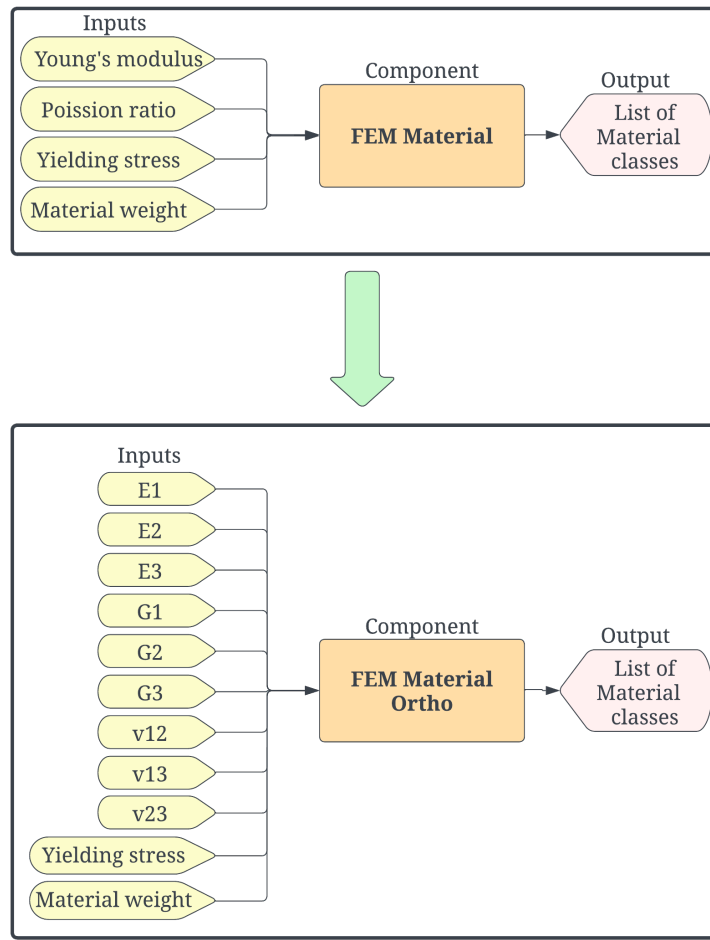


Figure 4.3.1: The change from Material component to Material Ortho component.

4.4 Development towards Combining Materials

Structures are in general consisting of multiple materials working together. Therefore, it would be preferable to be able to represent this interaction between materials with different properties in the model analysis. To be able to combine different materials in the same model, the development of the FERret plugin was taken further. The way the FERret plugin originally was developed, it was restricted to using only one material for the whole model. The following section describes the changes in the FERret plugin script to implement this possibility of combining materials in one model.

4.4.1 Component: FEM Part

A new component was made to achieve the correct structure with meshes and materials, named FEM Part. The inputs of this component are a mesh list and a list of the material for each part. The mesh list is structured so that the meshes of the structure that should be considered together as a part are gathered into one branch. Therefore, this branch structure represents the parts the structure consists of. Further, the component creates a material list with branches corresponding

to the parts in the mesh lists. Finally, this material list and the mesh list are returned, representing each element in the structure together with the specific material properties for the element.

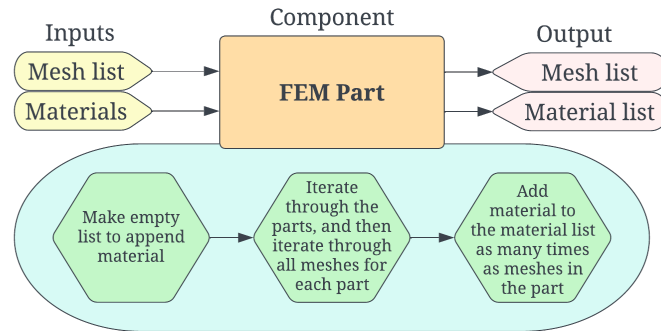


Figure 4.4.1: The Part component structures the materials to the desired format.

4.4.2 Component: Part FEM Solver

To make the solver compatible with the new structure in the material input, the authors changed the `NEW_FEMSOLVER` component from taking in one material item to taking in a whole material list. This list corresponds to the list of meshes the solver takes in, which again represents the elements in the global mesh. In this way, each element can be assigned a particular material. Since the material properties are used to construct local stiffness matrices representing each element, the correct global stiffness matrix would be possible to construct out of the local matrices, each representing a specific material. The processes in the Part FEM Solver work in the same manner as the `NEW_FEMSOLVER`, but this new component was made to save the original script. Thus, the workflow of this component is not illustrated here; see Figure 4.2.4 for reference.

In addition to the FEM solver, some classes in the FERret plugin had to be manipulated to work with the new list structure of the materials instead of the single-item structure. `FEM_Matrices` was developed to `FEM_Matrices_Part`, and `FEM_Utility` was developed to `FEM_Utility_Part`. The developments involved changing variables and functions to handle lists and not only an item. Loops were utilized to ensure that the correct material was connected to the correct mesh and, thereby, the correct element in the structure. With these changes, all calculations regarding elements are done, taking their corresponding material into account.

4.5 Development of FEniCSx

An extensively time-consuming procedure within FEM analysis involves constructing and solving the matrices outlined in the Theory section. In both the PreFERret and FERret solvers, maintaining low computational time is challenging when the mesh is refined. Here, the global K-matrix is established and calculated by looping over the elements individually and then solving by using a direct solver. To solve the time-consuming issue, the open-source computing platform FEniCSx is used. The platform has a large community surrounding it, but the University of Cambridge is the primary developer. With FEniCSx, scientific models described by Partial Differential Equations

(PDEs) are easily solved. The software translates them into efficient FEM code and solves the resulting linear system of equations, making the process efficient and accurate. All the components in the FEniCSx platform are fundamentally designed for parallel processing. This also means that instead of a direct solver, the program uses a much better approach, namely a multigrid solver. This framework allows for quick prototyping of FEM formulations and solvers on laptops, workstations, and high-performance computers (FEniCS, 2021).

One of the key advantages of FEniCSx is that its packages are structured in a manner that ensures the code closely aligns with the theoretical formulations. This approach makes the program more recognizable to users and lowers the barrier for constructing their own PDE solver, for instance, making a FEM solver for 2D and 3D geometries. To illustrate the close correspondence between code and theory, the fundamental elasticity equations are presented alongside a corresponding FEniCSx code in the following (Alnæs et al., 2015). To describe small elastic displacements of a body Ω , the equations are written as follows:

$$-\nabla \cdot \sigma = f \quad \text{in } \Omega, \quad (4.5.1)$$

$$\sigma = \lambda \text{tr}(\varepsilon)I + 2\mu\varepsilon, \quad (4.5.2)$$

$$\varepsilon = \frac{1}{2}(\nabla u + (\nabla u)^T). \quad (4.5.3)$$

The stress tensor, represented by σ , determines the distribution of internal forces in a material. Body force per unit volume is denoted by the variable f . The material's elasticity in the domain of Ω is defined by the parameters λ and μ . The identity tensor, represented by I , maintains the direction of a vector and alters its magnitude by a factor of 1. To calculate the diagonal elements' sum in a tensor, one can utilize the trace operator marked as tr . Additionally, the symmetric tensor ε measures the rate of material displacement, considering its intensity and orientation. Lastly, the displacement vector field, known as u , illustrates the motion of points within the material. By combining these equations, the stresses can be determined.

$$\sigma = \lambda(\nabla \cdot u)I + \mu(\nabla u + (\nabla u)^T). \quad (4.5.4)$$

```
1 def sigma(u):
2     return lambda_*nabla_div(u)*Identity(d) + 2*mu*epsilon(u)
```

Listing 4.5.1: Defining the stress function in Python.

In order to successfully solve the partial differential equation (PDE), it needs to be formulated in a variational form. This form can be summarised using the following functions:

$$a(u, v) = L(v) \quad \forall v \in \hat{V}, \quad (4.5.5a)$$

$$a(u, v) = \int_{\Omega} \sigma(u) : \nabla v \, dx = \lambda(\nabla \cdot u)I + \mu(\nabla u + (\nabla u)^T), \quad (4.5.5b)$$

$$L(v) = \int_{\Omega} f \cdot v \, dx + \int_{\partial\Omega} T \cdot v \, ds. \quad (4.5.5c)$$

```

1 #Define variational problem
2 u = Trialfunction(V)
3 d = u.geometric_dimension() #space dimension
4 v = TestFunction(V)
5 f = Constant((0, 0, -rho*g))
6 T = Constant((0, 0, 0))
7 a = inner(sigma(u), epsilon(v))*dx
8 L = dot(f, v)*dx + dot(T, v)*ds

```

Listing 4.5.2: Defining the variational problem in Python.

If ∇v is expressed as a sum of symmetric and anti-symmetric parts, the unsymmetrical part will be canceled out in the product of $\sigma(u) : \nabla v$. By replacing ∇u with the symmetric gradient $\epsilon(u)$, the variational form can be written more naturally considering elasticity problems.

$$a(u, v) = \int_{\Omega} \sigma(u) : \epsilon(v) dx, \quad (3.28) \quad (4.5.6)$$

$\epsilon(v)$ is the symmetric part of ∇v given by:

$$\epsilon(v) = \frac{1}{2}(\nabla v + (\nabla v)^T). \quad (4.5.7)$$

```

1 def epsilon(u):
2     return 0.5*(nabla_grad(u) + nabla_grad(u).T)

```

Listing 4.5.3: Defining the strain function in Python.

```

1 #Compute solution
2 u = Function(V)
3 solve(a == L, u, bc)

```

Listing 4.5.4: Solves the PDE in Python.

Naturally, more information must be defined for the solver to function properly. The complete Python code is available in Appendix A.

The code for FEniCSx has been developed for C++ and Python language but is only available for Linux operation systems. Since the authors are using Windows, the program is run using a docker development environment running Ubuntu. The FEniCSx-plugin collects data from Grasshopper, sends it to the FEniCSx-program for calculation, and then brings the results back to Grasshopper for visualization. The plugin contains five components; **InfoLBC**, **Load**, **Collect Data**, **FEniCSx Solver** and **Results**. A brief description of how data is collected, transported, and calculated is shown in the following sections.

4.5.1 Component: InfoLBC

The InfoLBC component takes all the nodes from the mesh, a BREP representing the BC-zone, and BREPs representing the loading zone as input. The nodes are further denoted as vertices. As output, the component gives the nodeinfo list, a list of BC points, and lists of different loading points. The nodeinfo list contains tags that provide information about the use of the different vertices in the calculation. This method minimizes the information that is transported to the FEniCSx Solver. Since many meshing tools give duplicated vertices as output, these duplicates are removed. Then the code loops through all the BC- and Load-BREPs, and check if the vertices are lying inside.

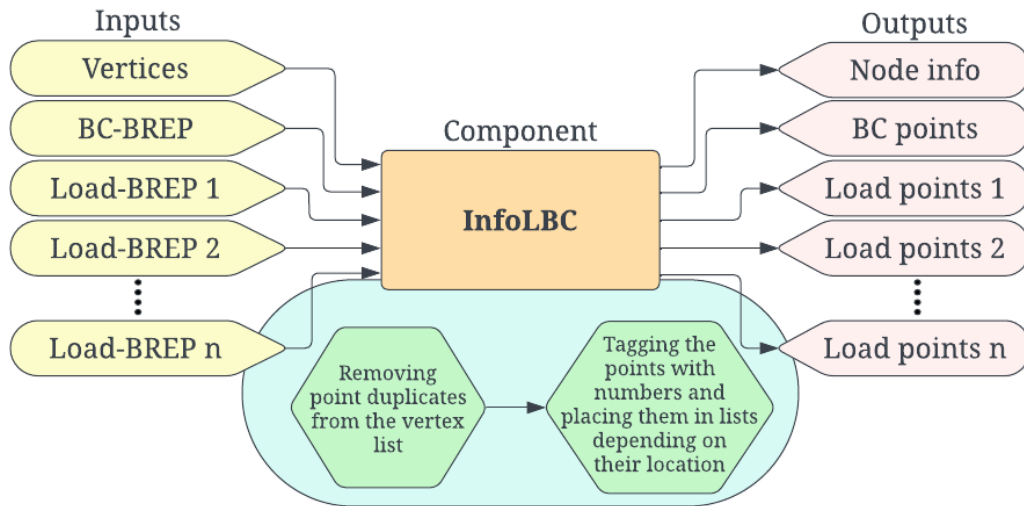


Figure 4.5.1: Workflow in the InfoLBC component

4.5.2 Component: Load

The Load component takes in a set of numbers representing loading in the x, y, and z-direction for up to 4 different loads. The output is a list of vectors describing the loads ready to be transported to the FEniCSx Solver.

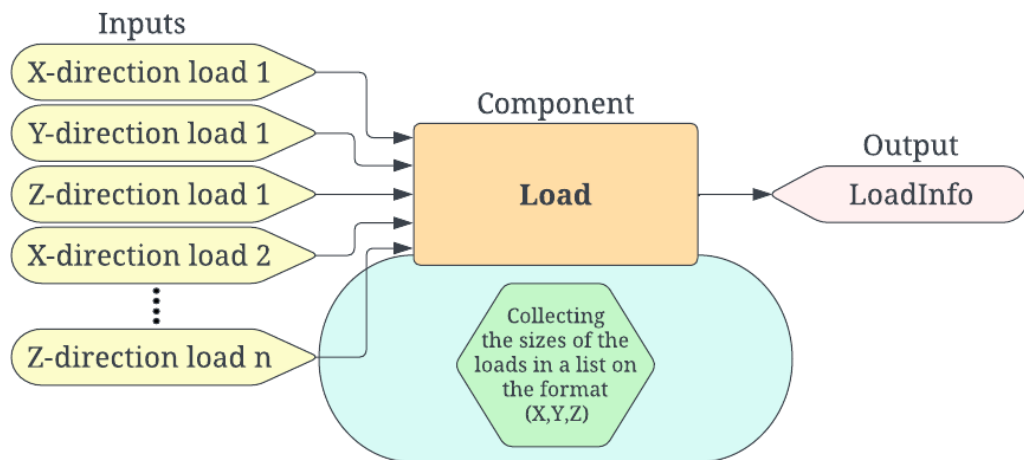


Figure 4.5.2: Workflow in the Load component

4.5.3 Component: Collect Data

The Collect Data component takes the vertices and connectivity list from the meshing tool, the nodeinfo list from InfoLBC, and LoadInfo from Load as input. Inside the component, the four different lists are written to a text file on the computer. The tag number is written together with its belonging point. As output, the component gives a bool and the time to write the files. The bool is set to true if the code is completed and ensures that the FEniCSx Solver reads the text files after they have been filled with information.

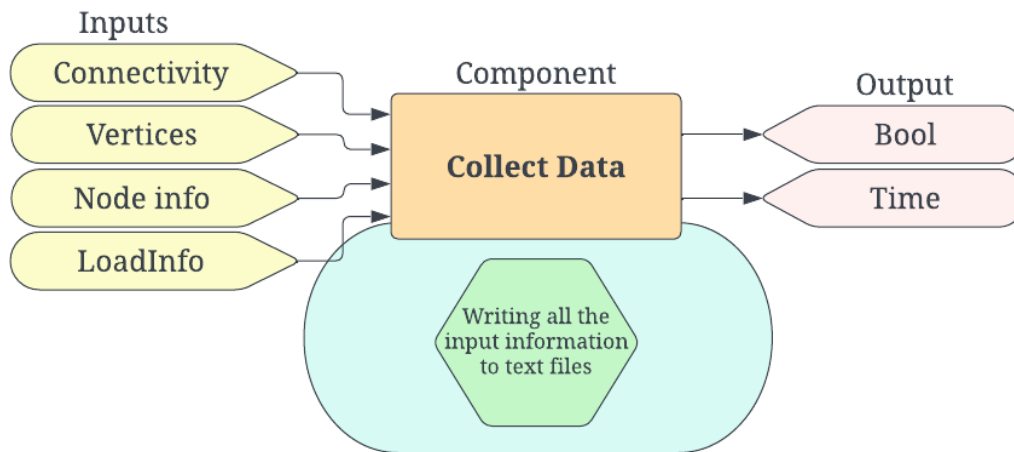


Figure 4.5.3: Workflow in the Collect Data component

4.5.4 Component: FEniCSx Solver

In the FEniCSx Solver component, all the information sent to the text files in the Collect Data component is used. When the component gets the true boolean, it starts a Python compiler running through Ubuntu. This program first collects the lists of connectivity, vertices, node info, and load info. Then the vertices are mapped from Grasshopper numbering to FEniCSx numbering. To solve the static linear elasticity problem, a smoothed aggregation algebraic multigrid solver is used. A *near-nullspace* which is the operator's nullspace in the absence of boundary conditions, is required. Consequently, a so-called PETscNullSpace object is built. For this solver, the null space is spanned by three translation modes and three rotation modes (FEniCS, 2023). Then the mesh is built again using the connectivity and the vertices collected from the text files.

The code is now made for tetrahedral elements with quadratic shape functions, but the shape function order or element type can easily be changed in the code. Further, the material properties, loads, boundary conditions, stress function, and the different function spaces are defined. The model is then assembled, and the partial differential equations are solved. This gives functions for the stress field, displacements, and for instance, the possibility of calculating the von Mises stress. The results are then mapped back to Grasshopper numbering and stored in text files. The component has the boolean and a SolverInfo as output. The boolean is set to true if the code is completed. The SolverInfo gives information about, among other things, the number of nodes, matrix size, and the solver's computational time.

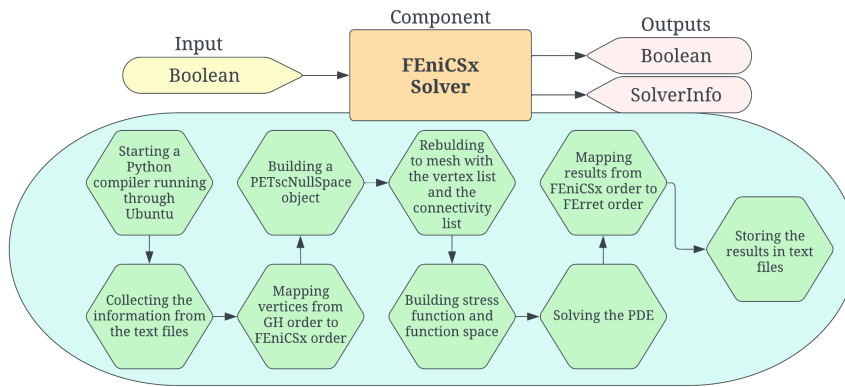


Figure 4.5.4: Workflow in the FEniCSx solver component

4.5.5 Component: Results

This component also inputs the boolean and starts if the bool is true. Inside the component, the calculated information is collected. The vertices are also brought as input. By adding the displacements to the points, the displaced points are obtained, representing the deformed geometry. The three displacement components, u , v , w , the von Mises stress, and the displaced points are given as output.

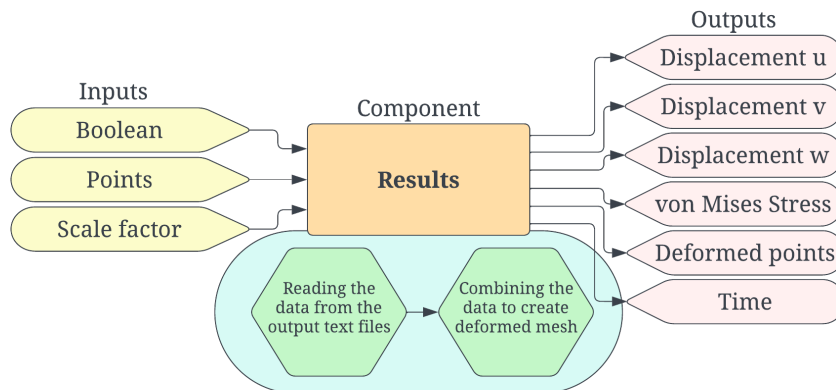


Figure 4.5.5: Workflow in the Results component

5 Case Studies

In this section, the six case studies are presented. Each case study is generally presented before the methodology for obtaining the desired structural problem is provided. Further, the results from the particular case study are presented and shortly discussed. For this thesis, displacements and stresses are the evaluated parameters. The stresses are in general assessed in terms of von Mises stress as defined earlier in Equation 2.1.29. To obtain fully comparable results, self-weight is excluded from the analyses for all case studies. For each case, a flowchart illustrating the Grasshopper script is provided. Figure 5.0.1 explains the color codes that are used for these charts. The discussion section for this thesis provides the overall discussion of the outcome from the six case studies; see Section 6.

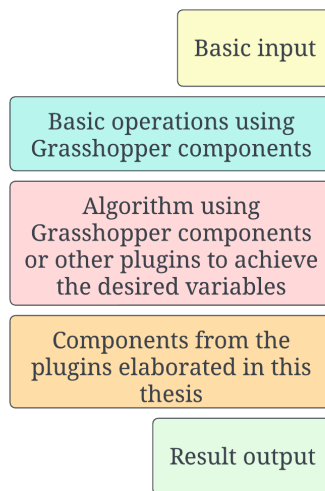


Figure 5.0.1: Color codes for the flowcharts presented in the case studies.

5.1 Case Study 1: Verifying the PreFERret Plugin

To confirm the method used for developing the PreFERret plugin, a benchmark is done to obtain comparable results for comparison with the commercial software Abaqus. Displacement in the vertical direction is the parameter evaluated in this benchmark. This parameter is chosen as this is the most significant displacement pattern due to loads working in the negative z -direction. In addition, since strains and stresses are dependent on displacements, this parameter is considered the most crucial to perform a benchmark on.

A comparison between the maximum displacements is added to account for eventual problems with node sorting for the PreFERret plugin. Here, the mesh structure and the number of elements are equal in PreFERret and Abaqus. This will work as a confirmation of the magnitude of displacement if the values in the specific nodes do not match.

The chosen geometry in the benchmark is a cubic cantilever with one point load at each of the two upper corners on the opposite side from the fixed surface, as illustrated on Figure 5.1.1.

Key values for the geometry and the load are presented in Table 5.1.1. This geometry is chosen to make a consistent mesh with cubic elements. In the same table, the material properties are listed. These values represent the isotropic material steel.

| Width | Height | Length | \sum Force | Youngs' Modulus | Poisson's ratio |
|----------|----------|----------|--------------|---------------------------|-----------------|
| 1 000 mm | 1 000 mm | 1 000 mm | 20 kN | 210 000 N/mm ² | 0.3 |

Table 5.1.1: Geometry, load and material properties for Case Study 1.

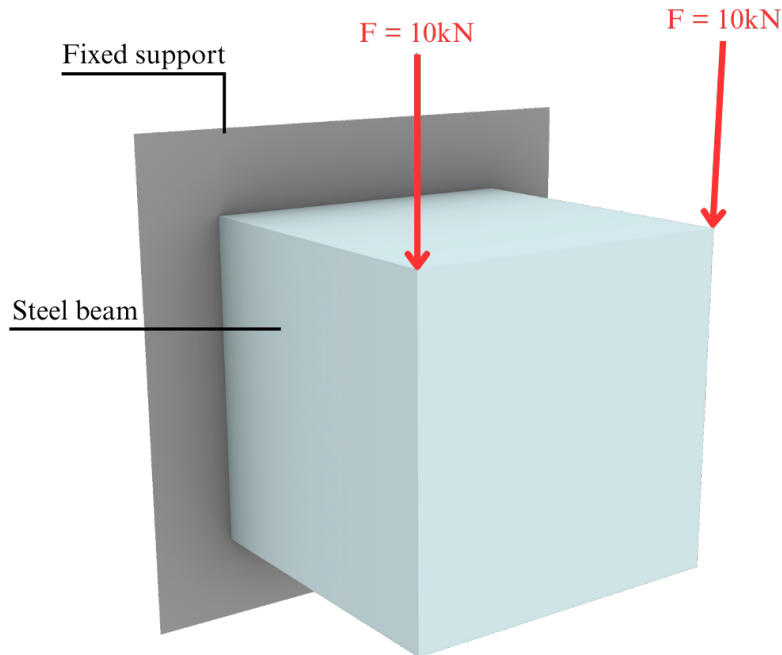


Figure 5.1.1: Geometry and load case for Case study 1.

