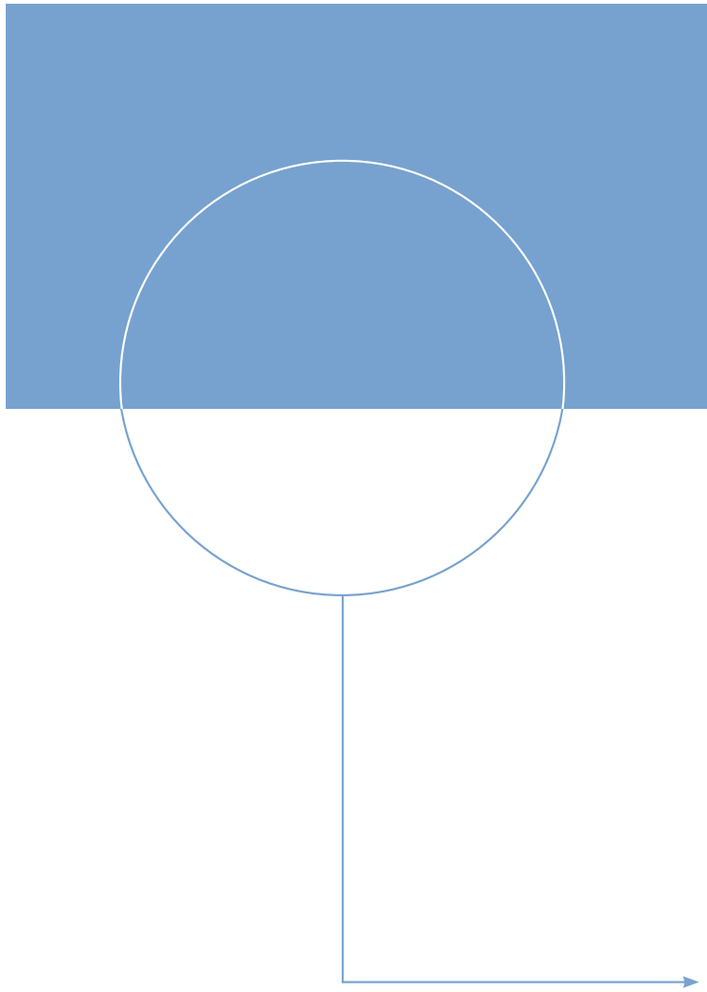


Master's thesis

2019

Martin Vee Akselsen, Synne Tjøtta Stenvold

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

Martin Vee Akselsen
Synne Tjøtta Stenvold

Virtual Reality – Only a Hype or Real Improvement for Structural Design?

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

Virtual Reality – Only a Hype or Real Improvement for Structural Design?

Martin Vee Akselsen

Synne Tjøtta Stenvold

Engineering and ICT

Submission date: June 2019

Supervisor: Anders Rønnquist, KT

Co-supervisor: Marcin Luczkowski, KT

Norwegian University of Science and Technology

Department of Structural Engineering

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

ACCESSIBILITY

Open

MASTER THESIS 2019

SUBJECT AREA:	DATE:	NO. OF PAGES:
Structural Engineering	06.06.2019	xii + 119

TITLE:

Virtual Reality – Only a Hype or Real Improvement for Structural Design?

Virtuell virkelighet – bare en hype eller reell forbedring for konstruksjonsdesign?

BY:

Martin Vee Akselsen and Synne Tjøtta Stenvold

SUMMARY:

The purpose of this thesis is to examine whether virtual reality (VR) only is a technological hype or if it could contribute to a real improvement for structural design. A product prototype was created to help explore the opportunities with VR.

The product uses the finite element method (FEM) to analyze 3D models, and combines VR and the parametric environment in Grasshopper. To create this product, a connection between Grasshopper and VR was established through existing plug-ins. The lack of analysis programs in Grasshopper using 3D elements for meshing motivated to the creation of a new plug-in.

The resulting product, called SolidsVR, is a plug-in to Grasshopper meshing solids into 8-node hexahedron elements. SolidsVR includes pre-processing and calculation components, as well as post-processing components dedicated to display the model and results to the user in VR. SolidsVR is kept parametric in VR, facilitating a two-way communication between the user and the VR interface. This allows the user to make changes to the model and get updated results in near real-time.

The analysis results were compared to an existing FEM program, and it is concluded that the accuracy is sufficient with an adequate number of elements in the mesh.

Challenges were discovered during research, but none of which was big enough to undermine the benefits and possibilities with VR in structural design. A clear benefit of using VR was the easy navigation and intuitive visualization of 3D structures.

It is concluded that VR could provide great possibilities for structural design improvement. To continue to explore this topic, SolidsVR can be used as the product prototype it is today, or it can easily be extended with additional functionalities.

RESPONSIBLE TEACHER: Professor Anders Rønnquist

SUPERVISOR(S): Professor Anders Rønnquist and PhD candidate Marcin Luczkowski

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

Preface

This thesis concludes our Master of Science degree in Engineering and ICT (both with ICT and Structural Engineering as main profile) at the Department of Structural Engineering at the Norwegian University of Science and Technology. The course code and name is TKT4920 Structural Design, Master's Thesis.

We would like to thank our supervisor Professor Nils Erik Anders Rønnquist and PhD candidate Marcin Luczkowski for their interest in supervising a thesis exploring a relatively new topic and their enthusiasm throughout the process. We would also like to express our gratitude to Professor emeritus Kolbein Bell and PhD candidate Katarzyna Ostapska-Luczkowska for their eagerness to provide insights into the theory and application of the finite element method.

Martin Vee Akselsen and Synne Tjøtta Stenvold

Trondheim, June 2019

Summary

The purpose of this thesis is to examine whether virtual reality (VR) only is a technological hype or if it could contribute to a real improvement for structural design. Many companies are already using VR as a visualization tool, but these software programs do not have a two-way connection providing the possibility to manipulate what is in VR. The increase of computational power and the popularity of parametric modeling motivates to move structural analysis in a parametric environment into VR.

To do this, a connection between Grasshopper and VR was established through existing plug-ins to explore the structural design possibilities in VR. The lack of analysis programs in Grasshopper using 3D elements for meshing motivated to the creation of a new plug-in. Based on the theory chapter dedicated to the finite element method, an analysis program meshing solids into 8-node hexahedron elements were made. The program, called *SolidsVR*, is a package consisting of 24 self-made components usable in a Grasshopper script by dragging them onto the canvas.

In addition to pre-processing and calculation components, SolidsVR includes post-processing components dedicated to display the model and results to the user in VR. Additional interaction components were created to keep SolidsVR parametric, even in VR. This facilitated a two-way communication between the user and the VR interface, allowing the user to make changes to the model and get updated results in near real-time.

SolidsVR was created in smaller portions at a time, enabling testing and discussion during development. The way such plug-ins are implemented made it possible to adjust the self-made components or add new ones as more features were desired. This resulted in a program meshing and analyzing relatively advanced geometry, with load and boundary conditions. As an attempt to add an exciting feature to explore in VR, the possibility to iteratively remove the most ineffective elements in terms of stress value was added. The VR interface visualizes the adjustable model, as well as the stress distribution in a chosen direction using colors and a scalable deformed geometry.

SolidsVR is a product prototype, and it will still benefit from further development. However, SolidsVR worked as the intended proof of concept – a realization of a concept to demonstrate its practical potential.

The analysis results were compared to Abaqus', and it is concluded that the accuracy is sufficient with an adequate number of elements in the mesh. For a squared cube meshed into 125 elements, the von Mises stress in a corner node only differs from Abaqus' by 0.32%. For more advanced geometry, a finer mesh is needed to obtain the same results. Further, increasing the number of elements increases the computation time of the analysis. With a cube meshed into 1000 elements, the computation time for one analysis by the test computer was 3 seconds. Finer mesh than this pushes the limit of this thesis definition of near real-time. However, this is still promising results, and with exponentially increasing computational power, this is not considered a concern.

The most significant limitation to SolidsVR was the existing software programs. As many components were created to get around the challenge of making SolidsVR parametric in VR, it may have affected the user experience when testing and exploring the product prototype in this temporary user interface. Despite this, the connection with VR worked as expected, and analyzing different types of geometry provided insight into VR as a tool for structural design. A clear benefit of using VR was the easy navigation and intuitive visualization of 3D structures.

Challenges were discovered during research, but none of which was big enough to undermine the benefits and possibilities with VR in structural design. Based on the insight gained through this thesis, the combination of VR and a parametric environment appear to be promising as an analyzing and visualization tool in structural design.

If VR is to be adopted into the industry, a customized VR connection and interface must be created to fit structural design and the parametric environment. The same argument is used regarding the VR-training of employees and software programs to be adapted to VR. If VR could be a real improvement for structural design, resources will be set aside by the relevant firms to cover these upgrades.

It is concluded that VR could provide great possibilities for structural design improvement. To continue to explore this topic, SolidsVR can be used as the product prototype it is today, or it can easily be extended with additional functionalities.

Sammendrag

Formålet med denne avhandlingen er å undersøke om virtuell virkelighet (VR) bare er en teknologisk hype eller om den kan bidra til en reell forbedring for kontruksjonsdesign. Mange bedrifter bruker allerede VR som et visualiseringsverktøy, men disse programmene har ikke en toveisforbindelse som gir brukeren mulighet til å manipulere det som befinner seg i VR. Den økende beregningskraften og populariteten til parametrisk modellering gjør det interessant å flytte konstruksjonsanalyse i et parametrisk miljø over til VR.

For å få til dette ble en forbindelse mellom Grasshopper og VR etablert gjennom eksisterende programtillegg for å utforske muligheter innenfor kontruksjonsdesign i VR. Mangelen på analyseprogrammer i Grasshopper som bruker 3D-elementer til meshing, motiverte til opprettelsen av et nytt programtillegg. Basert på teorikapitlet om elementmetoden ble det laget et analyseprogram som mesher massiv romgeometri i 8-nodet heksaederelementer. Programmet, kalt *SolidsVR*, er en pakke som består av 24 egenutviklede komponenter som kan brukes i et Grasshopper-skript ved å dra dem over til skjermen.

I tillegg til preprosseserings- og beregningskomponenter, innholder SolidsVR postprosseseringskomponenter som visualiserer modellen og resultatene til brukeren i VR. I tillegg er det laget interaksjonskomponenter for å holde SolidsVR parametrisk, også i VR. På denne måten ble det etablert en toveiskommunikasjon mellom brukeren og VR-grensesnittet, slik at brukeren kan gjøre endringer på modellen og få oppdaterte resultater i nær sanntid.

SolidsVR ble laget i mindre deler av gangen, noe som muliggjorde testing og diskusjon underveis i utviklingen. Måten slike programtillegg implementeres gjør det mulig å endre de egenutviklede komponentene eller legge til nye dersom flere funksjoner er ønsket. Dette resulterte i et program som mesher og analyserer relativt avansert geometri, med last- og grensebetingelser. Som et forsøk på å legge til en spennende funksjon å utforske i VR, ble muligheten å iterativt fjerne de mest ineffektive elementene i forhold til spenningsverdien lagt til. VR-grensesnittet visualiserer den justerbare modellen, samt spenningsfordelingen i en valgt retning ved hjelp av farger og en skalerbar deformert geometri.

SolidsVR er en prototype, og vil kunne forbedres gjennom videreutvikling. SolidsVR fungerte likevel i henhold til intensjonen – en realisering av et konsept for å demonstrere det praktiske potensialet.

Analyseresultatene ble sammenlignet med Abaqus sine, og det konkluderes med at nøyaktigheten er tilstrekkelig, gitt nok antall elementer i meshet. For en kube som er meshet i 125 elementer, varierer von Mises-spenningen i en node med 0,32% fra Abaqus. For mer avansert geometri er det nødvendig med et finere mesh for å oppnå de samme resultatene. Videre vil økt antall elementer øke beregningstiden for analysen. Med en kube meshet i 1000 elementer var beregningstiden for analysen utført av testdatamaskinen 3 sekunder. Med finere mesh enn dette nærmer man seg grensen til denne avhandlingen sin definisjon av nær sanntid. Uansett er dette lovende resultater, og med en eksponensielt økende beregningskraft anses ikke dette for å være en bekymring.

Den største begrensningen for SolidsVR var de eksisterende programmene. Ettersom mange komponenter ble laget for å komme rundt utfordringen med å gjøre SolidsVR parametrisk i VR, kan det ha påvirket brukeropplevelsen da prototypen ble testet og utforsket i et midlertidig brukergrensesnitt. Til tross for dette virket forbindelsen med VR som forventet, og det å analysere ulike typer geometri ga nyttig innsikt i VR som et verktøy for konstruksjonsdesign. En tydelig fordel med VR var den enkle navigasjonen og intuitiv visualisering av kontruksjoner i 3D.

Utfordringer ble oppdaget underveis, men ingen av dem store nok til å overskygge fordelene og mulighetene VR gir konstruksjonsdesign. Basert på innsikt tilegnet gjennom denne avhandlingen, ser kombinasjonen av VR og et parametrisk miljø ut til å være et lovende analyse- og visualiseringsverktøy i kontruksjonsdesign.

Hvis VR skal innføres i bransjen, må det lages en VR-tilkobling og et brukergrensesnitt som er tilpasset konstruksjonsdesign og det parametriske miljøet. Det samme argumentet gjelder VR-opplæring av ansatte og programvarer som må tilpasses VR. Hvis VR viser seg å være en reell forbedring for konstruksjonsdesign, vil relevante firmaer bruke ressurser på å dekke disse oppgraderingene.

Det konkluderes med at VR vil kunne bidra til forbedring for konstruksjonsdesign. For å fortsette å utforske dette temaet, kan SolidsVR brukes som den prototypen det er i dag, eller enkelt utvides med tilleggsfunksjoner.

Table of Contents

Abbreviations

FEM	=	Finite element method
FEA	=	Finite element analysis
CAD	=	Computer-aided design
OO	=	Object-oriented
BIM	=	Building information modeling
CPU	=	Central processing unit
NURBS	=	Non-uniform rational basis spline
VR	=	Virtual reality
AI	=	Artificial intelligence
PVD	=	Principle of virtual displacements
PVW	=	Principle of virtual work
gdof	=	Global degrees of freedom
ldof	=	Local degrees of freedom
Hex8	=	Triliniear hexahedron element

Chapter 1

Introduction

The computational power is increasing, and heavier calculations can be done faster. This motivates to wider use of 3D elements when analyzing solids, rather than simplifying to lower dimensions.

The popularity of parametric design is also growing among structural engineers. Parametric design lets you specify key parameters of the model and chge them interactively. When analyzing a parametric model and one or more parameters are changed, it will have to be analyzed again. With the model updating automatically, engineers may be able to explore options in a more efficient manner. To maintain the benefits of parametric design, it is preferable that the analysis is relatively fast.

When combining the two trends of increasing computational power and parametric design, it is relevant to explore the finite element analysis (FEA) of parametric 3D models where results are provided to the user in near real-time.

While designing and analyzing in 3D, visualizing may be more demanding than in lower dimensions. Representing 3D through a 2D screen provides a perspective view of the model, but clicking and navigating may not be as precise. VR is a tool that can be used to see how a structure will look like in real life and enables the user to interact with it.

The intent of this thesis is to explore the use of VR in structural design and state whether it could contribute to real improvements to the structural design industry or if it is just a way to play with new technology. The aim is to introduce a product prototype. This is a proof of concept study, meaning that the product is not intended to be complete, but a realization

of the idea to test if the concept has practical potential. The product should:

- Use the finite element method (FEM) to analyze 3D models.
- Be parametric.
- Be connected to VR.

As an attempt to achieve this, a software package called *SolidsVR* is created. SolidsVR is a plug-in for Grasshopper, a parametric modeling software, which provides extended functionality such as FEA of solids in 3D. The package includes components connecting with VR, facilitating a two-way communication between the product and the user.

In Chapter 2, the development of computational design software is briefly examined, and relevant software programs used in this thesis are presented. Next, in Chapter 3, a literature review is conducted on virtual reality, and the discussion on how to merge it with the structural engineering discipline is introduced. The theory behind all calculations is presented in Chapter 4. Chapter 5 is about how the workflow of the desired solution will look like from the software perspective and the user perspective. Further, the solution method used to develop the product prototype is described in Chapter 6. This is done in an iterative manner, where each step is a study case dividing Chapter 6 into sections. Case 1 is the base version of the product analyzing a simple geometry. Case 2 and 3 include more advanced geometry, and Case 4 is testing a new feature. In Chapter 7, the time performance of the product is analyzed. Lastly, Chapter 8 includes a discussion, followed by concluding remarks in Chapter 9. Chapter 10 presents possible extensions to the product created in this thesis and outlines future research areas.

Chapter 2

Software

This chapter starts with a historical note on computational design software in Section 2.1. In Section 2.2, we discuss the evolution of computational power, its future outlook, and how it may impact the possibilities of computational design. Section 2.3 introduces the software programs used in this thesis, followed by a flowchart illustrating the workflow between them in Section 2.4. Lastly, in Section 2.6, the programming tools used are presented.

2.1 The History of Computational Design Software

To understand the history of design software, three ideas are worth looking into. The Sketchpad, formal design methods and object-oriented software.

Sketchpad, developed by Ivan Sutherland in 1963 during his PhD thesis, is the original computer-aided design (CAD) system. This software was an early example of the model-view-controller pattern, where the model is the constraint system, the view is the display and the light pen is the controller [1]. Note that the Sketchpad is matched to the cognitive abilities of the user, meaning that the program's functionality is often limited by the skills of the engineer using it [2].

Formal design methods are the idea where hierarchical decomposition is used to understand a design problem. Research in the 1970s explored the use of a more general directed acyclic graph, where the nodes are design decisions, or simply components, and the arcs are dependencies between them. This idea is complex on a large scale, but the dependency

graph is easily implemented as a program and shared between the man and machine [2].

In the early 1980s, object-oriented (OO) software came to light [3]. A better user model is one that mimics some of the attributes and behavior of real-world objects. CAD is doing in some way the opposite, namely representing real-world objects as a computational system. To summarize, an object-oriented CAD system is re-applying the OO concepts, derived from real-world objects, to the design of further real-world objects.

The history of practical CAD may also be divided into three eras. The first one, the 2D drafting era, started in the 1980s. It was all about representing buildings as multiple 2D drawings. Some people see this as a way to make architecture more conservative and not use the available technology or as a travesty of the Sketchpad's intentions. Others say that it is a way to make the technology available to the public [2]. However, the challenge is not just to make the technology available, but also the underlying concepts such as understanding the constraint system and a more profound way to think. Again, the use is limited to the engineers' knowledge.

Second comes the building information modeling (BIM) era that surprisingly started before the 2D drafting era. The BIM era is based on creating one single 3D model, and the 2D drawings are extracted from the model later. Some users would say it is hard to be innovative developing models with BIM and that it is better used to create the "obvious" in an efficient and structured way [4].

The last era, the design computation era, is introducing the difference between the generative description of a building and the resulting model. The user is no longer modeling the structure, but a script. This script is editable and the model will change accordingly. This is what we call parametric design and enables the user to explore several alternatives. The parametric software programs used in this thesis will be introduced in Section 2.3.

Today, software programs using the finite element method can analyze any geometry with a sufficient degree of accuracy [5]. Every such finite element application must consider the difference between beams, shells and solids. Solids are 3D objects, while shell elements are abstracted to 2D elements by storing the thickness separately. Beam elements are abstracted to 1D elements by storing the cross section as a separate property. For thick walled and solid components, meshing into 3D elements would be preferable to achieve good results. Analyzing such elements require more computational power than in 2D and 1D. This is why solids are often simplified to shell structures when two of the three dimensions dominates. The same argument can be used for beams as they are most suitable when one dimension dominates the other two.

2.2 The Impact of the Increased Computer Performance

A relevant aspect to examine is the development of computational power. A central processing unit (CPU) is what carries out the instructions of a computer program by performing basic operations. The faster the CPU runs, the more processes it can run at any given time. There are several ways to increase the performance, and history shows that the power and speed of computers have increased exponentially. Gordon E. Moore picked up on this exponential growth rate in the 1960s and described it in what is known as Moore's Law [6]. Figure 2.1 illustrates this evolution of CPU performance.

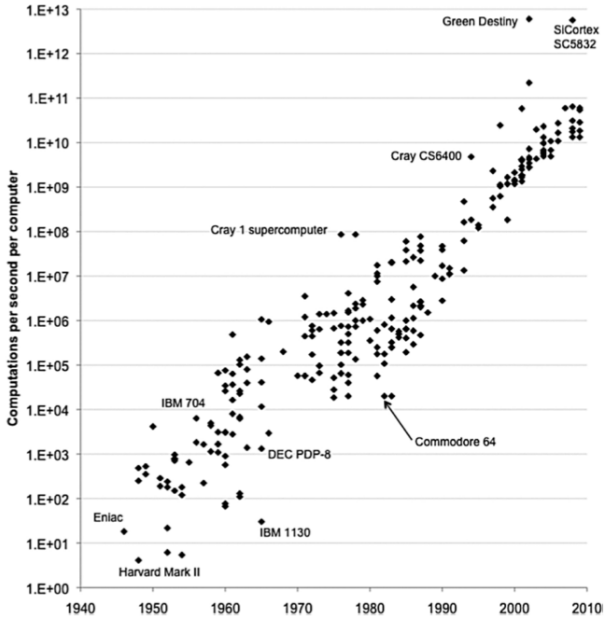


Figure 2.1: The historical increase of CPU performance. [6]

Even though it is debatable if Moores's law will continue to hold, there are still reasons to believe the growth will continue [7].

Changing the focus back to computational design and FEA, the increase in computational power may contribute to a wider use of 3D elements. This also came up in a meeting with Kolbein Bell (March 15th, 2019) where he explained how they decades ago spent much time using advanced elements for meshing to minimize the number of equations. Instead, with today's computational power, it is more efficient to use simpler elements, and compensate with a finer mesh.

2.3 Parametric Software: Rhinoceros and Grasshopper

Rhinoceros 6 (Rhino) is a computer-aided design application. According to their own web site, Rhino focuses on producing a mathematically accurate representation of curves and free-form surfaces based on the NURBS mathematical model [8]. However, Rhino is limited in keeping track of history. The lack of connection between parts of the model makes even small changes to it a comprehensive action. This issue motivated to develop today's Grasshopper [9], a free plug-in for Rhino.

Grasshopper is a visual scripting environment. Instead of regular programming, visual programming lets the developer manipulate the program graphically, a way that makes more sense to humans [10]. Using the concept of visual programming, a Grasshopper file consists of components with connections between them representing operations. One component's output may be another component's input. More importantly, Grasshopper can be used for parametric modeling. This is done by using number sliders or toggles as input to the components. Grasshopper allows you to go back to earlier steps, change these input values, and instantly affect the result. It should be noted that components are not created for one specified solution, and they can be reused in other solutions.

In Figure 2.2, a straight line is created parametrically in Grasshopper. The line consists of two input points, and the points consist of three values, one for each direction in the coordinate system. The value inputs are set to an adjustable number slider. The point will be modified as changes are done to its input parameters, and so will the line. Grasshopper uses Rhino to display the resulting model.

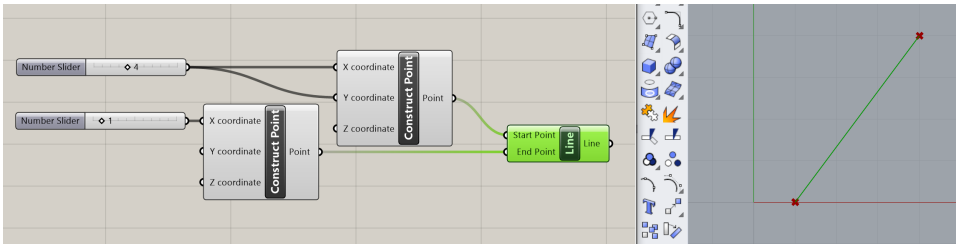


Figure 2.2: Parametric modeling in Grasshopper (left) creating a line in Rhino (right).

There exists fully embedded software packages in Grasshopper that provides analysis of trusses, frames and shells. This is one type of plug-ins made to enhance or extend Grasshoppers functionality. As of today, these analyzing tools do not provide the possibility to analyze solids in 3D.

2.4 Workflow

The software programs introduced in the past section have a connection that can be described as data sent between the programs. This workflow is illustrated in Figure 2.3 using a flowchart. The circles in the flowchart represent the programs and the arrows represent data being sent from one program to another. There is also a square in the workflow representing plug-ins and the straight line indicates what software program the plug-ins are downloaded to.

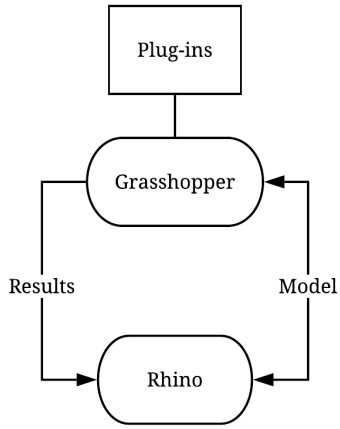


Figure 2.3: How Rhino and Grasshopper are connected.

The figure illustrates the two-way connection between Rhino and Grasshopper. It is described as sending the model back and forth. Every time a Rhino geometry or a parameter in Grasshopper is changed, it triggers the communication, and updated data is sent between the programs. The chosen action is performed in Grasshopper, either by built-in functions or as a plug-in, and the result from this is sent to Rhino. However, geometry created in Grasshopper is only *visualized* in Rhino. It is not possible to edit the geometry in Rhino before it is exported from Grasshopper to the current Rhino document.

The results sent from Grasshopper are previewed in Rhino as for example analysis results or geometry.

2.5 Abaqus/CAE

Abaqus/CAE, or simply Abaqus, is a software program used to model, analyze and visualize the finite element analysis result. The most precise software depends on the purpose, but Abaqus is a robust software for FEA in general [11]. Therefore, Abaqus is used as a benchmark when comparing the accuracy of results in this thesis.

The comparisons are done without *Reduced Integration* and *Incompatible Modes*.

2.6 Programming Support

A part of this thesis is to create a new plug-in for Grasshopper. To do this, the programming language C# is used in Visual Studio 2017 for Windows [12]. Grasshopper is written in C#, and it is therefore facilitated for creating components in the same language. The process of creating a new Grasshopper component is simplified by the Grasshopper add-on to Visual Studio providing an empty script for creating new components.

Being two people programming together, Git has been used as a tool to collaborate due to its many advantages. With simple commands, the code is uploaded to Git, shared with other members and added to their local version of the code. Git also has a complete history of all changes made to the code, giving the user the possibility to go back to former solutions if needed. GitHub [13] is a service for managing projects using Git, working as an interface for controlling and distributing the code.

Chapter 3

Virtual Reality

Section 3.1 introduces the term virtual reality. Virtual reality's current status is presented, and we discuss both challenges and opportunities in Section 3.2. Next, in Section 3.3, it is explored how structural engineers can take further advantage of virtual reality. Section 3.4 is about the virtual reality equipment used in this thesis and Section 3.5 presents the connection between virtual reality and the rest of the software, including an updated flowchart illustrating the workflow.

3.1 Virtual Reality and its History

Virtual reality (VR) is the use of computer technology that creates a simulated environment that emulates reality. Unlike other user interfaces, VR places the user inside this environment. The user is allowed to be influenced by and influence this computer-generated 3D world. To interact with VR, the user is given a head-mounted display and a set of controllers. The headset is used as an enhanced screen in front of the eyes, and the controllers allow manipulation of the virtual world using hands. Two or more sensors are placed around the room to track the equipment's movements.

The exact origin of virtual reality is unknown because it is debatable when the concept of an alternative existence started. However, the first reference to what we today call VR is from science fiction, already in the 1930s. The first VR head-mounted display came in 1960, and a year later motion tracking was added to it. The term *virtual reality* was not

coined until 1987 and the devices started to reach the public in the middle of the 90s [14]. The household ownership was limited, but people had high expectations. Many people got a feeling that VR did not live up to its early promises. As a result, they started to lose interest and the hype around VR faded [15].

Later, in 2014, a VR start-up named Oculus VR sold their company to Facebook for more than $2 billion and Facebook's resources probably played a role in taking the VR equipment from a developers product to a consumer product [16]. As computer technology has exploded, it is expected that VR will be even more accessible in the years to come.

3.2 State of the Art, Challenges and Opportunities

Most people associate VR with gaming [17], and it is in gaming VR has seen the earliest and highest level of development. Gamers are better consumers in many ways when it comes to new technology, as they want to improve their gaming experience and eager to try new things. As the gaming industry can be considered as the state of the art of VR, it is relevant to look at some challenges it is facing:

- VR is encountering criticism when trying to revolutionize the gaming industries as a lot of equipment is needed, and the cost of it is high [18].
- Games and other software programs have to be renewed or adapted to VR to work. Companies are struggling to provide a sufficient amount of VR content to satisfy the gaming environment [19].
- It is rarely possible to use VR for a longer time period without getting disoriented or experience motion sickness. That is why users are recommended taking a 10 to 15 minute break for every 30 minutes playing [20].

