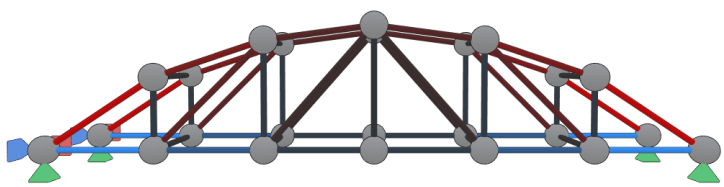
LUND UNIVERSITY



INTERACTIVE STRUCTURAL ANALYSIS FOR THE CONCEPTUAL DESIGN PHASE

MATTIAS LILJA

Structural Mechanics

Master's Dissertation

DEPARTMENT OF CONSTRUCTION SCIENCES
DIVISION OF STRUCTURAL MECHANICS

ISRN LUTVDG/TVSM--16/5209--SE (1-55) | ISSN 0281-6679
MASTER'S DISSERTATION

INTERACTIVE STRUCTURAL ANALYSIS FOR THE CONCEPTUAL DESIGN PHASE

MATTIAS LILJA

Supervisor: **DANIEL ÅKESSON**, Licentiate in Engineering, Div. of Structural Mechanics, LTH.
Examiner: **JONAS LINDEMANN**, PhD, Div. of Structural Mechanics, LTH & Lunarc, Lund.

Printed by Media-Tryck LU, Lund, Sweden, June 2016 *(Pl)*.

For information, address:
Division of Structural Mechanics,
Faculty of Engineering LTH, Lund University, Box 118, SE-221 00 Lund, Sweden.
Homepage: www.byggmek.lth.se

Interactive Structural Analysis for the Conceptual Design Phase

by

Mattias Lilja

Abstract

The importance of structural demands is usually overlooked in the conceptual design phase. Architects commonly conceive the geometry of a structure without much involvement of an engineer or with regards to structural demands. This is in part because a lack of tools available to engineers in this phase. The common paradigm for structural analysis software interface badly with the iterative and chaotic nature of conceptual design. Developing tools that enables structural demands to be a more prominent part of conceptual design should hypothetically result in overall better performing designs leaving the conceptual phase.

This thesis investigates the requirements for, and the development of, a simple structural design and analysis application using a more direct interaction model adapted to the conceptual design phase. Emphasis is on creating a user experience that incite design exploration by developing a suitable user interaction model and a design comparison tool. A game engine was used to aid the development of an interactive and engaging environment.

Feedback and observation of users testing the application indicated that a more direct interaction model can enhance user engagement as well as proneness to design exploration, strengthening similar results in previous work done in this field. Feedback also indicated that a developed tool, aimed at design comparison, became poorly used. Testers argued that the tool was potentially powerful, but needs a more integrated implementation than the one used in this application, which did not engage users enough.

The developed application is called StructSTUDIO and is available online at structarch.org/ids/.

Key words; conceptual structural design, direct manipulation, interactive structural analysis, structural design comparison

Interaktiv Strukturanalys i den Konceptuella Designfasen

av

Mattias Lilja

Sammanfattning

Vikten av att beakta strukturella krav redan i den konceptualla designfasen förbises ofta trots den stora inverkan tidiga designval har på efterföljande designbeslut. Detta beror delvis på att många strukturanalysverktyg är dåligt anpassade till den konceptuella designfasen.

I detta arbete utvecklades och undersöktes vilka krav som ställs på ett enklare strukturanalysprogram med en mer direkt interaktionsmodell anpassad till den konceptuella designfasen. Den interaktiva miljön möjliggges av en spelmotor och är en möjlighet att frångå den stela och oengagerande upplevelsen i vanligt förekommande strukturanalysprogram. En mer interaktiv användarmiljö kan bidra till att användare engageras och skapa incitament till större utforskning av designalternativ.

Resultatet är ett program där användartester indikerade på att mer direkta interaktionsmodeller och användarupplevelser engagerar användare, vilket stärker tidigare indikationer från andra arbeten inom området. Arbetet producerade även ett verktyg kallat *Snapshots*, ett sorts fotografi av ett mekaniskt system. Med verktyget kan designalternativ jämföras mot varandra med hjälp av en indexerad prestandaparameter. Verktyget ansågs som ett potentiellt starkt verktyg men implementationen i det producerade programmet behöver förbättras och bli en mer integrerd del i designprocessen.

Arbetet har bidragit med insikt i hur strukturdesign- och analysprogram kan utformas för att bättre anpassas till den konceptuella designfasen. Tanken är att verktyg som detta skall hjälpa till att bidra till utvecklingen av nya konceptuella analysverktyg med påföljden att bättre presterande lösningar lämnar den konceptuella fasen.

Programmet heter StructSTUDIO och finns tillgängligt online på structarch.org/ids/.

Nyckelord; konceptuell strukturdesign, direkta interaktionsmodeller, interaktiv strukturanalys, jämförelse av mekaniska system

Preface

This master thesis was carried out at the Division of Structural Mechanics at Lunds institute of technology. I want to thank Dr. Jonas Lindemann and Lic. Eng. Daniel Åkesson for the opportunity to do this work.