5.1.1 Method

Grasshopper

For the Grasshopper script used in this benchmark, the geometry is firstly made by standard Grasshopper components. A box is constructed with the desired width, height, and length inputs. This box is then used in a cluster code developed by the authors. This cluster code repeats the box in the desired amount of times in each of the three directions, simultaneously as the size is decreased to still fulfill the overall measurements for the model. This way, a regular mesh compatible with the PreFerret plugin is easily made.

Further on, this mesh is deconstructed to obtain its vertices. These vertices are then evaluated to state which points should be defined as load points and fixed points. These points are then sent to the Load and Support components, respectively. The element component takes in the mesh and is then connected to the PreFerret Solver and the Load and Support components. From the solver, the results in terms of displacement are processed. The values are sorted to obtain the extreme values. In addition, the displacements are evaluated at the desired index to compare to the coinciding points to evaluate the displacements at a particular vertex on the meshed geometry. For a detailed description of the entire setup, please refer to the GH file in Appendix A.

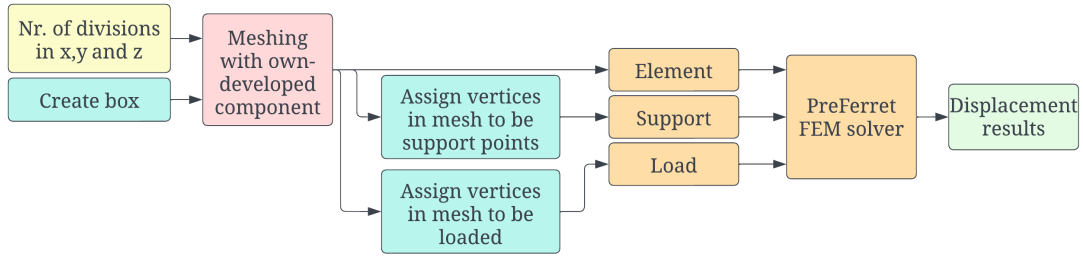


Figure 5.1.2: Flowchart of Grasshopper script for Case Study 1.

Abaqus

To obtain the same geometry in Abaqus, a rectangle of 1x1 meter was drawn and then extruded 1 meter in the z-direction. Subsequently, the material of steel, the support condition, and the load case were defined. The linear hexahedral element called C3D8 was used in Abaqus to obtain similar behavior of the elements. For mesh creation, one meshing was done to create a highly dense mesh to obtain numerical solutions as accurately as possible. This mesh was the base for evaluating the convergence of the displacement results obtained from Grasshopper as the mesh was gradually refined. Afterward, for comparing the maximum stresses, the meshing was done in the same manner as in Grasshopper to achieve comparable results.

5.1.2 Results

For the analytical solution, due to the height-to-length ratio of the beam exceeding the ratio of 1/10, shear deformations have to be taken into account. Therefore, the Timoshenko beam theory is necessary to calculate this solution. The formula in Equation 5.1.1 calculates the analytical solution for a cantilever beam subjected to a point load.

$$w_{\max} = \frac{PL}{\kappa GA} + \frac{PL^3}{3EI} = -0.000671\text{mm} \quad (5.1.1)$$

- $P = 20\text{kN}$, representing the point loads acting on the beam.
- $L = 1\,000\text{mm}$, representing the length of the beam.
- $\kappa = \frac{10(1+\nu)}{12+11\nu}$, the Timoshenko shear coefficient.
- $G = 81\,000\text{MPa}$, the shear modulus.
- $A = b \cdot h = 1\,000\,000\text{mm}^2$, the cross-sectional area.
- $E = 210\,000\text{MPa}$, the Young's modulus.
- $I = \frac{b \cdot h^3}{12}$, the second moment of area for the cross-section, where b and h represents the height and the width.

The results from the displacement benchmark for the three nodes at the end of the beam are presented in Table 5.1.2. From the benchmark of the maximum displacements of the beam, these results are presented in Table 5.1.3. Both benchmarks are illustrated in Figure 5.1.3 and Figure 5.1.4. The term *error* in the results is referred to the errors between the obtained solutions. Both the errors between the two numerical solutions and the errors between the numerical and analytical solutions are measured. The definitions of the absolute and relative errors are given in Equation 5.1.2 below, where X denotes the relevant variable for error measurements. The case of comparing the two numerical solutions is presented in this equation, but the same procedure applies to comparing the numerical methods and the analytical solution. The other case studies will use this method of computing the error. Thus, the definition is not repeated.

$$\begin{aligned} \text{Absolute error} &= X_{\text{Ferret}} - X_{\text{Abaqus}} \\ \text{Relative error} &= \left(\frac{X_{\text{Ferret}} - X_{\text{Abaqus}}}{X_{\text{Abaqus}}} \right) \times 100 \end{aligned} \quad (5.1.2)$$

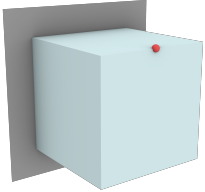
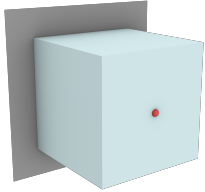
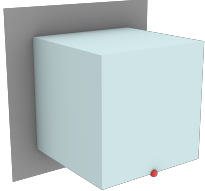
| Point of Measurements | FEM Solver | Number of Elements | Displacement [mm] | Error from Abaqus Sol. | Error from Analytical Sol. |
|---|------------|--------------------|-------------------|------------------------|----------------------------|
|  | Abaqus | 39 783 | -0.000586 | – | 85.1E-06 |
| | PreFerret | 8 | -0.000418 | 168E-06 | 253E-06 |
| | | 64 | -0.000678 | -91.7E-06 | -6.56E-06 |
| | | 512 | -0.000653 | -66.7E-06 | 18.4E-06 |
| | | 4 096 | -0.000483 | 103E-06 | 188E-06 |
|  | Abaqus | 39 783 | -0.000566 | – | 106E-06 |
| | PreFerret | 8 | -0.000211 | 355E-06 | 460E-06 |
| | | 64 | -0.000204 | 362E-06 | 467E-06 |
| | | 512 | -0.000160 | 406E-06 | 511E-06 |
| | | 4 096 | -0.000153 | -964E-06 | -859E-06 |
|  | Abaqus | 39 783 | -0.000565 | – | 106E-06 |
| | PreFerret | 8 | -0.000104 | 461E-06 | 567E-06 |
| | | 64 | 0.0000800 | 645E-06 | 751E-06 |
| | | 512 | -0.0000620 | 503E-06 | 609E-06 |
| | | 4 096 | -0.0000550 | 510E-06 | 616E-06 |

Table 5.1.2: Benchmark of displacements in z-direction for three node positions.

| FEM Solver | 2x2x2 | 4x4x4 | 8x8x8 | 16x16x16 |
|--------------------------|----------|---------|---------|----------|
| Abaqus [mm] | 0.00121 | 0.00216 | 0.00406 | 0.00785 |
| PreFERret [mm] | 0.00109 | 0.00229 | 0.00423 | 0.00829 |
| Absolute error [mm] | -0.00012 | 0.00013 | 0.00017 | 0.00044 |
| Relative error [%] | -9.69 | 6.06 | 4.18 | 5.57 |
| Time usage PreFERret [s] | 0.004 | 0.112 | 43.8 | 30 244 |

Table 5.1.3: Benchmark of maximum displacements in z-direction for certain mesh divisions.

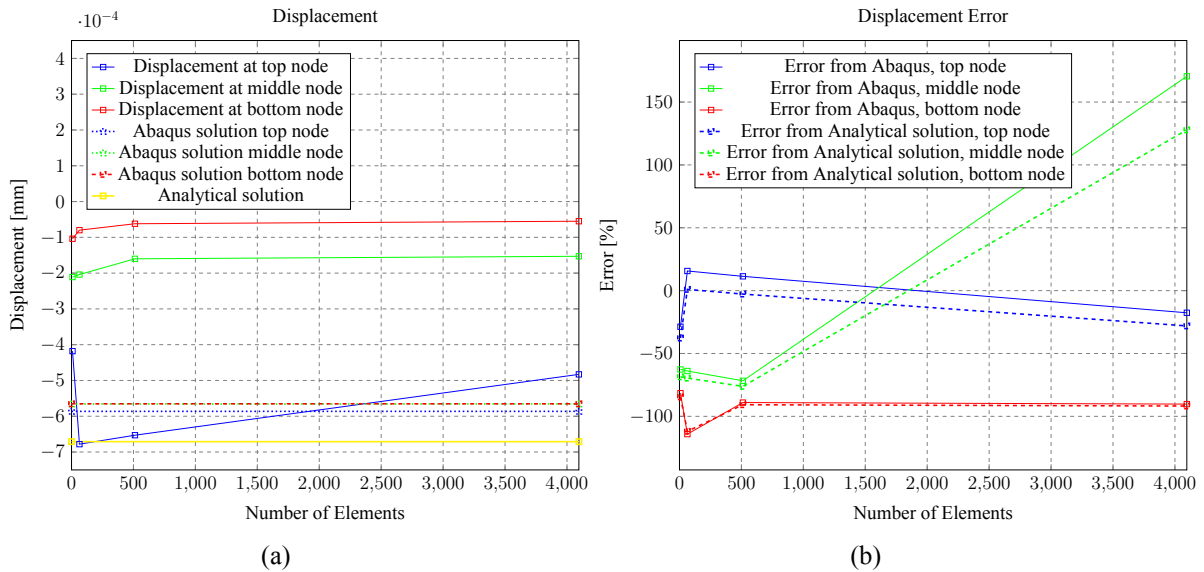


Figure 5.1.3: Comparison of displacements and errors compared to Abaqus and analytical solution.

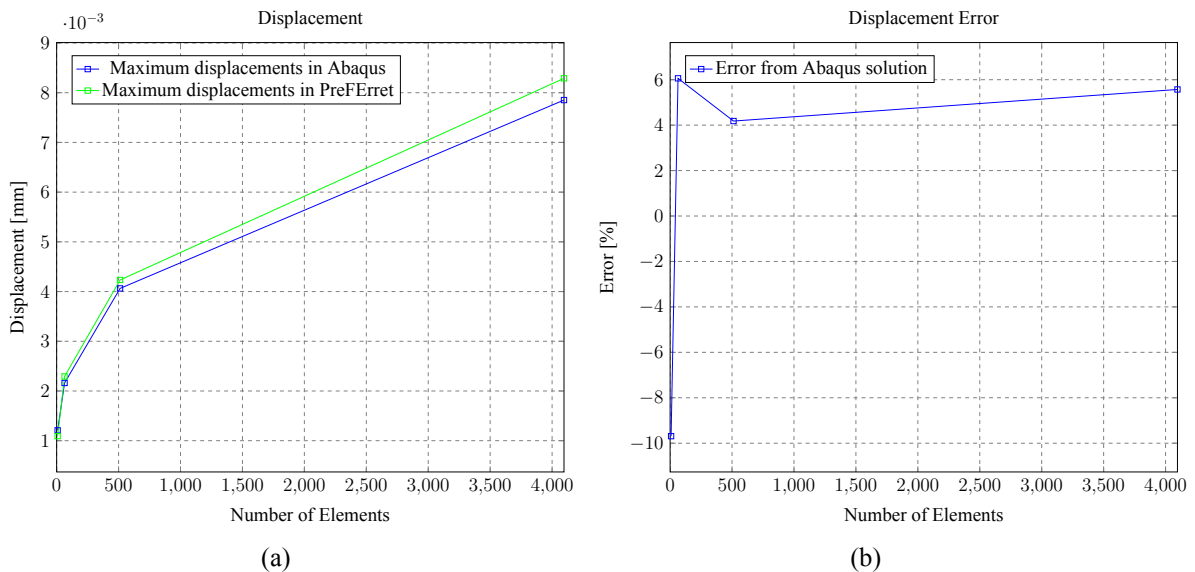


Figure 5.1.4: Comparison of displacements and deviations from the Abaqus solution.

5.1.3 Discussion

From Table 5.1.2, it is clear that the displacements from PreFERret are at a reasonable and expected level. Even though the absolute errors are minor, the relative errors will be quite large as the displacements are small for this load case. For the upper node, the displacement converges somewhat towards the solution from Abaqus but also goes beyond this value. Two mesh structures give excellent values compared to the analytical solution, but this trend is inconsistent. For the middle node, the displacements are most accurate at the coarsest mesh and seem to converge to a specific value quite lower than both the Abaqus and analytical solution. This development also occurs for the lower node. In other words, PreFERret does not give a good solution for displacements in the nodes, as these appear neither stable nor converged. The possibility of mistakes in sorting the elements' nodes may lead to this result. Therefore, looking at the maximum displacement benchmark in Table 5.1.3 is a better way to confirm the results from this plugin. However, this method is irrelevant for comparison to the analytical solution due to singularities that appear by defining point loads at particular nodes.

The displacements were more comparable when the meshes in PreFERret and Abaqus were similar. For the coarsest mesh, the solution from PreFERret was the stiffest one. However, the Abaqus solution was the stiffest for the other mesh sizes. The error's absolute value decays for each mesh refinement, from 9.69% to 5.57 %, with a slight exception for the 8x8x8 mesh. However, as these relative displacement errors are quite low, these results are evaluated as able to confirm the PreFERret computations. The problem seems to be about getting complete coherence between the nodes and their displacement values. In addition, the PreFERret shows a high increase in computational time with mesh refinement. The plugin requires hours to compute the solutions for a relatively small number of elements. A study to find the errors and possible improvements should be done. Still, the authors proceed to the already-made FERret plugin to solve the more significant problems related to this thesis' research question.

5.2 Case Study 2: Verifying the Development towards Orthotropic Materials

To assess the effectiveness of incorporating orthotropic materials into FERret's development, a case study on the displacement of a timber cantilever beam is being conducted. A series of measurements are done to evaluate how the solution converges as the mesh is refined. Refining the mesh should reduce the errors and converge towards the solution obtained in Abaqus, where a highly dense mesh is used. The displacements are measured at a point on the bottom end of the beam, as this is the spot where the displacements will be the largest. The point is located in the middle of the width of the beam.

The geometry in this case study, shown in Figure 5.2.1, is a more traditional cantilever beam with a slender shape and a rectangular cross-section, but with the same length as in the first case study. The grain direction of the timber spans along the longitudinal direction of the beam. Therefore, this high and slender cantilever shape is desirable to highlight the material's strong direction. The geometry and load values are presented in Table 5.2.1. As the material for this case study is timber, the orthotropic material requires more engineering constants to be determined. These constants are given in Table 5.2.2 and represent typical values for a Nordic Spruce (Malo, 2021). An illustration

of the longitudinal (L), radial (R), and tangential (T) directions are shown in Figure 5.2.2. The illustration also defines the correlation between the axis systems L, R, and T, and 1, 2, and 3, where the latter system is used further in this thesis.

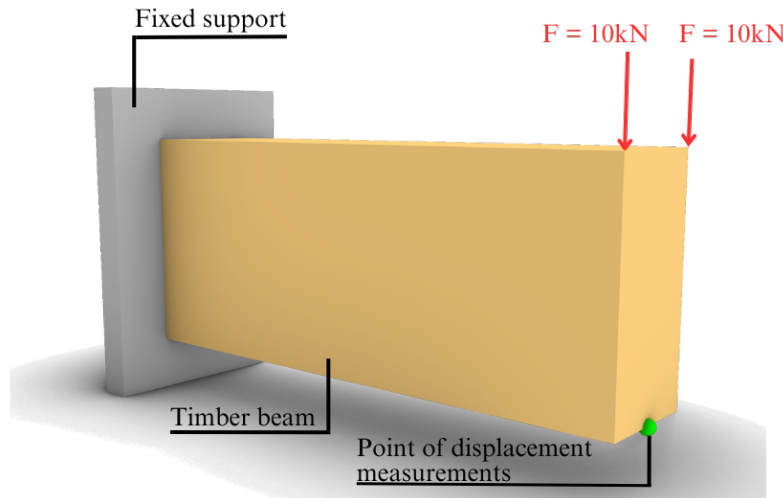


Figure 5.2.1: Geometry and load case for Case Study 2.

| Width | Height | Length | \sum Force | Boundary Condition |
|--------|--------|----------|--------------|--------------------|
| 200 mm | 400 mm | 1 000 mm | 20 kN | Fixed |

Table 5.2.1: Geometry and load values for Case Study 2.

| E_{11} | E_{22} | E_{33} | ν_{12} | ν_{13} | ν_{23} | G_{12} | G_{13} | G_{23} |
|----------|----------|----------|------------|------------|------------|----------|----------|----------|
| 10 000 | 800 | 400 | 0.5 | 0.6 | 0.6 | 600 | 600 | 30 |

Table 5.2.2: Material properties for Case Study 2.

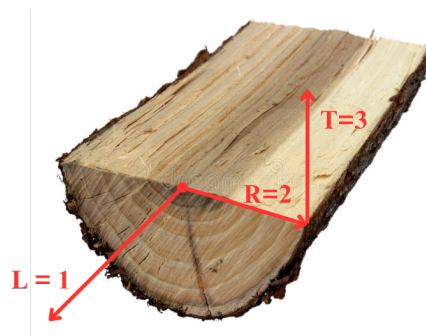


Figure 5.2.2: The three orthogonal axes in timber: Longitudinal, Radial, and Tangential.

As timber properties vary significantly between the axes, it is desirable to be able to set the directions of the material. Manipulating the stiffness matrices with rotational matrices goes beyond the scope of this thesis. However, the opportunity of determining the strong axis should be considered. In other words, the user should be able to decide if the model spans along either the x-, y- or z-axis. Even though it is quite a limited choice to be restricted to these orthogonal axes, some additional freedom is provided. The transformation also works as a confirmation of the theory in Equation 2.1.33. As the solver interprets the x-axis as the main axis, the 1-direction is set along the x-axis by default. Suppose it is desirable to model the beam along, e.g., the y-axis. In that case,

the relation between the Poisson's ratios can be utilized to manipulate the material properties so that the solver interprets the y-axis as the strong axis. This transformation example is shown in Table 5.2.3.

| E_{xx} | E_{yy} | E_{zz} | ν_{xy} | ν_{xz} | ν_{yz} | G_{xy} | G_{xz} | G_{yz} |
|--|----------|----------|---|------------|------------|----------|----------|----------|
| E_{11} | E_{22} | E_{33} | ν_{12} | ν_{13} | ν_{23} | G_{12} | G_{13} | G_{23} |
| 10 000 | 800 | 400 | 0.5 | 0.6 | 0.6 | 600 | 600 | 30 |
| ↓ Transforming y-axis to strong axis ↓ | | | | | | | | |
| E_{22} | E_{11} | E_{33} | $\nu_{21} = \nu_{12} \cdot \frac{E_{22}}{E_{11}}$ | ν_{23} | ν_{13} | G_{12} | G_{13} | G_{23} |
| 800 | 10 000 | 400 | 0.04 | 0.6 | 0.6 | 600 | 600 | 30 |

Table 5.2.3: Example of how the material properties can be manipulated to change the direction of the strong axis, here along the y-axis.

5.2.1 Method

Grasshopper

A flowchart for the GH file of Case Study 2 is provided below. Two different BREPs have been created for this benchmark to compare the results. One method involves creating the entire geometry in a BREP, which is directly inputted to the chosen meshing tool. For hexahedral elements, the SimplyfyMesh-component is used to make the mesh. For the tetrahedral elements, Tetrino is used. This results in a free mesh where the element size and orientation vary significantly. There are also more nodes along the geometry's outer edges than the interior. The second alternative involves a custom code that iterates a desired number of hexahedral boxes in the x, y, and z directions to represent the geometry. The division is based on dimensions in each direction and the desired number of subdivisions.

For the hexahedral element, these BREPs are sent through the SimplyfyMesh-component to make the mesh. For the tetrahedral ones, the list of BREPs is inputted to Tetrino, where six tetrahedral elements are generated per hexahedral BREP. The mesh is extracted from the chosen meshing tool and input into FEM Load, Boundary On Points, MaterialOrtho, and FEM Solver. The vertices of the mesh are also extracted from Tetrino. The load and boundary points are picked with a small code in Grasshopper, and assigned to the FEM Load and Boundary On Points, respectively. Subsequently, all the information is passed into FERret FEM Solver, which calculates the solution. For a detailed description of the entire setup, please refer to the GH file in Appendix A. A video visualizing this case study is available at [YouTube](#), see Appendix B.

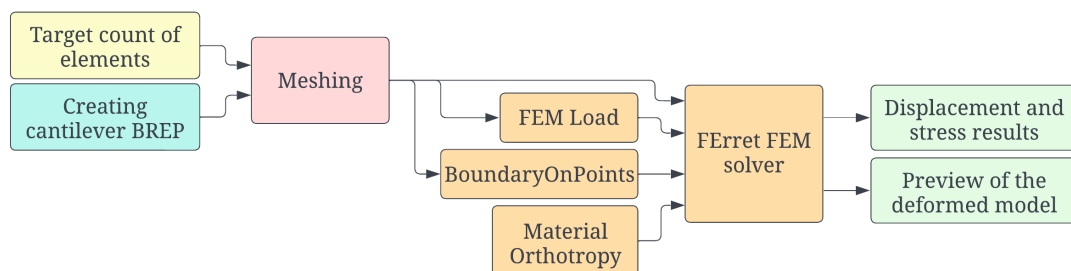


Figure 5.2.3: Flowchart of Grasshopper script for Case Study 2.

Abaqus

As the mesh structure is challenging to obtain identically in both Grasshopper and Abaqus, the authors chose to create a model in Abaqus consisting of a highly dense mesh to compare with the results from Grasshopper. Similarly to the first case study, the Abaqus solution was used to compare the results from Grasshopper while the mesh was gradually refined. The procedure in Abaqus was, therefore, quite similar to before. However, a new material had to be defined. The timber was described with the nine parameters from Table 5.2.2. Further, the main axis was defined along the longitudinal direction of the beam, and from there, the procedure was equal to the one in Case Study 1. The mesh comprises 57 682 elements.

5.2.2 Results

As in Case Study 1, the beam's height-to-length ratio exceeds 1/10. Thus, shear deformations must be considered to calculate the analytical solution to the structural problem. To calculate the displacement at the end of the beam, Equation 5.2.1 is used. The equation is equal to the one in Case Study 1, but since the material is now timber, some variable values have changed and are listed below.

$$w_{\max} = \frac{PL}{\kappa GA} + \frac{PL^3}{3EI} = -1.11520\text{mm} \quad (5.2.1)$$

- $P = 20\text{kN}$, representing the point loads acting on the beam.
- $L = 1\,000\text{mm}$, representing the length of the beam.
- $\kappa = \frac{10(1+\nu)}{12+11\nu}$, the Timoshenko shear coefficient.
- $G = G_{12} = 600\text{MPa}$, the shear modulus.
- $A = b \cdot h = 80\,000\text{mm}^2$, the cross-sectional area.
- $E = E_1 = 10\,000\text{MPa}$, the Young's modulus.
- $I = \frac{b \cdot h^3}{12}$, the second moment of area for the cross-section, where b and h represents the height and the width.

The benchmark results are provided in Table 5.2.4 and Table 5.2.5. Displacement patterns from Grasshopper and Abaqus are illustrated in Figure 5.2.6. The results are also presented by the graphs in Figure 5.2.4 and Figure 5.2.5. In the graphs, one can see the convergence behavior of the displacements and errors, as well as the increase of computational costs as the number of elements increases. The errors are defined in the same way as in Case Study 1, refer Equation 5.1.2.