Despite the difficulties the VR community has encountered so far, it is expected that the VR market will be worth $117 billion by 2022 [21]. According to the same report, this growth will be driven by business adoption, not gaming. The applications of VR has already expanded to other areas [22] and Facebook's founder, Mark Zuckerberg, was right in 2014 when he said that "after games, we're going to make [VR] a platform for many other experiences" [16]. Today, we see companies using VR for training employees. For example, doctors use it to practice difficult surgeries [23] and soldiers to prepare for military actions. Some universities also have their own virtual reality learning programs to make it easier for communities to gather and learn from each other [24].

In addition to growth in the businesses above, VR technology can be regarded as a natural extension to 3D computer graphics and brings a completely new environment to the CAD community. Many architectural firms show their customers and project owners how the result will look like before the construction period starts [25]. This helps to avoid dissatisfied customers as they may provide early input or changes to the model. Studies show that, compared to a regular 3D model, VR technology provides a promising way to visualize how the model will look in real life [26]. This is a one-way communication between the user and VR, where the user is only able to look at the design through the VR headset.

In light of this growth, the first two challenges mentioned may not seem as threatening. Where there is money, business will thrive, and the growing value of the VR market will become a boosting factor in creating new games, equipment and other software programs. All these products will become cheaper and more available to people as more producers get involved, creates competition and pushes the prices down. Oculus VR released their newest equipment in May 2019, and Tek.no highly recommends it [27]. This indicates that VR may yet become a massive mainstream hit.

Regarding motion sickness when using VR, there are two main reasons why this happens. The first, and the most obvious is the fact that the user is placed in a virtual environment where things appear to be moving when they are actually standing still. The challenge is difficult to get around, but ways to counteract VR motion sickness are currently being researched [28] [29]. The second challenge is the lag between the user's movements and the refresh rate. VR demands high computation power to run smoothly. As described in Section 2.2, the computer performance will continue to increase, and this lag may get close to unnoticeable in the future.

3.3 Further Possibilities in Structural Design

As the structural design industry is changing and more firms are interested in exploring technological opportunities, it is natural to examine future possibilities with VR. The use of VR in the gaming industry and other businesses proves that it is entirely possible to interact with the virtual world. In other words, VR does not necessarily have to be a one-way communication tool only used for visualization. The controllers enable the user to exercise commands, and with the appropriate software program, this can be used to manipulate and make changes to objects in the virtual world.

For a computational design software program used by a structural engineer, this feature

could make it possible to create the model *and* analyze it in VR. Imagine such a two-way communication where the user, without taking the VR headset off, can draw the geometry, put the loads and boundary conditions on and choose what type of analysis to execute in VR. The results from the analysis are also visualized in VR as soon as they are calculated – preferably in real-time.

Looking at a longer time horizon, ideas combining the visualization tools today and analyzing chosen parts of that parametric model in detail would likely be useful. The user might walk around in a virtual building, and when he or she comes across a beam or connection to analyze, it could be an option to click on it and get analysis results visualized in VR. Also, it could be possible to make changes to that particular object and watch how the stress distribution changes, and even how that change affects the rest of the building, just like in a parametric environment. The user could for example be an architect exploring creative possibilities or a structural engineer used to work with advanced analyzing tools.

If VR gets adapted to the industry, it will probably be more common to collaborate in VR. As software programs improve, it might be possible for several people to work on a structure simultaneously in VR. Combining the two visions above; the parametric building analyzer in real-time and collaboration in VR, one could imagine significant opportunities in cross-functional collaboration between architects and structural engineers. Having both disciplines in VR at the same time, the ability to visualize how changes will affect the structure and explore new ideas in real-time, could break the communication barrier and reduce misunderstandings between the two.

Looking even further into the future, and the rise of artificial intelligence (AI), one could imagine AI making suggestions for changes to the structure and the engineer and architect could make well founded decisions faster. Spacemaker [30] is a company already exploring this topic from an architectural perspective.

3.4 Software and Hardware: Oculus VR

Oculus VR [31] is a manufacturing company delivering hardware for VR use. This includes all the VR equipment used in this thesis: The VR headset Oculus Rift, two Oculus Rift Constellation sensors and two Oculus Touch controllers. This hardware is connected to the Oculus VR software program, which is the platform the user interacts within the virtual world.

3.5 Virtual Reality in Rhino and Grasshopper: Mindesk

In a meeting in September 2018, representatives from the architecture firm Bryden Woods and developers of Rhino, McNeel, discussed VR solutions for Rhino [32]. Bryden Woods expressed a need for a more interactive tool than the plug-ins for visualization there is today. The plug-ins possibilities to customize tools and analyze data are limited.

RhinoVR [33] is an attempt to solve this. It is an ongoing project developed by McNeel and is so far only used as a sample plug-in. RhinoVR is not intended to be a finished product, but more of a starting point for developers to build their own VR plug-in for Rhino. Giving away the source code on GitHub enables third-party development.

Mindesk has been doing just that RhinoVR was intended for, namely developed the open source code further. According to their website, Mindesk is the first and most advanced VR interface for CAD software like Rhino 6 and Grasshopper [34]. Mindesk and the software and hardware that come along with VR can now be added to the flowchart from Section 2.4 and results in Figure 3.1.

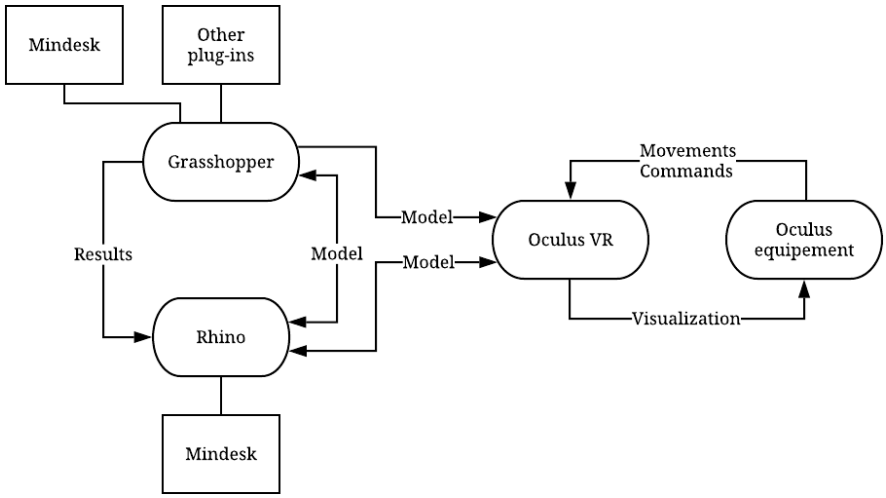


Figure 3.1: Current workflow representing how VR is connected to tools and software programs.

Mindesk is a plug-in installed in both Rhino and Grasshopper enabling the communication between the programs and Oculus VR. The user can build a Rhino 3D model, navigate through and edit it with the provided tools, all in VR. The Grasshopper geometry is not editable in VR, only previewed, unless it is exported to Rhino as explained in Section 2.4.

Chapter 4

Theory

The theory presented in this chapter only consists of the relevant theory for the implementation of the finite element method in this thesis. This chapter starts with introducing the finite element method in Section 4.1. It includes the basic knowledge and a detailed explanation of the components in the force-displacement relation. In Section 4.2, we focus on the special case of analyzing solids and how the calculations are done.

As this chapter only provides an introduction to the finite element method targeting the main purpose of this thesis, further reading can be found in the books used: "An engineering approach to FINITE ELEMENT ANALYSIS of linear structural mechanics" by Kolbein Bell [35], "Finite Element Modelling for Stress Analysis" by Robert D. Cook [36] and "Finite Element Procedures" by Klaus-Jürgen Bathev [37].

4.1 Introduction to the Finite Element Method

The finite element method (FEM) is a popular and powerful tool for computing displacement, strain and stress in a structure. FEM is based on the idea of dividing a geometry into smaller parts, called elements, stored in a mesh with the needed information for performing an analysis. It is called *finite* element because there is a finite number of degrees of freedom to describe the behavior of each element. After the meshing process of dividing the structure, a FEM solver produces the results typically using methods involving matrices. FEM only provides an approximation to the exact solution.

4.1.1 The Basics

4.1.1.1 Assumptions

The real world is complicated and it is in most cases impossible to replicate it in a model. Therefore, all FEM tools need some assumptions to simplify real-world problems. The most common assumptions, which are used in this thesis, are based on the two assumptions in linear theory as stated by Bell [35]:

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

In this thesis, all materials are considered homogeneous.

4.1.1.2 The Principle of Virtual Displacement

The formulation of FEM in this thesis is based on the principle of virtual displacements (PVD), which again is a special version of the principle of virtual work. PVD is stated by Bell as follows [35]:

"If a system of *real* external and internal forces (stresses) that are in *static equilibrium* are subjected to any set of *virtual*, but *kinematically compatible*, displacements (strains), then the virtual work performed by the real external forces over the virtual displacements is equal to the virtual work performed by the real internal forces over the virtual displacements."

Applying this principle on one single element is the beginning of the derivation of the relationship between force, displacement and stiffness.

4.1.1.3 Degrees of Freedom

All elements consist of nodes. Some elements have nodes only in their corners, and some also have nodes on their sides and surfaces. Each node has a finite number of degrees of

freedom (dofs), which represent independent nodal displacements that can vary freely. In general, displacements includes both transitional and rotational freedom.

The number of dofs in an element describes the complexity of the element. Similar to the number of nodes, a simple element has a few dofs, while a more complex element has more dofs describing complex variations within the element. The corner dofs are often associated with multiple elements, making them more effective than, for example, dofs inside the element.

There are also a difference between local and global degrees of freedom (ldof and gdof), where ldofs are related to each element and gdofs are related to the whole system.

4.1.1.4 Shape Functions

Shape functions are an essential part of the formulation of FEM. Equation 4.1 is expressing the direct interpolation

$$\mathbf{u} = \mathbf{N}\mathbf{v}, \tag{4.1}$$

where **N** is the shape function interpolating values between the degrees of freedom, **v**, for the displacement **u** inside the element.

Depending on the problem, proper shape functions must be used. Shape functions can have different orders and are closely connected with the degrees of freedom. To ensure correct analysis of a problem the shape functions must fulfill several requirements [35].

1. **Continuity**: If the order of differentiation in the strain-displacement relation is m, the field variables and their derivatives up to and including the order $m - 1$ must be continuous along the entire boundary between neighboring elements.

2. **Completeness:** The combination of **Nv** must be capable of representing the rigid body motions correctly, meaning pure rigid body motion must not produce stresses in the element, and for certain values of the nodal dofs, **v**, the shape functions must reproduce a state of constant stress.

3. The **interpolation requirement** can be formulated as follows:

   - $\sum_i N_i = 1$.

- N_i must yield v_i = 1 and $v_j = 0$ ($j \neq i$).
- $N_i = 0$ along all edges and surfaces that do not contain dof *i*.

The first point only applies to displacement dofs. The second applies to all elements, while the third only to 2- and 3-dimensional cases.

If an element satisfies all the requirements shown above, then the result will converge towards the correct result when the element size is decreased.

4.1.2 The Force-Displacement Relation

The most basic equation in FEM representing the relationship between force and displacement in each gdofs is given as

$$\mathbf{R = Ku}, \tag{4.2}$$

Where **R** is the force in each gdof and **u** is the displacement in each gdof. The relationship between these two is given by the system stiffness matrix, **K**, also called the global stiffness matrix. This equation is formed by building up a system of linear, algebraic equations from applying PVD on each element. Solving Equation 4.2 for **u** results in

$$\mathbf{u} = \mathbf{K^{-1}R}. \tag{4.3}$$

4.1.2.1 The Stiffness Matrix

The creation of the global stiffness matrix, **K**, is arguably the most critical part of FEM. This is achieved by adding together each element stiffness contribution, $\mathbf{K}^e$, into one single matrix.

The assembly of **K** is done by linking the ldofs in each element with gdofs in the model. Nodes shared by multiple elements will have stiffness contributions from all elements the nodes are part of. Therefore, when element stiffness matrices are added to the global stiffness matrix, some elements may add stiffness to the same entries in **K**. An example of the assembly process is illustrated in Figure 4.1, showing two distinct element stiffness matrices being added to **K**.

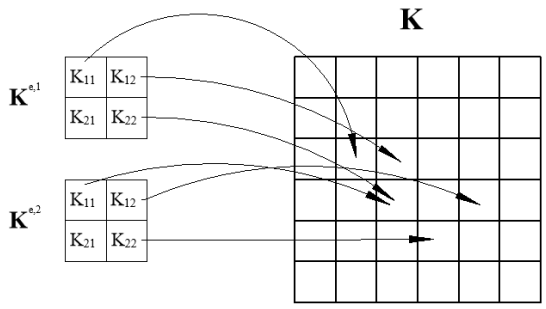


Figure 4.1: Simplified illustration of how element stiffness matrices are assembled to the global stiffness matrix.

As seen in Equation 4.3, **K** must be inverted to solve the linear system of equation. Unstable structures will have a singular **K** until the necessary boundary conditions are assigned to the model and the system may be solved. Some additional properties of **K** are pointed out:

- Both the element stiffness matrix and the global stiffness matrix are symmetric.
- In linear theory, **K** is a positive definite matrix, which is a symmetric matrix with all positive eigenvalues.
- **K** is a sparse matrix, meaning that most of the elements are zero.

4.1.2.2 Boundary Conditions

When some of the dofs are specified it is called boundary conditions. These specifications can be divided into two categories: Support conditions (zero displacement and rotation) and displacement or rotation (non-zero) constraints.

When only dealing with support conditions, the implementation will look like this:

$$\begin{bmatrix} k_{11} & \dots & 0 & \dots & k_{1n} \\ \vdots & \ddots & 0 & & \vdots \\ 0 & 0 & 1 & 0 & 0 \\ \vdots & & 0 & \ddots & \vdots \\ k_{n1} & \dots & 0 & \dots & k_{nn} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_i \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ 0 \\ \vdots \\ R_n \end{bmatrix} \tag{4.4}$$

The entries of the rows and column, except the diagonal, of the stiffness matrix **K** corresponding to the boundary conditions are set to zero and a trivial equation is introduced.

When allow displacement constraints are allowed, for example $u_i = \delta$, the corresponding column of **K** is multiplied by δ and subtract it from the original load vector **R**.

$$\begin{bmatrix} k_{11} & \dots & 0 & \dots & k_{1n} \\ \vdots & \ddots & 0 & & \vdots \\ 0 & 0 & 1 & 0 & 0 \\ \vdots & & 0 & \ddots & \vdots \\ k_{n1} & \dots & 0 & \dots & k_{nn} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_i \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} R_1 - k_{1i} * \delta \\ \vdots \\ \delta \\ \vdots \\ R_n - k_{ni} * \delta \end{bmatrix} \tag{4.5}$$

4.1.3 Isoparametric Formulation

Most real-life structural problems do not only consist of straight edges. To analyze complex geometry, a way to transform curvilinear shapes into the simplest possible geometry is needed. The basic idea of introducing isoparametric interpolation is to interpolate element geometry from coordinates of the node.

4.1.3.1 Mapping

Because of the irregular geometry, Cartesian coordinates are no longer suitable. A different type of coordinates for the interpolation process is introduced: natural coordinates, (ξ, η, ζ). Natural coordinates are dimensionless and associated with the element shape rather than its real size and orientation in space. The process of transforming geometry from Cartesian to natural coordinates is called *mapping* and is illustrated in Figure 4.2.

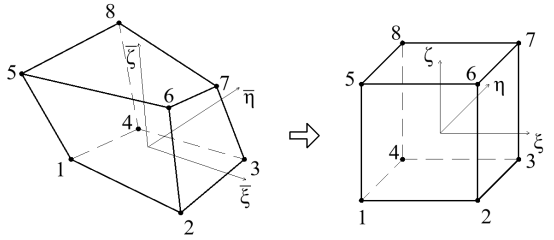


Figure 4.2: Actual and normalized geometries for a 3D element.

The element analysis still needs to be integrated into the physical world. Therefore, a relationship between Cartesian and natural coordinates must be established. This relationship is given by the Jacobian matrix for each element, $\mathbf{J}^e$, and is shown in Equation 4.6.

$$\begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{bmatrix} = \mathbf{J}^e \frac{\partial \mathbf{N}_i}{\partial u}, \tag{4.6}$$

$\mathbf{J}^e$'s determinant, the Jacobian scale factor, is used to scale the distances measured in two different coordinate systems.

4.1.3.2 Numerical Integration

Introducing the isoparametric formulation and mapping also introduces more complex integrals. In most cases, the exact analytical computation of the element integrals is not possible and numerical integration is needed. In this thesis, the Gauss quadrature will be used, which is also the most popular numerical integration method used in FEM.

The Gauss quadrature uses weights and Gauss points to express an integral as a sum of products of the function values in a number of known points and prescribed weights. Gauss quadrature integrates a polynomial function of degree $2g - 1$ with g Gauss points exact.

4.2 3D Elements

As discussed in Section 2.2, the increase of computational power creates the basis of why the focus is aimed towards finite element analysis of solids. Instead of simplifying to lower dimensions, the solid is represented as 3D elements. Beams, trusses and shells can be considered as special cases of a 3D solid. 3D elements require field variables for displacement in three different directions, u , v and w as shown in Figure 4.3.

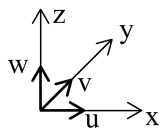


Figure 4.3: Field variables and coordinate system in 3D.

4.2.1 Common Solid Elements

There are two types of elements that are the most common for using FEM with solids: tetrahedral and hexahedral elements, both shown in Figure 4.4.

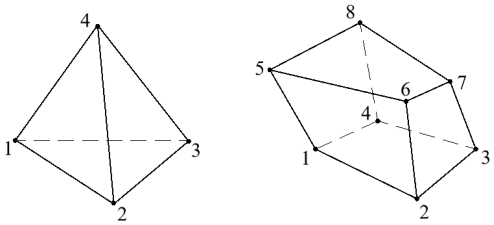


Figure 4.4: A tetrahedron (left) and a hexahedron (right).

The simplest tetrahedral element has four nodes, one in each corner. There are several other tetrahedral elements with different number of nodes and node locations. Tetrahedral elements contribute to simple computations and are easier to fit into complex geometry.

A hexahedron is any polyhedron with six faces. In comparison to the tetrahedral, the simplest hexahedral element is the trilinear hexahedron element with eight nodes. One advantage of using hexahedral elements is that it is more economical considering computation than tetrahedral elements. One hexahedron corresponds to six tetrahedral elements and less hexahedral elements are needed in a mesh to get the same accuracy.

Only trilinear hexahedron elements will be discussed further throughout this thesis.

4.2.2 FEM using Trilinear Hexahedron Element

A trilinear hexahedron element, or Hex8 hereafter, has eight nodes and 24 dofs - three dofs for each node (translation in x-, y- and z-direction). Therefore, the Hex8's displacement vector, **u**, and force vector, **R** is

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{24} \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_{24} \end{bmatrix}. \tag{4.7}$$

Because Hex8 only has corner nodes, the geometry and displacement between the nodes are interpolated between the nodes. A point in the element is interpolated as

$$\mathbf{u} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \sum_{n=1}^{n} N_i \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{bmatrix} = \mathbf{N}\mathbf{v}, \tag{4.8}$$

with the shape functions

$$\mathbf{N} = [\mathbf{N}_1 \ \mathbf{N}_2 \ \dots \ \mathbf{N}_8], \qquad \mathbf{N}_i = \begin{bmatrix} \mathbf{N}_i & 0 & 0 \\ 0 & \mathbf{N}_i & 0 \\ 0 & 0 & \mathbf{N}_i \end{bmatrix}. \tag{4.9}$$

Formulating Hex8 as an isoparametric element with arbitrary shape, it is necessary to write the shape functions in natural coordinates as in Equation 4.10. The shape functions for node 5 is shown in Figure 4.5.

$$N_i(\xi, \eta, \zeta) = \frac{1}{8}(1 + \xi_i\xi)(1 + \eta_i\eta)(1 + \zeta_i\zeta) \tag{4.10}$$

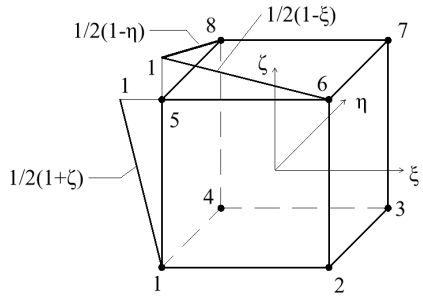


Figure 4.5: Hex8 with shape function for node 5.

The strain-displacement relationship can be expressed using Equation 4.8, where **B** is the strain matrix in Equation 4.12.

$$\boldsymbol{\epsilon} = \Delta\mathbf{u} = \Delta\mathbf{N}\mathbf{v} = \mathbf{B}\mathbf{v} \rightarrow \mathbf{B} = \Delta\mathbf{N}, \tag{4.11}$$

$$\mathbf{B} = [\mathbf{B}_1\ \mathbf{B}_2\ \dots\ \mathbf{B}_8], \qquad \mathbf{B}_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial z} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x} \end{bmatrix}. \tag{4.12}$$

The strain matrix, **B**, is a function of the coordinates within the elements. It may be expressed in natural coordinates following the relationship shown in Equation 4.6. The elasticity matrix, **C**, for linear elastic isotropic material in three dimensions is given:

$$\mathbf{C} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{4.13}$$

.

4.2.3 Stiffness Matrix

The element stiffness matrix for a solid can be computed by following PVD:

$$\mathbf{K}^e = \iiint_V \mathbf{B^T C B} dV = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \mathbf{B^T C B} |\mathbf{J^e}| d\xi d\eta d\zeta. \tag{4.14}$$

The strain matrix is evaluated in the natural coordinates ξ, η and ζ, and Equation 4.14 may therefore be difficult to solve analytically. For Hex8 using linear shape function, $\mathbf{K}^e$ can be integrated fully using Gauss quadrature and eight Gauss points (ξ_i, η_i, ζ_i) and weights (w_i, w_j, w_k). The Gauss quadrature in three directions can be defined as

$$I = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta,\zeta) d\xi d\eta d\zeta = \sum_{i=1}^{n}\sum_{i=1}^{n}\sum_{i=1}^{n} w_i, w_j, w_k f(\xi_i, \eta_i, \zeta_i). \tag{4.15}$$

The assembly of **K** for solids follows the methodology described in Subsection 4.1.2.1.

4.2.4 Approximation of Strain and Stress

For a three dimensional solid, the stress and strain will exist in three directions; x, y and z, shown in Figure 4.6.

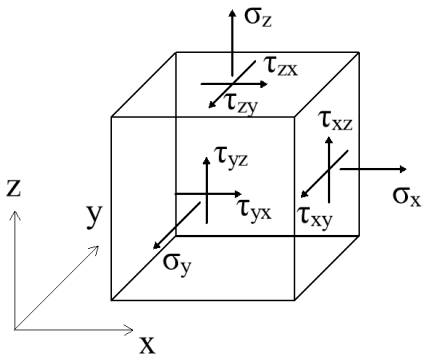


Figure 4.6: 3D stress state.

Strain

The strain vector is given as

$$\boldsymbol{\epsilon} = [\epsilon_x \ \epsilon_y \ \epsilon_z \ \gamma_{xy} \ \gamma_{yz} \ \gamma_{zx}]^T. \tag{4.16}$$

The relationship between the displacement and strain is shown in Equation 4.11. The strain matrix, **B**, from Equation 4.12 is evaluated in each Gauss point, meaning that the strain vector is calculated in the Gauss points of the element.

Stress

The linear relationship between strain, $\boldsymbol{\epsilon}$, and stress, $\boldsymbol{\sigma}$, using the elasticity matrix **C** from Equation 4.13 is

$$\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\epsilon}, \tag{4.17}$$

where the stress, $\boldsymbol{\sigma}$, is given as

$$\boldsymbol{\sigma} = [\sigma_x \ \sigma_y \ \sigma_z \ \tau_{xy} \ \tau_{yz} \ \tau_{zx}]^T. \tag{4.18}$$

To find the stress in the nodes of the element, not in the Gauss points, an extrapolation of the stress is required. The extrapolation is done using the same shape functions as used for calculating the displacements. A second set of dimensionless variables, r, s and t, which is proportional to the natural coordinates ξ, η and ζ, are introduced in Equation 4.19. All variables are shown in Figure 4.7.

$$r = \sqrt{3}\xi, \qquad s = \sqrt{3}\eta, \qquad t = \sqrt{3}\zeta. \tag{4.19}$$

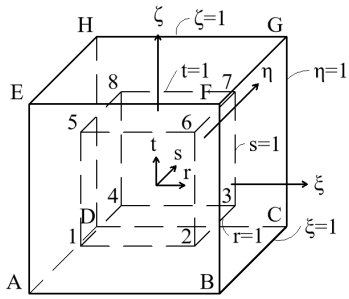


Figure 4.7: Reference coordinate system used in extrapolation of stresses from Gauss points.

The stresses in a node can be calculated by summing the product of the stress in each Gauss point multiplied with the shape function:

$$\sigma_a = \sum_{\alpha=1}^{8} N_\alpha \sigma_\alpha, \tag{4.20}$$

where σ_a is any of the stresses shown in Equation 4.18, σ_α is the corresponding stress component in Gauss point i and N_α is the shape function given by the new variables

$$N_\alpha = \frac{1}{8}(1 + r_\alpha r)(1 + s_\alpha s)(1 + t_\alpha t). \tag{4.21}$$

In a solid, most nodes are shared by multiple elements. Therefore, one node may have different stresses from each element creating a discontinuous stress field. A common way of smoothing the stress field is by applying nodal point averaging, simply averaging all stress contributions in each node. If σ_i describes the stress in an arbitrary node, n_i is the number of elements sharing node i, and σ_{ie} is the interpolated stress contribution in node

i from element e, the nodal averaging is defined as

$$\sigma_i = \frac{1}{n_i} \sum_{e=1}^{n_i} \sigma_{ie}. \tag{4.22}$$

The analysis in this thesis is applied to geometry consisting of metallic material. Therefore, the von Mises stress, σ_m, is calculated from the nodal averaged stresses. Von Mises accounts for all six stress components in each node and is a useful criterion to check if a given material will yield or fracture. The von Mises stress in 3D is given as

$$\sigma_m = \sqrt{\frac{1}{2}[(\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_z - \sigma_x)^2] + 3(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2)}. \tag{4.23}$$

Chapter 5

The Desired Workflow

In this chapter, we will set the scene for the development of our product prototype. To achieve this goal, we will need to use the parametric environment from Chapter 2 and the VR tools from Chapter 3. The theory behind FEM from Chapter 4 is the basis for all calculations performed by this product. How this is done is presented in Section 5.1. The section includes an updated flowchart and how our solution fits with the existing workflow. Section 5.2 is aimed towards the user's perspective and how the product is experienced by the user, who is assumed to have basic knowledge about structural analysis programs.

5.1 The Software Perspective

Section 3.5 shows the workflow between Oculus VR, Rhino and Grasshopper. This lays the foundation for further development and will be used to answer the central question of this thesis: "Could VR be a real improvement for structural design?". As stated in the thesis introduction, this will include the creation of a product prototype. Using existing software imposes limitations and the product will be adapted to these limitations. The product is intended to establish a link between three main concepts:

1. FEA of solids.
2. Parametric environment in Grasshopper.
3. Connection to Oculus VR.

The fact that existing plug-ins to Grasshopper don't handle FEA of solids motivates to create such a software package as a part of this thesis. The new package will be our product and is called *SolidsVR* hereafter. This covers the first concept listed above. SolidsVR will include a FEM solver based on the theory represented in Chapter 4. It will be flexible in the way that new components can be adjusted or added to the package anytime. As one of many Grasshopper plug-ins, it will also be possible to combine them with components from other plug-ins, for example, Mindesk.

SolidsVR fits with the existing workflow, and the updated flowchart in Figure 5.1 concludes the workflow for the software perspective in this thesis.

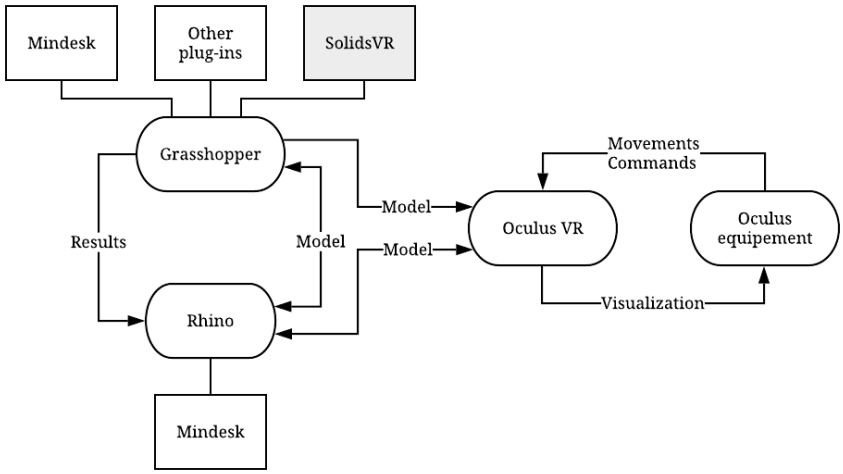


Figure 5.1: How programs will be connected from the software perspective.

The advantage of using Grasshopper is that the environment already facilitates for a parametric flow of data. With SolidsVR acting as a plug-in to Grasshopper, this package will be fully parametric and the second concept can be ticked off the list. A change of any parameters will instantly trigger a new analysis.

The last concept needed to be covered is the connection with VR. This is done through Mindesk described in Section 3.5. The features that Mindesk provides in VR focuses on creating and editing geometry. However, these features are not intended to fit with an analysis program, and therefore, the creation of SolidsVR will have to be adapted to Mindesk's existing features to test the parametric FEM program inside the VR environment.

5.2 The User's Perspective

Using virtual reality is new to most people and it is important not to demand much VR experience from the user. Programs using advanced technology is often limited by the user's abilities. Aiming to keep the user interface as simple as possible, the goal is to avoid the user switching between software programs and putting on and off the VR equipment. The user should provide the input data through the VR equipment, and read the output data from the same equipment. In other words, VR will not just be a place to observe and interpret the output on the display, but also a platform to provide input. This is the same two-way communication as discussed in Section 3.3.

Another motivating factor to keep the user interface as simple as possible is to maintain the usefulness of SolidsVR being parametric. Imagine the rather cumbersome action if the user has to take off the VR equipment to change an input parameter in Grasshopper, and put the equipment back on to see the new results.

With the aim to keep the VR equipment on, the results should be displayed in near real-time to avoid the user getting impatient and lose focus. Near real-time is referring to the time delay between the request for a new FEA, for example, after adjusting a parameter, and the display of the updated results. According to Microsoft, the average human attention span was 8 seconds in 2013 [38] and this gives an idea of how fast the FEA of a solid should be. The time performance of the analysis in SolidsVR will be an important topic and is discussed in Chapter 7.

SolidsVR does not only consist of the FEM solver but also components for pre- and post-processing. It is the pre-processing components that transforms the input data from the user into a suitable form for the FEM solver. The minimum input data required for SolidsVR to handle can be listed as follows:

- An adjustable solid geometry, including material properties.
- Meshing details.
- Loads.
- Boundary conditions.

After the calculations in the FEM solver, the output is transformed by the post-processing components in SolidsVR to a format that is understandable to the user. Again, the necessary output data should be made available to the user:

- Deformation of the geometry.
- Stress distribution in all directions, including the von Mises stress.
- Key values, such as maximum displacement and stress.

More features may be added to the product prototype as it is developed. However, if the input and output requirements above are fulfilled, it will be possible to gain a sufficient amount of insight into the VR user experience to answer the central question of this thesis.

Figure 5.2 illustrates the workflow from the user's perspective in a flowchart. Underlying the VR equipment which the user interacts with, are the software doing the actual calculations and transferring geometry and results between the programs.

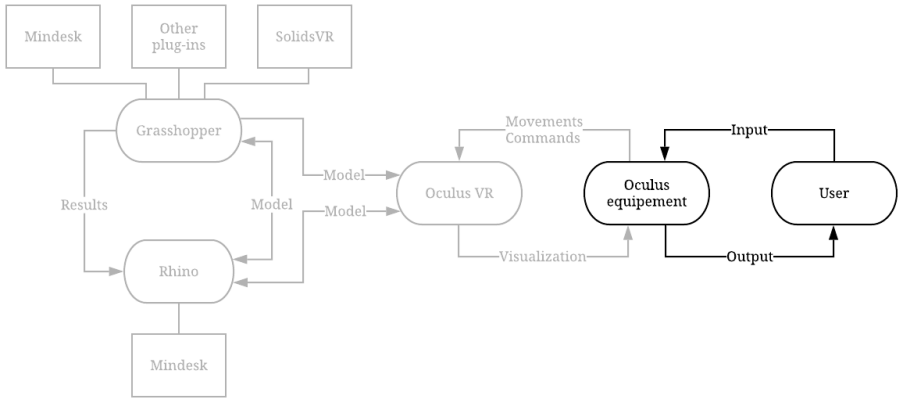


Figure 5.2: How programs will be connected from the user perspective. The grey circles represent what the user does not interact with.

Chapter 6

Study Cases and Results

To start this chapter, our working process is described in Section 6.1. This is essential to understand some of the choices made during the development of SolidsVR. Some clarifications are provided in Section 6.2 to avoid misunderstandings when reading the code. Section 6.3 encloses the first study case. Case 1 forms the basics of SolidsVR and its components are therefore explained in detail. The accuracy of the analysis results are presented. The section ends with a discussion of how Case 1 serves as a prototype of the desired product, as well as improvements regarding the VR user interface, features to be added and other adjustments. Applying these improvements to Case 1 forms the second study case. Section 6.4 about Case 2 has the same layout as the previous case. This way of working resulted in two more study cases, Case 3 and Case 4, represented in Section 6.5 and 6.6, respectively. Lastly, Section 6.7 provides a detailed description of how to install and run SolidsVR on your own computer.

6.1 The Working Process

Regarding the working process, inspiration is taken from what is formally called *agile software development methods* [39]. Agile is the ability to create and respond to change, and the key concepts adopted from this set of framework is an iterative and incremental development.

The idea of iterative and incremental development is to develop a product through repeated

steps and in smaller portions at a time [40]. The first step will result in a base version of the product. The advantage of this method is that the team members will learn from the development of earlier steps and product versions. Learning comes from both the development and testing of the product. In each iteration, improvements are made, and new features are added to the product. One iteration, which usually does not lasts more than a couple of weeks, is illustrated in Figure 6.1.

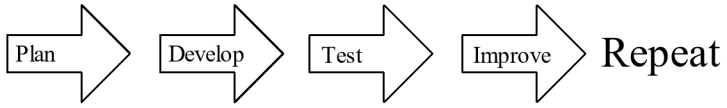


Figure 6.1: This thesis' version of an agile software development cycle.

In this thesis, the iterative steps have resulted in one study case each. Every case result is a usable prototype of SolidsVR. After the last study case, SolidsVR had evolved into a plug-in consisting of the 24 components shown in Figure 6.2. The components are programmed to cover the functionalities from Chapter 5, and the resulting code is attached in Appendix A. This is what is later referred to as "the code". There is also a video in Appendix B showing the VR user interface of SolidsVR.

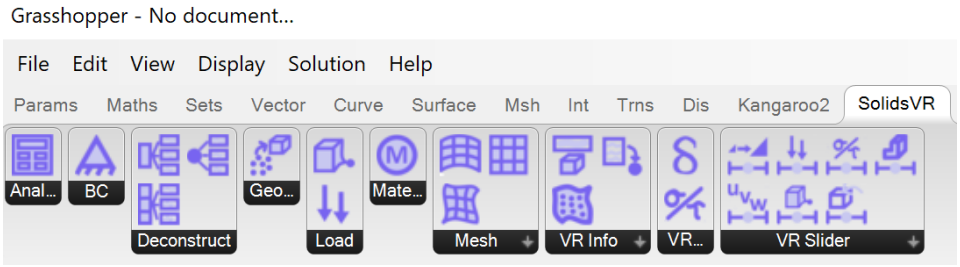


Figure 6.2: The resulting plug-in to Grasshopper. Every icon is a self-made component.

Each component is created based on the chosen features for analyzing and visualization in VR. One component can be used multiple times and is not limited to one model. If a feature is considered valuable, a component can be created and added to the list of components in SolidsVR.

6.2 Efforts for Robust Code

There are some lines in the code that might appear to be unnecessary. For clarification reasons, those parts of the code are explained in this section.

To maintain a generic code, it is essential that all types of geometry are processed similarly. Different types of geometry are not always consistent in their representation in Grasshopper. Therefore, all corners, edges, and surfaces are sorted using some customized control points and kept in this order throughout the calculations in SolidsVR. The control points are inputs where the order is determined, based on the mesh directions. In this thesis, the control points are placed so that u- and v-direction are in the plane of the cross section and w-direction is along the length of the geometry. The control points with numbering are shown as an example in Figure 6.3.

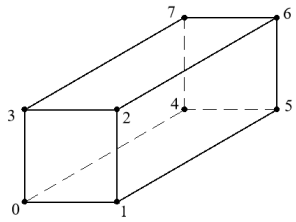


Figure 6.3: How the customized control points are placed.

The sorting is dependent on the exact values of each point. For complex geometry, the Rhino namespace may return vertices or points on the surface with small decimal errors in the coordinates in C#. Therefore, all input points or points generated by the input geometry are rounded off to one decimal to make sure that equal points will match when compared. As the input is given in mm, the rounding may only lead to minor changes. Note that for the calculations, the coordinate values have 8 decimals. An example of how rounding is implemented in C# is shown in Listing 6.1

```
1 for (int i = 0; i < vertices.Count; i++)
2 {
3     vertices[i] = new Point3d(Math.Round(vertices[i].X, 1),
          Math.Round(vertices[i].Y, 1), Math.Round(vertices[i].Z, 1));
4 }
```

Listing 6.1: Example of code rounding vertices coordinates.

6.3 Case 1: The Hexahedron

The goal of the first iteration, or study case, is to create the first prototype of SolidsVR as it is described in Chapter 5. For Case 1, a simple input geometry is chosen: a hexahedron. There are several reasons why:

- Six flat surfaces is a relatively simple geometry to handle. It can be adapted into a twisted cube.
- It can easily be meshed into smaller Hex8 elements, which are the chosen mesh elements for this thesis due to its simplicity and advantages presented in Subsection 4.2.1.
- A hexahedron may be interpreted as a beam. For example, with boundary conditions on one end surface and one of the lengths longer than the other two, a cantilever is created.
- It is simple to model a similar geometry in Abaqus, compare the FEA results and discuss the accuracy of SolidsVR.

6.3.1 FEM with Grasshopper Components

SolidsVR includes both new classes and new components. Object-oriented programming is associated with the concept of classes and self-made classes to represent nodes, elements, material and other objects that do not exist in the Rhino namespace, had to be made. The classes created are:

Node includes the coordinates (x, y, z) of the node, the global node number, which elements it belongs to and whether it is a corner, edge, or center node. It also contains the displacement, strain and stress in all directions.

Element store a list of **Node**-objects, the element number, the local stiffness matrix and the strain matrices in each Gauss point.

MeshGeometry includes information about the nodes and the elements in the mesh and how they are connected.

BrepGeometry was created to substitute the existing Rhino Brep-class, where a brep is representing the geometry. This was done to control the order of surfaces.

Material is a simple class containing Young's modulus, Poisson's ratio and the yield stress of the material.

The self-made components can be divided into three categories: pre-processing, the actual FEM solver and post-processing. The pre-processing components are the following:

MeshHex is the component that creates an instance of the **MeshGeometry**-class, given geometry and number of mesh divisions in three directions: u, v and w.

SurfaceLoad takes a surface and uniform load as inputs and distributes the load onto the nodes on that surface.

SurfaceBC applies pinned boundary conditions to all nodes on a given input surface.

CreateMaterial creates an instance of the **Material**-class with the chosen inputs.

The FEM solver is one single component with the same name:

FEMSolver is the component that takes the output from the previous components, executes FEA and calculates the displacements, strains and stresses and provides it as output.

The components mentioned are all a part of a Grasshopper script used to analyze an input hexahedron. An example Grasshopper script is illustrated in Figure 6.4.

More details on the logic of these components, except **CreateMaterial** due to its simplicity, and how they are implemented in C# is found in Section 6.3.1.1 through 6.3.1.4. The post-processing components are excluded for now as these components connect the output from **FEMSolver** to VR and will be introduced later when explaining the VR interface.

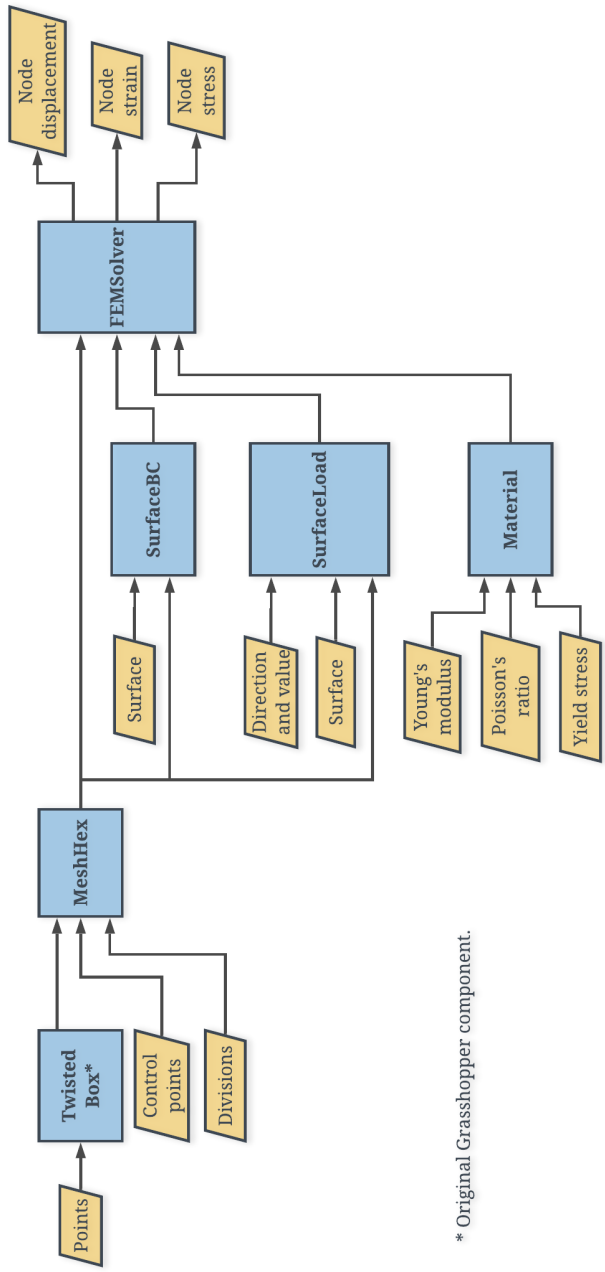


Figure 6.4: Workflow of a Grasshopper script analyzing a hexahedron with SolidsVR components, without the post-processing components and the VR connection.

6.3.1.1 Meshing with MeshHex

The input to **MeshHex** is a hexahedron, and after meshing, the information about the mesh is stored in an output **MeshGeometry**-object. **MeshGeometry** contains information about all elements as a list of **Element**-objects with information about the nodes inside the element. **MeshGeometry** also has a global coordinate list and connectivity for each element relating local and global node numbering. **MeshHex** takes the original geometry and the divisions in u, v and w-direction as inputs and determines how to mesh it into smaller Hex8 elements. The process of creating the mesh is divided into two main stages:

1. Create a global node list following the order shown in Figure 6.5
2. Create elements from the global node list using the location of the nodes in the list.

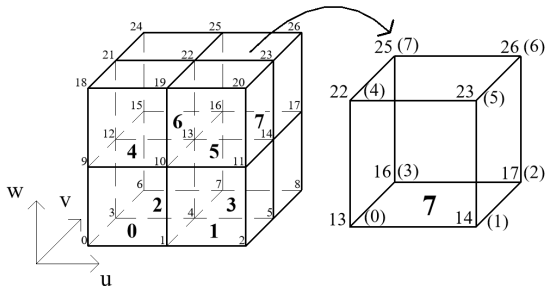


Figure 6.5: The relationship between element numbering and global and local node numbering. Numbers with parenthesis shows the local numbering inside the element. It is zero-based numbering.

Stage 1 uses three *for-loops* to go through w-direction, v-direction and then u-direction to create the nodes in the correct order. Having corner points, vectors can be created to iterate through the geometry. The steps of creating the global node list are shown in Figure 6.6, and also presented as code in Listing 6.2.

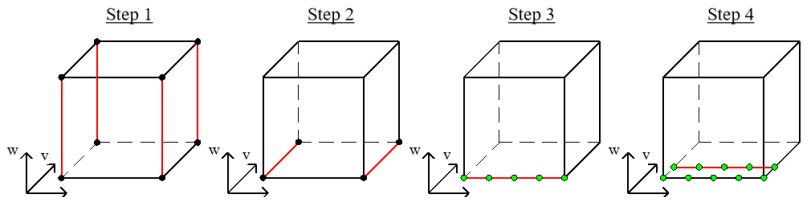


Figure 6.6: The first steps for generating the nodes from mesh division. The red lines shows the vector created and the green circles represents the added nodes.

```
for (int i = 0; i <= w; i++)//Creating points in w-direction
{
    Point3d p1_w = new Point3d(cornerNodes[0].X + lz1 * i * vec_z1.X,
        cornerNodes[0].Y + lz1 * vec_z1.Y * i, cornerNodes[0].Z + lz1 *
        vec_z1.Z * i);
    Point3d p2_w = new Point3d(cornerNodes[1].X + lz2 * i * vec_z2.X,
        cornerNodes[1].Y + lz2 * vec_z2.Y * i, cornerNodes[1].Z + lz2 *
        vec_z2.Z * i);

    Point3d p3_w = new Point3d(cornerNodes[2].X + lz3 * i * vec_z3.X,
        cornerNodes[2].Y + lz3 * vec_z3.Y * i, cornerNodes[2].Z + lz3 *
        vec_z3.Z * i);
    Point3d p4_w = new Point3d(cornerNodes[3].X + lz4 * i * vec_z4.X,
        cornerNodes[3].Y + lz4 * vec_z4.Y * i, cornerNodes[3].Z + lz4 *
        vec_z4.Z * i);

    Vector3d vecV1 = (p4_w - p1_w) / (p1_w.DistanceTo(p4_w));
    Vector3d vecV2 = (p3_w - p2_w) / (p2_w.DistanceTo(p3_w));

    Double length_v1 = p1_w.DistanceTo(p4_w) / v;
    Double length_v2 = p2_w.DistanceTo(p3_w) / v;

    for (int j = 0; j <= v; j++)//Creating points in v-direction
    {
        Point3d p1_v = new Point3d(p1_w.X + length_v1 * j * vecV1.X,
            p1_w.Y + length_v1 * j * vecV1.Y, p1_w.Z + length_v1 * j *
            vecV1.Z);
        Point3d p2_v = new Point3d(p2_w.X + length_v2 * j * vecV2.X,
            p2_w.Y + length_v2 * j * vecV2.Y, p2_w.Z + length_v2 * j *
            vecV2.Z);

        Vector3d vec_u1 = (p2_v - p1_v) / (p1_v.DistanceTo(p2_v));
        Double length_u1 = p1_v.DistanceTo(p2_v) / u;

        for (int k = 0; k <= u; k++)//Creating points in u-direction
        {
            Point3d p1_u = new Point3d(p1_v.X + length_u1 * k * vec_u1.X,
                p1_v.Y + length_u1 * k * vec_u1.Y, p1_v.Z + length_u1 * k
                * vec_u1.Z);
            globalPoints.Add(p1_u);

            Node node = new Node(p1_u, globalPoints.IndexOf(p1_u));
            nodes.Add(node); // Add to global node list
        }
    }
}
```

Listing 6.2: Stage 1: Finding points in the correct order for meshing in the component **MeshHex**.

Stage 2 of the meshing exploits the order of the nodes to create elements. Considering the order of elements shown in Figure 6.5, a logic to find the vertices of each element can be implemented. The logic is based on the pattern of the nodes using the information from u, v and w-divisions. Using a counter that always has the same value as the index of the first node in the element, the whole element can be gathered. The counter increases based on the index of the first node in each element and jumps to the next start node when it finds nodes that cannot be the first node in an element. The creation of each element using this logic and how each element is stored is shown in Listing 6.3.

```
//Putting together 8 nodes to craete hex
List<Node> hexNodes = new List<Node>()
{
    nodes[counter],
    nodes[(u + 1) + (counter)],
    nodes[(u + 1) + (counter + 1)],
    nodes[counter + 1],
    nodes[(u + 1) * (v + 1) + counter],
    nodes[(u + 1) * (v + 1) + (u + 1) + (counter)],
    nodes[(u + 1) * (v + 1) + (u + 1) + (counter + 1)],
    nodes[(u + 1) * (v + 1) + (counter + 1)],
};

//Showing the connectivity between local and global nodes
List<int> connectivity = new List<int>()
{
    counter,
    (u + 1) + (counter),
    (u + 1) + (counter + 1),
    counter + 1,
    (u + 1) * (v + 1) + counter,
    (u + 1) * (v + 1) + (u + 1) + (counter),
    (u + 1) * (v + 1) + (u + 1) + (counter + 1),
    (u + 1) * (v + 1) + (counter + 1),
};

//Creating a new element-object with eight Nodes, element number and
    connectivity for the element.
Element element = new Element(hexNodes, i, connectivity);
elements.Add(element);
```

Listing 6.3: Stage 2: Creating elements in the component **MeshHex**.

The code in **MeshHex** is represented as pseudo code using a flowchart in Figure 6.7.

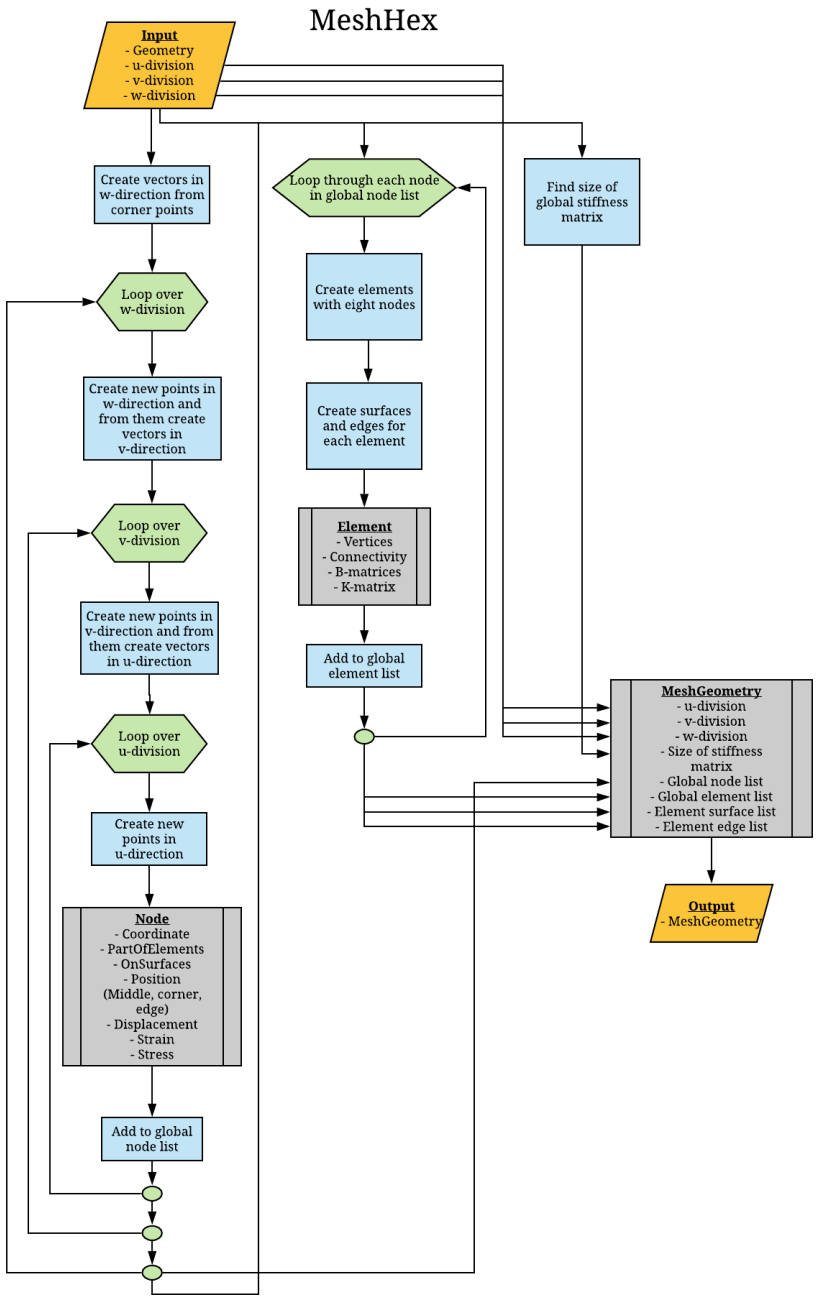


Figure 6.7: Flowchart showing the pseudo code of the meshing in **MeshHex**.

6.3.1.2 Pre-Processing Load Input with SurfaceLoad

The input to **SurfaceLoad** is a vector representing the direction and magnitude of a uniform load, a surface number and the mesh created in **MeshHex**. The unit for the load input is $\frac{N}{mm^2}$ and the load is converted to point loads and lumped onto nodes on the input surface.

As illustrated on a cube's surface in Figure 6.8, the magnitude of the nodal point load is based on the area covered by the particular node. The load in a central node is twice as large as the edge node's, and four times larger than the corner nodes. Since the surface is meshed into $u * v$ parts, this is a good estimate if the cube is squared and all the parts have equal areas. If the geometry is twisted or curved, the estimate will be slightly worse, but still sufficient for this study case.

The output of this component is a text string with the node coordinates and the decomposed force on the format "$x, y, z; F_x, F_y, F_z$". The force magnitudes are given in global x, y and z-direction. This output text string is passed along to the FEM solver.

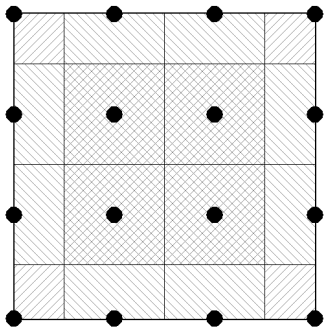


Figure 6.8: Illustration of the load lumping to nodes (black dots) on a surface meshed into $3 * 3$ parts.

The pseudo code in Figure 6.9 explains how the input parameters are lumped onto the nodes on the correct surface.

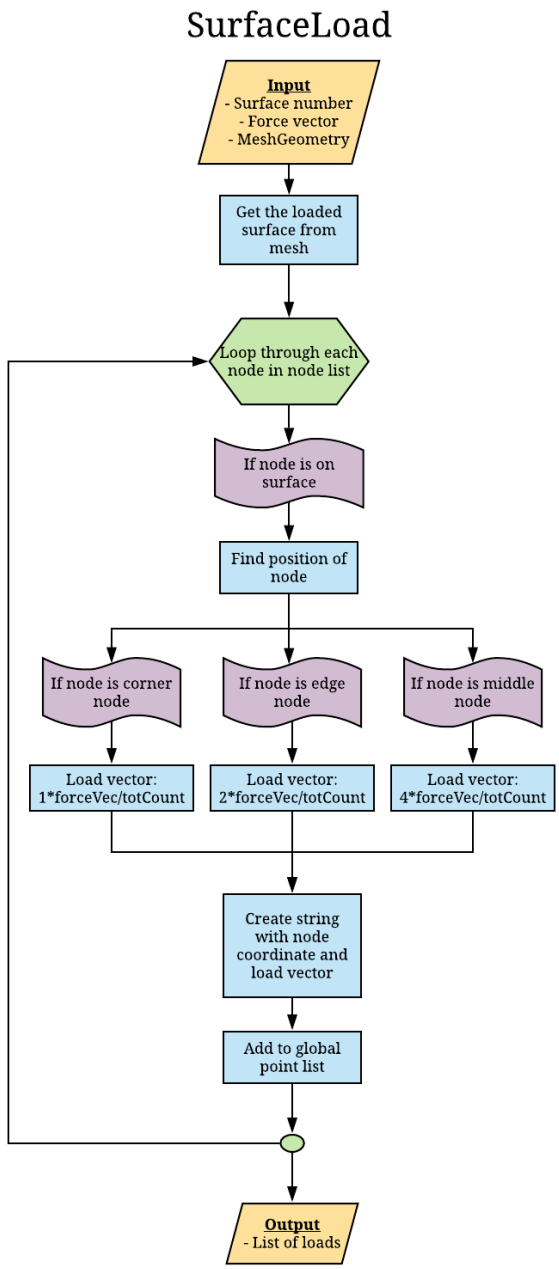


Figure 6.9: Flowchart showing the pseudo code of lumping a uniform load onto nodes on a surface in **SurfaceLoad**.

6.3.1.3 Pre-Processing Boundary Conditions Input with SurfaceBC

Similar to how a uniform load is assigned to surface nodes, the boundary conditions are assigned to nodes in **SurfaceBC**. For now, SolidsVR assumes zero translation in all directions to the nodes placed on the input surface. Therefore, the input is only a surface number and the mesh created in **MeshHex**. The output of this component is a text string with the node coordinates on the input surface and the restraints in each direction on the format: "$x, y, z; 0, 0, 0$". The reason it is explicitly stated zero displacement in these support nodes is to make it easier to expand to other displacement values later.

The pseudo code for pre-processing boundary conditions is illustrated in Figure 6.10. If boundary conditions should be applied to more than one surface, the user is free to include additional copies of **SurfaceBC** in the Grasshopper script.