In Table 5.2.6, the results from the change in direction of the strong axis are presented, here along the y-axis. A video illustrating the procedure of modifying the strong axis is available at [YouTube](#), see Appendix B.

| Number of Elements | Displ. [mm] | Error from Abaqus Solution | Relative Error [%] | Error from Analytical solution | Relative Error [%] | FErret Time [s] |
|--------------------|-------------|----------------------------|--------------------|--------------------------------|--------------------|-----------------|
| 48 | -0,939 | 62.2E-03 | -6,21 | 176E-03 | -15,8 | 0,0330 |
| 60 | -0,941 | 60.0E-03 | -5,99 | 174E-03 | -15,6 | 0,0510 |
| 88 | -0,971 | 30.6E-03 | -3,05 | 145E-03 | -13,0 | 0,0680 |
| 195 | -0,986 | 14.8E-03 | -1,48 | 129E-03 | -11,6 | 0,194 |
| 384 | -0,995 | 6,29E-03 | -0,628 | 120E-03 | -10,8 | 0,550 |
| 504 | -1,000 | 0.929E-03 | -0,0928 | 115E-03 | -10,3 | 0,780 |
| 1 200 | -1,008 | -7.13E-03 | 0,712 | 107E-03 | -9,59 | 3,05 |
| 1 848 | -1,010 | -8.51E-03 | 0,850 | 106E-03 | -9,46 | 6,56 |
| 2 496 | -1,011 | -10.3E-03 | 1,03 | 104E-03 | -9,30 | 11,1 |
| 3 332 | -1,012 | -11.0E-03 | 1,10 | 103E-03 | -9,23 | 18,9 |
| 4 320 | -1,013 | -11.5E-03 | 1,15 | 103E-03 | -9,20 | 34,0 |

Table 5.2.4: Displacements for mesh consisting of hexahedral elements. Compared to the solution from Abaqus with dense mesh giving a displacement $w = -1.0016\text{mm}$.

| Number of Elements | Displ. [mm] | Error from Abaqus Solution | Relative Error [%] | Error from Analytical solution | Relative Error [%] | FErret Time [s] |
|--------------------|-------------|----------------------------|--------------------|--------------------------------|--------------------|-----------------|
| 288 | -0,837 | 165E-03 | -16,4 | 279E-03 | -25,0 | 0,0890 |
| 528 | -0,899 | 102E-03 | -10,2 | 216E-03 | -19,4 | 0,252 |
| 2 304 | -0,953 | 47.8E-03 | -4,77 | 162E-03 | -14,5 | 2,54 |
| 4 032 | -0,979 | 22.3E-03 | -2,23 | 136E-03 | -12,2 | 6,94 |
| 5 940 | -0,986 | 15.5E-03 | -1,55 | 130E-03 | -11,6 | 14,3 |
| 7 200 | -0,991 | 10.1E-03 | -1,01 | 124E-03 | -11,1 | 20,0 |
| 10 296 | -0,995 | 6,29E-03 | -0,628 | 120E-03 | -10,8 | 40,0 |
| 12 960 | -0,998 | 2,79E-03 | -0,279 | 117E-03 | -10,5 | 62,3 |
| 14 976 | -1,001 | 0.317E-03 | -0,0317 | 114E-03 | -10,3 | 80,7 |
| 20 580 | -1,003 | -1,83E-03 | 0,182 | 112E-03 | -10,1 | 152 |
| 23 520 | -1,003 | -1,71E-03 | 0,171 | 112E-03 | -10,1 | 199 |

Table 5.2.5: Displacements for mesh consisting of tetrahedral elements. Compared to the solution from Abaqus with dense mesh giving a displacement $w = -1.0016\text{mm}$.

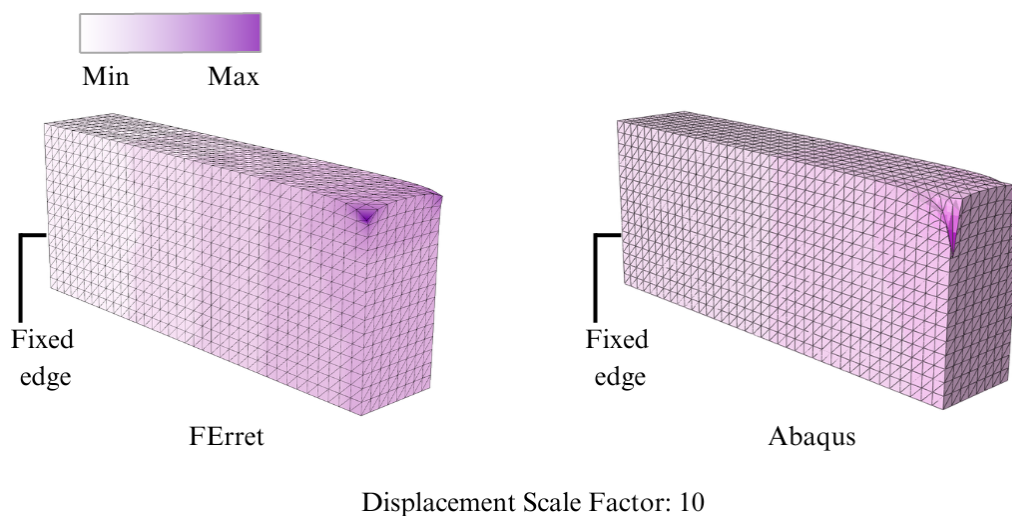


Figure 5.2.6: Displacement pattern for Case Study 2.

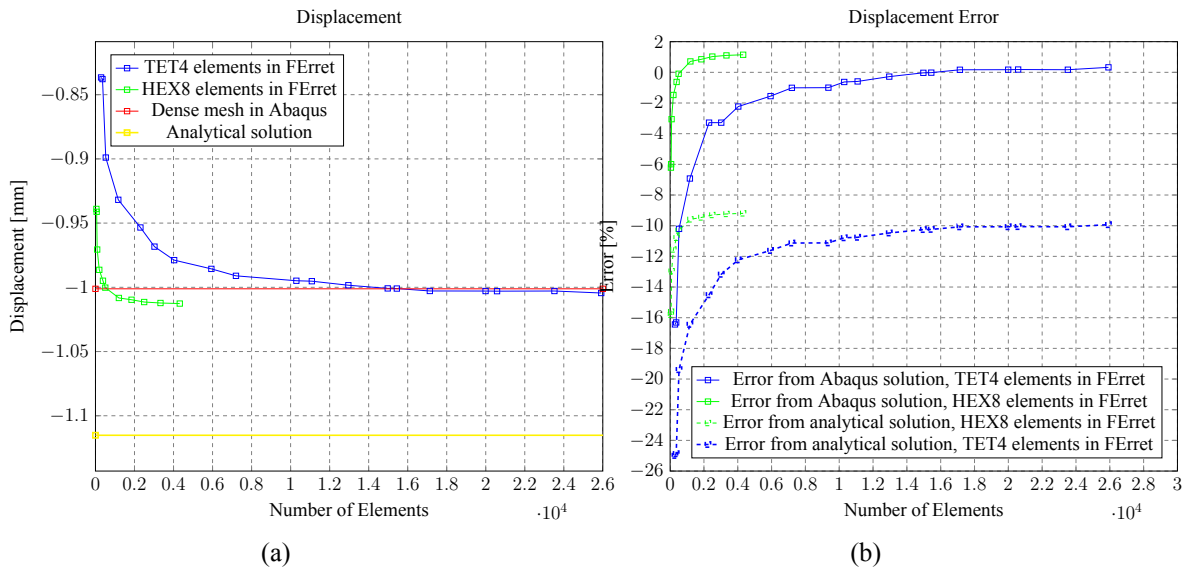


Figure 5.2.4: Comparison of displacements and errors from Abaqus and analytical solution.

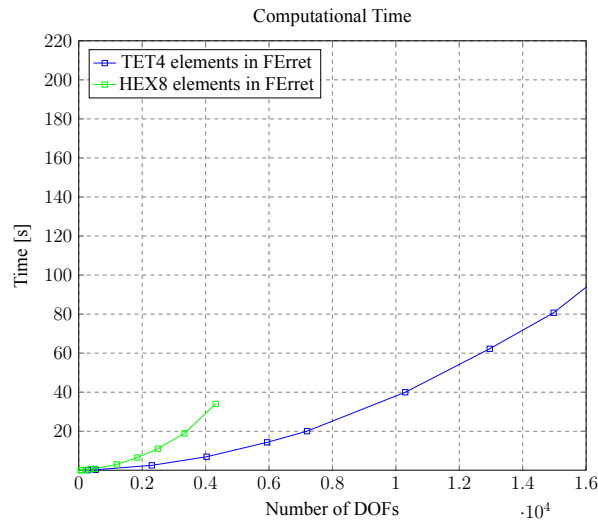


Figure 5.2.5: Computational time as a function of the number of DOFs in the system.

| Number of elements | x-axis | y-axis |
|--------------------|--------|--------|
| 288 | -0.837 | -0.837 |
| 4 032 | -0.979 | -0.979 |
| 10 296 | -0.995 | -0.995 |
| 23 520 | -1.003 | -1.003 |

Table 5.2.6: Comparison of displacements with the strong axis spanning along the x- and y-axis.

5.2.3 Discussion

The results above regarding displacements will be used to evaluate the solidity of the development of the FERret plugin for handling orthotropic materials. Due to the difference in the number of nodes between the hexahedral and the tetrahedral elements, the FERret plugin could not handle as many hexahedral elements as tetrahedra. There are some limitations in the C# language, in this case, related to exceeding the maximum length of an array. Therefore, there is a maximum amount of DOFs the plugin can handle due to this limitation. However, the trend is similar for the two benchmarks. The solution from FERret converges towards the Abaqus solution from the upper side and goes a bit beyond. The errors are minor in both an absolute and relative sense, with relative errors of 1.15% for the hexahedra and 0.17% for the tetrahedra. Comparing the results for the same number of elements for the two-element geometries, the hexahedra are more accurate but at a higher cost in a temporal sense. The hexahedra need about 34 seconds to provide the results, while the tetrahedra only need about seven seconds for the same amount of elements.

The graphs show these behaviors with clearly converging curves, especially for the tetrahedra. Despite the relatively low number of elements for the hexahedral mesh, it is clear that the solution converges towards a number a bit lower than what the tetrahedral gives. The tetrahedra create a solution that nearly aligns with the line for the Abaqus solution with a dense mesh. It is also worth noticing that a satisfactory value for the displacements is found for both element geometries for an element number significantly less than the largest amounts used in this benchmark. Only 384 elements are needed for the hexahedra to obtain a solution with a relative error of less than one percent. Similarly, for the tetrahedra, a mesh containing 7200 elements gives a solution with a relative error of one percent compared to the Abaqus solution.

The time consumption for these numbers of elements is 0.55 and 20 seconds, respectively. This shows that the hexahedra are far more efficient for computing the solution, as the convergence rate is clearly larger than for the tetrahedra. However, if the goal is to achieve the most accurate solution possible, tetrahedra should be used, but they come with a significant computational cost. Compared to the analytical solution, the displacements from both the FERret solution and the Abaqus solution converge to a value about 10% lower. This behavior might be explained by the complexity of the timber, as more material properties than the ones defined in the strong direction will influence how the beam deflects. This could explain the slightly conservative solution obtained by the analytical method.

For the rotation of the strong axis presented in Table 5.2.6, the result is capable of confirming the theory in Equation 2.1.33, as well as the method described in Table 5.2.3. With exactly equal displacement values for the two measurements, it is clear that there is an opportunity of choosing which axis the orthotropic material should span along. This makes the plugin more suitable and versatile as the user is not restricted to modeling the structure along one specific axis. This finding opens an opportunity of expanding the potential of the FERret plugin, which is presented and discussed further in the main discussion of this thesis.

5.3 Case Study 3: Verifying the Development towards Combining Materials

To verify that the method in 4.4 works, a steel dowel is inserted into a timber beam with the same geometry used in Case Study 2. Two point loads are applied to each side of the dowel, intending to stretch the timber beam along its grain direction. The geometry and load case are illustrated in Figure 5.3.1 and listed in Table 5.3.1. The material properties for both the timber beam and the steel dowel are listed in Table 5.3.2. For simplicity, the contact property between the dowel and the timber is considered glued in this case study. In other words, there is complete interaction between the two parts.

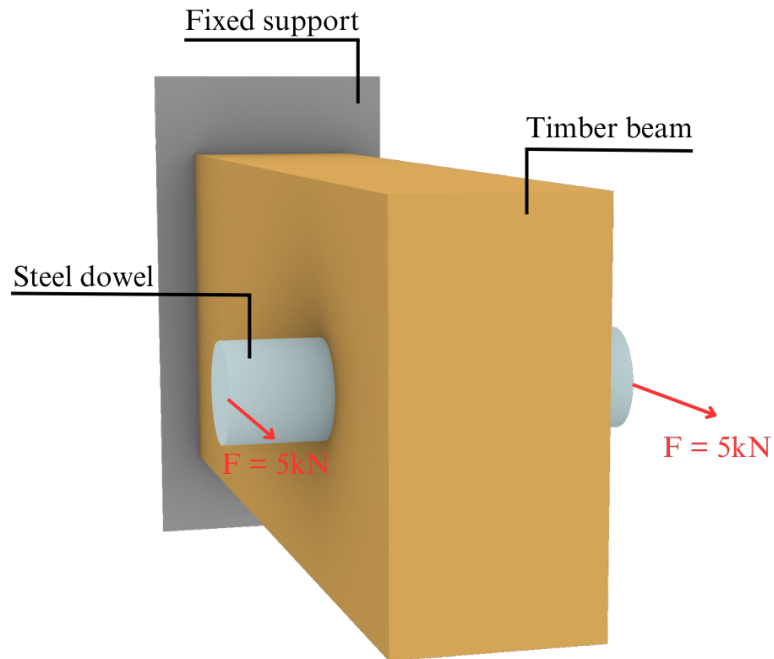


Figure 5.3.1: Geometry and load case for Case Study 3.

| Beam Width | Beam Height | Beam Length | Dowel Radius | Dowel Length | Σ Force |
|------------|-------------|-------------|--------------|--------------|----------------|
| 200 mm | 400 mm | 1 000 mm | 50 mm | 400 mm | 10 kN |

Table 5.3.1: Geometry and load values for Case Study 3.

| E_{11} | E_{22} | E_{33} | ν_{12} | ν_{13} | ν_{23} | G_{12} | G_{13} | G_{23} | E_{dowel} | ν_{dowel} |
|----------|----------|----------|------------|------------|------------|----------|----------|----------|-------------|---------------|
| 10 000 | 800 | 400 | 0.5 | 0.6 | 0.6 | 600 | 600 | 30 | 210 000 | 0.3 |

Table 5.3.2: Material properties for the timber beam and the steel dowel in Case Study 3.

Some approximations had to be made to obtain full compatibility between the two parts regarding meshing. A small script was made to define which part of the global mesh represents the dowel and which part represents the timber beam. An integer value is given as a parameter for this script to decide how strictly the elements should be chosen. This integer represents how many of the element's nodes should be inside the dowel's volume to be considered part of the dowel mesh. This means that if the input is the number 1, only one node from any element must be inside the volume representing the dowel to be considered part of the dowel mesh. On the other side, if the integer is 3, three nodes from an element must be inside this volume for the element to be included in the dowel mesh. This integer value has to be chosen by evaluating the resulting dowel.

An example is illustrated in Figure 5.3.2 with various values for this integer. The elements cannot precisely represent the cylinder using the Iguana plugin, but the representation is acceptable by refining the mesh. However, with the Tetrino plugin, the elements perfectly represent the cylinder by setting the input integer to 3. This variation in dowel representation is illustrated in Figure 5.3.2 and Figure 5.3.3, where the integer parameter is denoted i . Due to problems with the Tetrino meshing method, Iguana is chosen with a mesh size of 25, and the integer is set to 1 to represent the dowel preferably. This gives a slightly larger dowel, but as the dowel is relatively smaller than the timber part, this is evaluated as acceptable.

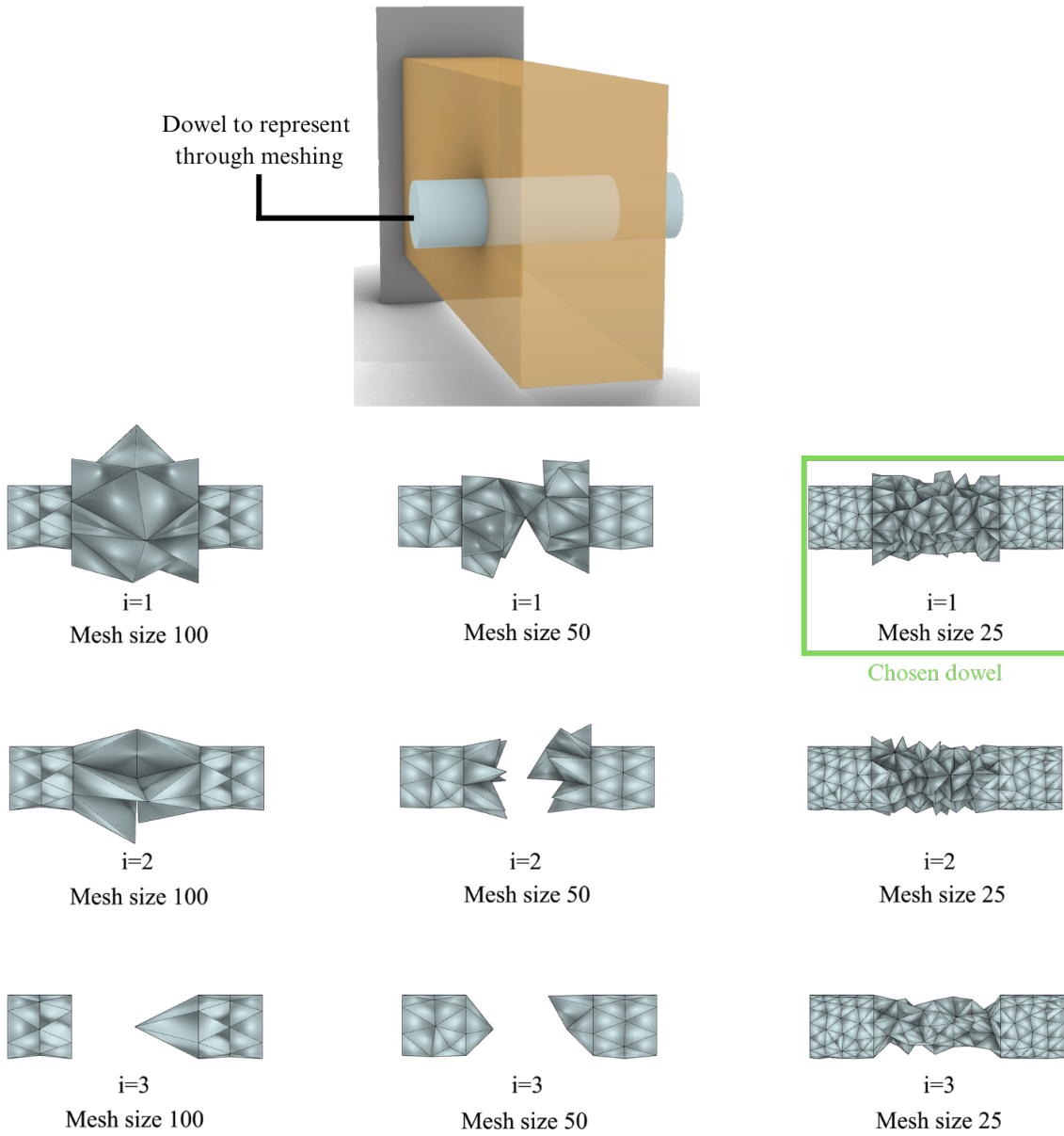


Figure 5.3.2: Approximation of the dowel volume with Iguana.

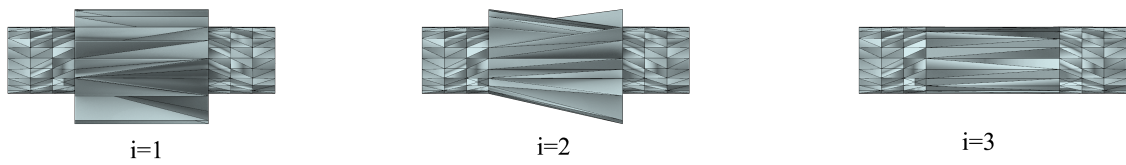


Figure 5.3.3: Approximation of the dowel volume with Tetrino. Mesh size is constant for the three dowels.

5.3.1 Method

The benchmark is based on comparisons between the model from FERret and the model with a dense mesh from Abaqus. The displacements and stresses are the parameters used for comparison. For the displacements, three points on the tip of the beam are chosen to obtain the variation on the end face. Two points in the middle of the beam's cross-section are selected for the stresses, about 500mm and 100mm inside the beam from the fixed support. This point is determined to obtain comparable results from the interior of the beam.

Grasshopper

Figure 5.3.4 shows the process in the Grasshopper script to obtain the results in this case study. The main difference from the previous processes is the definition of parts for the timber beam and steel dowel. This results in a different data structure going through the process, handled with new components as introduced in Section 4.4. For the Grasshopper model, the two parts are defined in a way that makes them fully interact, like they in practice were glued. For a detailed description of the entire setup, please refer to the GH file in Appendix A.

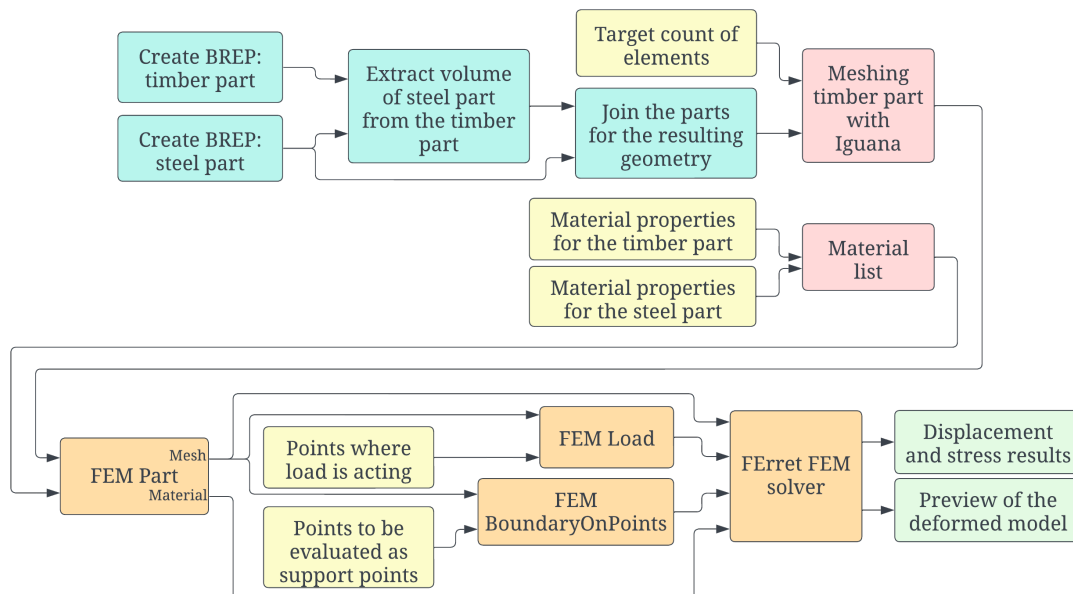


Figure 5.3.4: Flowchart of Grasshopper script for Case Study 3.

Abaqus

The mesh settings in Abaqus were chosen to gain the best results possible with a reasonable computational time. The quadratic tetrahedral element C3D10 was used, with an approximate element size of 15mm for the timber part and 10mm for the steel dowel. Smaller elements in the dowel are preferable for achieving more accurate results at a lower computational time. A constraint property of type Tie was introduced in the Abaqus model to obtain a connection between the timber beam and the steel dowel. The mesh in this case study comprises 130 895 elements.