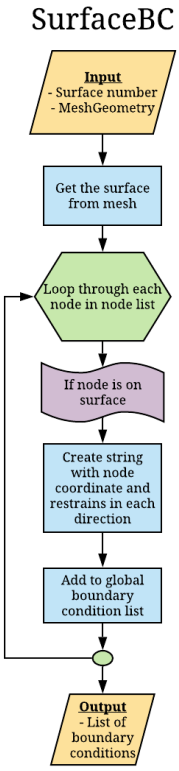


Figure 6.10: Flowchart showing the pseudo code of applying of boundary conditions in **SurfaceBC**.

6.3.1.4 Analyzing with FEMSolver

The analysis based on the theory chapter is performed in the **FEMSolver** component. The inputs for this component are prepared in the pre-processing components. **FEMSolver** calculates the displacement in mm, dimensionless strain and stress in MPa in each node and this is also the output of the component.

Element Stiffness Matrix and Assembly of Global Stiffness Matrix

The creation of the element stiffness matrices and the assembly of these matrices are done in the **FEMSolver** component. The element stiffness matrix and the contribution to the global stiffness matrix are based on the input mesh and the connectivity.

The element stiffness matrix for each element is created using the eight vertices in each element following the theory in Subsection 4.2.3. The calculation of the element stiffness matrix in SolidsVR uses linear shape function integrated in eight Gauss points located at $-\frac{1}{\sqrt{3}}$ and $\frac{1}{\sqrt{3}}$ in natural coordinates. Since the stiffness matrix for each element is calculated using strain matrices, the strain matrix for every node in the element are also added to the **Element**-class.

The assembly of the global stiffness matrix is also following the theory described in Subsection 4.1.2.1. To increase the efficiency of the assembly, the stiffness contributions from each element is directly added to the global stiffness matrix as shown in Listing 6.4.

```
1  for (int i = 0; i < connectivity.Count; i++)
2  {
3      for (int j = 0; j < connectivity.Count; j++)
4      {
5          //Inserting 3x3 stiffness matrix
6          for (int k = 0; k < 3; k++)
7          {
8              for (int e = 0; e < 3; e++)
9              {
10                 K_tot[3 * connectivity[i] + k, 3 * connectivity[j] + e] =
                       K_tot[3 * connectivity[i] + k, 3 * connectivity[j] +
                       e] + Math.Round(K_e[3 * i + k, 3 * j + e], 8);
11             }
12         }
13     }
14 }
```

Listing 6.4: Assembly of the global stiffness matrix in the component **FEMSolver**. The values are rounded off to eight decimals to ensure that decimals error do not lead to asymmetry.

Applying Boundary Conditions

The input string from **SurfaceBC** contains coordinates that indicate which entry of the global stiffness matrix the restraints should be applied to. Each point in the global point list is compared to these input points.

This is done by looping through each node in the boundary condition list, and if the coordinates correspond to a point in the global point list, the node is added to the list *bcNodes*, as in Listing 6.5. The corresponding entries to the nodes in *bcNodes* are set to zero in the global stiffness matrix following the theory described Subsection 4.1.2.2 and shown in the code.

```
for (int i = 0; i < bcNodes.Count; i++)
{
    for (int j = 0; j < K.ColumnCount; j++)
    {
        //Set all values in row to 0 except diagonal
        if (bcNodes[i] != j)
        {
            K[bcNodes[i], j] = 0;
        }
    }
    for (int j = 0; j < K.RowCount; j++)
    {
        //Set all values in column to 0 except diagonal
        if (bcNodes[i] != j)
        {
            K[j, bcNodes[i]] = 0;
        }
    }
}
```

Listing 6.5: Applying the boundary conditions to the stiffness matrix in the component **FEMSolver**.

Applying Loads

The next step is to create the force vector based on the input string from **SurfaceLoad**. Each point in the global point list is again compared to the input string, and if the coordinates match, the load of the node is assigned to the load vector. The load is placed in the correct entry of the force vector based on the global node number. The whole process of transferring the text input to loads in the force vector is shown in Listing 6.6.

```
foreach (string s in pointLoads)
{
    string coordinate = (s.Split(';'))[0];
    string iLoad = (s.Split(';'))[1];
    string[] coord = (coordinate.Split(','));
    string[] iLoads = (iLoad.Split(','));

    loadCoord.Add(Math.Round(double.Parse(coord[0]), 8));
    loadCoord.Add(Math.Round(double.Parse(coord[1]), 8));
    loadCoord.Add(Math.Round(double.Parse(coord[2]), 8));
    pointValues.Add(Math.Round(double.Parse(iLoads[0]), 8));
    pointValues.Add(Math.Round(double.Parse(iLoads[1]), 8));
    pointValues.Add(Math.Round(double.Parse(iLoads[2]), 8));
}
int index = 0;
foreach (Point3d p in points)
{
    for (int j = 0; j < loadCoord.Count / 3; j++)
    {
        if (loadCoord[3 * j] == Math.Round(p.X, 8) && loadCoord[3 * j +
            1] == Math.Round(p.Y, 8) && loadCoord[3 * j + 2] ==
            Math.Round(p.Z, 8))
        {
            R[index] = pointValues[3 * j];
            R[index + 1] = pointValues[3 * j + 1];
            R[index + 2] = pointValues[3 * j + 2];
        }
    }
    index += 3;
}
```

Listing 6.6: Assigning loads to each node in the component **FEMSolver**.

Calculating Deformation, Strain and Stress

Finally, with all the needed information, the displacements, strains and stresses in each node can be calculated. The process of finding the displacements is straight forward using the stiffness matrix and the force vector as described in Subsection 4.1.2. The displacements are obtained by inverting the stiffness matrix and multiplying it with the force vector. All nodes are updated with information about the displacements.

Strains in the Gauss points in each element are a result of the displacement in each node and the strain matrix. This is calculated by multiplying the strain matrix in the Gauss point with the displacement vector of the element. All the needed information is stored in the **Element**-class and the process of calculating the strains are shown in Listing 6.7.

```
Vector<double> u_e = Vector<double>.Build.Dense(24);
for (int i = 0; i < nodes_e.Count; i++)
{
    List<double> deformations = nodes_e[i].GetDeformation();
    u_e[i * 3] = deformations[0];
    u_e[i * 3 + 1] = deformations[1];
    u_e[i * 3 + 2] = deformations[2];
}

for (int j=0; j< B_e.Count; j++)
{
    Vector<double> nodeStrain = B_e[j].Multiply(u_e); // In Gauss points
    Vector<double> nodeStress = C_Matrix.Multiply(nodeStrain);
    elementStrain.Add(nodeStrain);
    elementStress.Add(nodeStress);
}
```

Listing 6.7: Calculation of strain in each node in the component **FEMSolver**

Stress is a product of the strain and the elasticity matrix and is therefore easily calculated in the Gauss points from the strains. The stresses are extrapolated to the node from the Gauss points following the theory described in Subsection 4.2.4 and shown in Listing 6.8.

```
Vector<double> nodeStress = Vector<double>.Build.Dense(6);

for (int i = 0; i < nodeStress.Count; i++)
{
    for (int j = 0; j < shapeF.Count; j++)
    {
        nodeStress[i] += gaussStress[j][i] * shapeF[j];
    }
}
```

Listing 6.8: Calculation of strain in each node in the component **FEMSolver**

Since most of the nodes belong to multiple elements, the stresses in each node are averaged from the contribution from the different elements. Then, the strains in the nodes are calculated from the extrapolated stresses. Finally, the von Mises stress is calculated following Equation 4.23 in the same theory chapter.

The code in **FEMSolver** is represented as pseudo code using a flowchart in Figure 6.11.

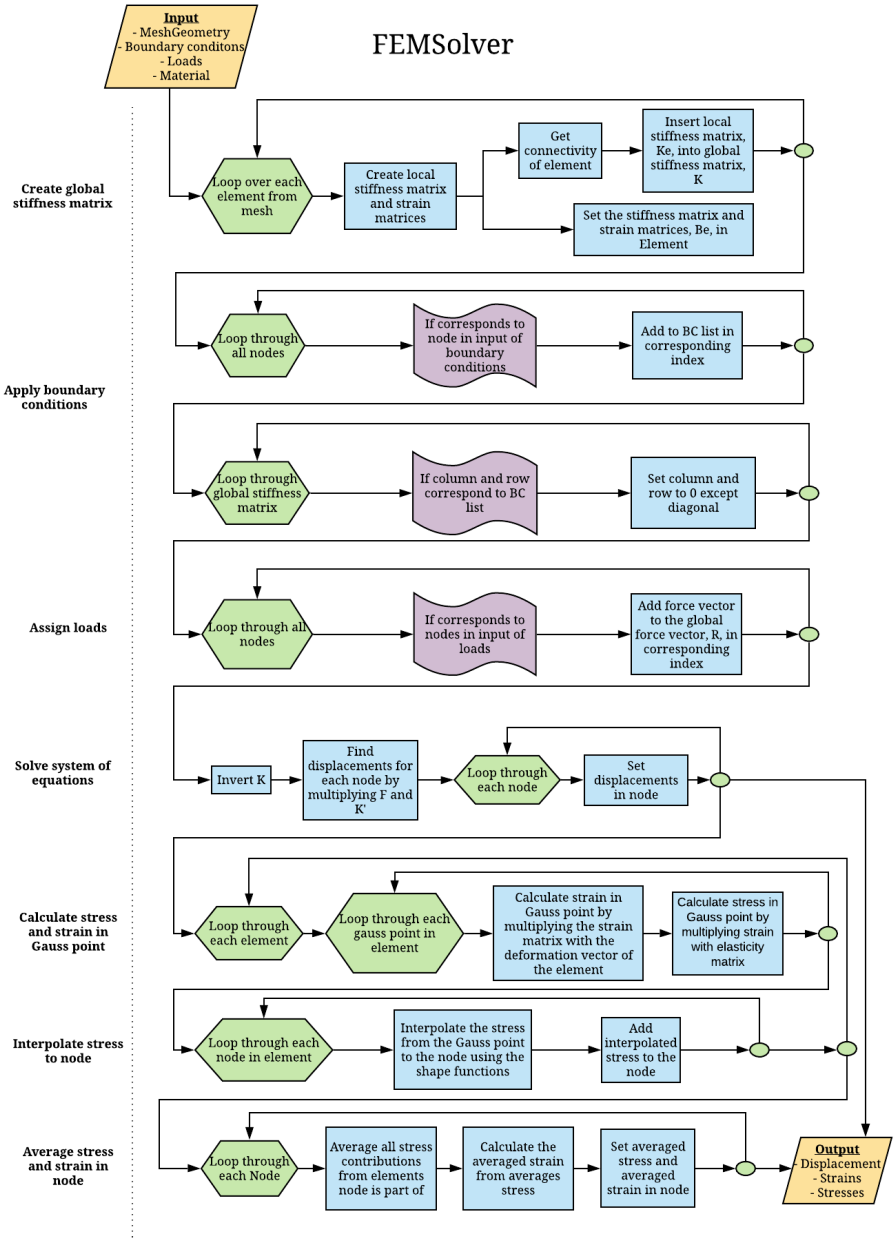


Figure 6.11: Flowchart showing the pseudo code of the main analysis component, **FEMSolver**.

6.3.2 The VR Interface

When developing the VR interface, the focus was on creatively utilize the properties of VR. One of the unique experiences in VR is that the user may walk and turn around to explore the virtual world. Today, many structural engineers might feel a need for additional computer screens at their workplace. In VR this is not a problem. To take advantage of this, the virtual world is divided into different areas as seen in Figure 6.12.

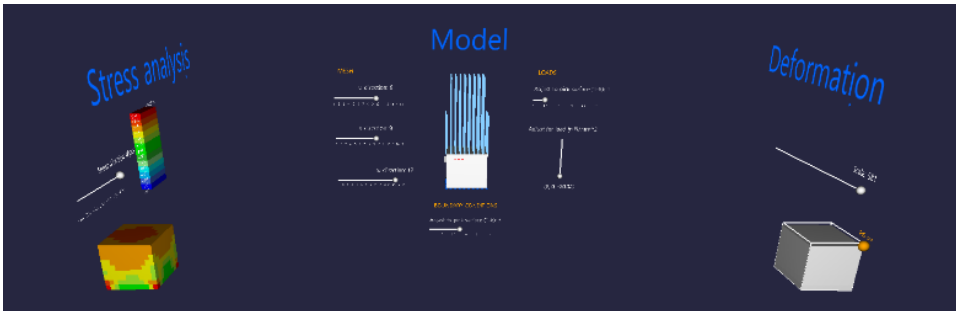


Figure 6.12: How the VR world looks like from the user's perspective.

The areas are referred to as the model area, the deformation area and the stress area. Each area will give the user some kind of information and the ability to interact and adjust what they are looking at. All visualizations are displayed through the Mindesk plug-in introduced in Section 3.5. Mindesk provides two additional Grasshopper components to transfer either text or geometry from Grasshopper or Rhino to VR. Figure 6.13 shows example input parameters to the Mindesk components, MDSK Text and MDSK Geometry.

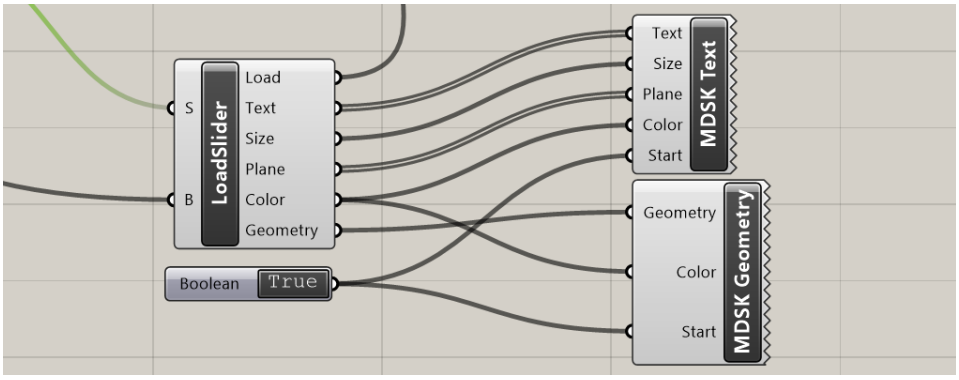


Figure 6.13: Mindesk's components for displaying text and geometry in VR.

The deformation area and the stress area are displayed after the results have been post-processed in two new self-made components in SolidsVR:

ViewDeformation displays the deformation results to the user in VR. The deformation area is located to the right of the model area, and the user will have to turn around to look at it. This area will show the outlines of the input geometry in addition to the deformed one, which the user may scale as desired. There is also an orange sphere indicating the magnitude and location of the maximum displacement in mm.

ViewStresses displays the stress results. The stress area is placed to the left of the model area. Colors are used to visualize the stress distribution in the direction chosen by the user. The stress direction can be chosen among σ_x, σ_y, σ_z, τ_{xy}, τ_{yz}, τ_{zx} and the von Mises stress and are displayed in MPa.

To allow the user to for example adjust load and boundary conditions in the model area or choose between stress directions in the stress area, inputs from the user are needed. Mindesk do transfer data *to* VR, but does not provide components transferring user input *from* VR back to Grasshopper. A classic number slider in Grasshopper is an input parameter that may vary within a range of numbers, and now this kind of number manipulation is needed in VR. This is the reason more self-made components, called *sliders*, are added to SolidsVR. They are created to fit its purpose, but the main idea is the same:

Slider components transfers straight lines in VR to input to other SolidsVR components. The straight line geometry is the input to the slider component, and the output is either a vector or a number, calculated based on the length and direction of the straight line. The length and direction of the lines may be adjusted by the user by clicking and dragging it with the VR controllers. This is done to keep SolidsVR parametric in VR. The sliders are also displayed in VR through Mindesk components.

More details on the logic of the three types of components above and how they are implemented in C# is found in Section 6.3.2.1 through 6.3.2.3.

The example Grasshopper script is updated and its workflow is illustrated in Figure 6.14, now with all components included.

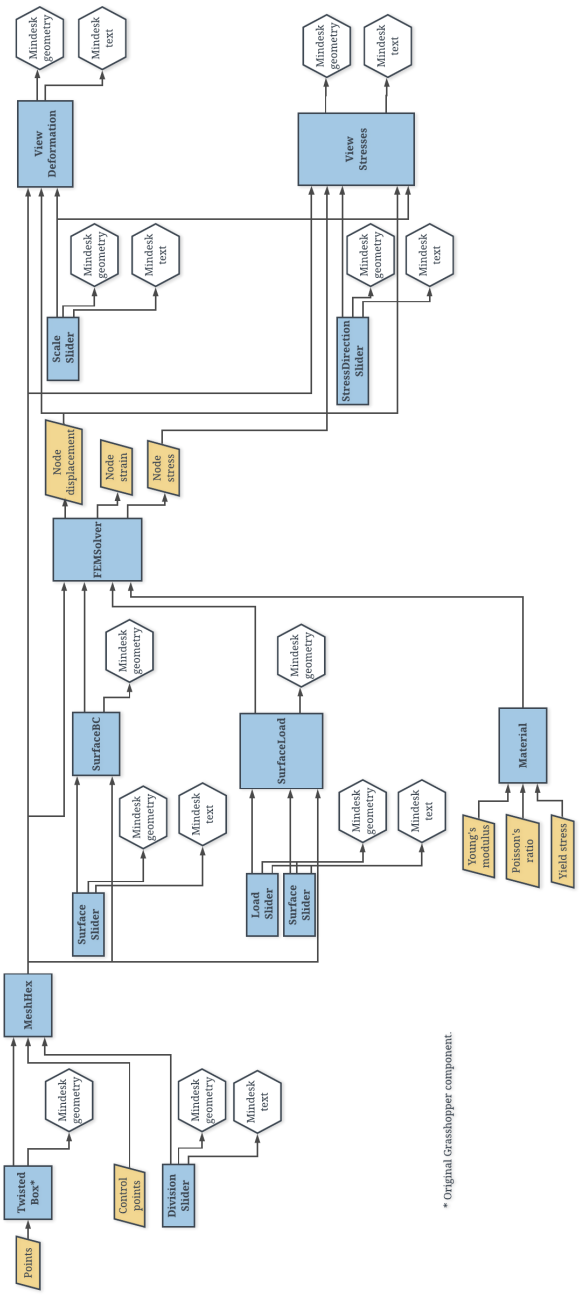


Figure 6.14: Workflow of the Grasshopper script analyzing a hexahedron and visualizing results in VR with SolidsVR components.

Before describing **ViewDeformation** and **ViewStresses**, the logic behind how to turn and place the geometry and text relative to the original model will be explained. To avoid a default size of text and geometry a number called *refLength* is introduced in Listing 6.9.

```
1 double sqrt3 = (double)1 / 3;
2 double refLength = Math.Pow(brp.GetVolume(), sqrt3);
```

Listing 6.9: Calculation of *refLength* in the **BrepGeometry**-class.

As *refLength* = $\sqrt[3]{V}$, where V is the volume of the geometry, *refLength* = L, for a perfect squared cube with sides of length L. For a different geometry it will not be exactly equal to any of the lengths, but about the same order of magnitude.

The stress area and the deformation area are rotated $90°$ and $270°$, respectively. The distance between the chosen center and the three different areas is set to be $6.5 * refLength$, as seen in Figure 6.15. These distances and angles may be adjusted if desired. Listing 6.10 shows how parts of the geometry are rotated based on the center and the angle.

```
1 Brep new_brep = Brep.CreateFromMesh(mesh, false);
2 Vector3d vecAxis = new Vector3d(0, 0, 1);
3 new_brep.Rotate(angle, vecAxis, center);
4 breps.Add(new_brep);
```

Listing 6.10: Rotating geometry to correct area.

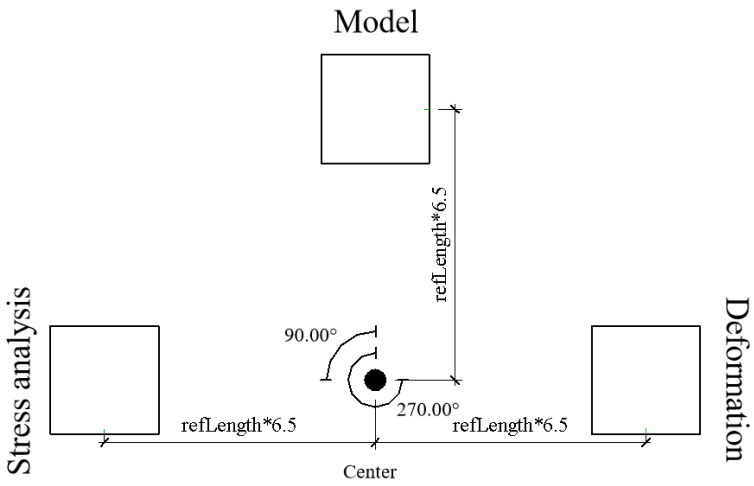


Figure 6.15: View of SolidsVR's model with placement of the different areas.

6.3.2.1 Post-Processing Deformation with ViewDeformation

The input to **ViewDeformation** is a **MeshGeometry**-object from **MeshHex**, all node displacements calculated in **FEMSolver**, and a scale factor. A scale input of 100 corresponds to a deformation of the structure at a scale of 100:1. For each element, a *for-loop* iterates through each node as seen in Listing 6.11. The displacement vector is calculated based on the input displacement, multiplied with the scale factor, and added to the old node coordinate. New elements are created and a new deformed geometry is the output displayed to the user with Mindesk's components.

```
for (int i = 0; i < vertices.Count; i++)
{
    Point3d p = vertices[i].GetCoord();
    Vector3d defVector = new Vector3d(vertices[i].GetDeformation()[0],
        vertices[i].GetDeformation()[1], vertices[i].GetDeformation()[2]);
    Point3d new_p = Point3d.Add(p, defVector * scale);
    mesh.Vertices.Add(new_p);
}
```

Listing 6.11: Creating new deformed elements in **ViewDeformation**.

The value and placement of maximum displacement are also calculated. A sphere with center at this maximum displacement point is added to the list of geometry displayed in VR. This is to show the user which point is critical regarding displacement. For the user to gain even more insight, the outline of the original geometry is also an output parameter to be displayed. The resulting deformation area in VR is shown in Figure 6.16a, and the pseudo code of **ViewDeformation** is shown to the left in Figure 6.17.

(a) The deformation area.

(b) The stress area.

Figure 6.16: The areas in the VR world for Case 1.

6.3.2.2 Post-Processing Stress with ViewStresses

The input to **ViewStresses** is the same as to **ViewDeformation**, but also the stresses calculated in **FEMSolder** and an integer indicating the chosen stress direction. For each element, the average element stress is calculated based on the stress in its belonging nodes in the input direction. To visualize the element stress, a color is assigned to it based on the average stress. The color range consists of 13 colors, and the difference between the minimum and maximum stress is divided by 13 to get the range size. For each element, a *for-loop* is iterating through the average stress and checks what range the stress is within and assigns the corresponding color to the element, as seen in Listing 6.12.

```
1  double[] averageValues = AverageValuesStress(elements, dir);
2  double max = averageValues.Max();
3  double min = averageValues.Min();
4  double range = (max - min) / 13;
5  //Creating header
6  if (dir == 0) rangeValues.Add("S,xx [MPa]");
7  else if (dir == 1) rangeValues.Add("S,yy [Mpa]");
8  else if (dir == 2) rangeValues.Add("S,zz [Mpa]");
9  else if (dir == 3) rangeValues.Add("S,xy [Mpa]");
10 else if (dir == 4) rangeValues.Add("S,yz [Mpa]");
11 else if (dir == 5) rangeValues.Add("S,zx [Mpa]");
12 else if (dir == 6) rangeValues.Add("von Mises [Mpa]");

14 for (int i = 0; i < breps.Count; i++)
15 {
16     if (averageValues[i] < min + range) color = Color.Blue;
17     else if (averageValues[i] < min + 2 * range) color = Color.RoyalBlue;
       .
28     else if (averageValues[i] < min + 12 * range) color = Color.OrangeRed;
29     else color = Color.Red;
30     brepColors.Add(color);
31 }
```

Listing 6.12: Coloring of the elements according to the average von Mises stress.

Colors alone may not give the user much insight into the stress distribution. Legends showing the colors and its corresponding values are displayed in the stress area as well. Again, all the outputs are inputs to Mindesk's components.

The geometry in the stress area is also deformed, and the calculations are exactly the same as in **ViewDeformation**. The resulting stress area in VR is shown in Figure 6.16b, and pseudo code of **ViewStresses** is shown to the right in Figure 6.17.

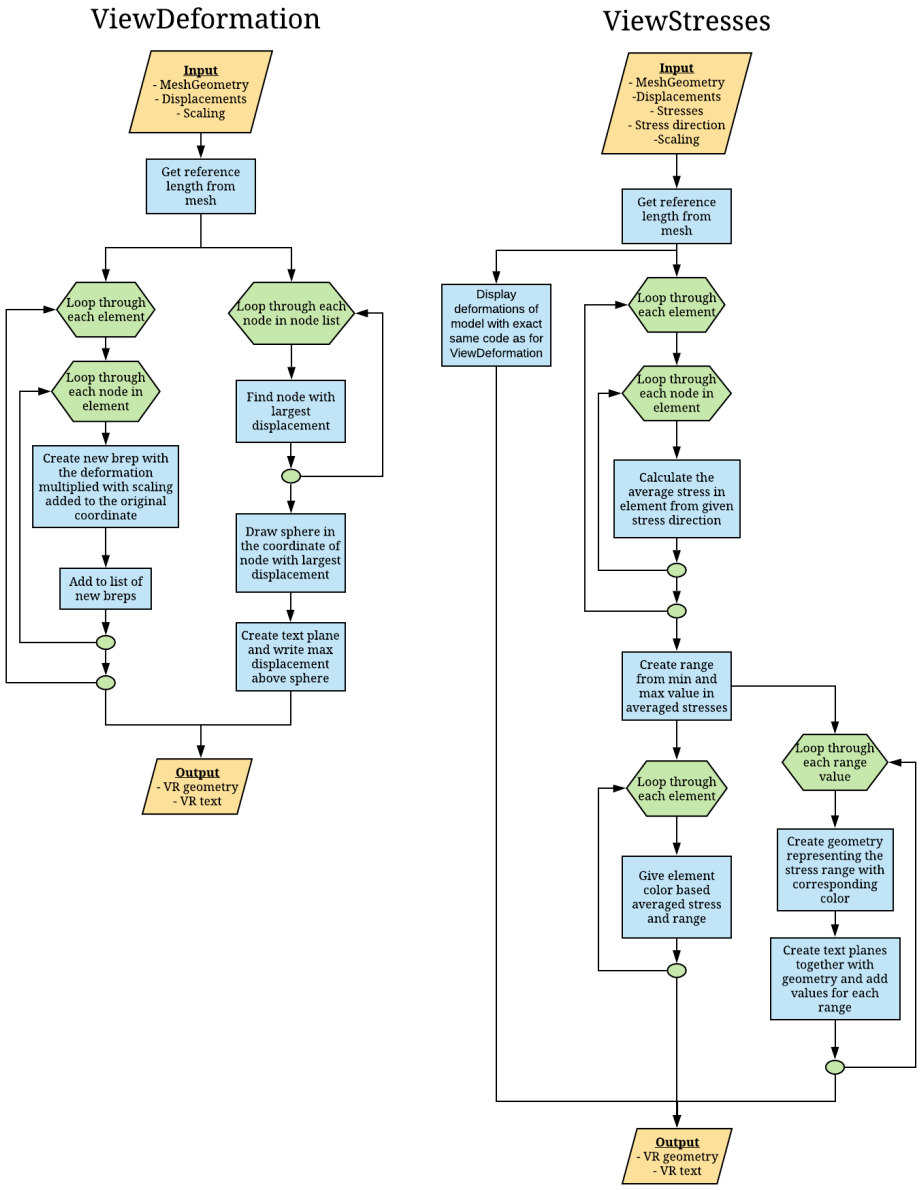


Figure 6.17: Flowchart showing the pseudo code of **ViewDeformation** (left) and **ViewStresses** (right).

6.3.2.3 Interacting in VR with Sliders

Sliders are created and displayed in VR – some in the model area as shown in Figure 6.18. As mentioned, the sliders allow the user to interact with the model in VR. In addition, to change the size and shape of the hexahedron with Mindesk's existing functionality, it is now possible to adjust the meshing details, load and boundary conditions. The self-made sliders are listed below.

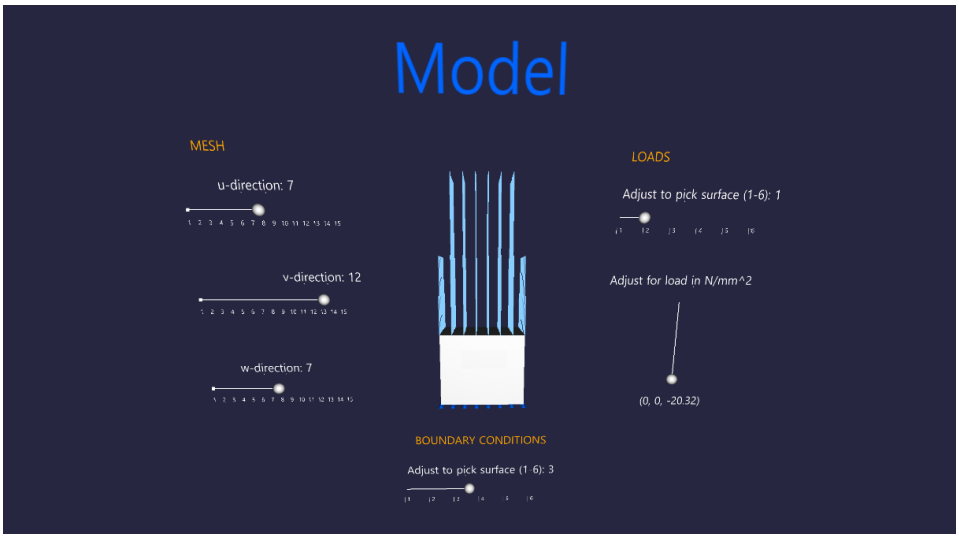


Figure 6.18: How the model area looks like from the user's perspective.

SurfaceSlider translates the length of a line in VR to an integer between 0 and 5, representing one of the geometry's surfaces. To make sure the slider's length is relative to the size of the geometry, the length is based on *refLength* calculated earlier. For this particular component, a slider with length equal to *refLength* will result in an output integer representing surface number 5. Sliders longer than this is limited to *max*, which in this case is 5. How this is implemented is shown in Listing 6.13.

```
1 double refLength = brp.GetRefLength();
2 double adjustment = 5 / refLength;
3 int surface = Convert.ToInt32(curve.GetLength() * adjustment);
4 surface = surface > max ? max : surface;
```

Listing 6.13: Using *refLength* in **SurfaceSlider**.

StressDirectionSlider works the same way, but with seven different stress directions (including von Mises stress) the maximum value is 6.

DivisionSlider takes three lines as inputs. They represent the mesh division in u, v and w-direction. The lengths are translated to an integer between 1 and 15. However, this integer range may be adjusted to the user's needs.

ScaleSlider translates the length of the line in VR to a number within a chosen number range. In this study case, it is set to 0-1000. The details behind the four sliders mentioned can be easier understood in the pseudo code to the left in Figure 6.19.

LoadSlider allows the user to adjust the value *and* direction of the load. It is now suitable to use the direction of the line as the direction of the load and the length of the line as the magnitude. The component translates the input curve to a vector based on the start point and end point of the curve, and the proper input vector for **SurfaceLoad** is created. Pseudo code for a vector slider is illustrated to the right in Figure 6.19.

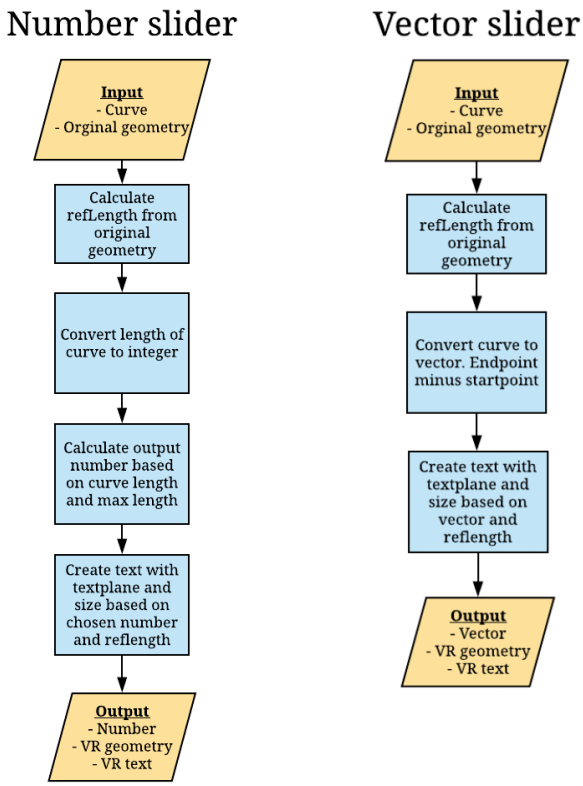


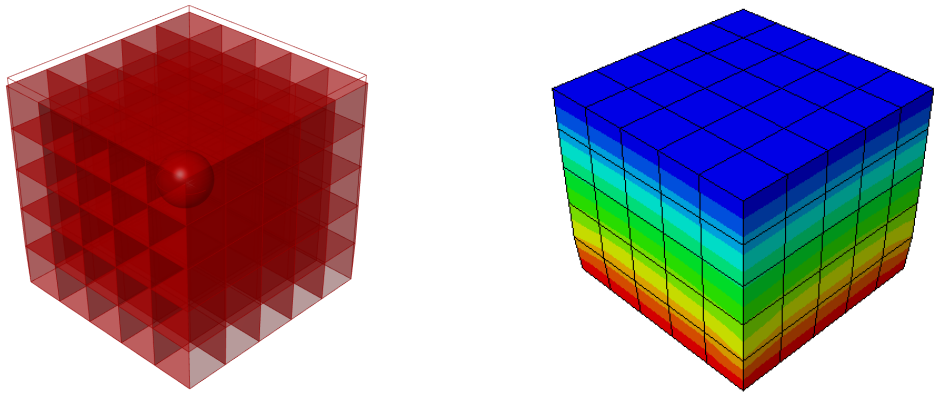
Figure 6.19: Flowchart showing the pseudo code for number sliders (left) and vector sliders (right).

6.3.3 Comparison of Results

6.3.3.1 Squared Cube

For the first comparison with Abaqus, a squared cube is used. It is modeled with a uniformly distributed load, F, on the top surface and boundary conditions on the bottom. Figure 6.20 illustrates the cube in both SolidsVR and Abaqus. The parameters are given below, where L is the side length, A is the area, E is the Young's modulus and ν the Poisson's ratio.

$$F = -10\frac{N}{mm^2} \quad L = 1000mm \quad A = 1000000mm^2 \quad E = 210000MPa \quad \nu = 0.3$$



(a) The comparison cube in SolidsVR.

(b) The comparison cube in Abaqus.

Figure 6.20: Deformed comparison cubes for Case 1 meshed into $5 * 5 * 5 = 125$ elements.

Deformation

The maximum vertical displacement, taking place in the upper corner node, of the cube calculated in SolidsVR is compared to Abaqus' and the analytic solution. This is done with an increasing number of elements to see how the results potentially converge. The analytic solution estimating displacement, ΔL_a, is found using the following equations:

$$\Delta L_a = \epsilon L \tag{6.1}$$

$$\epsilon = \frac{\sigma}{E} = \frac{F}{E} \tag{6.2}$$

$$\Delta L_a = \frac{FL}{E} = \frac{-10 * 1000}{210000} = -0.0476mm \tag{6.3}$$

The vertical displacement in the upper corner node in SolidsVR, Abaqus and the analytic solution are shown in Table 6.1 and plotted in Figure 6.21.

Mesh	**Elem. [#]**	**SolidsVR [mm]**	**Abaqus [mm]**	**Diff. [%]**	**Analy. [mm]**
1x1x1	1	-0.04333	-0.04603	5.87	
2x2x2	8	-0.04513	-0.04746	4.91	
3x3x3	27	-0.04581	-0.04597	0.35	
4x4x4	64	-0.04607	-0.04612	0.11	
5x5x5	125	-0.04621	-0.04630	0.19	-0.0476
6x6x6	216	-0.04630	-0.04637	0.15	
7x7x7	343	-0.04636	-0.04643	0.15	
8x8x8	512	-0.04640	-0.04646	0.13	
9x9x9	729	-0.04643	-0.04649	0.13	

Table 6.1: Comparison of maximum vertical displacement in corner node in Case 1 cube.

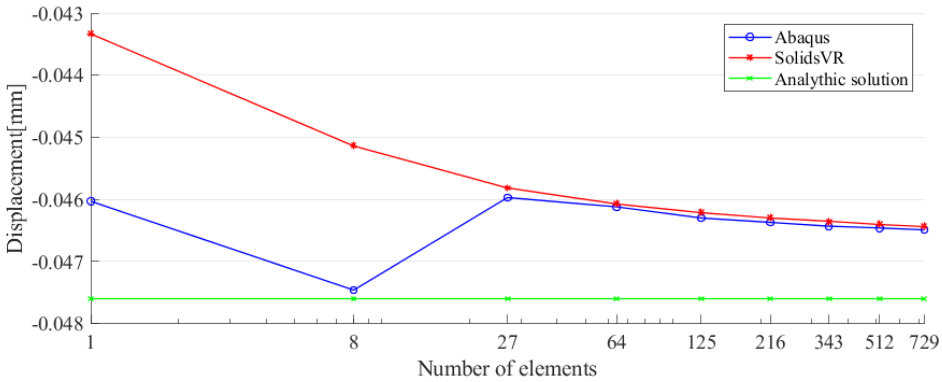


Figure 6.21: Maximum vertical displacement in Case 1 cube.

There is a clear convergence in both software programs when using finer mesh than $2 * 2 * 2 = 8$ elements. For all mesh cases, the cube modeled in SolidsVR seems to be slightly

stiffer, resulting in smaller displacement. However, the difference is 0.35% when meshing into 27 elements. When looking at the analytic solution, the difference from Abaqus is 2.33%, and from SolidsVR it is 2.40%.

Von Mises Stress

The von Mises stress in the same upper corner node is also compared to Abaqus. The results are shown in Table 6.2 and plotted in Figure 6.22.

Mesh	Elements [#]	SolidsVR [MPa]	Abaqus [MPa]	Difference [%]
1x1x1	1	10.16735	11.71080	-13.18
2x2x2	8	9.68156	8.87117	9.14
3x3x3	27	9.85812	9.96154	-1.04
4x4x4	64	9.91462	9.85105	0.65
5x5x5	125	9.94726	9.97913	-0.32
6x6x6	216	9.96506	9.96155	0.04
7x7x7	343	9.97604	9.98369	-0.08
8x8x8	512	9.98325	9.98429	-0.01
9x9x9	729	9.98820	9.98996	-0.02

Table 6.2: Comparison of the von Mises stress in selected node in Case 1 cube.

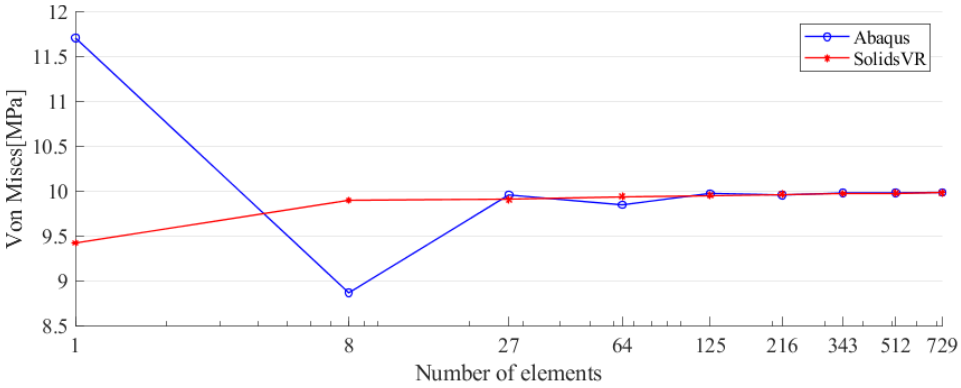


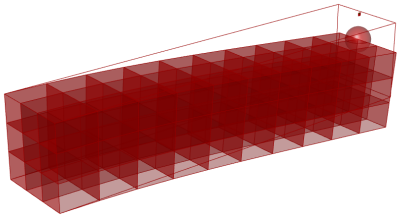
Figure 6.22: Plot of the von Mises stress in Case 1 cube.

Abaqus' results vary but do converge towards a value. With finer mesh than $4 * 4 * 4 = 64$ elements, both solutions converge relatively quick towards the same value. The difference makes up less than 1% from here, and less than 0.1% with 216 or more elements.

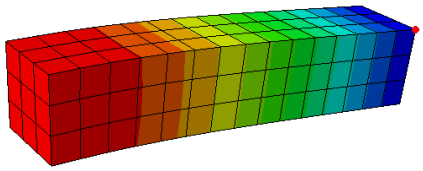
6.3.3.2 Cantilever

The second comparison geometry, the cantilever, is made of one length, L_l, longer than the other two, L_s. Again with a uniformly distributed load, F, on the top surface and boundary conditions at one end. Figure 6.24 illustrates the cantilever in SolidsVR and Abaqus, and the parameters are given below. E is the Young's modulus and ν the Poisson's ratio.

$$F = -10\frac{N}{mm^2} \quad L_s = 200mm \quad L_l = 1000mm \quad E = 210000MPa \quad \nu = 0.3$$



(a) The comparison model in SolidsVR.



(b) The comparison cantilever in Abaqus. The comparison node is marked in red.

Figure 6.23: Deformed comparison cantilevers for Case 1 meshed into $3 * 3 * 15 = 135$ elements.

Deformation

Similar to the cube, the deformation of the cantilever in SolidsVR is compared to Abaqus' and the analytic solution. This is done by comparing the vertical displacement of the corner node pointed out in Figure 6.23b with an increasing number of elements. The analytic solution, δ_{max}, from Figure 6.24, is calculated. Note that $q = F * L_s = -2000\frac{N}{mm}$ since Equation 6.5 is a 1D beam formula, and the depth of the beam is not considered.

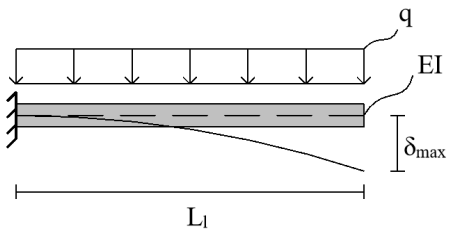


Figure 6.24: 1D cantilever with a uniform load, q.

$$I = \frac{L_s^4}{12} = 833333333.3mm^4 \tag{6.4}$$

$$\delta_{max} = \frac{qL_l^4}{8EI} = \frac{-2000 * 1000^4}{8 * 210000 * 833333333.3} = -8.92857mm \tag{6.5}$$

The vertical displacement in the selected node in SolidsVR, Abaqus and the analytic solution are shown in Table 6.3 and plotted in Figure 6.25.

Mesh	Elem. [#]	SolidsVR [mm]	Abaqus [mm]	Diff. [%]	Analy. [mm]
1x1x5	5	-5.96301	-10.92480	45.42	
2x2x10	40	-7.94962	-8.98262	11.50	
3x3x15	135	-8.55139	-9.04721	5.48	
4x4x20	320	-8.79792	-9.08798	3.19	
5x5x25	625	-8.92180	-9.11295	2.10	-8.92857
6x6x30	1080	-8.99285	-9.12872	1.49	
7x7x35	1715	-9.03747	-9.14344	1.16	
8x8x40	2560	-9.06742	-9.14684	0.87	
9x9x45	3645	-9.08855	-9.15235	0.70	

Table 6.3: Comparison of vertical displacement in selected node in Case 1 cantilever.

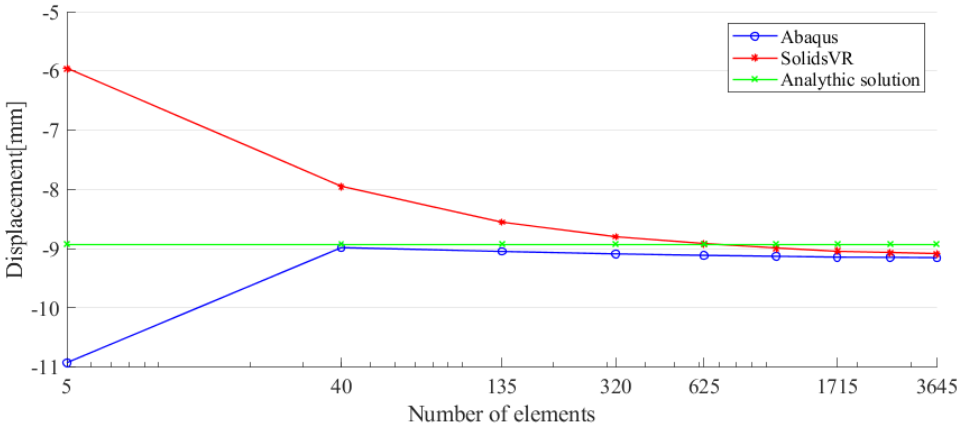


Figure 6.25: Plot of the vertical displacement in Case 1 cantilever.

It is a clear convergence in both software programs. However, SolidsVR's solution converges slower than for the squared cube. It takes 2560 elements to get a difference that makes up less than 1% of Abaqus' result. The cantilever modeled in SolidsVR is slightly stiffer, resulting in smaller displacement. When comparing to the analytic solution using 3645 elements, the difference from Abaqus is 2.51% and from SolidsVR it is 1.75%. Note that the analytic solution is not exact. It assumes the load to be applied in the center. In the 3D models, the load is applied on top, 100 mm from the center.

Von Mises Stress

The von Mises stress in the same node is shown in Table 6.4 and plotted in Figure 6.26.

Mesh	Elements [#]	SolidsVR [MPa]	Abaqus [MPa]	Difference [%]
1x1x5	5	11.8266	18.0644	-35.53
2x2x10	40	10.0266	10.1609	-1.32
3x3x15	135	9.9266	9.8798	0.47
4x4x20	320	10.0094	9.9440	0.66
5x5x25	625	10.0719	9.9450	0.77
6x6x30	1080	10.1049	10.0283	0.76
7x7x35	1715	10.1191	10.0471	0.72
8x8x40	2560	10.1224	10.0571	0.65
9x9x45	3645	10.1120	10.0615	0.58

Table 6.4: Comparison of the von Mises stress in corner node in Case 1 cantilever.

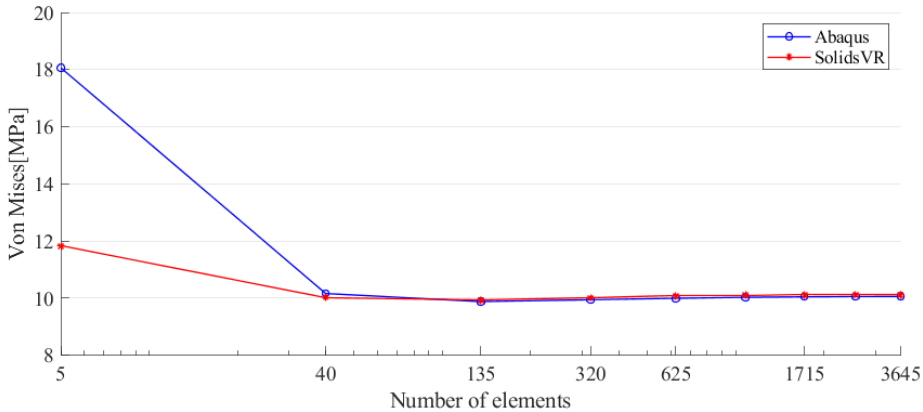


Figure 6.26: Plot of the von Mises stress in Case 1 cantilever.

SolidsVR and Abaqus differ a lot for the first mesh, but for mesh finer than 135 elements the difference stays below 0.77%.

6.3.4 Discussion

Case 1 testing a cube and cantilever provided the first insights into how SolidsVR performed compared to the finite element program Abaqus, the benchmark for testing SolidsVR. For both geometries, SolidsVR's nodal von Mises stress differs from Absqus' by less than 0.47% when meshing into 135 elements. This shows that SolidsVR is sufficiently accurate and may therefore be developed even further. It should be noted that SolidsVR uses a larger amount of elements in order to converge towards the correct results for the cantilever than Abaqus and is in both cases slightly is stiffer than Abaqus and the analytic solution.

This is also the first test using a FEM-program analyzing solids inside the VR environment. Splitting the VR world into areas with model, deformation and stress analysis is a way of exploiting the characteristics of VR. In this way it is possible to move around inside VR and see multiple results at once. In SolidsVR the pre- and post-processing is done side by side. One change creates a new analysis and could be seen by just turning around. The user interface in this study case worked well, and the main idea will remain the same for Case 2.

The interaction explained above is possible since SoldisVR keeps the VR environment parametric. This two-way connection works as expected and the user is given feedback on how adjustments affect the analysis results relatively fast.

Case 1 only handle simple geometry, where all sides of the geometry have straight edges. **MeshHex**'s way of meshing would not work for a curved geometry and that is a big restriction for a FEM program. Therefore, a natural step for developing SolidsVR further will be to develop the meshing of the geometry. However, the use of Hex8 elements provided sufficiently well results and will be used throughout the next study cases.

6.4 Case 2: The Curve

The goal of the second study case is to improve the geometry of the model from Case 1. The aim for Case 2 is that the input geometry should be built by two cross sections and a rail between them, thereby analyzing more advanced geometry. Examples of such a rail may be a polyline, arch or a NURBS curve.

The VR interface for Case 2 is the same as in Case 1. However, with the new geometry, the user may adjust it by pulling the control points of the NURBS curve or a corner point in one of the cross sections. The possibility to change mesh, boundary conditions and loads with sliders is still possible as shown in Figure 6.27.

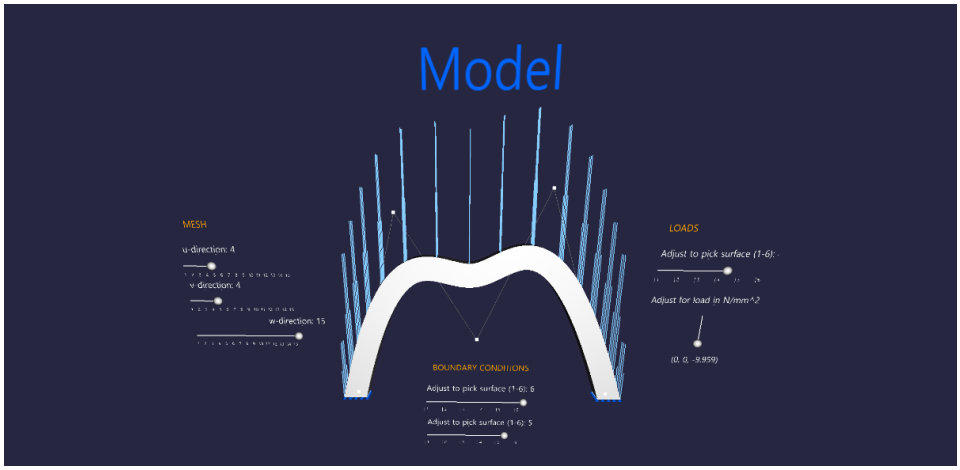


Figure 6.27: How the model area looks like from the users perspective. The user may adjust the three control points of the NURBS curve in VR.

6.4.1 Mesh Sweeping with MeshCurve

In Case 1, the meshing of the geometry was based on the fact that all edges were straight. Now, only the cross section must have straight edges while the rest of the geometry can be curved in multiple directions. The cross section is meshed and then swept through the geometry. The meshing of the curved input geometry is based on the former meshing component **MeshHex**, and is now a new component called **MeshCurve**.

Again, the meshing follows the two stages described in Case 1. In Stage 1, the difference

is that the meshing in w-direction follows the edges of the geometry, and is created from dividing the edges of the curves into w numbers of points. The creation of points in v- and u-direction is done in the exact same way as for the hexahedron in Case 1. Figure 6.28 shows the first steps in creating the mesh points and Listing 6.14 shows the beginning of the mesh code where the points in w-direction are created.

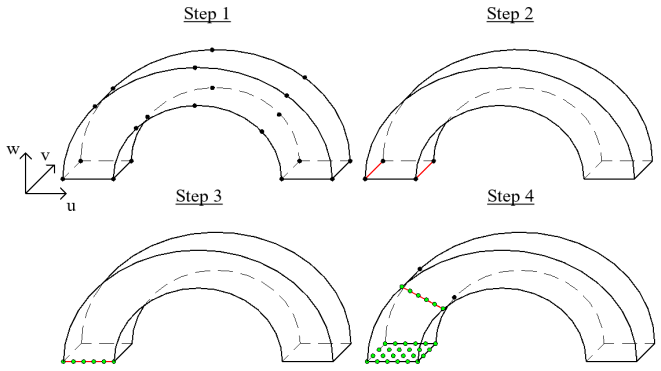


Figure 6.28: The first steps for generating nodes from mesh divisions. First, the edges in w-direction are divided in w points and then the vectors are used to create nodes.

```
for (int i=0; i<4; i++)
{ //Divide each edge in w-direction
    edges[i+8].DivideByCount(w, true, out wP);
    wDiv.Add(wP.ToList()); //Add list of divided points
}
for (int i = 0; i <= w; i++)
{//Creating points in w-direction
    Point3d p1_w = wDiv[0][i];
    Point3d p2_w = wDiv[1][i];
    Point3d p3_w = wDiv[2][i];
    Point3d p4_w = wDiv[3][i];

    Vector3d vecV1 = (p4_w - p1_w) / (p1_w.DistanceTo(p4_w));
    Vector3d vecV2 = (p3_w - p2_w) / (p2_w.DistanceTo(p3_w));
    Double length_v1 = p1_w.DistanceTo(p4_w) / v;
    Double length_v2 = p2_w.DistanceTo(p3_w) / v;
        ..... //Rest of code is the same as for case 1
```

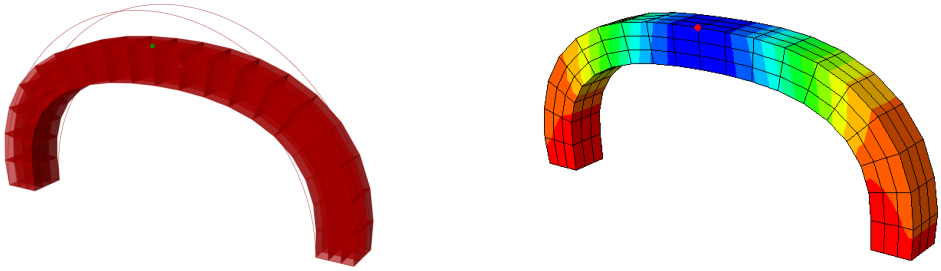
Listing 6.14: Part of meshing curved geometry in **MeshCurve**. Only start of code is shown.

Stage 2 of the meshing follows the same methodology and code as in Case 1.

6.4.2 Comparison of Results

A curve is modelled in both SolidsVR and Abaqus as an arch by specifying three points. A uniformly distributed load, F, on the top surface and boundary conditions on the bottom surfaces are applied. Figure 6.29 is illustrating this. The parameters are given below, where H is the height and L is the span of the curve. The cross section have two equal sides of lengths b. E is the Young's modulus and ν the Poisson's ratio.

$$F = -10\frac{N}{mm^2} \quad H = 500mm \quad L = 2H \quad b = 100mm \quad E = 210000MPa \quad \nu = 0.3$$



(a) SolidsVR. **(b)** Abaqus

Figure 6.29: The deformed comparison models for Case 2 meshed into $3 * 3 * 20 = 180$ elements.

Deformation

The vertical displacement of the edge node on top of the curve pointed out in Figure 6.29b is compared to Abaqus. The results are shown in Table 6.5 and plotted in Figure 6.30.

Mesh	Elements [#]	SolidsVR [mm]	Abaqus [mm]	Difference [%]
1x1x6	6	-0.281243	-0.291285	3.45
1x1x10	10	-0.358025	-0.408233	12.30
2x2x20	80	-0.449277	-0.470938	4.60
3x3x40	360	-0.495904	-0.509202	2.91
4x4x50	800	-0.505271	-0.513305	1.57
4x4x80	1280	-0.512712	-0.519655	1.34
5x5x100	2500	-0.51628	-0.520981	0.90

Table 6.5: Comparison of the vertical displacement of selected node for Case 2.

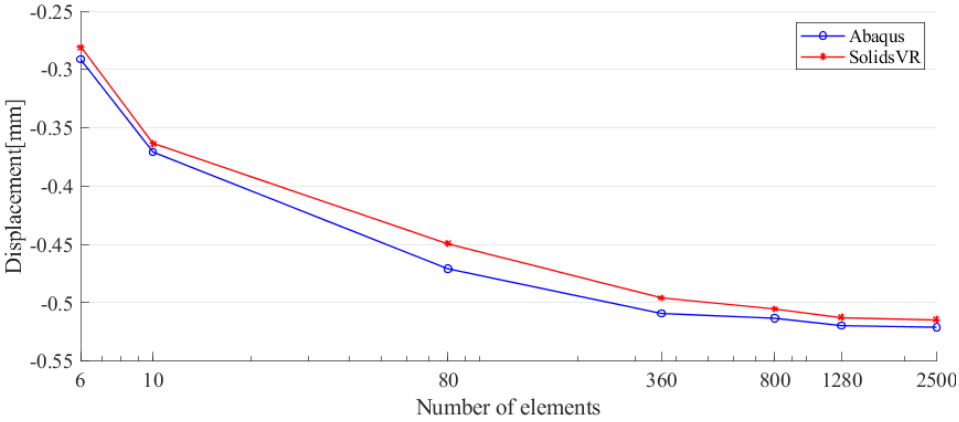


Figure 6.30: Plot of the vertical displacement in Case 2

The displacement of the top node in both SolidsVR and Abaqus converges towards the same value. Again, SolidsVR's model seems to be stiffer, resulting in smaller displacement. However, using 2000 elements or more the error is less than 1.01% of the value.