5.3.2 Results

Displacements

Unlike the structural problems in the two first case studies, obtaining an analytical solution to the situation in this case study is not readily done. Therefore, a model from Abaqus with a highly dense mesh is used as a reference for the displacement measurements. Due to the load direction along the x-axis, the displacement in the x-direction is evaluated in this section since the ones in the y- and z-direction will be minimal.

In Table 5.3.3, the results from the analysis in FERret are compared with the solution gained from Abaqus. The node position is given for each comparison, as well as the absolute error, as defined by Equation 5.1.2, between the FERret solution and the Abaqus solution. These results are also presented visually in Figure 5.3.5, while the displacement patterns are illustrated in Figure 5.3.6.


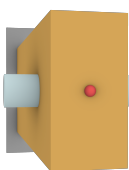

| Point of Measurements | FEM Solver | Number of Elements | Displacement [mm] | Absolute Error [mm] | Relative Error [%] |
|---|------------------------|--------------------|-------------------|---------------------|--------------------|
|  | Abaqus | 130 895 | 0.00509 | – | – |
| | FERret, Iguana mesh | 319 | 0.00726 | 0.00217 | 42.7 |
| | | 3 059 | 0.00570 | 0.000609 | 12.0 |
| | | 9 668 | 0.00575 | 0.000658 | 12.9 |
| | | 23 918 | 0.00542 | 0.000334 | 6.57 |
|  | Abaqus | 130 895 | 0.0140 | – | – |
| | FERret, Iguana mesh | 319 | 0.00913 | -0.00489 | -34.9 |
| | | 3 059 | 0.00984 | -0.00418 | -29.8 |
| | | 9 668 | 0.0103 | -0.00376 | -26.8 |
| | | 23 918 | 0.0106 | -0.00338 | -24.1 |
|  | Abaqus | 130 895 | 0.00510 | – | – |
| | FERret, Iguana mesh | 319 | 0.00744 | 0.00234 | 45.9 |
| | | 3 059 | 0.00578 | 0.000680 | 13.3 |
| | | 9 668 | 0.00577 | 0.000669 | 13.1 |
| | | 23 918 | 0.00540 | 0.000306 | 6.00 |

Table 5.3.3: Benchmark of displacements in the x-direction for three node positions.

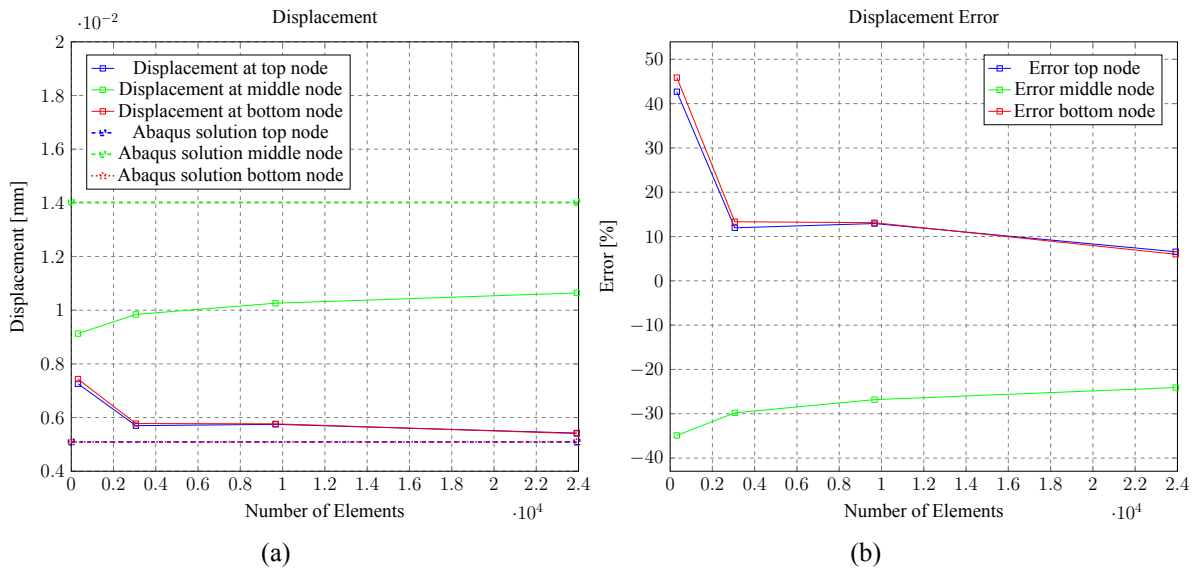


Figure 5.3.5: Comparison of displacements and errors compared to the Abaqus solution.

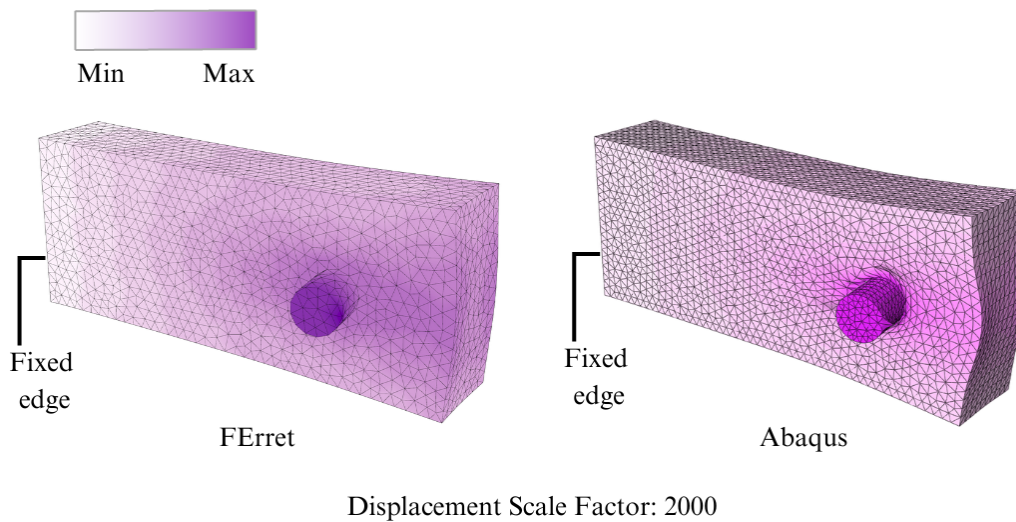


Figure 5.3.6: Displacement pattern for Case Study 3.

Stresses

All six stress components are compared to obtain the most comparable numbers between FERret and Abaqus. These components are the three normal stresses σ_x , σ_y , and σ_z , as well as the three shear stresses σ_{xy} , σ_{xz} , and σ_{yz} . The stresses are further denoted according to the main axes as S_{11} , S_{22} , S_{33} , S_{12} , S_{13} and S_{23} , respectively. The results are presented in Table 5.3.4 and Table 5.3.5 for stresses 50mm and 100mm inside the beam, respectively.

| FEM Solver | Node Position | S_{11} | S_{22} | S_{33} | S_{12} | S_{13} | S_{23} |
|----------------|----------------|-----------|-----------|------------|--------------|-------------|-------------|
| FERret | (48, 109, 200) | 128E-03 | 0.896E-03 | 1.98E-03 | -0.00505E-03 | -0.0510E-03 | -0.0230E-03 |
| Abaqus | (51, 102, 198) | 133E-03 | 3.71E-03 | 2.74E-03 | -0.0517E-03 | 0.110E-03 | 0.00374E-03 |
| Abs. error | | -4.62E-03 | -2.81E-03 | -0.767E-03 | 0.0466E-03 | -0.161E-03 | -0.0267E-03 |
| Rel. error [%] | | -3.47 | -75.8 | -28.0 | -90.2 | -147 | -715 |

Table 5.3.4: Stresses 50mm into the beam from the fixed support. The results are provided with both absolute and relative errors.

| FEM Solver | Node Position | S_{11} | S_{22} | S_{33} | S_{12} | S_{13} | S_{23} |
|----------------|-----------------|-----------|-------------|------------|-------------|------------|-------------|
| FERret | (106, 109, 209) | 130E-03 | -0.0570E-03 | 0.337E-03 | 0.0160E-03 | -0.560E-03 | -0.0520E-03 |
| Abaqus | (101, 102, 195) | 136E-03 | 1.36E-03 | 0.850E-03 | 0.00414E-03 | 0.217E-03 | 0.00713E-03 |
| Abs. error | | -5.77E-03 | -1.41E-03 | -0.513E-03 | 0.0119E-03 | -0.831E-03 | -0.0591E-03 |
| Rel. error [%] | | -4.23 | -104 | -60.3 | 287 | -307 | -830 |

Table 5.3.5: Stresses 100mm into the beam from the fixed support. The results are provided with both absolute and relative errors.

The stress maps are shown in Figure 5.3.7 for the three normal stresses to confirm the stress distribution throughout the model. The figures to the left are from the FERret analysis, while those to the right are from Abaqus.

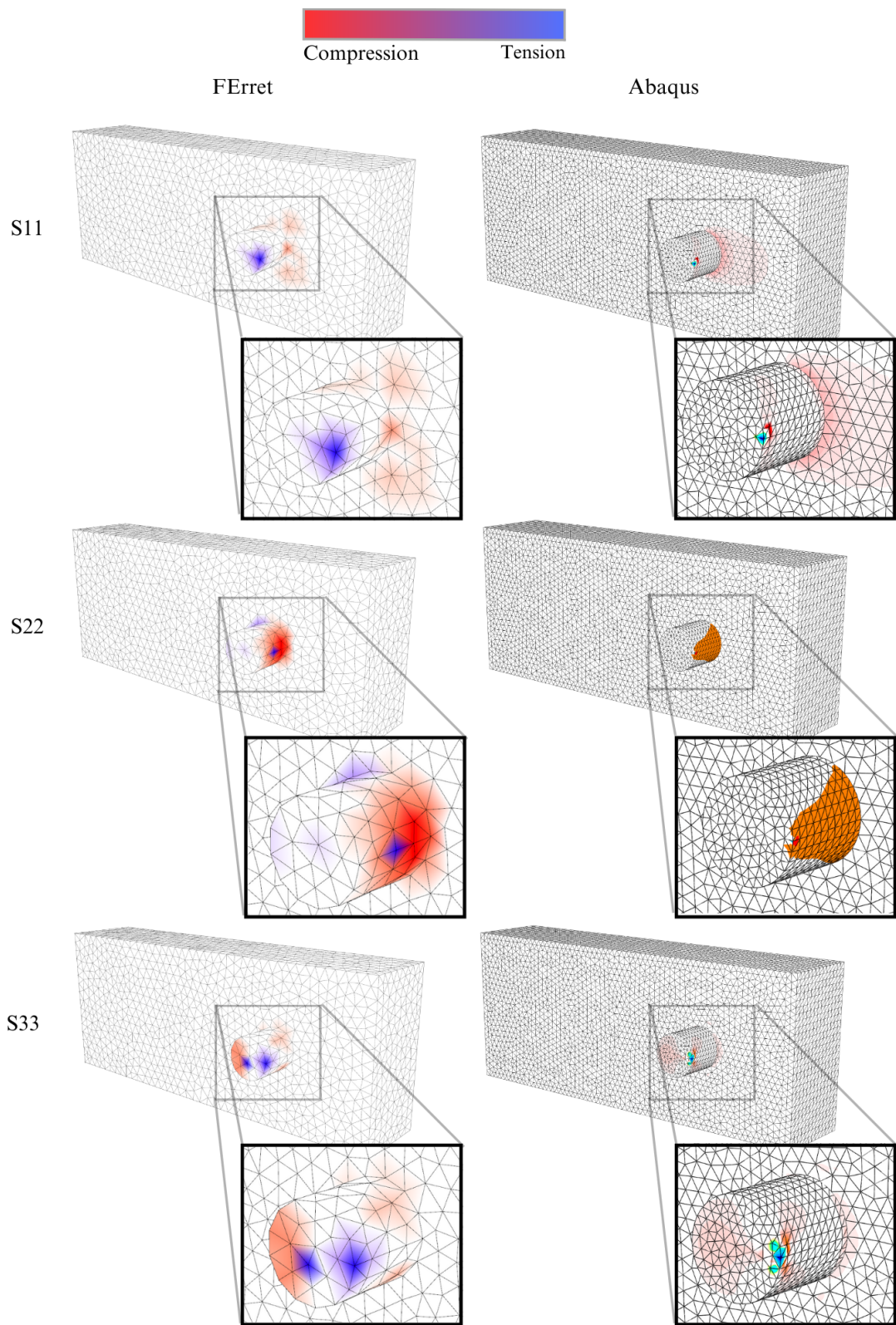


Figure 5.3.7: Stress distributions for the normal stresses.

5.3.3 Discussion

From the benchmark on both displacements and stresses, it is clear that the FERret plugin is able to analyze a structural problem that includes more than one material after the development in this case study was done. The results are not equal, which is not expected due to the challenges in representing the individual parts of the mesh, as mentioned above. The displacement values are clearly converging towards solutions quite close to the solution obtained with Abaqus. The nodes on the top and bottom of the beam show quite precise and similar values, which reflects the symmetry in the structure about the middle of the beam height.

However, it is worth noticing that the relative error is larger for the mid-node. This is also visible in Figure 5.3.6, where it is clear that the dowel deforms the middle of the beam to a larger extent in the Abaqus model. This could be due to the interaction definitions between the timber beam and the steel dowel. As explained in Section 5.3.1, the FERret model defines the parts as fully interacted, while in the Abaqus model, the timber beam is defined as a constraint for the dowel to move. This can lead to different behaviors around the dowel, as all the timber around the dowel will restrain the load in the FERret model, while for the Abaqus model, only the part of the timber subjected to compression will be restraining the motion. This would lead to higher displacements around the dowel for the Abaqus model, which matches well with the results. It also explains the slightly higher displacements at the top and bottom nodes for the FERret model; as for this model, the displacements are distributed more evenly over the beam.

For the stresses, there are good correlations between the dominating stresses, here mainly in the x-direction. As the other stresses are so small, the errors are comparably high even though the absolute error is low. In addition, the selected points are a source of error since the coordinates do not coincide due to the different mesh structures inside the volumes. As seen on both Figure 5.3.6 and 5.3.7, the displacement and stress patterns are quite similar for the solutions from the two different finite element solvers. This is a good indication that the FERret plugin correctly calculates the overall behavior of the structure.

By this, it seems that the challenge to improve this development is the meshing. Representing the parts more precisely will improve the quality of the solution. The problems related to the meshing will be further discussed in the main discussion section of this thesis. A video of the structure from this case study is available at [YouTube](#), see Appendix B.

5.4 Case Study 4: Analyzing Timber Elements from 3D-scanned House with FERret

The environment has become an important topic related to climate and CO₂ emissions. Human-induced climate change is now a higher priority on the political agenda because we are concerned about its effects. The government's main objective is for Norway to become carbon-neutral by 2030. This will pose significant challenges for all sectors of society (TreFokus, 2017). In 2020 the European Union required that 70% of all construction industry waste should be material recycled, a goal that is hard to accomplish. Today this industry is responsible for 25% of all waste in Norway, and approximately 40% of this is reclaimed in new materials (Laake, 2020). Reclaiming materials presents several challenges, including establishing an inventory of available reusable materials,

determining their suitability for specific constructions, assessing their capacity, and communicating the effect of reusing materials. Several solutions are made to accomplish this challenge.

The case study presented in this section is based on an under-development housing project. It is a small two-story building built only from reclaimed timber elements. The design and analysis of the highly unique structure required bespoke computational methods, which creates an interactive process between architecture and structural timber engineering. The project is called *Sletteløkka* and will be constructed in the autumn of 2023.

To select the reusable timber elements, an algorithm is made that takes in a list of 3D-scanned objects and saves their length and cross-section area. The elements from the building are also inserted as an input, where the component iterates through the bank of 3D-scanned elements and returns the elements that have or could have the desired length and cross-section. The cross-sections in the structural model are then updated, and Karamba 3D is utilized to analyze the structure.

The paper about this concept is not yet published but is scheduled for publication at the WCTE conference in June 2023. The solution could be improved since the cross-section of the reclaimed material is not fully used. By identifying the smallest cross-section area of the 3D-scanned beam, this area is added to the structural model. A more accurate approach is, therefore, to add the actual geometry and do the structural calculations using FERret. This also reduces the design time, as the model can be automatically updated as structural elements are replaced with reused components.

Adding FERret to the given algorithm enables a much more detailed design process where all the imperfections can be considered. The 3D-scanned geometry can be opened in Rhino and connected to FERret in Grasshopper. In this way, the capacity of various building components can be evaluated, and new solutions using reused materials can be tested to meet the current building requirements. In this case study, the building from the paper is used to test how FERret handles 3D-scanned elements. A 3D model of the structure is shown in Figure 5.4.1(a).

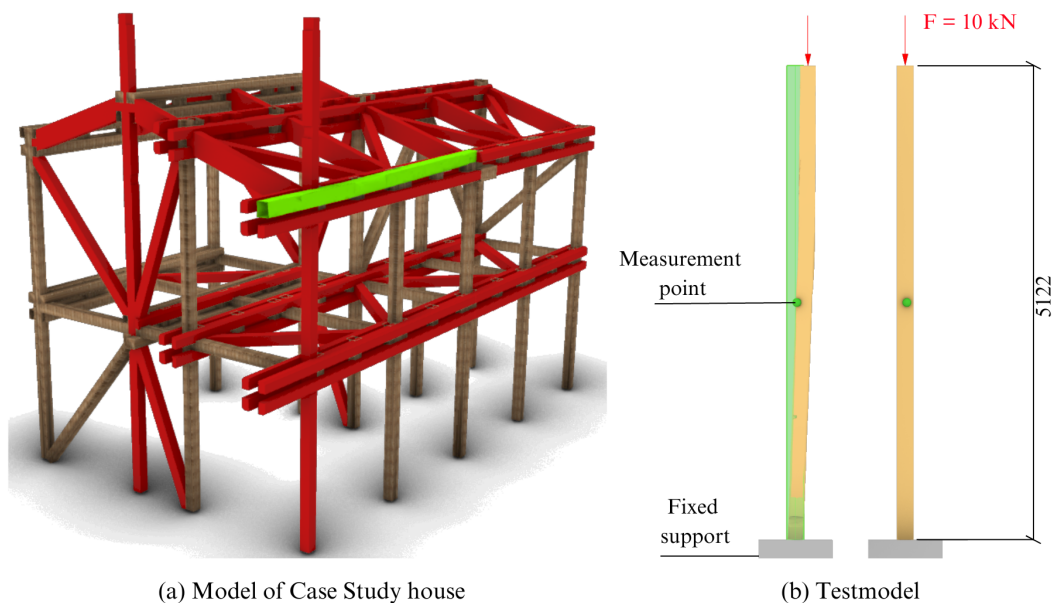


Figure 5.4.1: Illustration of test-object

To illustrate the solution’s effectiveness, a deformed building component has been selected, marked in green in the model. The beam has been rotated and tested as a column. This makes it possible to test it in compression to enable a straightforward assessment of displacement and stress distribution. Hereafter, the building component will be referred to as a column. Figure 5.4.1(b) shows the setup for the experiment.

Due to some irregularities at the top and bottom of the column, the ends of the column are cut. At the bottom of the column, all end vertices are constrained in x, y, and z directions to represent fixed support. At the top, the column is loaded with point loads at all end vertices. These loads together correspond to 10 kN. For comparison purposes, a perfect column is also modeled and shown to the right in figure 5.4.1(b). The green sphere in the middle of both columns shows the measurement point for displacement and stresses. Properties regarding geometry, load case, boundary conditions, and material used in this case study are summarized in Table 5.4.1 and 5.4.2.

| Width | Height | Length | \sum Load | Boundary Condition |
|--------|--------|----------|-------------|--------------------|
| 178 mm | 178 mm | 5 122 mm | 10 kN | Fixed |

Table 5.4.1: Geometry for the perfect column, and load and boundary conditions for both columns.

| E_{11} | E_{22} | E_{33} | ν_{12} | ν_{13} | ν_{23} | G_{12} | G_{13} | G_{23} |
|----------|----------|----------|------------|------------|------------|----------|----------|----------|
| 10 000 | 800 | 400 | 0.5 | 0.6 | 0.6 | 600 | 600 | 30 |

Table 5.4.2: Material properties for Case Study 4.

5.4.1 Method

Grasshopper

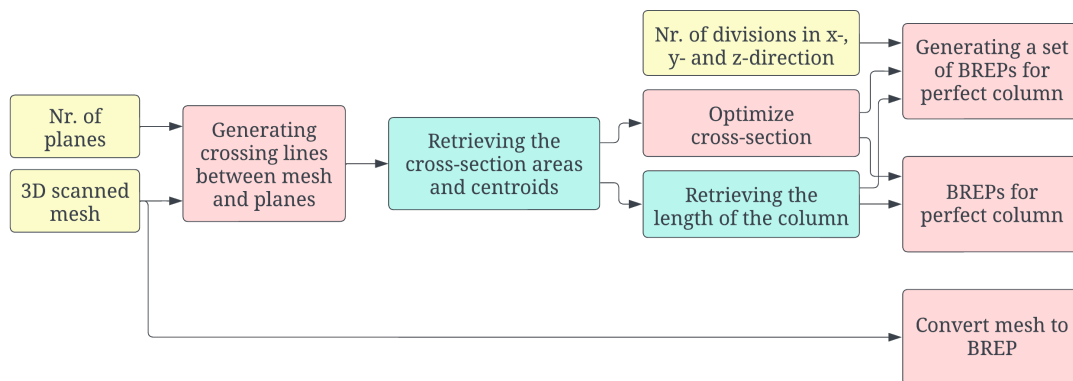


Figure 5.4.2: Flowchart for modeling the three different columns.

A flowchart for generating the three different columns is illustrated in Figure 5.4.2. The input to the Grasshopper script is the selected mesh from the 3D model and a given number of planes. Utilizing this data, several intersection lines are generated between the 3D-scanned column and the specified number of planes in the input. This information calculates the area at each intersection point and

the cross-section's centroid. By examining numerous intersection points and creating a polyline along the column center, the length can be determined, and the average area of the 3D-scanned column can be calculated. Using the optimization tool Galapagos in Grasshopper, the lengths of the cross-section are set to be as close as possible to the average area of the 3D-scanned column. With this information, the two perfect columns with structured and free meshing can be generated.

The custom code created for Case Study 2 is utilized for the structured mesh. The boxes' size and number are adjusted based on the cross-section, length, and input subdivision. For the 3D-scanned column, the BullAnt plugin in Grasshopper is used, which converts the input mesh into a BREP. The same setup as in Case Study 2, illustrated in Figure 5.2.3, is used for the remaining part.

Due to the difficulty in quality assuring the results obtained from analyzing the 3D-scanned column, a benchmark has been conducted on the perfect column. This analysis aims to assess the accuracy and performance of the meshing tool Tetrino. This also allows for verification that the experimental setup yields sensible solutions. An Abaqus model has been created to evaluate the solutions computed by the FERret. Various types of mesh from Tetrino are illustrated in Figure 5.4.3.