Von Mises Stress

A comparison is also done for the von Mises stress in the selected node. Table 6.8 and Figure 6.31 gives the same results as for deformation – the stress in both programs converge towards the same value, and with a finer mesh than 2000 elements, the difference is less than 1%.

Mesh	Elements [#]	SolidsVR [MPa]	Abaqus [MPa]	Difference [%]
1x1x6	6	27.8210	55.0883	-49.50
1x1x10	10	49.0584	77.5718	-36.76
2x2x20	80	75.1626	87.7692	-14.36
3x3x40	360	88.5821	92.7583	-4.50
4x4x50	800	91.0811	93.6149	-2.71
4x4x80	1280	93.2244	94.5659	-1.42
5x5x100	2500	94.1441	94.8890	-0.79

Table 6.6: Comparison of the von Mises stress in the selected node in Case 2.

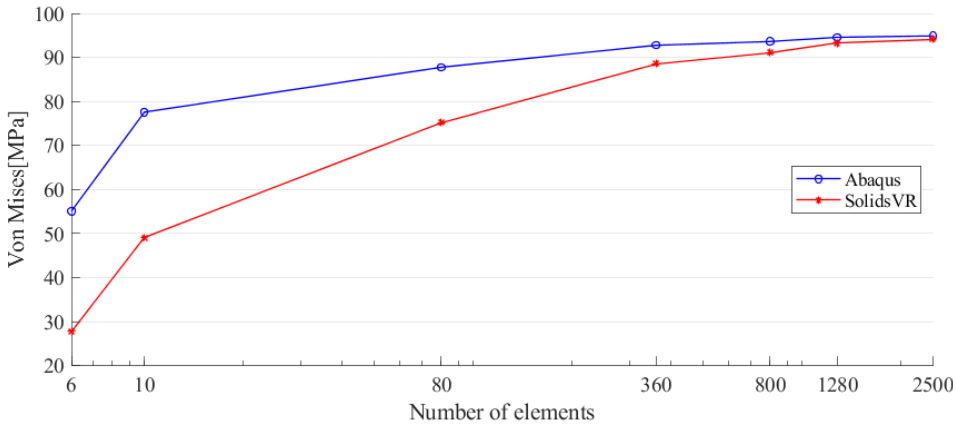


Figure 6.31: Plot of the von Mises stresses in the selected node in Case 2.

6.4.3 Discussion

The comparisons showed that the new type of meshing worked well, and more advanced geometry can be analyzed with SolidsVR. However, there are still many restrictions to the geometry. The cross section used is limited to consist of straight lines. It is still the meshing that is the limitation for analyzing even more arbitrary geometry.

There was also an experience that the post-processing components displaying the VR interface worked properly with a different geometry. The relative placement of the deformation and the stress area and the scaling of the sliders still worked well. The fact that the sliders work properly motivates to further improvement of the user interface in terms of more sliders or other types of interaction components.

SolidsVR makes it possible for the user to adjust the placement of boundary conditions, or support conditions. In the real world, a geometry may also have more complex preconditions, for example, a prescribed displacement, where a displacement constraint defines the non-zero value that is assigned to a node. In VR, the user should be able to pick what node, not only surface, to assign a user-defined displacement to.

6.5 Case 3: Advanced Cross Section

Case 2 showed that minor changes to the mesh code could increase the complexity of the geometry that can be analyzed. In this chapter, this will be taken a greater step further as the focus will be on making the cross sections of the geometry more arbitrary, while the rest of the geometry still is adjustable as in Case 2. In Case 3 of developing SolidsVR, a further generalization of the meshing is needed. Now, the goal is that the cross section should not be limited to straight lines, but may consist of any shape as long as it is built by four curves.

Features like prescribing displacement to a particular node will be explored, as a result of the discussion in Case 2. When this is done, there are also reasons to give the user insight into a particular node – not just the overall geometry.

Even though the VR interface worked properly in Case 1 and Case 2, further exploration of the VR interface would be interesting and other solutions to interact through VR should be explored. Also, when the cross sections get more advanced, a better view of the cross section is needed for the user to be able to adjust it properly.

6.5.1 Further Meshing Improvements with MeshSurface

While **MeshHex** and **MeshCurve** took advantage of straight edges and used vectors to create nodes, **MeshSurface** iterates through the geometry creating new surfaces and evaluates each surface to create nodes. The only two restrictions for this meshing is that the geometry must have eight corners and no holes. The geometry can be curved in any direction only requiring a more dense mesh to keep the precision. **MeshSurface** would work on all geometry studied earlier and could replace the mesh work in all former analyses.

Again, **MeshSurface** follows the same two stages as in **MeshHex**. Stage 1 is to create a node list in the correct order and Stage 2 is to create elements based on the order of the list. Therefore, Stage 2 of the meshing uses the exact same code as for Case 1 and Case 2.

Stage 1 is again divided into two steps, where the first step of the meshing follows the first step in **MeshCurve**. The edges in w-direction going from the cross section on one side to the other side are divided into w number of points. For each new iteration of looping through the geometry four new points on the edges are retrieved from the division of the four edges. From these four points new curves are created following the shape of the geometry. The first step is finished by creating a surface between these four curves acting

as a cut through the original geometry. This process is shown as code in Listing 6.15 and the surfaces are illustrated in Figure 6.32.

```
for (int i = 0; i <= w; i++)
{
    Point3d p_1 = p1w[i];
    Point3d p_2 = p2w[i];
    Point3d p_3 = p3w[i];
    Point3d p_4 = p4w[i];

    List<Point3d> ps1 = new List<Point3d> { p_1, p_2 };
    List<Point3d> ps2 = new List<Point3d> { p_2, p_3 };
    List<Point3d> ps3 = new List<Point3d> { p_3, p_4 };
    List<Point3d> ps4 = new List<Point3d> { p_4, p_1 };

    NurbsCurve c1 = surF1.InterpolatedCurveOnSurface(ps1, 0);
    NurbsCurve c2 = surF2.InterpolatedCurveOnSurface(ps2, 0);
    NurbsCurve c3 = surF3.InterpolatedCurveOnSurface(ps3, 0);
    NurbsCurve c4 = surF4.InterpolatedCurveOnSurface(ps4, 0);

    curve = new List<NurbsCurve>() { c1, c2, c3, c4 };
    Brep brepSurf = Brep.CreateEdgeSurface(curve);
    Surface surface = brepSurf.Faces[0].DuplicateSurface();

    List<Point3d> pointList = new List<Point3d>() { p_1, p_2, p_3, p_4 };

    (globalPoints, nodes) = CreatePoints(surface, u, v, w, i,
        cornerPoints, pointList, globalPoints, nodes);
}
```

Listing 6.15: Creating surfaces from the divided points in w-direction in **MeshSurface**.

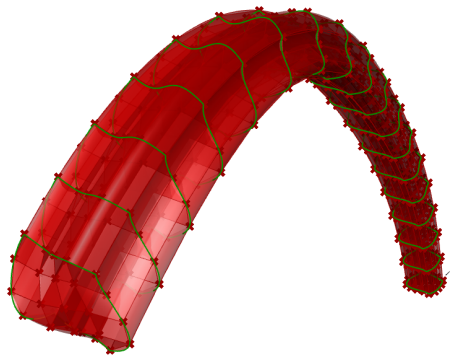
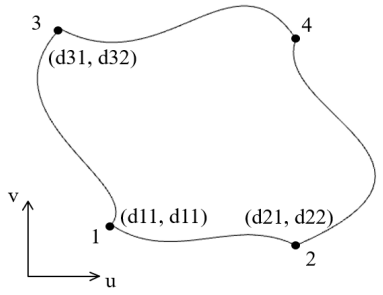


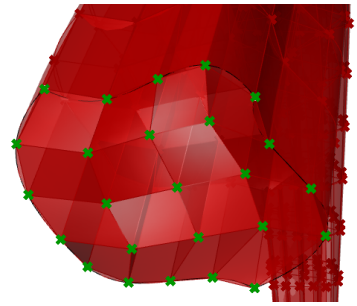
Figure 6.32: The green curves that creates a surface for each iteration in w-direction.

The second step of Stage 1 takes the created surface as input and populates it by points evaluating the surface. The function *surface.PointAt(tu, tv)* shown in line 12 in Listing 6.16 returns the point from the domain-coordinate, (tu, tv). Therefore, iterating through the domains in each direction returns points creating the mesh.

However, the u-direction and v-direction is not consistent in Grasshopper geometry. It is therefore necessary to check the direction in order to create nodes in the correct order. This check is done by the *if-sentences* shown in line 3 and line 7 in Listing 6.16. The variable names shown in Listing 6.16 corresponds with the names shown in Figure 6.33a. Figure 6.33b shows how the surface is populated with points.



(a) Order of points and domain.



(b) Populating surface with points

Figure 6.33: Step one and step two in Stage 1.

```
1  for (int j = 0; j <= v; j++)
2  {
3      if ((d31 - d11) != 0) tu = (double)(d11 - j * (d11 - d31) / v);
4      else tv = (double)(d12 - j * (d12 - d32) / v);
5      for (int k = 0; k <= u; k++)
6      {
7          if ((d21 - d11) != 0) tu = (double)(d11 - k * (d11 - d21) / u);
8          else tv = (double)(d12 - k * (d12 - d22) / u);
9          Point3d p1 = surface.PointAt(tu, tv);
10         points.Add(p1);
11         Node node = new Node(p1, points.IndexOf(p1));
12         nodes.Add(node);
13     }
14 }
```

Listing 6.16: Creating nodes by populating the surface in **MeshSurface**.

As mentioned, Stage 2 of the meshing uses the same code as for Case 1 and Case 2.

The pseudo code for **MeshSurface** is illustrated in a flowchart in Figure 6.34.

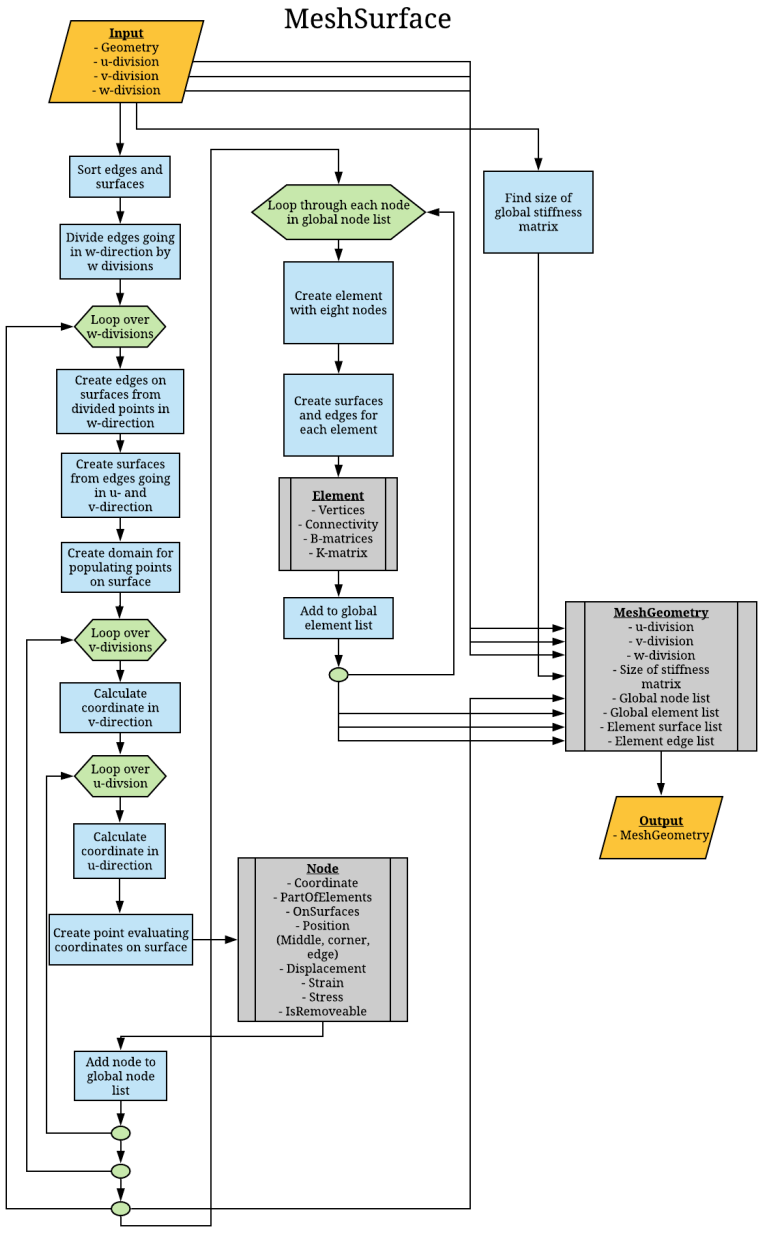


Figure 6.34: Flowchart showing the pseudo code of the meshing in **MeshSurface**.

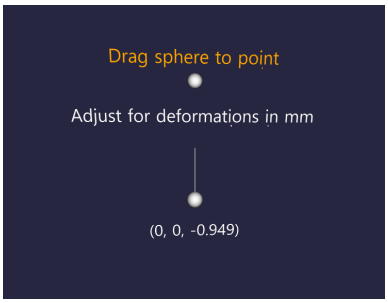
6.5.2 Prescribed Node Displacement

A new feature is added to SolidsVR which enables the user to specify prescribed displacements in a selected node. This is done by creating a new component that can be added to the list of our self-made components:

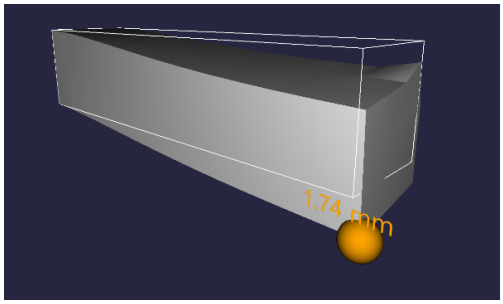
PreDisp prescribes a displacement to an input node. The second input, the displacement, should be given as a vector, while the last input is the mesh to get the coordinates of all existing nodes. The output is a text string with the point and displacements, u_x, u_y and u_z, in the same format as the boundary conditions in **SetBC**, "$x, y, z; u_x, u_y, u_z$", and the text is now also an input to **FEMSolver**.

To select a node, the user drags a sphere to the coordinates of the wanted node in VR. This is how the user can pick which point the prescribed displacement should be assigned to. It may be difficult to place the sphere in the exact coordinates of a node. Therefore, the node selected is the node closest to the point of the sphere's center. This is comparable to a snapping function. The pseudo code for **PreDisp** is showed in Figure 6.36.

The direction and magnitude of the displacement vector are found the same way as for loads in **LoadSlider** in Subsection 6.3.2. The new slider component is called **PreDisp-Slider**, and the VR interface for prescribing node displacement is shown in Figure 6.36.



(a) Sphere for choosing node and vector slider for specifing displacement.



(b) Deformed cantilever with prescribed displacement in corner node.

Figure 6.35: Interface of prescribed displacement in VR.

Adding prescribed displacements requires changes to **FEMSolver**, now applying the displacements to the global stiffness matrix according to theory in Subsection 4.1.2.2.

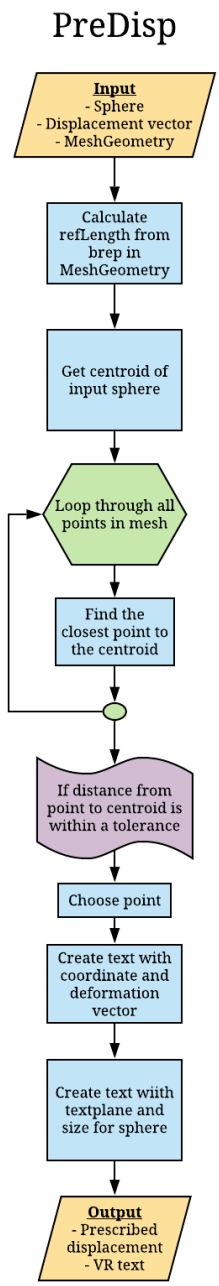


Figure 6.36: Flowchart showing the pseudo code of describing displacement to a node in **PreDisp**.

6.5.3 Selection of a Node in VR

With the logic of finding the closest point created, it could easily be reused to select a node and display information about it. This triggered the creation of another interaction component:

InfoNode simply uses the same logic of selecting a node as **PreDisp**, and displays the information about the selected node in VR. A new sphere is placed in the VR world and the user is provided with the node's coordinates, displacements and stresses as shown in Figure 6.37.

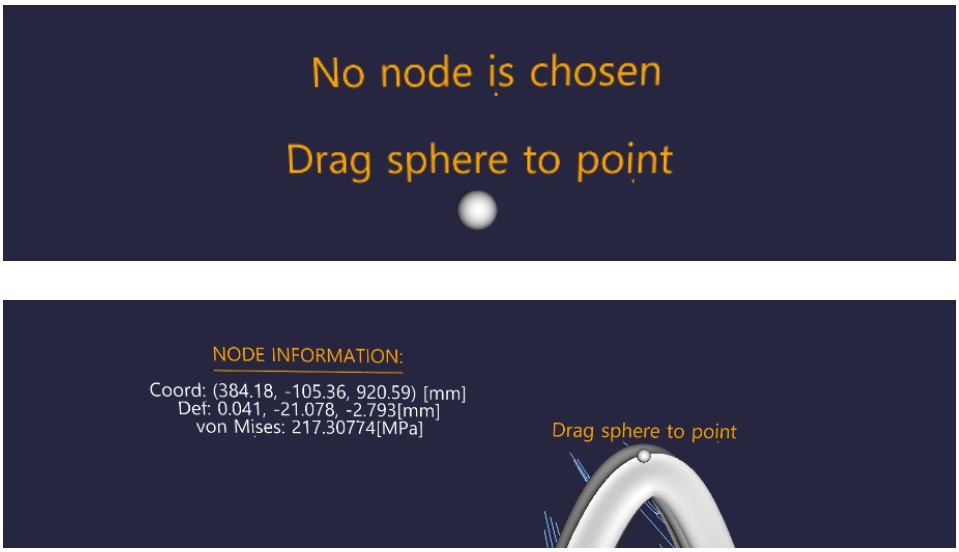


Figure 6.37: Interface for showing information of selected node.

6.5.4 Display of Cross Section in VR

When experimenting with creating cross sections in Grasshopper, the idea of displaying the cross section as a separate part came to mind. Changes to the separate part affect the geometry by assigning the exact same cross section to the geometry. The cross section can easily be changed by moving the control points of the curves creating the cross section. The cross section is assigned to the geometry by using Grasshopper operations to move and rotate the cross section according to the shape of the geometry. For displaying information

about the cross section and mesh in VR, as shown in Figure 6.38, a new component is created:

CrossSection displays the information describing the cross section to the user, such as information about u- and v-direction and how the where the nodes are placed on the surface. The inputs to the component are the curves forming the cross section and the mesh class with information about the mesh. The outputs are simply the points for showing the nodes and text to be displayed.

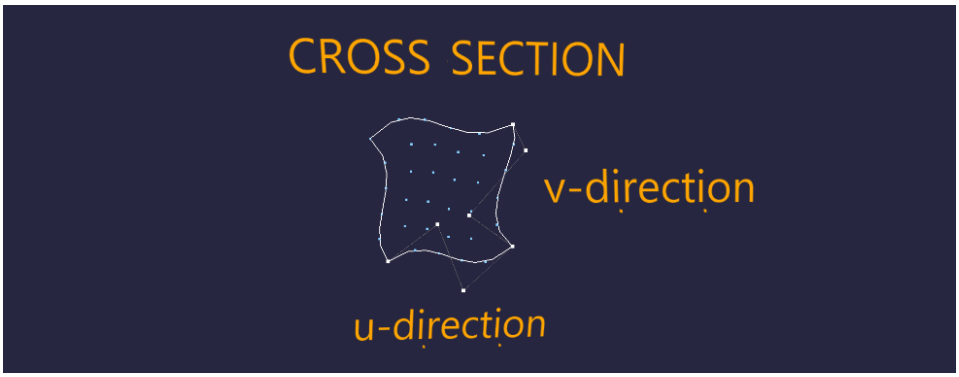


Figure 6.38: Adjustable cross section in VR.

The added feature of changing the cross section is placed to the left in the model area. This, together with an overview of all the areas with the new features, is shown in Figure 6.39.

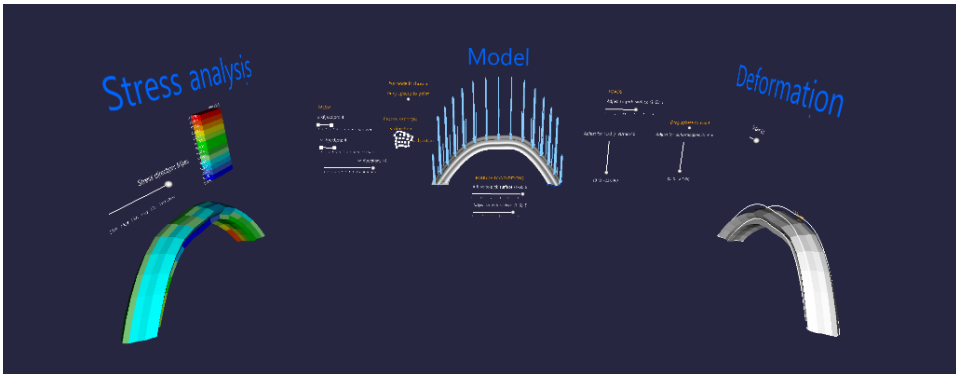


Figure 6.39: Overview of the VR interface.

6.5.5 Exploring Free-Form Geometry

This chapter focuses on analyzing more complex geometry with more arbitrary cross sections. Shell theory assumes constant thickness. Introducing the new meshing component makes it possible for SolidsVR to analyze shell structures with varying thickness.

The von Mises distribution for a shell structure is shown Figure 6.40. This structure is controlled by three curves with control points, meaning that the geometry can easily be reshaped and analyzed.

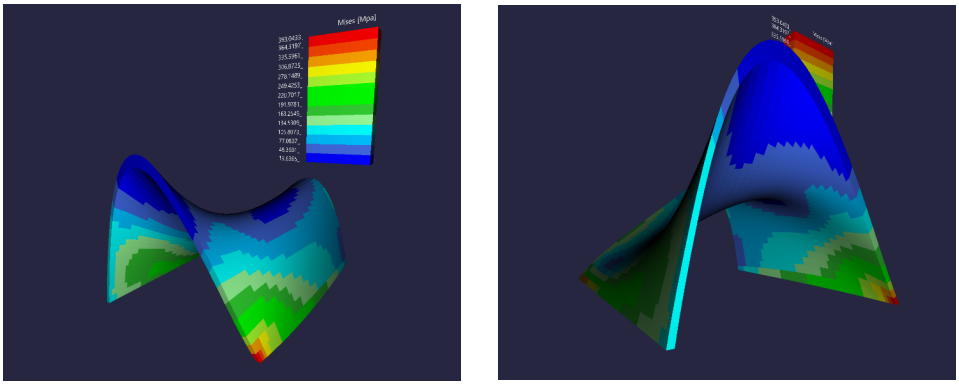


Figure 6.40: Stress distribution in shell structure in VR.

However, the analysis of the free-form shell structures showed that the code for applying the load and boundary conditions introduced in Subsection 6.3.1.2 and Subsection 6.3.1.3 is not optimal for shell structures.

6.5.6 Comparison of Results

For comparison reasons, the test curve is modeled in SolidsVR and Abaqus as half an arch by specifying three points along the arch. The comparison cross section is made of four curves, but it is plane. As described in this section, SolidsVR handles considerably more complex cross sections than the one used for this comparison, but as both programs must analyze the same geometry, such a cross section is chosen.

Figure 6.42 illustrates the geometry in both programs. Again, a uniformly distributed load,

F, is placed on the top surface and boundary conditions on the edge surfaces were applied. The measures are given below, where H is the height from the ground to the center line and L is the span of the curve. The cross section in Figure 6.41 is created from two half circles with radius, r, and two straight lines b. E is the Young's modulus and ν the Poisson's ratio.

$$F = -10\frac{N}{mm^2} \qquad H = 1000mm \qquad L = 1000mm \qquad b = 100mm$$
$$r = 50mm \qquad E = 210000MPa \qquad \nu = 0.3$$

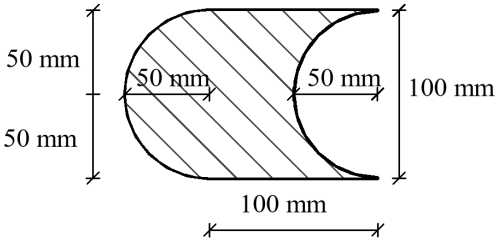
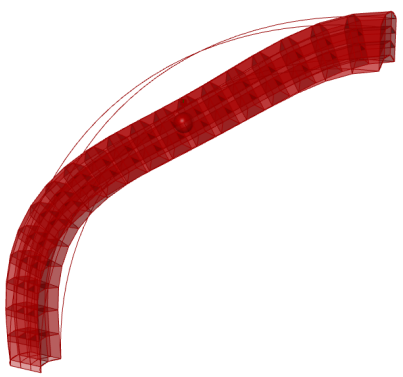
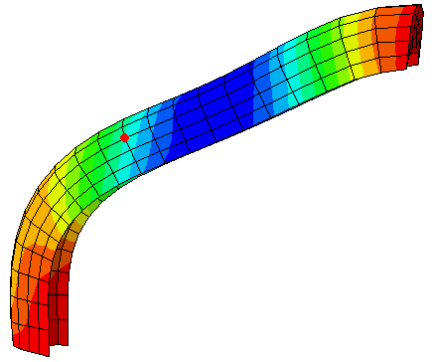


Figure 6.41: The cross section to the comparison model for Case 3.



(a) The comparison model in SolidsVR.



(b) The comparison model in Abaqus. The comparison node is marked in red.

Figure 6.42: The deformed comparison models for Case 3 meshed into $3 * 3 * 20 = 180$ elements.

Deformation

The vertical displacement of a node located at point $(257.54, 50, 742.46)$ is compared to Abaqus. The node is pointed out in Figure 6.42b. Again, this is done with a different number of elements and the results are shown in Table 6.7 and plotted in Figure 6.43.

Mesh	Elements [#]	SolidsVR [mm]	Abaqus [mm]	Difference [%]
2x2x6	24	-0.692098	-0.680519	-1.7
3x3x10	90	-0.890484	-0.886851	-0.41
4x4x16	256	-1.02163	-1.02365	0.20
5x5x20	500	-1.07248	-1.07728	0.45
6x6x26	936	-1.11407	-1.11893	0.43
7x7x30	1470	-1.13358	-1.13879	0.46
8x8x36	2304	-1.15129	-1.15532	0.35
8x8x46	2944	-1.16410	-1.16665	0.22

Table 6.7: Comparison of the vertical displacement of the selected node in Case 3.

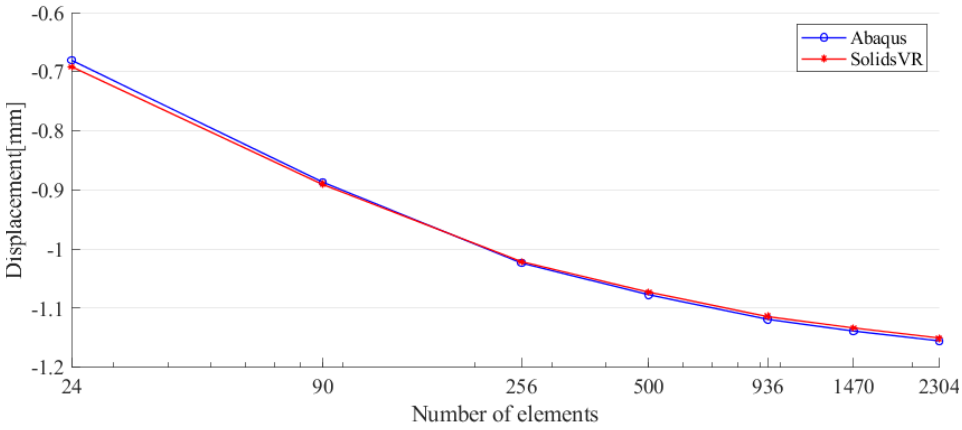


Figure 6.43: Plot of the vertical displacement of the selected node in Case 3.

SolidsVR's results differ from Abaqus's results less than 0.5% when meshing with a finer mesh than 90 elements. Figure 6.43 is illustrating that the results have a similar convergence as Abaqus.

Von Mises Stress

A comparison is also done for the von Mises stress in the same node. Table 6.8 provides the results, and a plot is found in Figure 6.44.

Mesh	Elements [#]	SolidsVR [MPa]	Abaqus [MPa]	Difference [%]
2x2x6	24	75.922	117.497	-35.38
3x3x10	90	95.407	116.065	-17.80
4x4x16	256	106.378	115.333	-7.76
5x5x20	500	109.961	114.338	-3.83
6x6x26	936	113.379	115.814	-2.10
7x7x30	1470	114.771	116.378	-1.38
8x8x36	2304	116.261	116.824	-0.48

Table 6.8: Comparison of the von Mises stress in the selected node in Case 3.

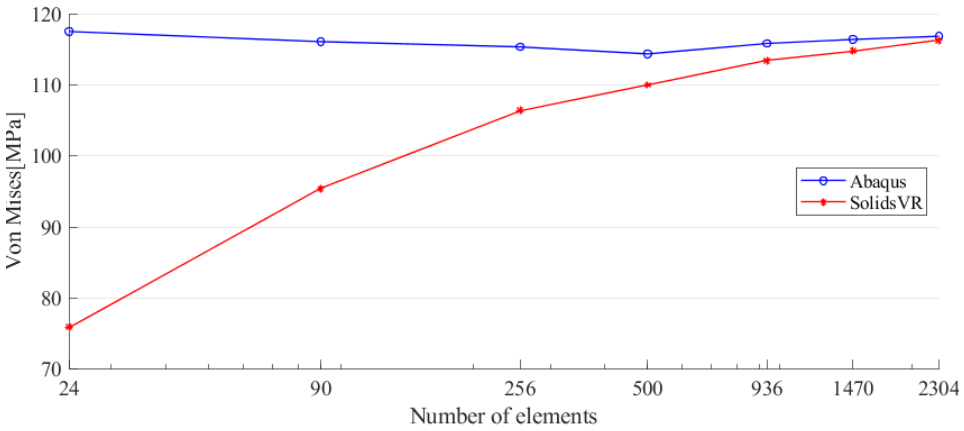


Figure 6.44: Plot of the von Mises stresses in the selected node in Case 3.

The stress results are showing that SolidsVR converges towards Abaqus' solution as the number of elements increases. However, for a smaller difference than 1% a 2300 element mesh is needed.

6.5.7 Discussion

The improvements of meshing open up a lot of new possibilities for creating geometry, and really exploits the possibilities when it comes to analyzing 3D solids. With comforting results regarding the comparison of accuracy, the meshing can be trusted. Again, only minor changes to the other components were needed, and this proofs a generic code. Of course, there are still restrictions to the input geometry, but it is sufficient for exploration in this thesis. Therefore, the meshing component will not be further improved.

Introducing the possibility to choose nodes in the VR environment makes it possible to add several new features to the interface. In Case 3, prescribed displacement to a node was added along with the possibility to get information of one chosen node. Both of these features worked as desired. However, if a prescribed displacement is to be placed in more than one node, the user interface would look chaotic and other solutions to do this should be considered.

The idea of editing the cross section in a separate location than the model also worked well. Small changes are easily made with the VR controllers, by adjusting control points of the edges. If bigger changes should be made, like changing one of the curves to a straight line, it is debatable whether it is easier to do in Rhino or VR.

Even though the code is generic for analyzing different types of geometry, it is not optimal for all types. Exploring the free-form geometry showed a significant increase in computation time for analyzing the shell structures with varying thickness. Examinations of the code showed that the increase of nodes with assigned load or boundary conditions slowed down the creation of the force vector and reducing the stiffness matrix massively. In retrospect, the assigning of loads and boundary conditions should have been done differently by assigning the load directly to the node objects instead of having to compare each node in the node list for each new load or boundary condition assigned.

So far, the meshing algorithms have been improved, functionalities have been added and the VR interface modified. The user is provided with a lot of features and insight into FEA results. Still, the user has to interpret these results and choose whether to keep the geometry as it is or change it and how. It could be valuable for the user to get some indication on *how* to improve the geometry, given some requirements or constraints. As it is more difficult to understand how the load is distributing in 3D than in 2D, this will increase the value of adding this feature to SolidsVR and look at the results in VR.

6.6 Case 4: Improving the Geometry

To improve the geometry can mean to reduce material, to reduce deformation or even make it look prettier in terms of architecture. To evaluate what is a *better* geometry, some functions and equations defining this have to be established. In this case, only basic ones will be used, as the goal is to adapt our existing code and analysis to this new feature.

During the research of this topic, the method *topology optimization* came across. It is maximizing the performance of a structure, given design space, load, boundary conditions and other constraints. The method has traditionally been used in construction where the goal is to save material and still ensure stiffness, but has also been described as a sparring partner during the design phase [41]. The vision of architects and engineers collaborating in VR as described in Section 3.3 is also a motivation to look at this topic. According to Beghini et al. (2013) a tailored topology optimization approach can be used as a tool that might lead to a better collaboration of these two disciplines when designing buildings [42].

Using the saving of material argument as an example, the performance could be measured in minimizing the mass of the structure, and the constraint is to not override the yield stress of the material used. The method will remove ineffective elements from the structure, analyze it again and repeat until constraints are reached.

6.6.1 Iterative Removal of Elements

With inspiration from the topology optimization method, an iterative solver will be explored. To explore this feature, a goal and restrictions are needed.

The goal is chosen to be to minimize the volume of the geometry:

$$min \sum_{e=1}^{n} v_e, \tag{6.6}$$

where v_e is the volume of element e.

Since the geometries in this thesis consist of the same kind of material and are meshed into elements of approximately the same volume, Equation 6.6 is simplified to eliminating elements from the geometry. This is done by removing one by one element, as long as the constraint is fulfilled. In this study case, there are two constraints. First, there should not be removed more material than a limit chosen by the user. This limit is a percentage of

the original volume. Second, the maximum von Mises stress should not override the yield stress of the material.

The constraints are expressed as

$$\sigma_{mises,e} < \sigma_{mises,max}, \tag{6.7}$$

where $\sigma_{mises,e}$ is the von Mises stress of element e and $\sigma_{mises,max}$ is the yield stress of the material, and

$$\sum_{e=1}^{n} v_e > v_{min} = V - V * v_{remove}, \tag{6.8}$$

where v_{min} is the lower boundary for the volume of the new geometry. It is calculated based on the original volume V and the input percentage from the user, v_{remove}.

6.6.1.1 Improving FEMSolver

The **FEMSolver** component has to run every time one element is removed to calculate the updated stress state. This is implemented as a *while-loop* in the component code as in Listing 6.17. If the requirements inside the parenthesis are fulfilled, it will continue to execute the following code. Only elements with no load and no boundary conditions can be removed.

```
1 while (numberElements > minElements && max < material.GetY()  first)
```

Listing 6.17: While loop in **FEMSolver** with the requirements deciding if the code should be executed.

These requirements are just a translation of the constraints in Equation 6.7, Equation 6.8 and the fact that the FEA has to be run at least once (if *first* is true, it means that it is the first time running through the while-loop).

Figure 6.45 shows this iterative deletion of elements and how it works with the rest of the code. The user may choose whether such an improvement suggestion or one single finite element analysis should be run. If the chosen percentage to be removed is zero, $v_{remove} = 0\%$, only one FEA is executed and no elements are removed.

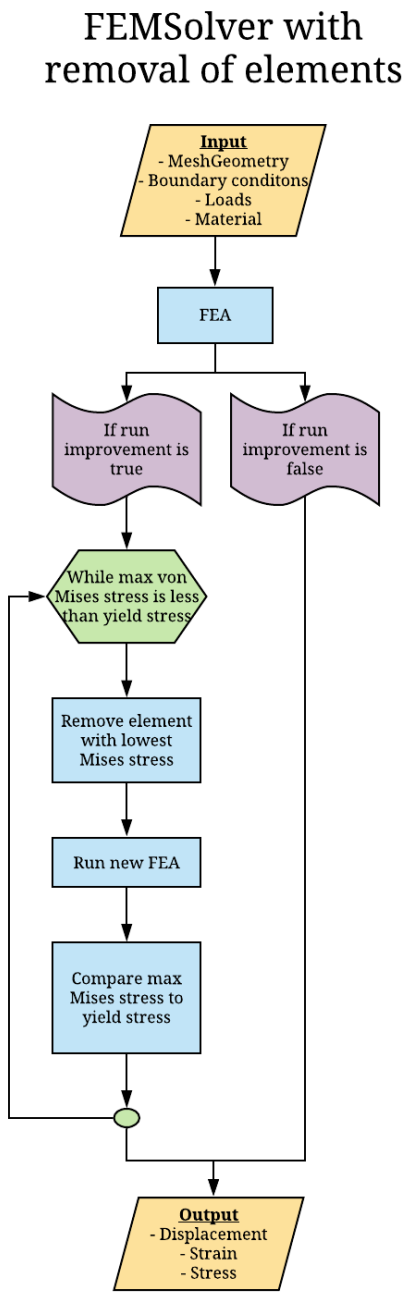


Figure 6.45: Flowchart showing the pseudo code of the new **FemSolver**.

Some changes are made to the **FEMSolver** component:

- The method *RemoveOneNode* returns the element number of the element with the lowest von Mises stress, and the current maximum von Mises stress in the geometry, $\sigma_{mises,max}$.
- A method, *RemoveElementAndUpdateNodes*, that runs if an improvement analysis is chosen, is added. If $\sigma_{mises,max}$ is less than the yield stress, the selected element is removed from the list of elements before analyzing the geometry. The list of the belonging elements which the node is a part of is updated.
- *UpdateKandR* is a method updating the stiffness matrix and load vector. This is done by removing the entries corresponding to the removed nodes. These rows and columns will not have any contribution to the stiffness of the structure, but need to be removed to avoid singularity and make matrix inversion possible.

6.6.1.2 Updating the VR interface

For Case 4, an input representing the volume percentage to be removed is needed from the user. A new number slider, **VolumePercentageSlider**, is added to the list of components. It is displayed in the VR interface and the user may adjust the length. The output value is a number, v_{remove}, which is the input to the new **FEMSolver** component.

One thing makes this slider different from the other. If the value is zero, the user wishes not to remove elements from the geometry. This is often the case, and it should be easy to choose this setting. This is solved by setting a larger interval on the slider where the value is zero. This is illustrated in Figure 6.46.

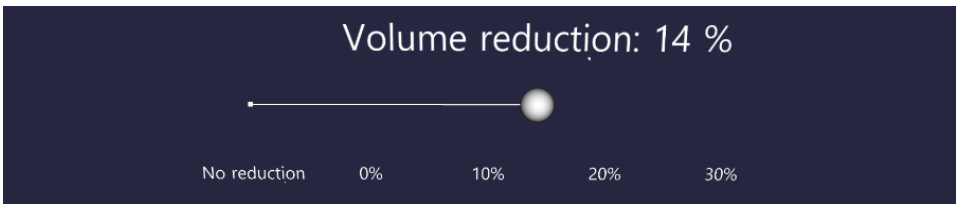


Figure 6.46: A 14% volume reduction is requested.

It follows an example of how a geometry looks like in VR before and after volume reduction. The geometry is a double clamped beam with a uniform load on top.

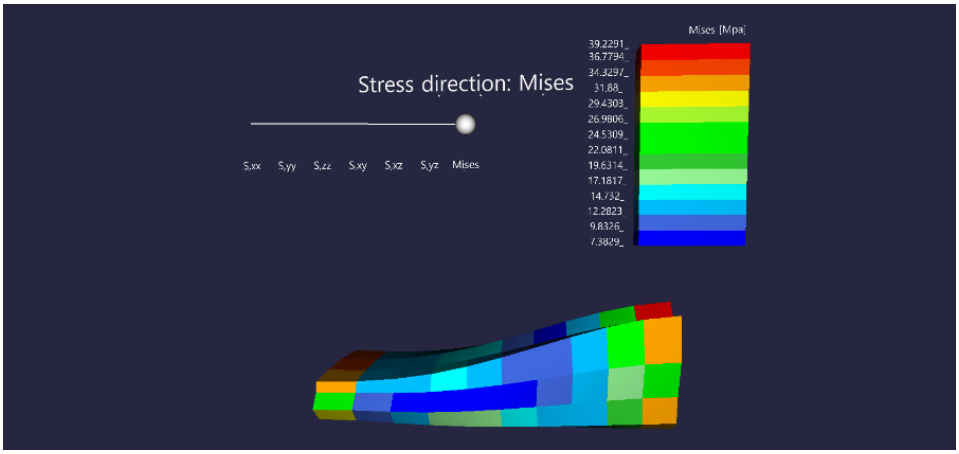


Figure 6.47: Stress distribution in VR before volume reduction.

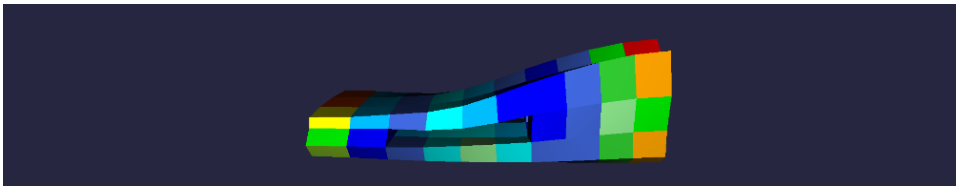


Figure 6.48: Stress distribution in VR after 14% volume reduction.

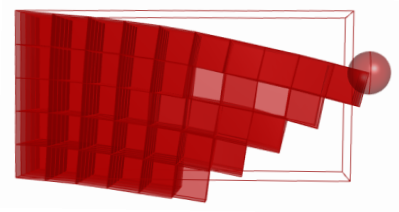
When visualizing the element stress in Figure 6.47 it is clear that the elements with lowest von Mises stress, which is colored in dark blue like the bottom of the legend, is removed in Figure 6.48. This example shows that a hole appear in the geometry as a result of the volume reduction.

6.6.2 Comparison of Results

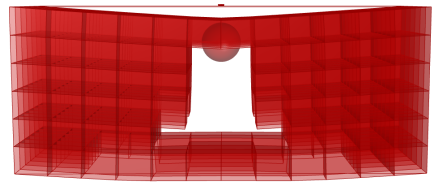
For this study case, there are two things to be considered. One of them is to verify that SolidsVR is still giving the correct FEA results. As in earlier study cases, a geometry is created in SolidsVR and Abaqus' and compared. Before this is done, it is also necessary to argue that it is actually the correct elements that are removed.

Removal of Elements

As a simple check to see if the results make sense, it has been removed parts of the volume from two known structures, a cantilever and a double clamped beam, both with uniform load on the top surface. The resulting geometry is shown in Figure 6.49.



(a) Cantilever after a 30% volume reduction.



(b) Doubly clamped beam after a 20% volume reduction.

Figure 6.49: Geometry after removal of elements. The sphere's location represents the maximum displacement.

Figure 6.49a shows the removal of the bottom elements of the free end. From basic mechanics, it is known that this is where it is less stress. Similar arguments hold for Figure 6.49b where the center elements are removed. The stress is greater on the top and bottom surface than in the middle. This indicates that the new **FEMSolver** component is doing what it is supposed to.

To supplement, some additional analyzes are performed on the cantilever in Figure 6.49a with dimensions of $1000mm*1000mm*2000mm$, meshed into $5*5*10 = 250$ elements. 15%, 30%, 45% and 60% is removed of the volume and the results are summarized in Table 6.9.

Vol. reduction [%]	Max Mises [MPa]	Min Mises [MPa]	Max disp. [mm]
0	83.0705	1.7460	-1.3789
15	83.0242	14.6976	-1.4673
30	83.0795	15.5205	-1.8031
45	94.9909	17.7956	-2.5360
60	169.9039	28.4855	-6.0624

Table 6.9: Results of cantilever of a model before and after volume reduction in Case 4.

The values makes sense to some degree. Since the element with less von Mises stress is removed, the structure's minimum and maximum von Mises stress are increasing, and so is the displacement. However, looking at which elements are removed when increasing v_{remove}, there is a reason to become more skeptical. As SolidsVR's algorithm only removes one element between each execution of FEA, the symmetry is not kept. This may result in a completely lopsided structure, as the cantilever illustraded in Figure 6.50. Here, 60% of the volume is removed, which in many real-life projects are a lot, but it illustrates the point.

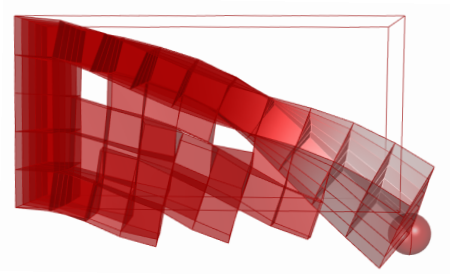


Figure 6.50: Cantilever after 60% of the volume is removed.

Still, it is arguable that **FEMSolver** does what it is told, namely removing elements with less von Mises stress. To gain more valuable results, it is the constraints that need to be advanced. With reliable results in terms of removing the element with less stress, the deformation and stress results from the analysis are compared below.

Deformation and Von Mises stress

For the comparison with Abaqus a cantilever is created, but this time with a hole through the cantilever. In Abaqus, the geometry is modelled with a hole with side lengths h, as shown in Figure 6.51. For SolidsVR the hole is created by specifying which elements to remove for each iteration of the removal of elements.

Keep in mind that this test is for a geometry where elements are removed to create the same geometry in both programs. In other words, the comparison geometry is not necessarily the optimal one.

A uniform load, F, is placed on the top of the surface and boundary condition on one of the ends. As the process of creating a consistent hole by removing elements is quite

extensive, only three different mesh cases are used in this comparison. The parameters are given below, where L_s is the shortest side length and L_l is the longer one. E is the Young's modulus and ν the Poisson's ratio.

$$F = \frac{N}{mm^2} \qquad L_s = 200x200mm \qquad L_l = 1000mm$$
$$h = \frac{200}{3}mm \qquad E = 210000MPa \qquad \nu = 0.3$$

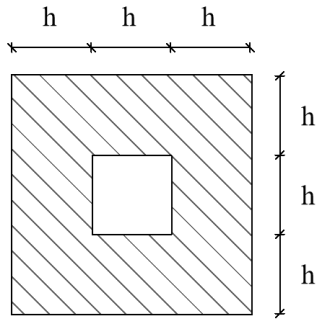
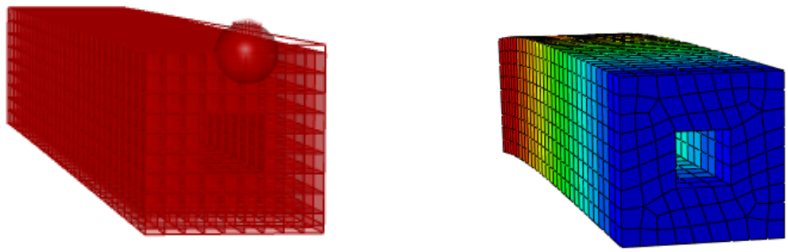


Figure 6.51: Cross section for the comparison model in Case 4.



(a) The comparison model in SolidsVR.

(b) The comparison model in Abaqus.

Figure 6.52: The deformed comparison models for Case 4 meshed into 2160 elements.

The vertical displacement of an upper corner node at the end of the cantilever is used to compare the values. Table 6.10 shows the comparison of the displacement.

Mesh	Elements [#]	SolidsVR [mm]	Abaqus [mm]	Difference [%]
3x3x10	80	-8.38599	-8.85986	5.36
6x6x20	640	-9.11620	-9.26453	1.60
9x9x30	2160	-9.28324	-9.33582	0.56

Table 6.10: Comparison of vertical displacement in selected node in Case 4.

Table 6.11 shows the comparison of the von Mises stress.

Mesh	Elements [#]	SolidsVR [MPa]	Abaqus [MPa]	Difference [%]
3x3x10	80	11.85939	11.77240	0.74
6x6x20	640	10.37584	9.92631	4.53
9x9x30	2160	10.23945	10.08130	1.57

Table 6.11: Results and comparison of the von Mises stress in Case 4.

The displacement is converging towards the same value as Abaqus', with a difference of 0.56% when meshing into 2160 elements. Regarding the von Mises stress, this trend is not as clear due to few mesh cases. However, there are good reasons to believe that SolidsVR's accuracy is sufficient after removing elements.

6.6.3 Discussion

Case 4 emphasizes the point mentioned in Section 5.1, where it is discussed how components may be changed and adapted to the user's needs. The optimization is not a needed feature for SolidsVR to work, but it is a feature that enhances the functionality of the package, and therefore added to the prototype in later study cases.

This case is only exploring the possibilities of an implementation of an improvement proposal for the geometry to SolidsVR and proves that this feature is possible. Looking at the stress results from the comparison to Abaqus, it is safe to state that SolidsVR's analysis is accurate enough for reduced geometry. The components developed in earlier study cases are generic enough to work properly on geometry with elements removed – no matter what mathematical functions are used to eliminate elements. The fact that SolidsVR can remove certain elements in the geometry, indicates that it can be improved to handle geometry with holes in it.

However, when looking at which elements that are removed, many improvements to the constraints are needed. This study case uses very simple functions and constraints, and it is not robust enough to conclude that the proposed improvement will make geometry better. It only proves that it is *possible* to add such a function to SolidsVR. If the goal is to provide a structural engineer with a new suggestion for a structure, several additional factors need to be considered. The aesthetics, safety and deformation are just some of them. In addition to this, the method could be extended to handle also addition of elements, not only removal. It could be interesting to take this iterative process a step further and dig deeper into optimization of geometry design.

One more drawback of this feature is that it seems to be very time consuming since the analysis has to be run multiple times. When the user is waiting for the results in the VR world, intended not to take the VR headset off, a long waiting time is not accepted. Specific for Case 4, the time usage could be reduced by removing more than one element at the time. More about SolidsVR's execution time in Chapter 7.

6.7 Installation of SolidsVR

All the needed files are attached to this thesis as a ZIP file with the name *Files_Akselsen&Stenvold.zip*, explained in Appendix A.

SolidsVR can be added as a plug-in to Grasshopper in two different ways. The first alternative is to drag the Grasshopper plug-in to the components folder in Grasshopper. This will only work if you use Rhino 6 version 6.15.19148.13161, as the plug-in in the ZIP file was built for this version. The other option is to build the code to the same folder. If you do this, it does not matter version of Rhino 6 you use. Both ways are explained in step 3.

1. Open Rhino, and type "Grasshopper" in the Command line, as shown in Figure 6.53. This will launch Grasshopper.

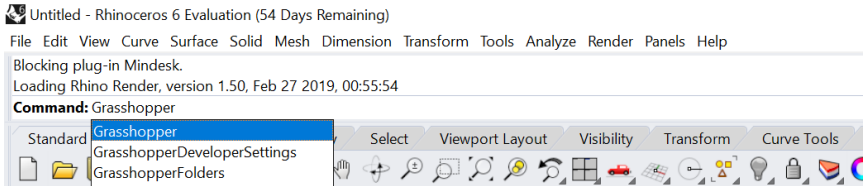


Figure 6.53: Launch Grasshopper.

2. In Grasshopper, open the *Components Folder* by following the steps in Figure 6.54. *File* → *Special Folders* → *Components Folder*.

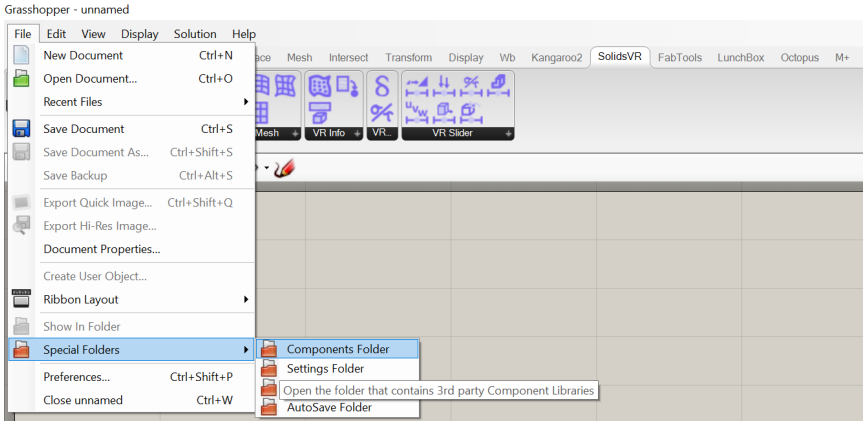


Figure 6.54: Open *Components Folder*.

3. If the same version of Rhino 6 as used in this thesis is installed, the *SolidsVR*-folder in the *Grasshopper Plug-In*-folder in the ZIP file, can be dragged into Grasshopper by following:

   The component folder, named *Libraries* in the file explorer, contains all plug-ins to Grasshopper that is not part of the original program. This folder will be empty if no packages are added. Drag the folder *SolidsVR* to this folder as shown in Figure 6.55. Then, close Rhino and Grasshopper and start the programs again.

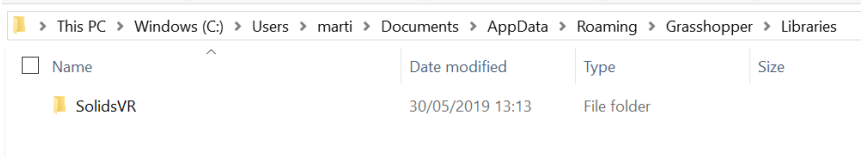


Figure 6.55: Drag *SolidsVR* to the components folder, *Libraries*.

   If the Rhino 6 version is different from this thesis', SolidsVR must be built to the same folder. When in the component folder, *Libraries*, copy the path name. Then, open the project, in the *SolidsVR*-folder from the *The Code*-folder in the ZIP file, in Visual Studio and double-click on *Properties* below the name of the project, SolidsVR. Inside the properties go to *Build* and paste the path name into the *Output Path* as shown in Figure 6.56.

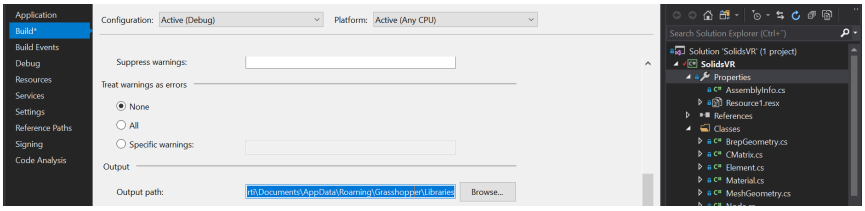


Figure 6.56: Paste the component folder's path name to the *Output path*.

   Click *Start* at the top ribbon in Visual Studio. Rhino will now start and SolidsVR will be created inside the component folder. Restart Rhino and Grasshopper.

4. When opening Grasshopper again, SolidsVR with all components will be added to the ribbon at the top as shown in Figure 6.2. The plug-in is now ready to be used.

Note that many of the components in SolidsVR have outputs that directly are connected to the components of Mindesk, and Mindesk should therefore be downloaded separately.

Chapter 7

Time Performance

To analyze the performance of SolidsVR in terms of the code's execution time, we use a built-in tool in Visual Studio called performance profiling. The programming language used is discussed in Section 7.1. Section 7.2 is presenting the results found during the performance profiling and we discuss the user experience in light of the time usage. There is potential for improvement and some of the measures are written about in Section 7.3.

7.1 Programming Implementation

The code behind SolidsVR is written in an object-oriented programming language called C#. C# is considered an economic language regarding memory and processing power, but cannot compare with languages like Fortran and C++. Fortran and C++ are native languages, meaning that the code can run without conversion, unlike other program languages like C#, Java, and Python. For instance, Abaqus uses Fortran for compiling its calculation code.

SolidsVR uses the package Math.NET Numerics [43], Math.NET hereafter, for most calculations. Math.NET provides methods and algorithms for mathematical operations on .NET-applications, like inverting and other matrix operations. Math.NET also provides the possibility of running operations using native language inside the C# program language. Therefore, all matrix operations in SolidsVR are done without having to convert before compiling, thereby increasing the performance of the program massively.

7.2 Performance Profiling

For SolidsVR to be as user-friendly as possible, the time usage of the analysis algorithms is an important topic to discuss. To analyze the performance of the code, Visual Studio's profiling tool is used. This presents the time usage of every method in each component in the running program and is used to learn what part of the code that demands most of the CPU time. The computation time depends heavily on the testing computer, and in this thesis, a four core 2,3 GHz Intel Core i5 processor and memory of 8 GB RAM, is used.

A performance profiling of **FEMSolver** is shown in Figure 7.1 to find the bottleneck in the component. The figure shows the time usage of the whole component, the creation of the stiffness matrix, **K**, applying loads and boundary conditions, inverting **K** and the calculation of deformation, stress and strain. This is done for the same squared cube as in Subsection 6.3.3.1 for a varying number of elements, and consequently also dofs.

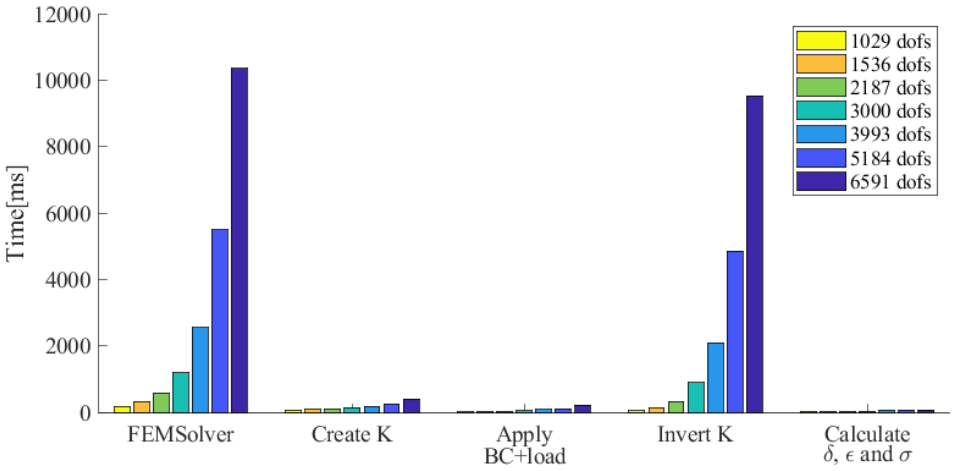


Figure 7.1: Plot of the time usage of different methods in **FEMSolver** for different number of dofs.