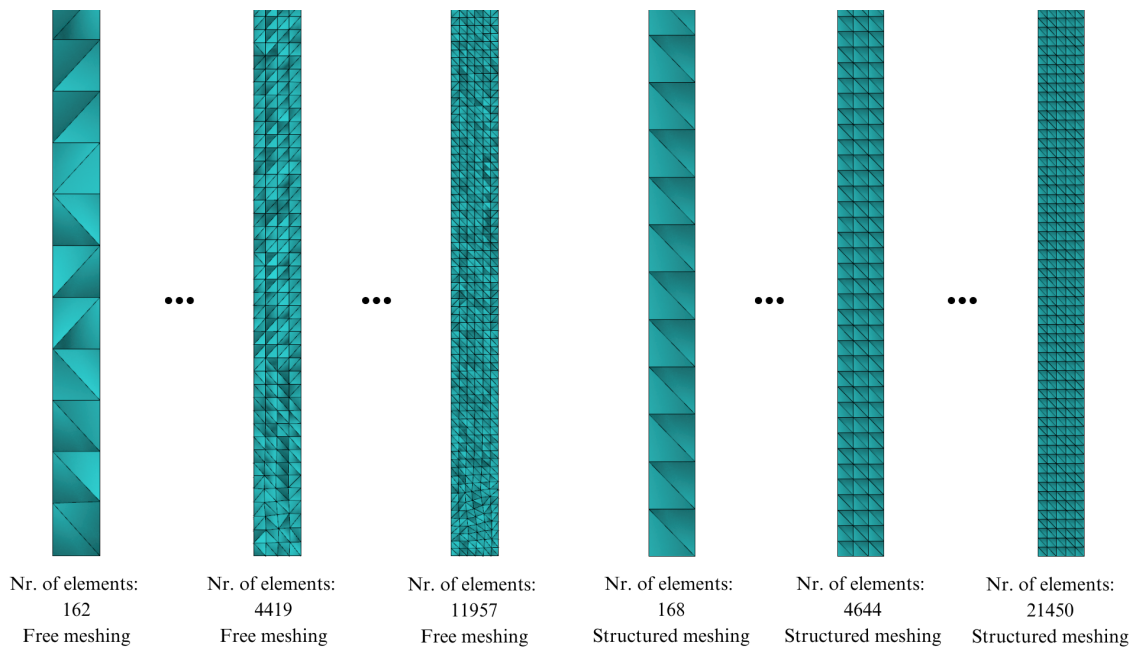


Figure 5.4.3: Free meshing and structured meshing using Tetrino.

The 3D-scanned column has a highly variable surface, which poses a challenging task for the meshing tools. As mentioned in Case Study 2, fine meshes can result in such a large K-matrix that the solver cannot calculate the solution due to a matrix size limitation. This makes controlling the number of elements essential, especially with an uneven surface. Small openings and cracks generate many elements and, therefore, many nodes. Iguana makes more similarly sized elements, but it has proven difficult because it is hard to control the number of elements. It was, therefore important to check that Tetrino gives reasonable results. Since mesh size and homogeneity can affect results, it is important to test the level of deviation when Tetrino is used.

After the BREP is generated in the BullAnt-component, this geometry is further regenerated into a tetrahedral mesh with four faces and four vertices. This mesh type is one of the two mesh types

that FERret accepts. Figure 5.4.4 shows some of the meshes and illustrates how Tetrino operates on the geometry. When the number of elements is low, the geometry representation is somewhat poor, and the volume deviates significantly. However, the meshing tool can capture tiny details as the number of elements increases. To calculate the volume deviation, Equation 5.4.1 is used. For a detailed description of the entire setup, please refer to the GH file in Appendix A.

$$\text{Deviation} = \left(\frac{V_{\text{Mesh}} - V_{\text{BREP}}}{V_{\text{BREP}}} \right) \times 100 \quad (5.4.1)$$

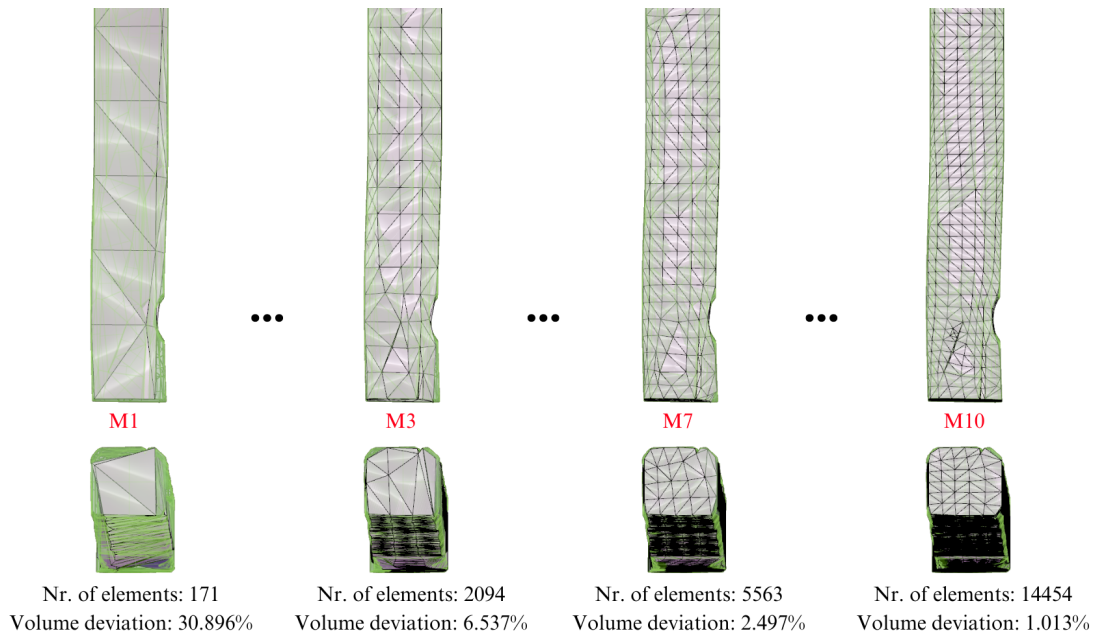


Figure 5.4.4: Difference in mesh

Abaqus

The Abaqus model was created with the same geometry, load, and support conditions as in Grasshopper to generate the most similar model possible. A test run was performed to ensure convergence. The favorable element was again a 20-node serendipity hexahedral element with reduced integration, known as C3D20R in Abaqus. This element was chosen for its excellent convergence ability and to achieve the closest approximation possible to the actual solution. The mesh comprises 10 045 elements.

5.4.2 Results

Perfect column

In the following subsection, the results for the perfect column are displayed. Figure 5.4.5 shows the colored displacement and stress map for the column calculated with free meshing and structured meshing in FERret and structured meshing in Abaqus. Figure 5.4.6 shows the error from the Abaqus solution for structured- and free meshing. To calculate the errors, the same definitions from Equation 5.1.2 are used. In Table 5.4.3 - 5.4.6, more details for the results are given, such as the number of elements, displacements, stresses and errors at the measurement point, and maximum values.

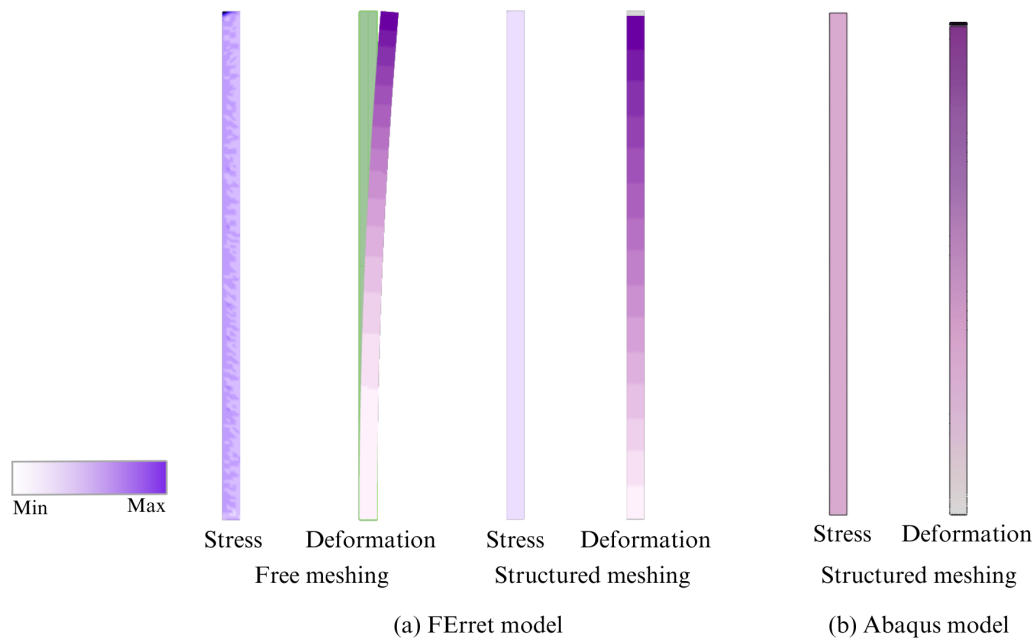


Figure 5.4.5: Result from analyzing perfect column.

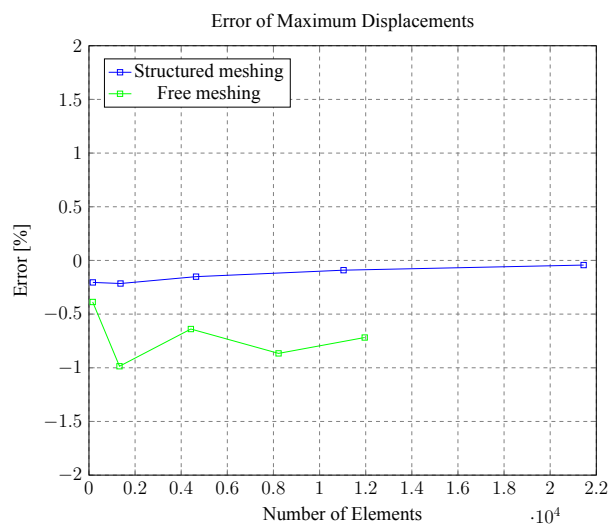


Figure 5.4.6: Error from the numerical solution obtained with Abaqus. Tetrino is used for meshing the free and structured meshes.

| Number of Elements | Displacement in middle [mm] | Displ. Error [%] | Maximum Displ. [mm] | Maximum Displ. Error [%] |
|--------------------|-----------------------------|------------------|---------------------|--------------------------|
| 168 | -1.98 | 1.72 | -4.04 | -0.205 |
| 1 368 | -2.03 | -0.540 | -4.04 | -0.215 |
| 4 644 | -2.00 | 0.930 | -4.04 | -0.151 |
| 11 040 | -2.03 | -0.771 | -4.04 | -0.0908 |
| 21 450 | -2.02 | -0.0692 | -4.03 | -0.0433 |

Table 5.4.3: Displacement results from the benchmark in the perfect column with a structured mesh. Abaqus provides the solutions $w_{\max} = -4.032$ mm and $w_{\text{mid.node}} = -2.0153$ mm at the measurement point.

| Number of Elements | Stress in middle [MPa] | Stress Error [%] | Maximum Stress [MPa] | Maximum Stress Error [%] |
|--------------------|------------------------|------------------|----------------------|--------------------------|
| 168 | 0.315 | 0.124 | 0.331 | 1.03 |
| 1 368 | 0.315 | 0.124 | 0.345 | -2.96 |
| 4 644 | 0.315 | 0.124 | 0.349 | -4.24 |
| 11 040 | 0.315 | 0.124 | 0.351 | -4.75 |
| 21 450 | 0.315 | 0.124 | 0.351 | -4.88 |

Table 5.4.4: Stress results from the benchmark in the perfect column with a structured mesh, compared with the solution from Abaqus with a dense mesh. Abaqus provides the solutions $\sigma_{\text{V.M.max}} = 0.335$ MPa and $\sigma_{\text{V.M.mid.node}} = 0.315$ MPa at the measurement point.

| Number of Elements | Displacement in middle [mm] | Displ. Error [%] | Maximum Displ. [mm] | Maximum Displ. Error [%] |
|--------------------|-----------------------------|------------------|---------------------|--------------------------|
| 162 | -2.06 | -2.21 | -4.05 | -0.388 |
| 1 324 | -2.00 | 0.700 | -4.07 | -0.986 |
| 4 419 | -2.02 | -0.475 | -4.06 | -0.639 |
| 8 221 | -2.01 | 0.454 | -4.07 | -0.866 |
| 11 957 | -2.01 | 0.477 | -4.06 | -0.719 |

Table 5.4.5: Displacement results from the benchmark in the perfect column with a free meshing. Abaqus provides the solutions $w_{\max} = -4.032$ mm and $w_{\text{mid.node}} = -2.015$ mm at the measurement point.

| Number of Elements | Stress in middle [MPa] | Stress Error [%] | Maximum Stress [MPa] | Maximum Stress Error [%] |
|--------------------|------------------------|------------------|----------------------|--------------------------|
| 162 | 0.315 | 0.124 | 0.428 | -27.9 |
| 1 324 | 0.315 | 0.123 | 0.925 | -176 |
| 4 419 | 0.315 | 0.123 | 1.11 | -230 |
| 8 221 | 0.315 | 0.124 | 1.33 | -299 |
| 11 957 | 0.315 | 0.124 | 1.34 | -301 |

Table 5.4.6: Stress results from the benchmark in the perfect column with a free meshing. Compared with the solution from Abaqus with dense mesh. Abaqus provides the solutions $\sigma_{\text{V.M.max}} = 0.335$ MPa and $\sigma_{\text{V.M.mid.node}} = 0.315$ MPa at the measurement point.

3D-scanned column

The results from the analysis of the 3D-scanned column are presented below. Figure 5.4.7 shows the colored displacement and stress map for the 3D-scanned column. Figure 5.4.8 displays the displacement and stresses compared with the volume deviation. In Table 5.4.7 - 5.4.8, more details for the results are given, including the number of elements, displacements, stresses, and deviations. A video illustrating the results in this case study is available at [YouTube](#), see Appendix B.

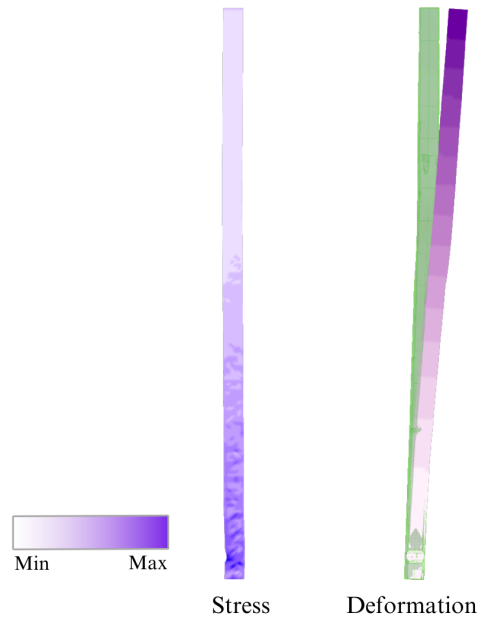


Figure 5.4.7: Stress and displacement of the 3D-scanned column.

| Sample Name | Number of Elements | Displacement in middle [mm] | Maximum Displacement [mm] | Volume [mm ³] | Volume Deviation [%] |
|-------------|--------------------|-----------------------------|---------------------------|---------------------------|----------------------|
| M1 | 171 | -2.92 | -9.73 | 107E+06 | 30.90 |
| M2 | 719 | -2.97 | -9.69 | 138E+06 | 10.78 |
| M3 | 2 094 | -2.61 | -9.83 | 145E+06 | 6.54 |
| M4 | 1 406 | -2.68 | -11.1 | 146E+06 | 5.99 |
| M5 | 2 928 | -2.19 | -9.70 | 149E+06 | 3.72 |
| M6 | 4 520 | -1.29 | -12.41 | 150E+06 | 3.14 |
| M7 | 5 563 | -1.44 | -14.08 | 151E+06 | 2.50 |
| M8 | 9 051 | -1.04 | -14.94 | 153E+06 | 1.54 |
| M9 | 9 440 | -1.59 | -15.24 | 153E+06 | 1.57 |
| M10 | 14 454 | -1.79 | -16.26 | 153E+06 | 1.01 |
| M11 | 16 031 | -1.82 | -16.11 | 153E+06 | 1.02 |

Table 5.4.7: Displacements from analysis of 3D-scanned column.

| Sample Name | Number of Elements | Stress in middle [MPa] | Maximum Stress [MPa] | Volume [mm ³] | Volume Deviation [%] |
|-------------|--------------------|------------------------|----------------------|---------------------------|----------------------|
| M1 | 171 | 0.425 | 1.75 | 107E+06 | 30.9 |
| M2 | 719 | 0.292 | 1.59 | 138E+06 | 10.8 |
| M3 | 2 094 | 0.257 | 1.54 | 145E+06 | 6.54 |
| M4 | 1 406 | 0.264 | 1.65 | 146E+06 | 5.99 |
| M5 | 2 928 | 0.235 | 1.63 | 149E+06 | 3.72 |
| M6 | 4 520 | 0.145 | 1.72 | 150E+06 | 3.14 |
| M7 | 5 563 | 0.129 | 1.68 | 151E+06 | 2.50 |
| M8 | 9 051 | 0.173 | 1.95 | 153E+06 | 1.54 |
| M9 | 9 440 | 0.0914 | 2.33 | 153E+06 | 1.57 |
| M10 | 14 454 | 0.0714 | 2.13 | 153E+06 | 1.01 |
| M11 | 16 031 | 0.0720 | 2.07 | 153E+06 | 1.02 |

Table 5.4.8: Stresses from analyzing 3D-scanned column

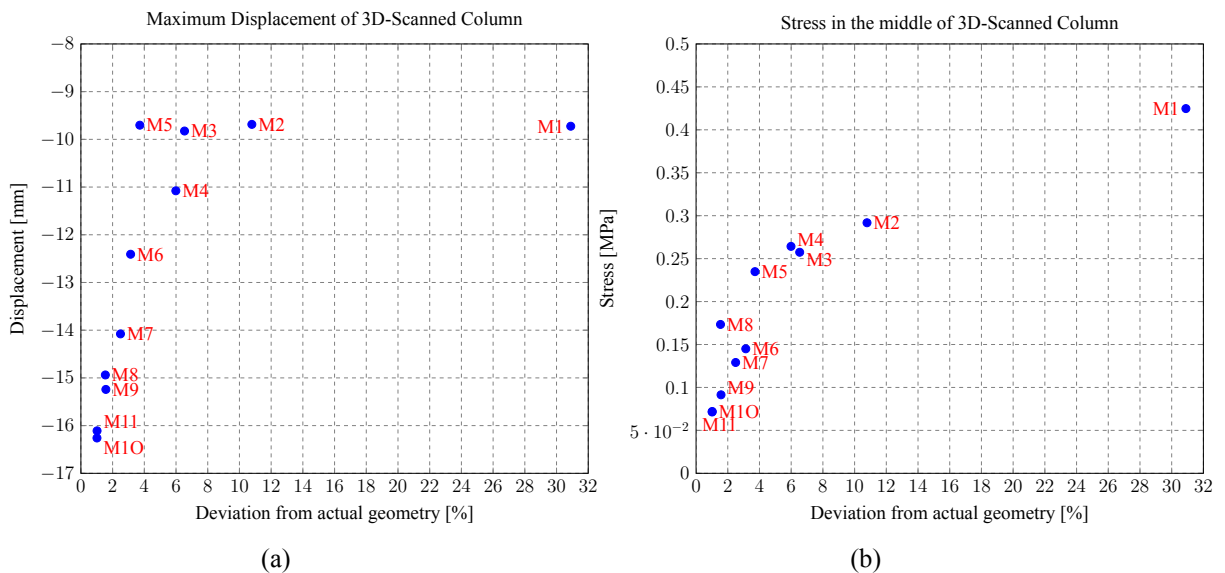


Figure 5.4.8: Displacements and stresses for the 3D-scanned column.

5.4.3 Discussion

The benchmarking performed on the perfect column reveals that FERret performs well. The stresses and displacements are of reasonable magnitudes, confirming that the test model works as it should. However, the free and structured meshing results demonstrate that the meshing technique can significantly influence the outcomes. This is evident in Figure 5.4.6, where the difference between the FERret and Abaqus solutions converges toward zero for the structured mesh. At the same time, it varies considerably for the free mesh. Nevertheless, the error is minimal and appears to converge toward zero. The error is also clearly visible in Figure 5.4.5, where a stress concentration occurs in the free mesh. This causes the column to deform to one side.

The structured mesh exhibits nearly uniform stresses throughout the column, and displacement occurs only in the vertical direction. In Abaqus, structured meshing is also employed, but with

hexahedral elements. As a result, the stress distribution and displacement behavior are similar to those observed in FERret's structured mesh. Table 5.4.4 and Table 5.4.6 demonstrate that the stresses at the middle of the column remain constant for the various meshes, aligning closely with the results obtained from Abaqus. Based on the location of the measurement point, it is reasonable that the stresses remain unchanged throughout this region, even if the mesh is modified.

The results from the 3D-scanned column also appear reasonable. Figure 5.4.7 displays concentrated stress at the bottom, which makes sense considering this area contains a cutout. The column's displacement is considerable, which could be attributed to imperfections that cause horizontal forces to act upon it. This displacement leads to a more significant displacement in the vertical direction at the top of the column. However, a challenge with the results lies in the difficulty of determining the exact solution since it never converges. The cause of this issue is the meshing. Figure 5.4.4 illustrates the challenge due to significant geometric variations resulting from different mesh sizes. A coarse mesh produces an approximately rectangular column corresponding to a solution that is likely far from the actual response. As the number of elements increases, the mesh geometry approaches the actual geometry. Still, even though the volume discrepancy is negligible, the solution does not flatten out completely.

It would be interesting to run a few solutions with volume deviation below 1% to observe if the results deviate less. However, one of the challenges with FERret arises in this scenario, namely the C# limitation on the size of the K-matrix. The solution may not converge entirely as it would for simpler geometries since the meshing in Tetrino significantly affects the results, as discussed for the perfect column above. Since loads are applied at the nodes, a denser mesh can potentially result in a less favorable scenario. The cross-sectional geometry may become less evenly distributed than desired. Consequently, the load distribution can become more eccentric for certain meshes, leading to varying loads. An example of this challenge is illustrated in Figure 5.4.9.

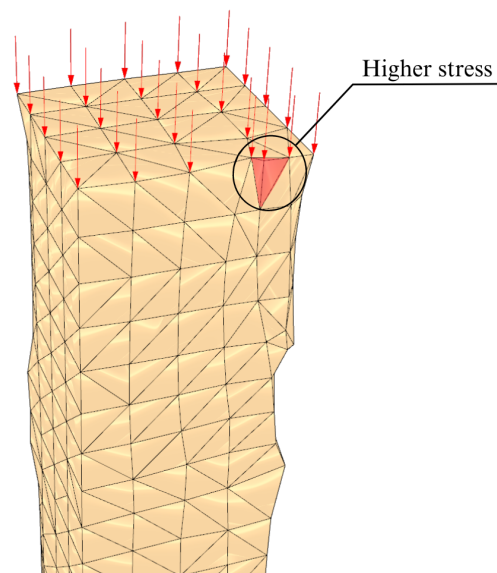


Figure 5.4.9: Refining the mesh of the uneven cross-section leads to a less favorable load distribution.

5.5 Case Study 5: Verifying the Implementation of FEniCSx in Grasshopper

Another benchmark was carried out to ensure the correct implementation of FEniCSx and its ability to provide precise results. The cantilever from Case Study 2 was used. In the model, there are two distinctions. Firstly, the applied load is represented as a surface load, and secondly, steel is chosen as the material. These modifications were made as FEniCSx was primarily designed to work with steel, and simulating surface loads is more manageable than point loads on this platform.

The geometry and details of the test model are provided in Table 5.5.1. Measurements have been conducted to assess the solution's convergence as the mesh transitions from coarse to fine. The solution is compared against an analytical solution and a numerical solution conducted in Abaqus. The experiment's parameters of interest are being compared, being the maximum values of displacement and von Mises stresses. Additionally, the information within the region enclosed by the green sphere in Figure 5.5.1 is being examined. The interest surrounding FEniCSx primarily lies in the speed of calculations, which is a significant concern in Case Study 2 as the mesh is refined. Therefore, this information has also been extracted from the analyses.

| Width | Height | Length | \sum Load | Boundary Condition | Youngs' Modulus | Poisson's Ratio |
|--------|--------|----------|-----------------------|--------------------|---------------------------|-----------------|
| 200 mm | 400 mm | 1 000 mm | 0.5 N/mm ² | Fixed | 210 000 N/mm ² | 0.3 |

Table 5.5.1: Dimensions, loads, boundary conditions and material properties for cantilever.

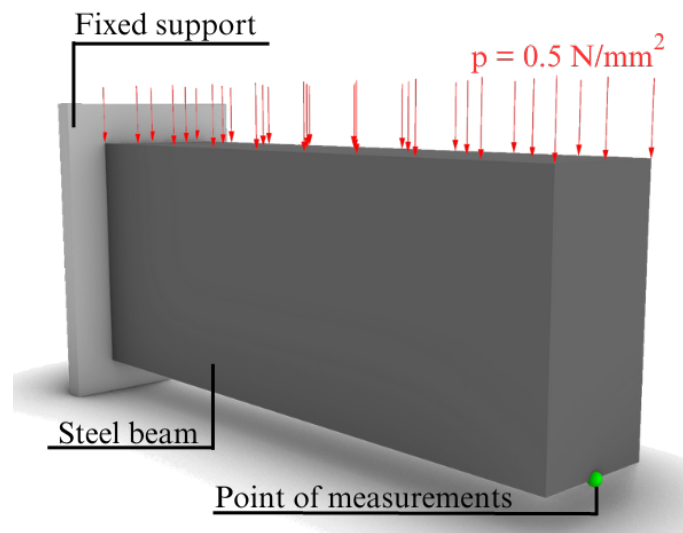


Figure 5.5.1: Test model for Case Study 5

5.5.1 Method

Grasshopper

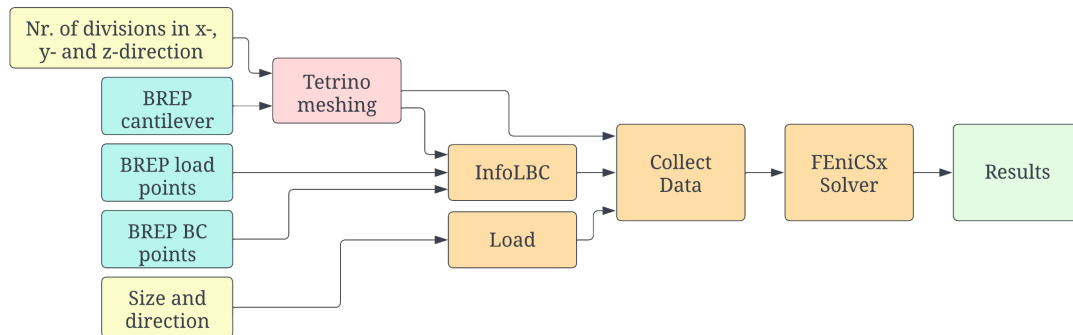


Figure 5.5.2: Flowchart for Case Study 5.

Figure 5.5.2 shows a flowchart representing the GH file utilized for the cantilever using FEniCSx as the solver. The geometry is constructed as a box, and Tetrino is employed as the meshing tool. When it comes to transferring the mesh to the solver, the difference between FEniCSx and FERret lies in their respective approaches. Instead of directly transferring the mesh from Tetrino to the solver, the vertices and connectivity between them are selected.

The InfoLBC component is employed to determine the points serving as boundary conditions and those subjected to loading. For this prototype, the solver has been designed to accommodate four different loads and a single type of boundary condition. The component utilizes the vertex list and the provided BREPs to determine the purpose of each point. The solver is currently designed only for surface loads, as it defines an area surface based on the points included in the load BREP.

All information is written to a file and further imported into the FEniCSx program for computation. The results are subsequently returned to the file and read by the Results component in Grasshopper. To create a colored displacement and stress display, the mesh is reconstructed using the Construct Mesh component in Grasshopper. The component takes the deformed points and a connectivity list for the input faces. This connectivity information is retrieved from Tetrino and is reformatted as $Q[n_1, n_2, n_3, n_4]$. To generate the colored pattern, the Gradient component from Grasshopper is connected, where the desired values are normalized before inputting. For a detailed description of the entire setup, please refer to the GH file in Appendix A.

As mentioned in Case Study 4, Tetrino creates an unstructured mesh, where element sizes and orientations can vary significantly. Consequently, there are fewer nodes in the geometry's central region than the number of nodes present on the surface. The meshed cantilever is shown in figure 5.5.3.

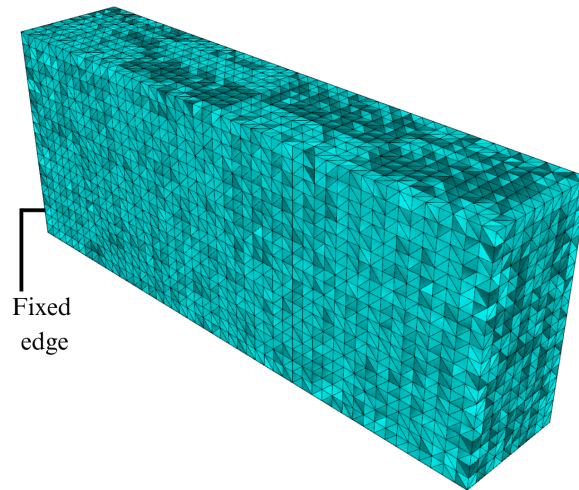


Figure 5.5.3: Free meshing of geometry.

Abaqus

The Abaqus model is created similarly as in Case Study 2, but this time with steel as the material. The support conditions remain the same, while the loading is now a surface load. A test run was performed to ensure convergence. The favorable element was a 20-node serendipity hexahedral element with reduced integration, known as C3D20R in Abaqus. This element was chosen for the same reasoning as in Case Study 4. The mesh comprised 35 805 elements.

5.5.2 Results

Due to the height-to-length ratio of the beam exceeding 1/10, shear deformations must again be considered. Timoshenko beam theory is, therefore, necessary to calculate the analytical solution. The following, Equation 5.5.1 has been used.

$$w_{\max} = \frac{qL^2}{2\kappa GA} + \frac{qL^4}{8EI} = -0.06488mm \quad (5.5.1)$$

- $q = 100\text{N/mm}$, representing the distributed load acting on the beam.
- $L = 1\,000\text{mm}$, representing the length of the beam.
- $\kappa = \frac{10(1+\nu)}{12+11\nu}$, the Timoshenko shear coefficient.
- $G = 81\,000\text{MPa}$, the shear modulus.
- $A = b \cdot h = 80\,000\text{mm}^2$, the cross-sectional area.
- $E = 210\,000\text{MPa}$, the Young's modulus.
- $I = \frac{b \cdot h^3}{12}$, the second moment of area for the cross-section, where b and h represents the height and the width.

Like the previous case studies, displacements and stresses have been extracted and illustrated in the results. Maximum values and values within the green sphere are therefore listed. In Figure 5.5.4, a comparison between colored displacement maps calculated using FEniCSx and Abaqus is presented. Figure 5.5.6 compares colored stress maps obtained from calculations performed with FEniCSx and Abaqus.

Results from the benchmark are listed in Table 5.5.2, and plotted graphically in Figure 5.5.5. The values are compared to the numerical solution in Abaqus and the analytical solution. Similarly, the von Mises stress results are provided in Table 5.5.3 and Figure 5.5.7, but in this case only compared with the Abaqus solution. To calculate the error from the analytical and Abaqus solution, Equation 5.1.2 has been used.

Displacements

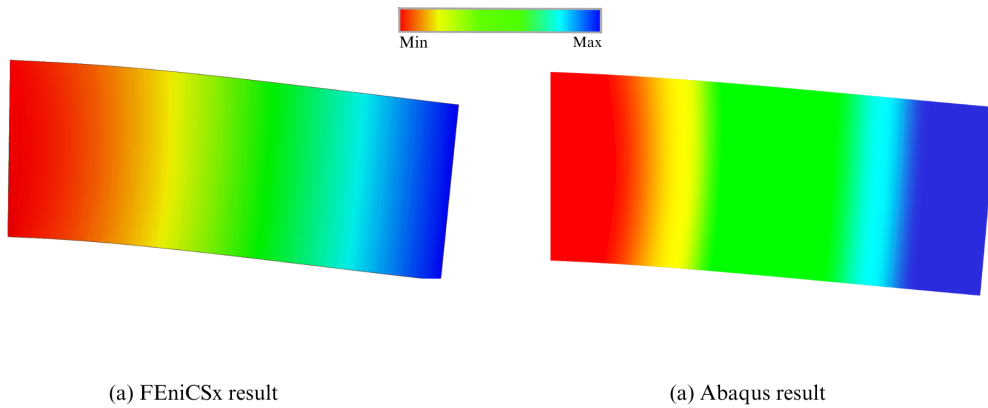


Figure 5.5.4: Colored displacement maps.

| Number of Elements | Displacement at Measurement Point [mm] | Displ. Error from Abaqus Solution [%] | Maximum Displacement [mm] | Max. Displ. Error from Abaqus Solution [%] | Max. Displ. Error from Analytical Solution [%] |
|--------------------|--|---------------------------------------|---------------------------|--|--|
| 457 | -0.0640 | -0.621 | -0.0645 | -0.601 | -0.617 |
| 1 416 | -0.0642 | -0.357 | -0.0646 | -0.355 | -0.370 |
| 3 263 | -0.0643 | -0.202 | -0.0648 | -0.185 | -0.200 |
| 6 438 | -0.0643 | -0.109 | -0.0648 | -0.108 | -0.123 |
| 11 076 | -0.0644 | -0.0466 | -0.0648 | -0.0462 | -0.062 |
| 18 046 | -0.0644 | -0.0155 | -0.0649 | -0.0154 | -0.0308 |
| 26 023 | -0.0644 | 0.0000 | -0.0649 | 0.0154 | 0.0000 |
| 41 932 | -0.0644 | 0.0311 | -0.0649 | 0.0308 | 0.0154 |

Table 5.5.2: Displacements compared to the solution obtained with Abaqus, as well as the analytical solution. Displacements at the measurement point, as well as maximum stresses, are evaluated. Abaqus provides the solutions $w_{\max} = -0.06487$ mm and $w_{\text{mp}} = -0.06439$ mm.

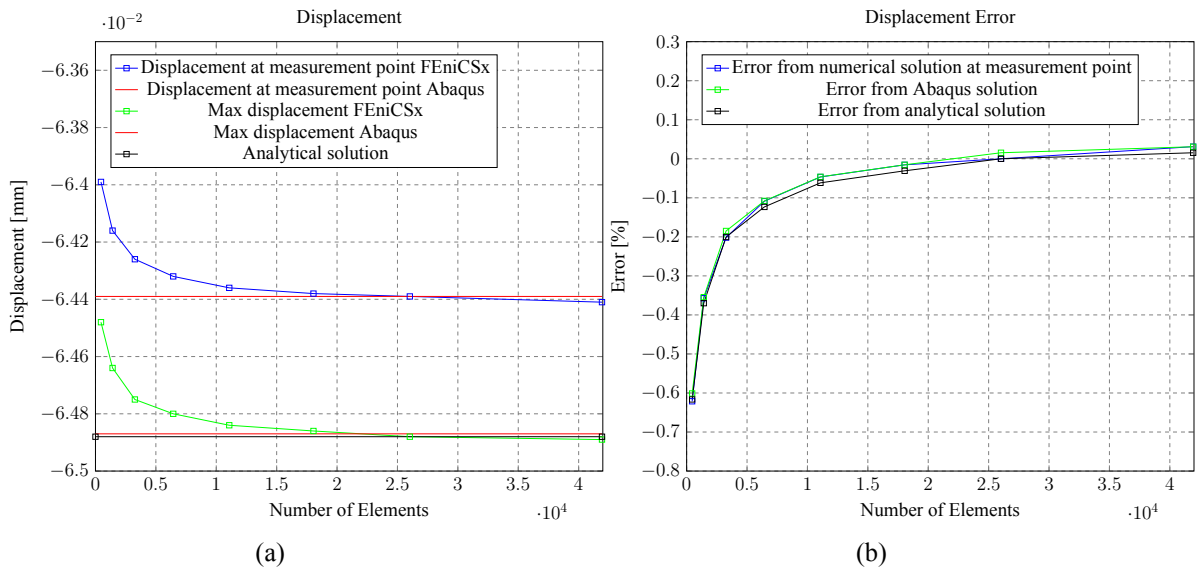


Figure 5.5.5: Comparison of displacements and errors from the numerical and analytical solutions.

Stresses

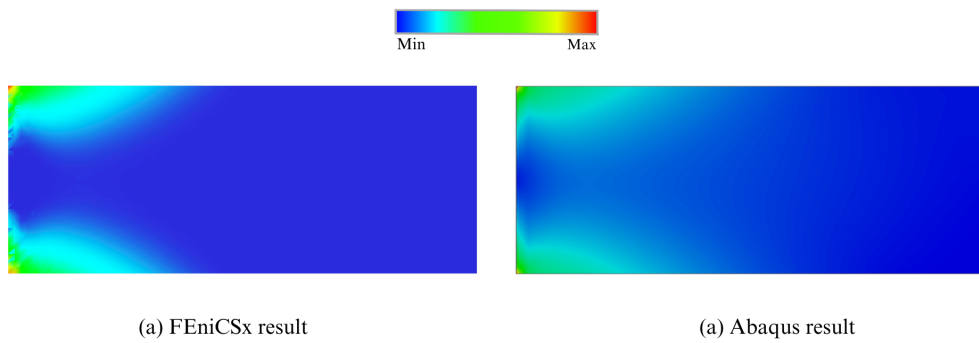


Figure 5.5.6: Colored stress map

| Number of Elements | Stress at Measurement Point [MPa] | Stress Error from Abaqus Solution [%] | Maximum Stress [MPa] | Maximum Stress Error from Abaqus Solution [%] |
|--------------------|-----------------------------------|---------------------------------------|----------------------|---|
| 457 | 0.0465 | 156 | 11.3 | -35.0 |
| 1 416 | 0.0248 | 36.6 | 12.9 | -25.9 |
| 3 263 | 0.0176 | -3.03 | 13.6 | -22.0 |
| 6 438 | 0.0173 | -4.90 | 16.8 | -3.51 |
| 11 076 | 0.0174 | -4.40 | 16.3 | -6.40 |
| 18 046 | 0.0195 | 7.05 | 20.1 | 15.6 |
| 26 023 | 0.0183 | 0.771 | 23.2 | 33.5 |
| 41 932 | 0.0184 | 0.991 | 23.7 | 36.4 |

Table 5.5.3: Stresses compared to the solution obtained with Abaqus. Stresses at the measurement point, as well as maximum stresses, are evaluated. Abaqus provides the solutions $\sigma_{V.M.max} = 17.37$ MPa and $\sigma_{V.M.mp} = 0.01817$ MPa at the measurement point.

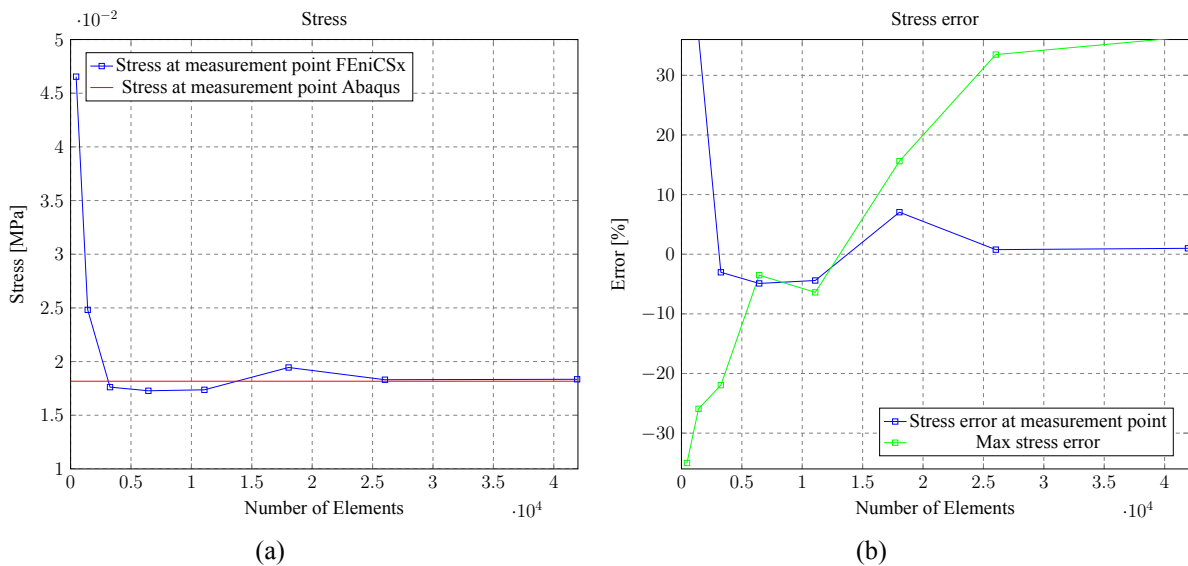


Figure 5.5.7: Comparison of stress with corresponding errors from the solution obtained with Abaqus.

5.5.3 Discussion

The implementation of FEniCSx has clearly been successful, as evidenced by the results presented above. The solver provides solutions that closely resemble Abaqus's numerical results while also aligning with the analytical solution. The von Mises stresses are clearly converging, however, the maximum stresses do not, as shown in (Figure 5.5.7b). This behavior is expected because stress concentration increases as the mesh becomes finer, and the solution does not converge toward an exact value. Variations in stress concentrations may also be a consequence of the meshing method.

As mentioned in previous case studies, Tetrino produces some free meshes where element size and orientation may vary significantly. Each time the mesh is refined, a new mesh is generated with varying sizes and orientations. This can affect stress distribution and result in varying maximum stresses, as well as varying stresses in the region near the point of measurement.

It would have been preferable to perform the analysis with a structured mesh, as was done for the perfect column in Case Study 4. However, the challenge lies in how the authors have built the solver. Currently, vertices and connectivity are extracted from Tetrino and passed into the FEniCSx solver instead of directly importing the mesh. The connectivity contains four connectivity numbers per line and describes which vertices are used to build each element. To construct a perfect mesh, the same method as in Case Study 4 would have to be used. Then the connectivity lines will contain more than four connectivity numbers per line, creating difficulties in rebuilding the mesh within the solver. This would require a significant amount of work for minimal benefit, considering the results are already satisfactory.

Something exciting about the results from FEniCSx is the speed of the solver. This is illustrated in Table 5.5.4 and Figure 5.5.8, where e.g. the finest mesh with a K-matrix size of 212 070x212 070 only takes 17.3 seconds. This includes everything that happens within the entire GH file. If we only consider the time the solver takes from reading the file to writing the results, it only takes 13.8 seconds.

| | | | | |
|---------------------------------|----------|--------|---------|---------|
| Number of elements | 18 | 10 361 | 26 023 | 37 627 |
| Number of DOFs | 189 | 60 012 | 147 594 | 212 070 |
| Total time[s] | 2.20 | 4.40 | 10.3 | 17.3 |
| FEniCSx-computation time[s] | 0.897 | 4.00 | 9.30 | 15.7 |
| Rebuild mesh in FEniCSx time[s] | 0.00203 | 0.0198 | 0.0521 | 0.102 |
| Assemble model time[s] | 0.000825 | 0.194 | 0.494 | 0.758 |
| Solver time[s] | 0.00242 | 2.54 | 6.95 | 11.4 |
| Total time in FEniCSx[s] | 0.110 | 3.19 | 8.48 | 13.8 |
| Write and read time[s] | 0.134 | 0.286 | 0.712 | 1.11 |

Table 5.5.4: Time usage for different processes using FEniCSx as the solver in Grasshopper.

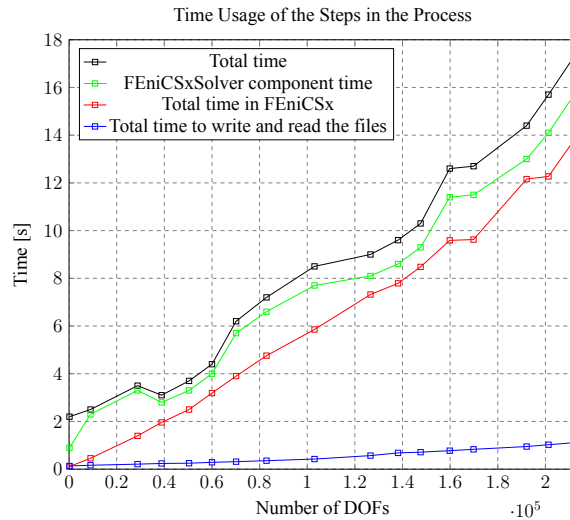


Figure 5.5.8: Time usage for the different processes in FEniCSx, as well as total time usage for this solver.

The green graph in Figure 5.5.8 illustrates the time the FEniCSxSolver component uses for the calculation, while the red graph shows the time used for FEniCSx outside Grasshopper to calculate the solution. There is a consistent difference between the graphs throughout the benchmarking process. The time difference primarily comes from the time it takes to start the FEniCSx file. The slight irregularities in the graph can also be attributed to the component taking some time to receive and pass on inputs.

One big challenge with the implementation is transferring information between the two programs. The authors attempted a solution where data was stored in a cloud solution through Speckle, in addition to a solution where it was only stored in the computer's memory and then retrieved. Both solutions presented issues due to a blockage that occurred in Ubuntu, so text files were chosen instead.

The time it takes to transport all the information through text files is illustrated in the blue graph. It shows that the solution has minimal impact on the overall speed, even with the finest mesh, which involves writing 162 077 lines in the CollectData component. It is evident that if the geometry becomes very large, the solution will increase the solver time, but it will still be low compared to the time the solver takes to provide the solution. For visualization of the case study, a video illustrating the problem is available at [YouTube](#), see Appendix B.

5.6 Case Study 6: Analyzing Complex Geometry with FEniCSx

Given FEniCSx's impressive track record regarding result convergence, managing large K-matrices, and computational speed, it would be prudent to explore its capabilities in handling complex geometries. In Case Study 4, a two-story community building was used as a test object, where a 3D-scanned beam was subjected to compression. For the same project, a modern solution was also made, as shown in Figure 5.6.1. All the model's beams, columns, and bracings are new and free of deformations and imperfections. The building material remains wood, but the joints are made of steel and connected to the wood using dowels. Table 5.6.1 presents the relevant geometry, load, boundary conditions, and material.

For this case study, a joint in the roof of the building has been selected, as it exhibits a more interesting load-bearing behavior than the joints along the columns. The particular connector is marked in green Figure 5.6.1. Details of the joint are presented in Figure 5.6.2.

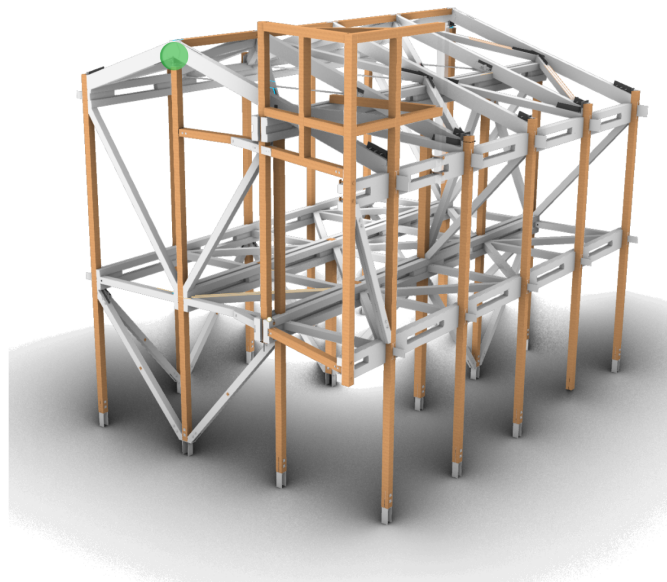


Figure 5.6.1: 3D model of new house

| Cross-Section Wood (b x h) | \sum Load | Boundary Condition | Youngs' Modulus | Poisson's Ratio |
|-------------------------------|-------------|-----------------------|---------------------------|--------------------|
| 119 mm x 235 mm | 20 kN | Fixed | 210 000 N/mm ² | 0.3 |

Table 5.6.1: Geometry for the timber, and loads, boundary conditions, and material properties for steel connector.