For finer meshing, or more dofs, it is clear that the inverting of the stiffness matrix is the most time-consuming task in **FEMSolver**, using almost the same time as the component itself. Increasing the speed of the other methods would not contribute to a significant improvement of the performance.

The inverting of the system stiffness matrix is arguably the most discussed numerical problem in FEM. There are a lot of different numerical methods that solve the linear system of equations. SolidsVR does not implement its own method for solving the force-

displacement relationship but uses Math.NET for inverting the system stiffness matrix. A comparison of the computation time using Math.NET and the Cholesky decomposition method is shown in Table 7.1. Here, the Cholesky-Banachiewicz algorithm is used. The results show that the Cholesky decomposition method is not even close to Math.NET. Considering that Math.NET uses native language and is specially optimized for matrix operations, and the Cholesky decomposition method is written in normal C# code, the results are not surprising.

dofs [#]	Math.NET [s]	Cholesky-Banachiewicz [s]
1029	0.2	1.3
1536	0.3	4.2
2187	0.6	13.7
3000	1.2	36.3
3993	2.6	78.4
5184	5.4	174.0
6591	10.3	348.9

Table 7.1: Comparison for computation time using Math.NET and Cholesky-Banachiewicz.

Even though the calculation component, **FEMSolver**, is the most time consuming of the components, a comparison of the different mesh methods is done. When improving the meshing component in Case 3 to handle more advanced geometry, as described in Subsection 6.5.1, some concerns were expressed about the time performance of this component. This concern can be rejected after seeing the plot of the comparison in Figure 7.2.

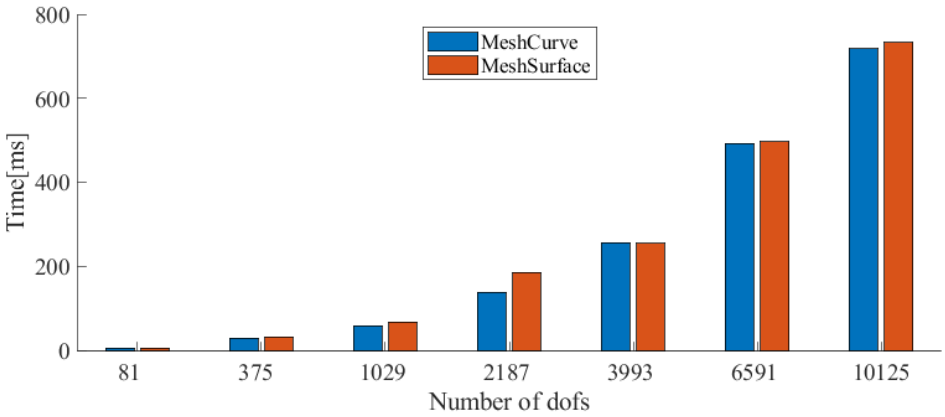


Figure 7.2: Comparison of total runtime of the components MeshCurve and MeshSurface.

The geometry improvement feature in Case 4 and the fact that **FEMSolver** may be run multiple times, emphasizes the importance of the time performance. Even though the stiffness matrix is slightly reduced when removing elements, the feature of removing elements is still time-consuming. Removing n number of elements means that the analysis have to be performed n times. Hence, the stiffness matrix must be inverted n times, exponentially increasing the computation time.

Table 7.2 shows the time usage of **FEMSolver** when removing a varying number of elements from the cantilever with a hole in the middle from Subsection 6.6.2.

Meshing	**Elements [#]**	**Removed elements [#]**	**Analyzing time [min]**
3x3x10	90	10	0.007
6x6x20	720	80	3.9
9x9x30	2430	270	510

Table 7.2: Comparison of time usage for varying number of elements for the cantilever used in the comparison of results in Case 4.

7.3 Performance Improvements

During the research and the development of SolidsVR, several measures to reduce the execution time were discovered. Some were implemented immediately and others were not prioritized as they were too comprehensive compared to the improvement it would bring. However, it is considered meaningful to mention the findings not implemented yet:

- Improve the method for solving the linear system of equations. Math.NET is optimized for inverting matrices, but not for solving the system of equations. Writing a method like the Cholesky decomposition, in a native language, would most likely be faster than Math.NET. Using native program languages, like Fortran and C++, as a core of the FEM solver, can also be a good solution for creating a FEM program in C# which have other advantages than the computation power.
- Only store the non-zero values of the global stiffness matrix. SolidsVR uses dense matrices because Math.NET, as of this day, have not optimized solvers for sparse matrices. Storing only non-zero values will save more memory and decrease the computation time as long as the force-displacement relationship can be solved with sparse matrices.

- Only store half of the global stiffness matrix. It is symmetric, and the complete matrix can quickly be built if needed. Also, rows and columns corresponding to nodes with boundary conditions can be removed before solving for the displacement and added again after the calculations.
- Renumber all nodes to decrease the bandwidth of the system stiffness matrix. The bandwidth describes how far from the stiffness matrix' diagonal non-zero values are placed. It is preferable to renumber nodes such that nodes belonging to the same elements are placed close together.
- Parallelize the code by running different parts of the code on separate parts of the computer simultaneously. In fact, parts of the problem of solving the linear system of equations can be solved simultaneously, thereby decreasing the computation time.
- Use arrays, rather than lists. If preallocated, looping through and accessing elements are faster for arrays than lists. On the other side, it should be kept in mind that lists are more flexible and have more functions. Dictionaries should also have been considered as they provide faster look-ups on average.

Chapter 8

Discussion

In Chapter 5, we sat the scene for how we would answer the central question in this thesis: "Is virtual reality only a hype or real improvement for structural design?" SolidsVR was intended to be a product prototype combining FEA on solids, parametric modeling and virtual reality to find an answer to this question. Therefore, SolidsVR is discussed in Section 8.1. A discussion on whether the use of such an analysis tool in virtual reality may improve or not the structural design industry is carried out in Section 8.2.

8.1 Was SolidsVR Implemented Successfully?

When implementing SolidsVR, some simplifications were done. The load lumping and only meshing into one kind of element are examples of this. After comparing the results to Abaqus', it can be concluded that the accuracy of SolidsVR can be trusted. In each study case, the deformation and stress results converged correctly. For Case 1, the von Mises stress in the selected node only differs from Abaqus' by 0.32% when meshing into 125 elements. In Case 3, where the geometry was more advanced, a mesh with 2300 elements was needed to obtain a difference of 0.48%. The feature of removing elements was implemented in Case 4, and SolidsVR's selection of removed elements was defended with basic mechanics. However, the von Mises stress differed by 1.57% when meshing into 2160 elements. This indicates that a higher number of elements are needed when the geometry is more advanced. Fewer simplifications could have been made but were not prioritized as the accuracy was sufficient for this thesis.

SolidsVR is adapted to existing programs, and some expected limitations occurred as a result of this. The limitation with the most significant impact on our work was the restrictions Mindesk gave us. Mindesk only provides tools for creating geometry in a non-parametric environment, and are not connecting the changes made in VR back to Grasshopper. We looked at possibilities to develop our own connection but quickly decided that it was too comprehensive. Despite this, we created additional components in SolidsVR to get around this challenge. In addition to creating the geometry beforehand and only analyze it in VR, the user was now able to do it all in VR. The disadvantage by providing this interaction through Mindesk is that the user interface is not the most intuitive. In situations where it is natural to click on a surface to choose is, we had to use straight lines as sliders and adjust the length, which corresponds to a surface number. This might affect the user experience when testing the product prototype.

Keeping SolidsVR parametric even in VR is important as the new results may now be provided in near real-time. This advantage is understood better when comparing with a non-parametric program like Abaqus. In each study case, we compared the same model in SolidsVR and Abaqus. We experienced that every time we adjusted the mesh or the model it was more time-consuming in Abaqus than SolidsVR as you had to navigate back to the pre-processing stage to make the changes and manually execute a new analysis. In SolidsVR, all this was done automatically after changing one parameter.

Today, SolidsVR contains 24 components. We created the components we found necessary to exploit VR. Components placing load and boundary condition on parts of a surface could have been created, but it demands the same logic and coding skills as placing it on a full surface with **SurfaceLoad** and **SurfaceBC**, and the increased insight in terms of exploring VR by creating these components is limited. It is important to mention that it is always room for making changes to components in SolidsVR or even add new ones when additional features are needed. This includes changes to the VR interface as well.

The meshing of the input geometry is done in self-made components in SolidsVR. The creation of the **MeshHex** component, and the improvement of it resulting in **MeshSurface**, was one of the most time-consuming tasks during this thesis. With a retrospective look, even though the meshing was implemented successfully, other meshing possibilities should have been explored. Adapting to existing meshing tools might have been a better solution. Mesh generation has become a discipline of its own and a lot of research is done on the topic [44], and using existing algorithms and software to do this could be advantageous.

Concerning the time performance of SolidsVR, it runs fast enough when the meshing

is limited to some degree. From Section 7.2, the analysis result of a mesh with 3993 dofs (1000 Hex8 elements) is provided in less than 3 seconds on the computer used for profiling, and is clearly within our definition of near real-time. If a more detailed analysis is needed, faster calculations would be preferred. In the same section, it is found that it is the inverting of the stiffness matrix in **FEMSolver** that should be in focus when improving computation time. Improvement of the computation time could be done by following the improvements mentioned in Section 7.3. More importantly, as discussed in Section 2.2, computational power is expected to increase even more in the future, so this is a challenge to be overcome. This advantage will clearly be beneficial using VR as well.

8.2 Is Virtual Reality Only a Hype or Real Improvement for Structural Design?

Having a program analyzing geometry in virtual reality, we had the unique opportunity to explore VR in structural design. The benefit is clear: It is easier to visualize the deformations and stress in a 3D structure when the user works in VR. The user may move and walk around in their "virtual office" to look at it from different angles, and even from the inside. The need for additional computer screens a structural engineer might feel, will be eliminated working inside this virtual office. Also, the depth perception is better than on a computer screen, which makes it is easier to, for example, pick the correct point or corner with the VR controllers than with a mouse and a 2D screen.

Even though the interaction with a 3D model is easier in VR, our experience was that the user loses precision when creating geometry. Unlike in Rhino, moving the geometry could not be done by typing in the exact coordinates on the keyboard. As a counteract, combining customized components and an individualized Grasshopper script will help the user to design the correct geometry in VR. It is possible to do this by changing SolidsVR, as discussed in the previous section or make a completely new customized plug-in. Adding functionalities as snapping, or the ability to specify exact coordinates, will help the user to create accurate geometry. This is useful if precision is important, which is often the case for a structural engineer.

In other real-life projects, it could be adequate with approximate results to get an overall understanding of the behavior of a structure. Research performed in this thesis provides reasons to believe that the biggest improvements VR will bring to structural design, is when exploration and communication are in focus rather than precision and accuracy. The

misinterpretation of a structure may be reduced when analyzing and visualizing in VR. As proposed in Section 3.3, the communication barrier and misunderstandings between two disciplines or even two engineers, could decrease when collaborating in VR.

The advantage of the two-way communication between the user using SolidsVR and virtual reality is dependent on the presence of the parametric environment's characteristics in VR. This is essential for VR to be an enhanced tool in structural design. The model can be adjusted in VR and new results will be displayed without much lag to the user in VR.

The challenges regarding VR in the gaming industry in Section 3.2 can be recognized. However, the equipment cost and resources needed to adapt existing software to VR is not the biggest concern as the use of VR is aimed towards the structural design industry, not private use. If there are benefits for the industry – firm's resources will be set aside. The fact that it is rarely possible to use VR for a longer time period without experience motion sickness is a bigger concern. Even though there are expected ways to counteract VR motion sickness in the future, other solutions to this should be considered in the meantime. As of this day, software programs not requiring the user to stay in VR for more than 30 minutes would be beneficial.

The last thing to address is the similarities to earlier historical innovations. As introduced in Section 2.1, the Sketchpad's functionality is matched to the user's abilities and skills. Regarding structural design, experience with 3D modeling and a good sense of spatial awareness is preferred when starting to work in VR. As with many new technologies or advanced software programs, it is a challenge for the user to fully utilize its functions if not enough training is done. Research substantiates that differences in age, gender and even gaming experience affect people's task performance and user preferences in 3D [45]. It is not always about providing the most technologically advanced analyzer if the user does not manage to use it properly. We experienced at first hand how a lot of practice is needed to navigate properly in VR, and education and VR-training for employees predicted to use this tool should be a matter of course.

After discussing challenges and advantages with VR in light of our own experiences using SolidsVR, it is relevant to look into the future and the potential improvements we see in structural design if VR is adopted into the industry.

The advantage of the two-way communication in a parametric VR environment motivates to look at the combination of existing visualization tools and a parametric analysis program, such as SolidsVR. This will contribute to both detailed insight into the structure and more efficient communication, which again improves the end result. In Section 3.3, the

idea of walking inside a building in VR and analyze a part of a structure by clicking on it with a controller, is discussed. This is a concept that utilizes these advantages and is also realistic in the near future.

In the same section, artificial intelligence (AI) making live suggestions for changes to the structure, was discussed. This thesis gives no reason to discard this vision. When including other disciplines as AI in the discussion about VR, we imagine the number of opportunities to be close to unlimited.

Chapter 9

Conclusion

In this study, we examined the use of VR as a structural design tool. The goal of this study was to state whether VR is only a technological hype or if it actually could improve the structural design industry. We successfully created SolidsVR, a FEM program as a parametric Grasshopper plug-in, including components connecting the analysis to VR through Mindesk.

The accuracy of SolidsVR is concluded to be sufficient with an adequate number of elements in the mesh. Increasing this number of elements increases the computation time of the analysis. With a high number of elements, the computed result may take longer timer time than the desired near real-time, which is essential to work parametric in VR. As computational power is increasing, this is not considered as a concern.

The biggest limitation of SolidsVR was given by the existing programs. By adapting to Mindesk, we managed to examine SolidsVR's temporary user interface in VR. However, if VR is to be adopted into the industry, a customized connection must be created. A menu with all the needed structural features and the possibility of selecting nodes or parts of the geometry by clicking on it, is fundamental in a new and improved VR connection for structural design. Another limitation is the simplifications done when developing SolidsVR. This may have a smaller impact on the analysis results but did not affect our ability to examine the use of SolidsVR in VR to answer this thesis' question.

Given a two-way connection between the user and VR, it is concluded that VR could be an improvement for structural design. It is essential for the connection with VR to facilitate a parametric way of working. Without a parametric model, VR is just a visualization tool

with no possibility to make changes to the model inside the VR environment.

In VR, it is simple to understand the force distribution in the structure and where the stresses are concentrated. A greater understanding of the structure's behaviour makes it possible to improve the analysis by adjusting the pre-processing parameters. The increased insight could also help the user to optimize the structure.

Challenges were discovered during research and development, but none of which was big enough to undermine the benefits and possibilities with VR in structural design. VR is definitely a hyped technology with high expectations, but it is not *only* a hype. Based on the insight gained through this thesis, the combination of VR and a parametric environment appear to be promising as an analyzing and visualization tool in structural design. VR will provide great possibilities for improving structural design and merit further research.

Chapter 10

Future Research

There are many possible areas for future research on VR in structural design, most of which were too comprehensive to include in this thesis. This chapter highlights the research questions considered to be most relevant. Even though SolidsVR is a usable product given the terms in this thesis, some potential improvements for the future are listed in Section 10.1. Section 10.2 is introducing an alternative to Mindesk and its limitations.

10.1 Improve SolidsVR

SolidsVR was not intended to serve as an end product, but could still benefit from further development. Some of the points we would prioritize is listed below:

- Ability to handle more general geometry by developing more advanced meshing components. This could include the use of different element types, for example, tetrahedral elements and lead to other meshing methods. The use of existing meshing tools may be a solution to this.
- The time performance is always possible to improve, and suggested improvements are found in Section 7.3.
- Load lumping can be examined further. Instead of an approximate approach to this, where the area is assumed to be the same for all elements on the surface, the area

could be calculated and the load lumped accordingly. Gravity load or projection of the load could also be implemented for more load options.

- Possibility to apply load and boundary conditions to parts of a surface.
- A continuation of the last study case about the topology optimization could be interesting. A deeper mathematical understanding of optimizing should be achieved to make sufficient improvements to the input geometry.

10.2 Create a Virtual Reality Connection

As discussed, Mindesk limited the VR interface for SolidsVR. A topic for further work is to implement a self-made VR connection. This should be adapted to a parametric environment and the features needed in the structural design industry.

The most important feature is to automatically transfer new geometry drawn in VR to Grasshopper and the parametric environment. By focusing on structural design when creating a self-made VR connection, it could be implemented features adapted to analyze structures and interpret the results. A menu with structural components is a wanted feature inside VR. Also, not having to adapt to existing software, but having a direct connection between the analysis and the VR environment would be a significant improvement for using VR as a tool in structural design.

This requires programming skills but is doable as there exists open source framework as RhinoVR discussed in Section 3.5.

Bibliography

[1] SutherLand, Ivan Edward. *Sketchpad: A man-machine graphical communication system*. PhD thesis, Massachusetts Institute of Technology, 1963.

[2] Brady Peters and Terri Peters. *Computing the Environment: Digital Design Tools for Simulation and Visualisation of Sustainable Architecture*. John Wiley Sons, Ltd, United Kingdom, 2013.

[3] Parsons, David. *Object Oriented Programming with C++*. Continuum, Great Britain, 2nd edition, 1997.

[4] Fenesan, Oriana. BIM Implementation in Parametric Building Software. Bachelor's thesis, VIA University College Aarhus, 2014.

[5] M. Anson, J.M. Ko and E.S.S. Lam. *Advances in Building Technology*. Elsevier, 2002.

[6] Max Roser and Hannah Ritchie. Technological progress. *Our World in Data*, 2019. Retrieved on March 13th, 2019 from: `https://ourworldindata.org/technological-progress`.

[7] Ramesh Kumar, Priye Ranjan and Mr.Dipesh Das. Moore's law. *International Journal of Modern Trends in Engineering and Research*, 3(4):508–514, 2016.

[8] Robert McNeel Associates. Rhinoceros 6. Retrieved on February 16th, 2019 from: `https://www.rhino3d.com/`, 2019.

[9] Scott Davidson. Grasshopper. Retrieved on February 4th, 2019 from: `https://www.grasshopper3d.com/`, 2019.

[10] Matthew Revell. What Is Visual Programming? Retrieved on March 12th, 2019 from: `https://www.outsystems.com/blog/what-is-visual-programming.html`, 2017.

[11] Dassault Systèmes. Abaqus Unified FEA. Brochure retrieved on May 19th, 2019 from: http://www.feaservices.net/abaqus-brochure-2014.pdf, 2014.

[12] Microsoft. Visual studio. Retrieved on February 5th, 2019 from: https://visualstudio.microsoft.com/vs/, 2019.

[13] Inc. GitHub. Github. Retrieved on February 5th, 2019 from: https://github.com/, 2019.

[14] History Of Virtual Reality. Retrieved on February 17th, 2019 from: https://www.vrs.org.uk/virtual-reality/history.html, 2017.

[15] Seth Giddings. *Gameworlds: Virtual Media and Children's Everyday Play*. Bloomsbury Academic, 2014.

[16] Stuart Dredge. Facebook closes its $2bn Oculus Rift acquisition. What next? Retrieved on May 20th, 2019 from: https://www.theguardian.com/technology/2014/jul/22/facebook-oculus-rift-acquisition-virtual-reality, 2014.

[17] Hanna Levy. How VR is helping transform the way we eat and drink. Retrieved on May 19th, 2019 from: https://www.builtinla.com/2019/01/08/la-vr-industry-trends-2019, 2019.

[18] Mark Mann. The real cost of virtual reality. Retrieved on February 26th, 2019 from: https://www.cnet.com/news/the-real-cost-of-virtual-reality/, 2017.

[19] Kevin Parrish. Pricing and lack of content are still barriers against the adoption of VR. Retrieved on May 2nd, 2019 from: https://www.digitaltrends.com/computing/vr-pros-see-pricing-and-content-as-mainstream-barriers/, 2018.

[20] Alex Hern. I tried to work all day in a VR headset and it was horrible. Retrieved on May 19th, 2019 from: https://www.theguardian.com/technology/2017/jan/05/i-tried-to-work-all-day-in-a-vr-headset-so-you-never-have-to, 2017.

[21] Inc. MarketWatch. Virtual and Augmented Reality Market is Expected to Exceed US$ 117 Billion by 2022. Retrieved on March 7th, 2019 from: https://www.marketwatch.com/press-release/virtual-and-augmented-reality-market-is-expected-to-exceed-us-117-billion-by-2022-2018-07-26, 2018.

[22] Avantis Systems Ltd. Virtual & Augmented Reality in Further & Higher Education. Retrieved on April 10th, 2019 from: https://www.classvr.com/virtual-reality-in-education/vr-university-higher-education/, 2017.

[23] Osso VR - Virtual Reality Surgical Training Platform. Retrieved on May 10th, 2019 from: https://ossovr.com/, 2019.

[24] Louisa Kendal. VR has the potential to revolutionise universities – here's how. Retrieved on March 10th, 2019 from: https://www.studyinternational.com/news/vr-future-revolutionise-universities/, 2018.

[25] Andreas Rivera. How Virtual Reality Is Changing Construction. Retrieved on May 19th, 2019 from: https://www.business.com/articles/how-virtual-reality-is-changing-construction/o, 2018.

[26] Josef Wolfartsberger. Analyzing the Potential of Virtual Reality for Engineering Design Review. *Automation in Construction*, 104(1):27–37, 2019.

[27] Niklas Plikk. VR-revolusjonen begynner nå, med Oculus Quest i spissen. Retrieved on May 29th, 2019 from: https://www.tek.no/artikler/test-oculus-quest/465382, 2019.

[28] Univercity of Waterloo. Virtual reality motion sickness may be predicted and counteracted. Retrieved on May 26th, 2019 from: https://www.eurekalert.org/pub_releases/2018-09/uow-vrm092518.php, 2018.

[29] Mengli Yu, Ronggang Zu, Huiwen Wang and Weihua Zhao. An Evaluation for VR Glasses System User Experience - The Influence Factors of Interactive Operation and Motion Sickness. *Applied Ergonomics*, 74(1):206–213, 2019.

[30] Spacemaker. Spacemaker. Retrieved on May 25th, 2019 from: https://spacemaker.ai/, 2019.

[31] LLC. Facebook Technologies. Oculus. Retrieved on May 29th, 2019 from: https://www.oculus.com/, 2019.

[32] AR/VR for Rhino UGM – September 2018. Retrieved on March 7th, 2019 from: https://www.rhino3d.co.uk/ar-vr/ar-vr-for-rhino-ugm-september-2018/, 2018.

[33] McNeel. RhinoVR - a virtual reality sample plug-in for Rhino WIP. Retrieved on February 6th, 2019 from: https://github.com/mcneel/RhinoVR, 2019.

[34] Mindesk Inc. Mindesk. Retrieved on February 20th, 2019 from: https://mindeskvr.com/, 2019.

[35] Kolbein Bell. *An engineering approach to FINITE ELEMENT ANALYSIS of linear structural mechanics problems*. Fagbokforlaget, Norway, 2013.

[36] Robert D. Cook. *Finite Element Modeling for Stress Analysis*. John Wiley Sons Inc., United States of America, 1994.

[37] Klaus-Jürgen Bathe. *Finite Element Procedures*. Prentice Hall, Pearson Education, Inc., United States of America, 2006.

[38] Microsoft. Attention Spans. Brochure retrieved on May 20th, 2019 from: https://www.scribd.com/document/265348695/Microsoft-Attention-Spans-Research-Report#download, 2015.

[39] Agile Alliance. What is Agile Software Development? Retrieved on March 1st, 2019 from: https://www.agilealliance.org/agile101/, 2019.

[40] Mike Cohn. Agile Needs to Be Both Iterative and Incremental. Retrieved on March 10th, 2019 from: https://www.mountaingoatsoftware.com/blog/agile-needs-to-be-both-iterative-and-incremental, 2014.

[41] J. Kingman, KD. Tsavdaridis and VV. Toropov. *Applications of topology optimization in structural engineering*. The University of Leeds, 2014.

[42] LL. Beghini, A. Beghini, N. Katz, WF. Baker and GH. Paulino. *Connecting architecture and engineering through structural topology optimization*. Elsevier, 2013.

[43] Marcus Cuda Christoph Ruegg and Jurgen Van Gael. Math.net numerics. Retrieved on February 5th, 2019 from: https://numerics.mathdotnet.com/, 2019.

[44] Inc Itasca Consulting Group. Meshing Solutions. Retrieved on March 5th, 2019 from: https://www.itascacg.com/software/products/meshing-solutions/mesh-generation, 2019.

[45] Zudilova-Seinstra, E., van Schooten, B., Suinesiaputra, A. et al. Exploring individual user differences in the 2D/3D interaction with medical image data. *Virtual Reality*, 14(2):105–118, 2010.

Appendix

A ZIP file

The ZIP file, *Files_AkselsenStenvold.zip*, contains three folders. One folder, *Grasshopper Plug-In*, includes the files needed to add SolidsVR to Grasshopper as a plug-in. The second folder, *RGH*, is the Rhino and Grasshopper files used in the study cases. The last folder contains the code used to develop SolidsVR, called *The Code*.

Grasshopper Plug-In

The folder *Grasshopper Plug-In* contains one folder with all the needed files for implementing SolidsVR as a plug-in to Grasshopper. This includes the packages used and the compiled filed created by the code. The plug-in is added by dragging this folder to the component folder for Grasshopper.

RGH

RGH contains four folders with the Rhino and Grasshopper files used in Case 1 to Case 4 with the same name, and also a folder called *Case 0*.

Case 0 contains two files. A clean Rhino-file with no geometry drawn but lines acting as sliders in VR and spheres for other features. A prepared Grasshopper file which is connected with the Rhino-file and to Mindesk, but does not contain any information about the geometry as this has not been drawn. The purpose of this folders is to use it as a basis for creating new cases.

The Code

The Code contains one folder called *SolidsVR* which is the project in Visual Studio. SolidsVR contains multiple folders including the icons which is shown in Grasshopper, packages used in the code, and the code which creates the Grasshopper plug-in. The code can be divided into two main parts, the components and the classes. Information about the components and classes are given in Table A.1 and Table A.2, respectively.

Component	Description
CreateMaterial	Creates a Material class with the information about Young's modulus, Poisson's ratio and the yield stress.
CrossSection	Shows information about the cross section in VR.
DeconstructElement	Deconstruct Element-objects and gives all the information about the object.
DeconstructMesh	Deconstruct Mesh-objects and gives all the information about the object.
DeconstructNode	Deconstruct Node-objects and gives all the information about the object.
DivisionSlider	Slider in VR for mesh-division in u-, v- and w-direction.
FEMSolver	Responsible for the analysis of the model.
HeadLine	Creates headlines for meshing, boundary conditions and loads in VR.
LoadSlider	Slider in VR for setting the values of the load vector.
MeshCurve	Meshing component developed in Case 2. Handles curved geometry with straight edges cross section.
MeshHex	Meshing component developed in Case 1. Handles geometry with only straight edges.
MeshSurface	Meshing component developed in Case 3. Handles curved geometry and curved cross section.
NodeInfo	Displays information of a selected node in VR.
PointsToGeometry	Creates a geometry from eight corner points.
PreDisp	Assigns a prescribed displacement to chosen nodes.
PreDispSlider	Slider in VR for setting the values of the prescribed displacement vector.
ScaleSlider	Slider in VR for scaling the visualization of the deformation of the structure.
StressDirectionSlider	Slider in VR for choosing which stress direction to be visualized.
SurfaceBC	Assigns boundary conditions to nodes on chosen surface.
SurfaceLoad	Assign loads to nodes on chosen surface.
ViewDeformations	Creates the deformation area in VR.
ViewStresses	Creates the stress analysis area in VR.
VolumePercentageSlider	Slider in VR for choosing percentage of volume to be removed when optimizing the geometry.

Table A.1: Description of all components in the ZIP file.

Classes	Description
BrepGeometry	Information about the geometry of the model.
CMatrix	Contains the elasticity matrix.
Element	Information about the elements in the mesh.
Material	Information about the material used in the model.
MeshGeometry	Information about the mesh for the model.
Node	Information about the nodes in the mesh.
StiffnessMatrix	Calculates the element stiffness matrix

Table A.2: Description of all classes in the ZIP file.

All files in the ZIP file is also found in the open repository on GitHub:

`https://github.com/synnestenvold/thesis`

B Video

A video, *Video_AkselsenStenvold.mp4*, is attached showing the VR user interface of SolidsVR.

The video is also available on CSDG NTNU's YouTube account:

`https://youtu.be/a7noK6irpeY`