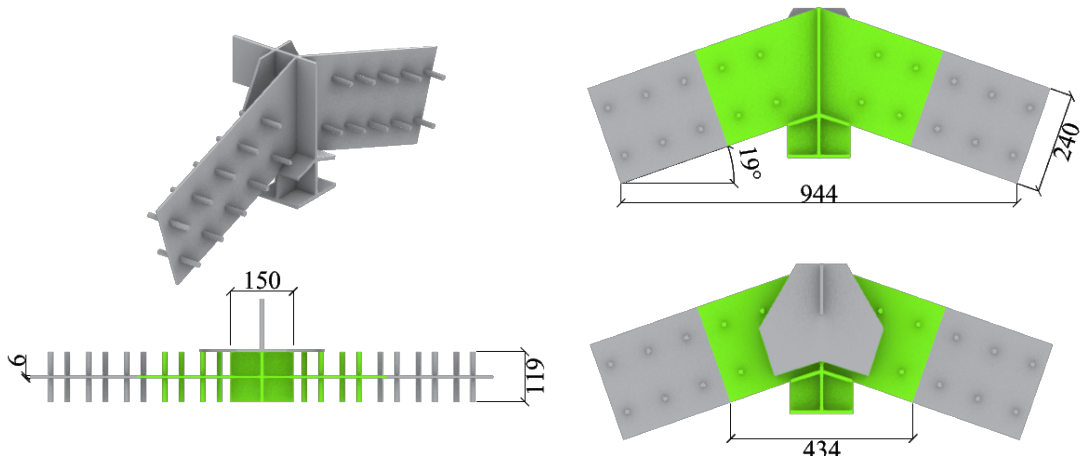


Figure 5.6.2: Detailed model. Measurements are given in mm.

The model has been chosen to be simplified as this case study aims to test whether FEniCSx can analyze and represent a complex loaded geometry. The reduced model is highlighted in green in Figure 5.6.2. In reality, the joint consists of rafters and dowels, but an attempt has been made to create a model that can provide the same response without including rafters and dowels in the analysis. This has been done to reduce the model's size and consequently decrease computational time.

To replicate the loading conditions, it has been chosen to load the edge of the steel plate as shown in Figure 5.6.3. The placement of this red zone will vary based on the directions of the applied loads. Figure 5.6.4 illustrates a typical loading scenario for such a joint. The selected test model is expected to yield a response similar to what such a model would provide.

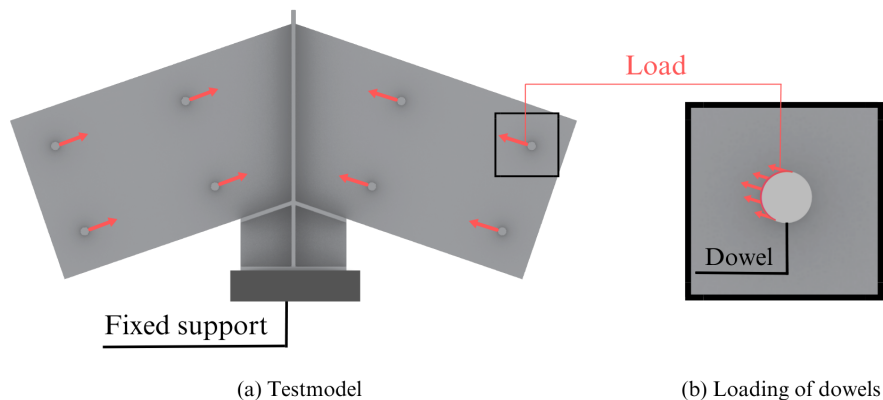


Figure 5.6.3: Illustration of how loads and boundary conditions are applied.

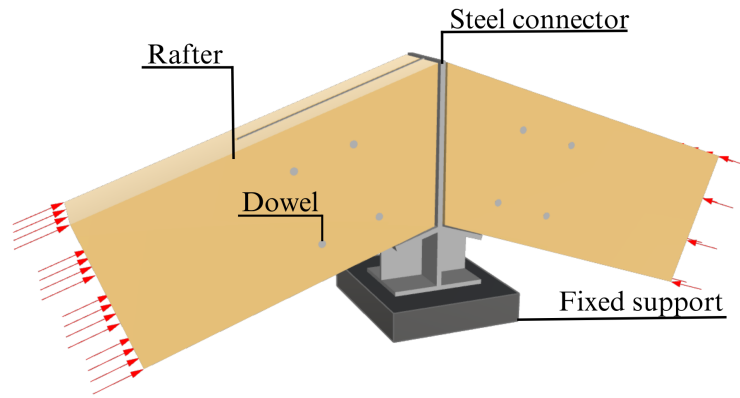


Figure 5.6.4: Test model of steel connector with rafters.

5.6.1 Method

Grasshopper

Figure 5.6.5 illustrates the flowchart for the GH file of Case Study 6. The setup is very similar to the setup in Case Study 5, with the only differences being the geometry to be analyzed, the location of the load application, and the direction of the load. The geometry used in this case study is more complex than in Case Study 5, requiring some preprocessing to make it compatible with meshing in Tetrino. In Rhino, the geometry was defined as separate surfaces, resulting in a rectangular plate composed of six surfaces. This posed a challenge and necessitated recreating the geometry. To minimize challenges with Tetrino, the geometry was reproduced in Grasshopper. The model was created as a solid and symmetrical BREP, resulting in a considerably better mesh. By doing so, the model became fully parametric, allowing lengths, widths, thicknesses, and angles to be adjusted by manipulating sliders. The number of dowels can also be chosen in this approach. Due to a large number of elements, the dowels were decided to be left out of the model. For a detailed description of the entire setup, please refer to the GH file in Appendix A.

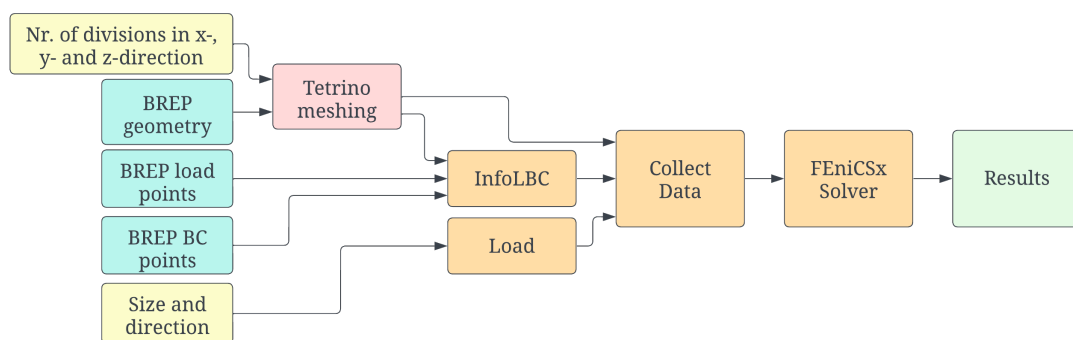


Figure 5.6.5: Flowchart for Case Study 6.

Three load cases were conducted to observe how the connection behaves under different loads. For each load case, three different meshes were used; coarse, medium fine, and fine. The adaptive sizing in Tetrino is used, allowing mesh refinement in the zones where it is necessary. This was to highlight the importance of accurately modeling the connections with a sufficient number of elements. Figure 5.6.6 shows a picture of the coarse mesh.

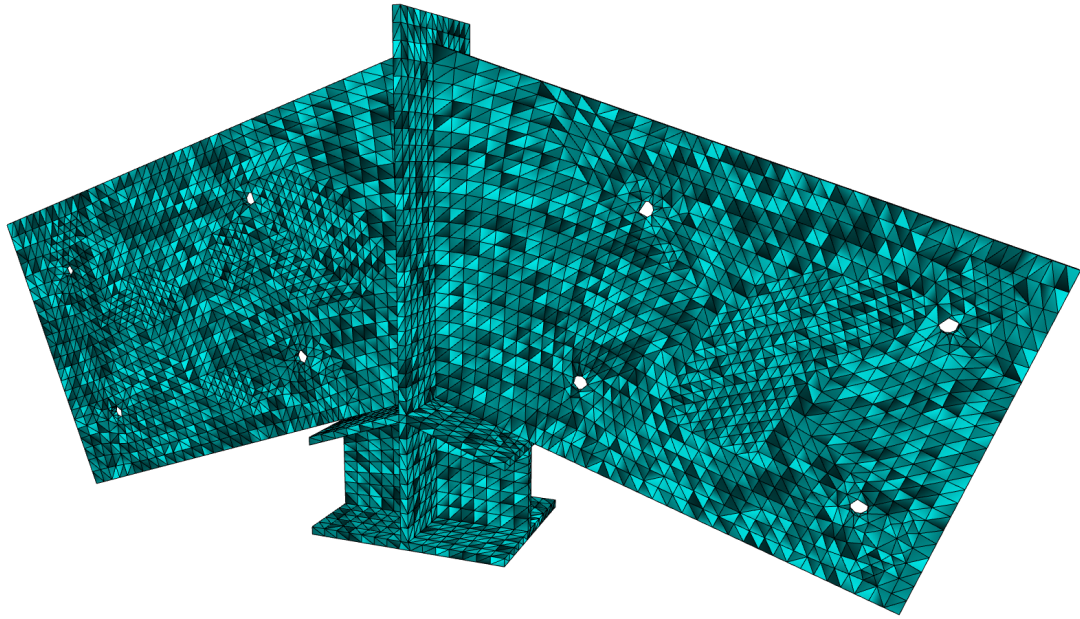


Figure 5.6.6: Meshed steel connection

5.6.2 Results

An analysis was conducted on the given geometry, and vital parameters such as displacement, von Mises stress, computational time, and the number of elements are extracted and presented in this section. In this case study, the results are exclusively presented through images, as the intention is to demonstrate that the solver can provide reasonable displacements and stress concentrations for the given load cases and geometry. Each load case is analyzed with the tree meshes, where meshing 1 corresponds to the coarse mesh, meshing 2 to the medium fine mesh, and meshing 3 to the fine mesh. The results are presented for load cases 1-3, where the left side depicts the deformed model with a color pattern illustrating absolute displacement calculated using Equation 5.6.1. The right side illustrates von Mises stresses in the geometry. A purple sphere indicates the location of maximum displacement, and an orange sphere represents the location of maximum von Mises stresses. In addition, a video illustrating this case study is available at [YouTube](#), see Appendix B.

$$r = \sqrt{u^2 + v^2 + w^2} \quad (5.6.1)$$

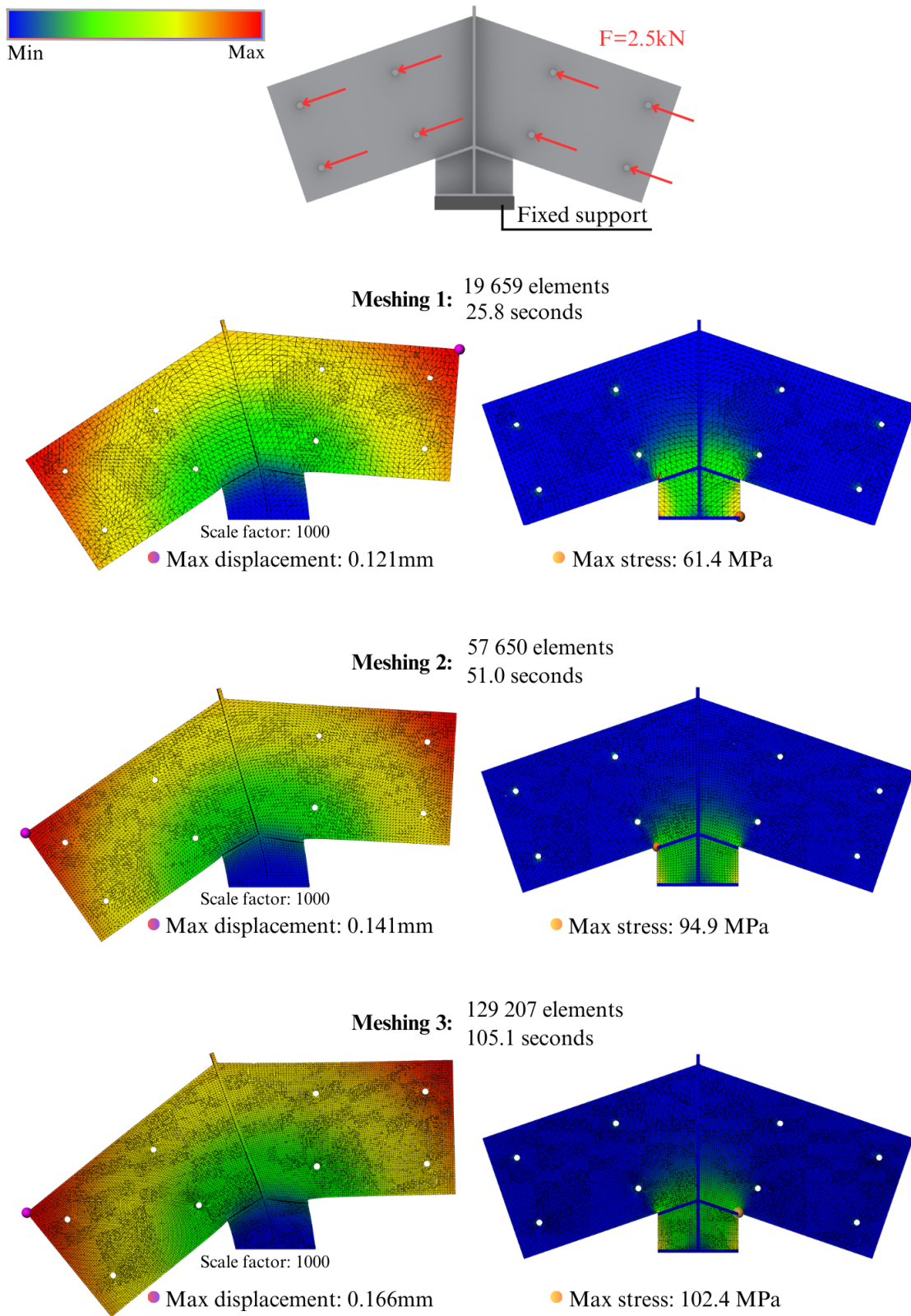


Figure 5.6.7: Load case 1.

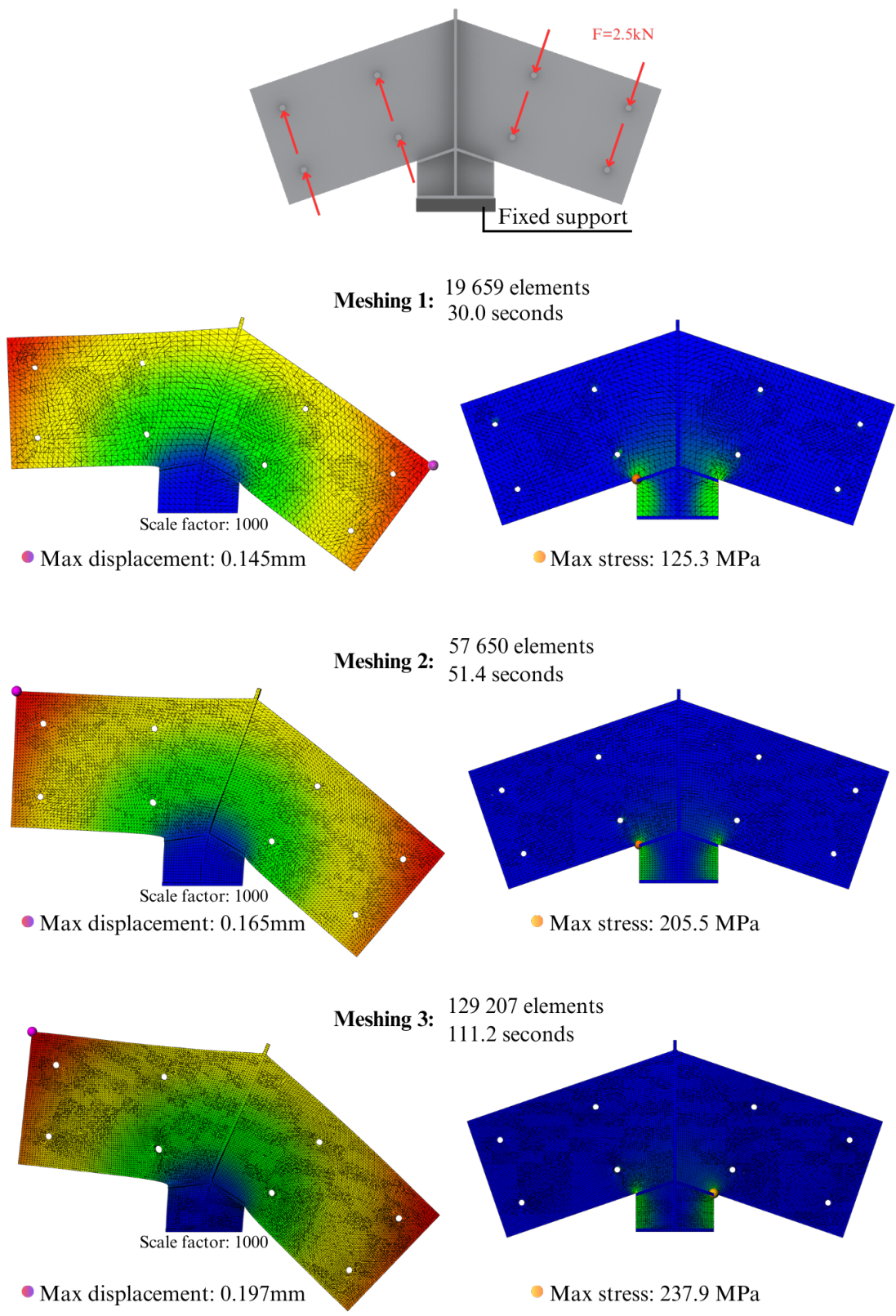


Figure 5.6.8: Load case 2.

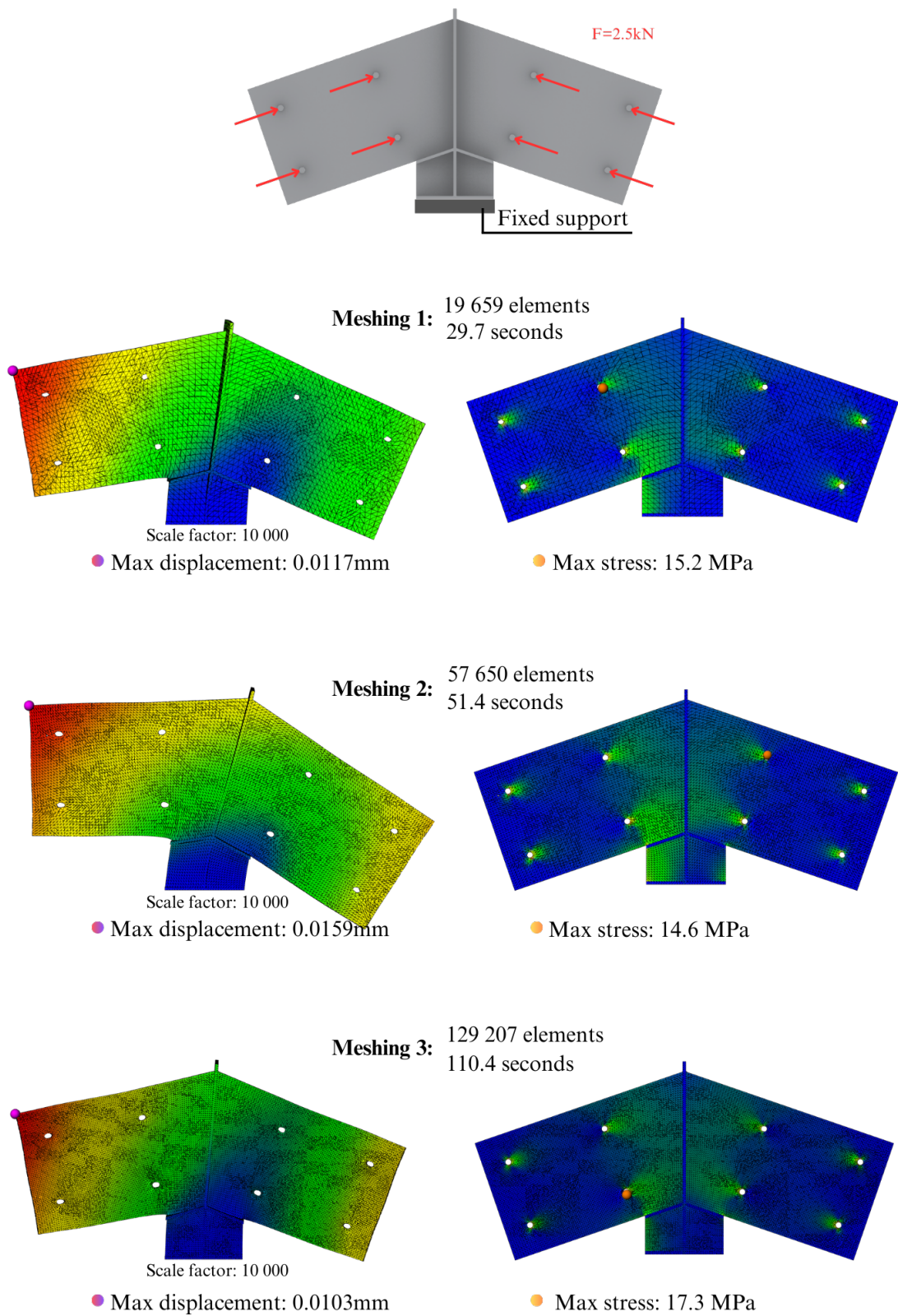


Figure 5.6.9: Load case 3.

5.6.3 Discussion

Through Case Study 6, FEniCSx is tested on a complex geometry that is highly relevant in everyday engineering. The presented results demonstrate that the solver is capable of performing reasonable analyses of the geometry. Based on the load case, the response and the variation of results with changes in mesh size are consistent with the authors' expectations. When comparing displacements and von Mises stresses a clear correspondence between stress concentration and large displacements can be observed when comparing displacements and von Mises stresses.

In Load Cases 1 and 2, the displacement patterns show a high level of symmetry in their coloration. This is expected since Equation 5.6.1 calculates absolute values without differentiating between positive and negative displacement. However, for load case 3, the symmetry seems to be absent. This may seem strange as this load case involves symmetric loads moving toward each other. The stress distribution map reveals inequality, with higher stress appearing on the left side. Consequently, the nodal point deforms towards the right. Upon investigation, it was discovered that this movement was due to the size of the loaded areas.

As previously stated, load zones are established by attaching tags to particular points. These points are subsequently utilized to create a load region in the FEniCSx module. The issue arises when the mesh is free and unsymmetrical. There may be a discrepancy in the number of points assigned to different loading zones sent to the solver. Consequently, this leads to slightly different loaded areas, which may produce an asymmetrical case. The various load zones for Case Study 3 are illustrated in Figure 5.6.10 and numbered in Table 5.6.2. The table also shows that the loading zone gets bigger every time the model is refined.

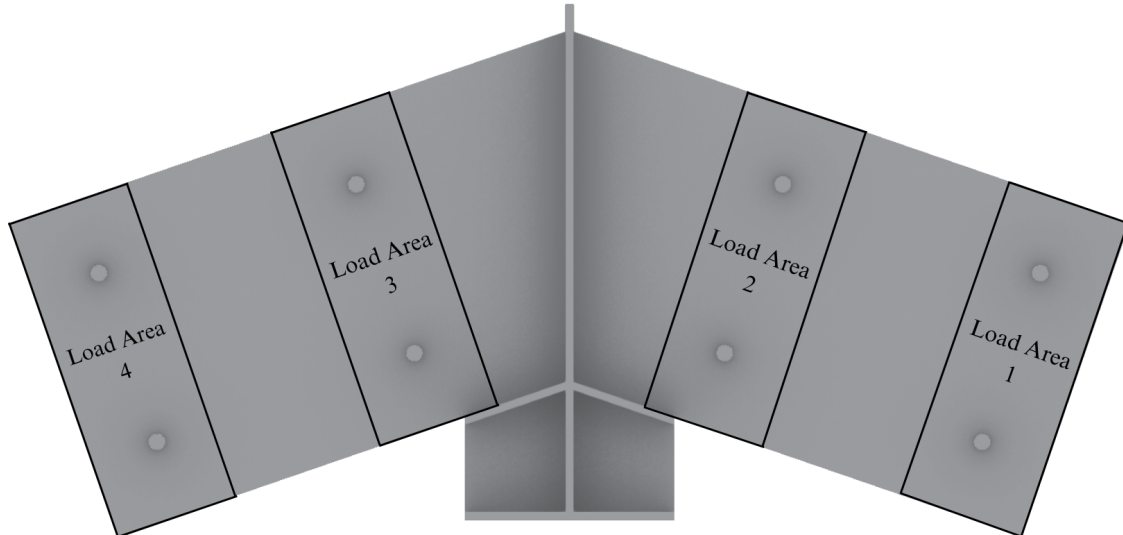


Figure 5.6.10: Showing the different loading zones that are modeled in the analysis

| | Load Area 1: | Load Area 2: | Load Area 3: | Load Area 4: |
|------------------|---------------------|---------------------|---------------------|---------------------|
| Coarse mesh | 147 mm ² | 121 mm ² | 130 mm ² | 167 mm ² |
| Medium fine mesh | 135 mm ² | 157 mm ² | 172 mm ² | 175 mm ² |
| Fine mesh | 196 mm ² | 180 mm ² | 194 mm ² | 191 mm ² |

Table 5.6.2: Areas for the different load areas.

According to the data presented in the table, the test model assumed by the authors is not functioning precisely as intended. It is possible that including the complete model, with a load on the end of the rafters, would result in a more evenly distributed load. However, this will not provide a realistic solution since FEniCSx can only handle one material for the model. Thus, the plugin is not able to represent the test model in Figure 5.6.4, and therefore not elaborated further in this thesis.

6 Discussion

Based on the six case studies, the research question will be answered in the following. The section is divided into two parts. Firstly, the implementation and development of the FEM solvers Pre-Ferret, FERret, and FEniCSx are discussed. In the second part, the discussion deals with the main challenge of applying FEM solvers in the Grasshopper environment, which relates to the geometries' meshing. The benefits and limitations of involving FEM in the conceptual phase through the AAD environment will be discussed, as well as the further work that should be done to improve the solutions developed in this thesis.

What are the opportunities and challenges associated with implementing Finite Element Method analysis using solid elements during the conceptual phase of projects, through the use of an Algorithms-Aided Design environment?

6.1 FERret and FEniCSx

The first impression of parametric modeling possibilities came with the development of the Pre-Ferret plugin in Case Study 1. Even though the plugin had some problems probably connected to the coherence between result values and their respective nodes, it clearly showed the potential of having a FEM solver inside an AAD environment like Rhino Grasshopper. The simple and user-friendly layout of the software makes it easy, even for people unfamiliar with the program to work with the FEM solver. Using sliders as parameters to define geometry, load, boundary conditions, and material makes the modeling process way more efficient and readily done compared to standard commercial FEM solvers like Abaqus.

The advantages of utilizing parametric modeling in the concept stage were further investigated in Case Studies 2 and 3. In the first of them, the implementation of the material timber showed that the FERret plugin could solve problems related to not only isotropic materials like steel. FERret showed good results also when the structural problem consisted of more comprehensive orthotropic materials like wood. With 384 hexahedral elements, FERret could provide a solution in 0.55 seconds with less than one percent error compared to the commercial software Abaqus. Using tetrahedral elements, the number of elements needed to obtain a relative error of less than 1 percent was significantly higher. Nevertheless, the tetrahedra provided more accurate solutions as they enabled many elements to be applied.

For now, the plugin has a limitation as it does not support structural elements with local axes angled to the global cartesian axes. This problem can be solved by introducing rotational matrices to define the engineering constants correctly concerning the global axes. This problem is left out for further work. However, the user can already define which global axis the main local axis should be set along by transforming the Poisson ratios.

Another limitation that comes with the development from Case Study 2 is that the whole structural model can only have one single direction for the strong axis in a global sense. This means that the model cannot be, e.g., a frame containing beams and columns with the grain direction spanning the components' longitudinal direction. The following case study, which deals with splitting the structural model into different parts, solves this problem and opens new possibilities for combining parts with different material definitions and properties.

Illustrated by a loaded steel dowel inserted into a timber beam, Case Study 3 showed that the potential of FERret was increased by implementing the use of parts in the global structural model. By defining certain volumes as independent parts, the structure could be divided into several pieces, each with its material properties. The results of the FE analysis were convincing, and the displacement and stress charts showed that the overall behavior of the structural model was trustworthy.

The case study used two materials with highly different properties. Steel's hard, ductile, heavy, and stiff properties were combined with the softer, less ductile, and lighter timber material. By combining these materials, the reliability of the results should be easy to evaluate as the strong steel dowel should be less deformed than the softer timber. This behavior was confirmed by the displacement plots, even though the steel dowels suffered from the singularity that appeared due to point loads with some large displacements at those nodes in particular.

The implementation of parts enables the user to define every material that the parameters of isotropic and orthotropic materials can represent. This means that the problem mentioned above about modeling a frame containing beams and columns is solved by the opportunity of defining different materials with their strong axis set along the desired global axis. In this way, even a 3-dimensional frame can be modeled by defining three different timber material classes, each with its strong axis defined along one of the x-, y-, and z-axes.

The developments of the FERret plugin in Case Study 2 and 3 has led to a plugin that is more solid and general than before. The solver can now handle different parts consisting of different materials that can be analyzed in interaction with each other. In other words, the plugin can now analyze more realistic and comprehensive structural problems, as the fewest real-life structures consist only of one material. Some of the potentials of the FERret plugin were explored in the following case study, analyzing a timber element from an actual project in Oslo. The results showed to be reliable and highly comparable to the solution obtained in Abaqus. This confirmed that the development of FERret is also useful in real-life cases like reusing timber materials in constructing new buildings.

Despite the convincing results from the development of FERret, there are some problems when modeling in Grasshopper with the PreFERret and FERret plugins, and those problems are related to the meshing process of the geometry. These problems are discussed in the final part of the section.

In Case Study 5, a steel beam with the same dimensions as in Case Study 2 was benchmarked. The analysis provided convincing results and showed that the FEniCSx environment was a powerful and efficient solver for FEA problems. The impressive low time consumption of solving the PDEs to obtain displacements and stresses was eye-opening. Computations were done in a few seconds while still providing accurate results compared to the commercial software.

To get a better impression of the time usage, a small benchmark was done only considering the time consumption in Abaqus and FEniCSx. This benchmark is illustrated in Figure 6.1.1 and clearly shows the remarkable performance of FEniCSx. The solver uses significantly less time analyzing and obtaining solutions than Abaqus. Nonetheless, it should be noted that Abaqus runs several more complex computations than the FEniCSx at this stage. However, for the desired results to evaluate at a conceptual stage, Abaqus provides a much more extensive and complete analysis than what is probably necessary so early in the process. This finding is highly valuable, as both the parametric design process and numerical computations are done faster with the FEniCSx solver

than in Abaqus. The low time consumption increases the possibility of providing accurate results within a short amount of time. For meetings in the conceptual stage, this plugin can provide help for decision-making efficiently and seamlessly.

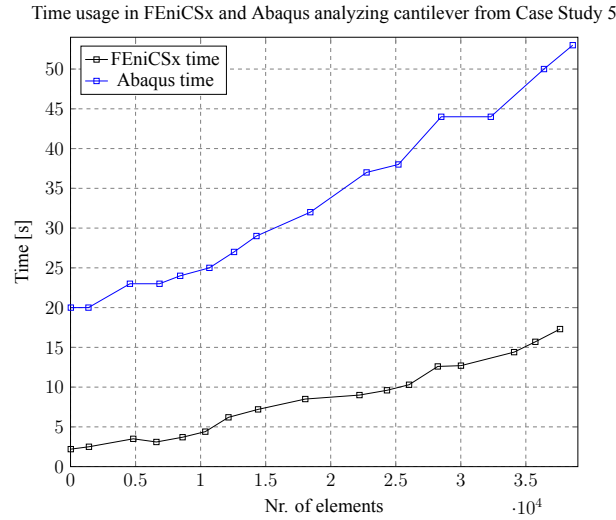


Figure 6.1.1: Time usage for the two different FEM solvers Abaqus and FEniCSx.

In the development stage, there were discussions about whether to write the information in memory or to file. As writing to memory provides quicker procedures, it requires more memory capacity, as well as more challenging programming algorithms. As shown in Table 5.5.4, the write and read time stands for less than 10% of the total time usage. This was evaluated as insignificant enough to use the procedure of writing and reading to text files rather than writing to memory. However, it should be noted that because of this choice, there is some potential to reduce the time consumption for further improvements of the FEM solver.

While Case Study 5 worked as a standard benchmark for testing the accuracy and reliability of the method, Case Study 6 uses this solver to analyze a real-life structural problem related to the same project as in Case Study 4. This geometry is quite complex and requires a solver capable of handling much larger meshes than what is achievable with FERret. Based on the analysis results, it appears that FEniCSx-solver can accurately model complex geometry while maintaining a low computational time. The promising results from this case study open up exciting possibilities for the detailed design of joints. By employing optimization tools, various free-formed joints can be tested, incorporating different dimensions, shapes, stiffness properties, dowels, etc. This allows for the design of the optimal joint configuration for the given structure.

Unlike FERret, FEniCSx has so far been developed for one single material only and cannot handle multiple materials simultaneously. In Case Study 6, the results were simplified to account for wooden rafters and achieve the desired response. However, the test model did not perform as expected. To achieve realistic results in this aspect, further software development is required. FEniCSx encountered several challenges related to meshing, and the following section presents the specific issues encountered in each case study.

6.2 Meshing Challenges

Three main meshing plugins are used in this thesis. Two of them are open-source applications available for download from the internet, while the last one is a simple mesh algorithm made in Grasshopper by the developers. The two open-source applications, Tetrino and Iguana, are the dominating meshing methods used in this thesis, and both of them showed great potential but also limitations.

Firstly, Tetrino was used in the case studies, but the problems with this plugin showed up quickly. When meshing with Tetrino, the surface of the volume is divided into the desired mesh settings set by the user. This structured mesh creates solid elements by connecting the surfaces through lines between the meshed surfaces' nodes. This method works fine for geometry with only one element over the thickness, as the resulting mesh is structured and regular.

However, for meshes with more than one element over the thickness, this meshing method becomes problematic as the mesh gets irregular and somewhat chaotic. For this reason, it became hard to obtain a mesh comparable to the mesh obtained in the commercial FEM software Abaqus. As the geometries in the first two case studies of the thesis consisted of right prisms, it was possible to use the authors' own-developed meshing algorithm to obtain the desired mesh, with a structured mesh also inside the volume.

Regarding Case Study 1, it was sufficient with this own-developed algorithm only, as the case dealt with 8-node brick elements. For Case Study 2, the structured mesh was used as input to the Tetrino plugin to obtain a tetrahedral mesh, now in a structured way due to the regularity obtained with the first algorithm. After this improvement, the mesh was more reliable to provide good results.

Case Study 3 was the first case where the Iguana plugin was used. Inserting a steel dowel into the timber beam was not a big problem for the Tetrino plugin. As seen on Figure 5.3.3, the Tetrino plugin could define the steel dowel better than Iguana. Nevertheless, the Tetrino plugin created a mesh not only consisting of 4-node tetrahedra but also some 5-node elements, which the FEM solver does not support. As the dowel representation using Iguana was acceptable, though not perfect, it was evaluated as the most usable meshing method to proceed with. The Iguana plugin creates a more free and less structured tetrahedral mesh than Abaqus, so comparing solutions from the two methods based on the mesh structure was not relevant. Solutions obtained from Abaqus with a highly dense mesh of higher-order tetrahedra were used as references for the results obtained from the FERret solver.

The tables and figures in Case Study 3 showed that the Iguana plugin was able to provide good results through the FERret solver. Even though the mid-node results were more inaccurate than the ones in the top and bottom nodes, this error is not evaluated as a problem mainly due to the meshing, as mentioned above, but rather a question of interaction properties between the parts. However, the different meshing structures will surely give some differences in the answers.

The main problem in Case Study 3 related to the meshing was representing the steel dowel precisely. Tetrino solved this problem easily, so it should be the preferable meshing algorithm in this case. Therefore, in similar circumstances, the Tetrino meshing plugin would probably be the best choice of meshing plugins, given that it creates mesh elements that the FEM solver supports. If

this is not the case, it could be investigated if the elements that are not supported can either be manipulated into a supported element or safely be removed from the mesh without causing problems or unacceptable accuracy. This investigation is not done in this thesis but is encouraged as further work if the issue remains relevant in future structural problems.

Case Study 4 was a structural problem where despite a challenging geometry, the Tetrino proved to be the best alternative for meshing. Figure 5.4.4 showed that Tetrino was able to describe the distorted and irregular shape of the reused timber element well when the element sizes were sufficiently small. Due to the uneven surface, Iguana showed to be weaker than Tetrino in adjusting the mesh size around these irregularities to represent the geometry efficiently and precisely.

For Case Study 5, the geometry was equal to the one in Case Study 2. Thus the same meshing procedure was used for the same reasoning for that case. In Case Study 6, some problems with Tetrino again occurred as the small cylindrical dowels required a large number of elements and thus caused a significant increase in computational time. The solution of reducing the model to only the significant parts solved the main issue, but in future problems, this might not be an alternative. From the discussion above, it is clear that the meshing problems occur independently of the FEM solver used, as the two last case studies use a rather different solver inside the FEniCSx environment.

The work done in the case studies in this thesis clearly shows that the meshing procedure is a large and vital part of the FE analysis. The time measurements during the case studies reveal that Tetrino is the fastest meshing tool. It only uses around 3.4% of the total analysis time to mesh the geometry. On the other hand, Iguana generates elements with relatively similar sizes but consumes up to 35.3% of the total time for meshing.

7 Conclusion

To conclude the research question, the six case studies highlight the potential of implementing FEM analysis using solid elements during the conceptual phase through using an AAD environment like Grasshopper. As the first case study was rather an intro to the FERret solver, the following two case studies focused on developing and improving this particular FEM solver. By enabling the possibility of analyzing problems that contain several parts of both isotropic and orthotropic materials, the plugin has become more general and flexible, as it can handle more realistic and complex structural problems in future cases. This generality of the solver makes it more relevant to utilize at meetings for decision-making during the conceptual phase of a project.

Two different methods were used in developing the plugins in the case studies. While FERret was based on matrix algebra, FEniCSx solved PDEs to obtain the results in the FEM analysis. While FERret has a wider range of applications, FEniCSx showed impressive accuracy and low time consumption for relatively complex structural problems. Ultimately, both plugins have possible improvements that would increase their potential even more.

The advantage of implementing FEM in such an early stage is that problems that traditionally appear later in the project may be detected at an earlier stage, where it is easier and cheaper to make the necessary changes. The case studies show that the FEM solver inside an AAD environment is not necessarily quicker than the commercial software Abaqus. However, the development of the FEniCSx solver has led to a plugin that provides the desired results quicker than the well-known commercial software. With an adequately dense mesh, solutions can be obtained with acceptable accuracy within a short amount of time. The parametric approach of the FEM solver enables the user to easily change the geometry and conditions while retrieving updated results in real-time.

The most challenging part of the implementation of FEM in Grasshopper is the mesh procedure. Multiple mesh algorithms have been utilized in this thesis. However, none of them are able to easily construct a mesh of complex geometry that the FEM solver effectively interprets. Improvements related to meshing tools can, for this reason, increase the process's efficiency and accuracy, making the potential even more remarkable.

8 Further work

As this is an early development of both FERret and FEniCSx, there are a few points that the authors believe should be further investigated in order to achieve versatile and powerful plugins for use in design in the future.

In FERret, the development of isotropic materials has progressed considerably, and this solver appears to perform well. However, for orthotropic materials, the solver requires further work to function effectively with real projects. The method employed by the authors, implementing different parts, is suitable for simple systems where all elements act along the principal axes. However, complications arise when introducing elements at an angle to the global axes. To resolve this problem, it's crucial to find a way to incorporate rotational matrices that accurately define the engineering constants in relation to the global axes.

In addition, the FERret software faces a significant challenge due to the limitations imposed by the C# programming language on matrix sizes. It is crucial to overcome this problem to analyze more extensive problems, such as Case Study 6, with a satisfying mesh division. One solution to the problem is to store the indices and the values of non-zero numbers instead of constructing the entire matrix within the program. This approach could potentially manage larger matrices, but it may come at the expense of higher computational processing. It may be beneficial to consider creating and solving the matrices in a different programming language, as the limitations are imposed by C# itself. If the problem is not resolved, it could become a significant obstacle, requiring the implementation of alternative software like FEniCSx into the FERret plugin.

FEniCSx is still in its early development stages and has some notable limitations. Currently, it can only handle one specific type of boundary condition, where all directions are restricted. However, for the software to be useful in practical applications, it needs to be able to support a variety of constraints. For FEniCSx to fulfill the role of FERret, the solver must be enhanced to cover orthotropic materials and allow for integrating several material types in one model.

Furthermore, there are significant limitations regarding the software's ease of use when accessed directly through Grasshopper. Currently, users are only able to choose geometry, loads, and boundary conditions. However, determining material parameters and incorporating self-weight require manual execution using separate code. The current input methodology, which involves vertices and indices, should also be modified to enable direct mesh input. Thus, the possibility of connecting multiple BREPs entities instead of a collection of all entities becomes feasible. As a result, the software needs significant improvements to make it user-friendly and practical for use in professional settings on a daily basis.

Generating appropriate meshes is a significant challenge for both meshing programs. To make Finite Element Analysis with solid elements accessible for everyday use in an AAD environment, a meshing tool tailored explicitly for such software should be developed. This tool should be capable of creating structured and unstructured meshes quickly and efficiently, enabling effective analysis with low computational time.

Bibliography

- Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., & Wells, G. N. (2015). The fenics project version 1.5. *Archive of Numerical Software, Vol 3*, Starting Point and Frequency: Year: 2013. <https://doi.org/10.11588/ANS.2015.100.20553>
- Bell, K. (2013). *An engineering approach to finite element analysis of linear structural mechanics problems*. Fagbokforlaget.
- Cook, R. D., & Malkus, D. S. (2002). *Concepts and applications of finite element analysis 4th edition*. John Wiley; Sons, Inc.
- FEniCS. (2021). *Fenicsx*. Retrieved 19th April 2023, from <https://fenicsproject.org/>
- FEniCS. (2023). *Elasticity*. Retrieved 7th May 2023, from https://docs.fenicsproject.org/dolfinx/v0.6.0/python/demos/demo_elasticity.html
- Laake, K. (2020). *Ngs løsninger gjør det mulig å nå eus krav om 70% materialgjenvinning*. Retrieved 13th May 2023, from <https://blogg.norskgjenvinning.no/losninger-som-gjor-det-mulig-a-na-eus-krav-om-70-materialgjenvinning/>
- Malo, K. A. (2021). Anisotropy in wooden materials [lecture note]. TKT4212.
- Mathisen, K. M. (2021). Finite element formulations for solid problems [lecture 16]. TKT4192.
- Microsoft. (2023a). *Visual studio code faq*. Retrieved 19th April 2023, from <https://code.visualstudio.com/docs/supporting/FAQ/>
- Microsoft. (2023b). *Visual studio: Ide and code editor for software developers and teams*. Retrieved 28th February 2023, from <https://visualstudio.microsoft.com/#vs-section/>
- Nätt, T. H. (2023). *C#*. Retrieved 28th February 2023, from <https://snl.no/C%23/>
- SimplyRhino. (2023). *Grasshopper for rhino 3d*. Retrieved 28th February 2023, from <https://simplyrhino.co.uk/3d-modelling-software/grasshopper/>
- Simulia. (2023). *Why abaqus?* Retrieved 8th June 2023, from <https://www.3ds.com/products-services/simulia/products/abaqus/>
- Staab, G. H. (2015). *Laminar composites*. Butterworth-Heinemann.
- Suzuki, E. (2021). *What is cad (computer-aided design)?* Retrieved 29th May 2023, from <https://www.autodesk.com/products/fusion-360/blog/what-is-cad-computer-aided-design/>
- Synthesis. (2021). *What is the cost of change?* Retrieved 29th May 2023, from <https://www.synthx.com/engineering-change-request/>
- Tedeschi, A. (2014). *Aad - algorithms-aided design : Parametric strategies using grasshopper*. Le Penseur.
- TreFokus. (2017). *Hvorfor er tre et miljøvennlig byggemateriale*. Retrieved 13th May 2023, from <http://www.trefokus.no/treveilederen/temaer/miljo-og-berekraft/hvorfor-er-tre-et-miljovennlig-byggemateriale-/>
- Ubuntu. (2023). *What is an ubuntu appliance*. Retrieved 7th June 2023, from <https://ubuntu.com/appliance/about>
- Zienkiewicz, O., & Taylor, R. L. (2013). *The finite element method: Its basis and fundamentals*. Butterworth-Heinemann.

Appendix

A GitHub Repositories

| Project | GitHub link | Relevant Branch |
|-----------------|---|--------------------|
| PreFERret code | github.com/vegardoyre/PreFERret.git | Vegard_branch |
| FERret code | github.com/marcinluczkowski/SolidFEM_ERret.git | Master_Vegard_Lars |
| FEniCSx code | github.com/augustjohansson/GH-FE.git | develop_01 |
| GH Case Study 1 | github.com/vegardoyre/CaseStudies.git | Case-Study-1 |
| GH Case Study 2 | github.com/vegardoyre/CaseStudies.git | Case-Study-2 |
| GH Case Study 3 | github.com/vegardoyre/CaseStudies.git | Case-Study-3 |
| GH Case Study 4 | github.com/vegardoyre/CaseStudies.git | Case-Study-4 |
| GH Case Study 5 | github.com/vegardoyre/CaseStudies.git | Case-Study-5 |
| GH Case Study 6 | github.com/vegardoyre/CaseStudies.git | Case-Study-6 |

Table A.1: GitHub repositories for code and Grasshopper files used in the thesis.

B Videos

| Filename | Relevant Study Case | Link |
|-------------------------|---------------------|---|
| CaseStudy2.mp4 | Case Study 2 | youtube.com/watch?v=07PgDBP4aLc |
| CaseStudy2_Rotation.mp4 | Case Study 2 | youtube.com/watch?v=8plN5ai6GMs |
| CaseStudy3.mp4 | Case Study 3 | youtube.com/watch?v=E1HQ_i6ECAA |
| CaseStudy4.mp4 | Case Study 4 | youtube.com/watch?v=dIsUBiz2K_8 |
| CaseStudy5.mp4 | Case Study 5 | youtube.com/watch?v=1oD-GON1vBE |
| CaseStudy6.mp4 | Case Study 6 | youtube.com/watch?v=KpKBkenk07g |

Table B.1: Videos visualizing the case studies in the thesis.



 **NTNU**

Norwegian University of
Science and Technology