I would like to especially thank my supervisor Lic. Eng. Daniel Åkesson for the outstanding guidance and availability during the time of this work as well suggesting it. Additionally, thank you for the opportunity to visit the Smartgeometry conference, which inspired me.

Lund, June 2016
Mattias Lilja

Contents

Chapter 1

Introduction

1.1 Background

Structural design can be recognised as a process where the aim is to design a structure that fulfils an intended functionality with respect to structural demands. Design work is usually an iterative process [1]. The iterative nature of the process is especially noticeable in the conceptual design phase. Generally, a structural engineers daily work can be categorized as [2]:

Conceiving: Considered as the most important design step where major or significant structural details or concepts are conceived.

Modelling: Part of which a simplified model of the conceived conceptual structure is created for analytical purposes.

Dimensioning: Establishing dimensions derived from the choice and combination of materials.

Detailing: Fine detailing of connections and the production of construction documentation.

Design work can move backwards and forwards between the different categories. In practice, there is no distinct incident where a team of designers move from one category to another. The process to a final design manifest itself differently between projects. Among the many differences is which design tools that are used. Design tools can be anything that add value or information to the decision process. Used extensively is structural design and analysis tools. Such software is commonly designed for the *late modelling*, *dimensioning* and *detailing* steps. There are few alternatives for the conceptual design phase.

This thesis emphasis is on the *conceptual design phase*. It is the earliest phase of the structural design process, attributing to the *conceiving* and *early modelling* steps.

The importance of structural design is usually overlooked in the conceptual design phase [2]. In part, this is due to common structural design and analysis alternatives interface badly with the iterative nature of conceptual design. A problem is the fuzzy nature of the process. Lack of defined problem parameters and the precise input required for proper analysis complicates matters. Additionally, the time and effort needed to properly set up a model makes structural analysis of early prototypes potentially unrewarding. This is a contributing factor to why structural analysis is usually preformed later in the process [3]. It is problematic as subsequent design choices naturally have dependencies on previous. Design choices leaving the conceptual design phase may see little change over time. Impact of design decisions are initially high but decline as a design matures, further emphasizing the importance of accounting structural demands early on, see Figure 1.1.

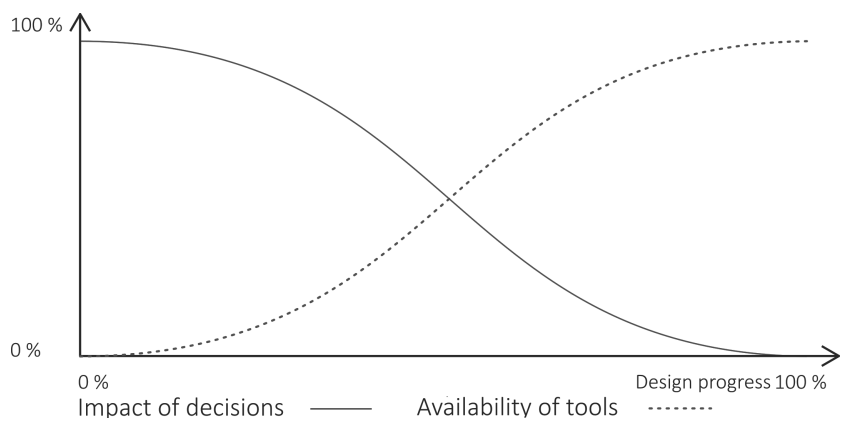


Figure 1.1: Generalisation of the impact of decisions and availability of tools during the design process [4].

The gemoetry of a structure is often conceived by architects. Commonly without much involvement of an engineer or with regards to structural demands. Little regard to structural demands allows for greater *design exploration*, in contrast to egineering work where structural demands can restrict design exploration. Architects have many digital modelling and visualizing tools, available for different stages of their corresponding design process. These tools seldom include functionality for structure analysis to verify the geometry [5].

Hypothetically, making structural demands a more prominent part of conceptual design can lead designers to obtain an earlier understanding of the consequences of their design choices with regard to structural performance. Subsequently, this leads to more informed decisions. This benefits the design process, resulting in overall better preforming designs.

In an investigation by M. Fröderberg and R. Crocetti [3], a number of practising engineers were tasked with a conceptual design task. It indicated that premature use of advanced analysis tools could negatively affect designers proneness to search for different, better preforming alternative structural designs. If possible, tools should allow for or incite users toward design exploration.

The rigid nature of structural design can be a hinderance towards the implementation of functionallity with the versatility needed to enable design exploration. Attributing to the rigid nature of structural analysis software is in part due to its interaction model. This interaction model have not changed much since the introduction of such software, but work have been done where the interaction model was made more intuitive [6–9]. The interaction models in [6] and [7] positively affected users proneness towards design exploration as well as being able to follow the iterative nature of the conceptual design phase. Additionally, even with simple interaction models in comparison to more heavyweight structural analysis software, it was still possible for these applications to give users constructive and meaningful feedback in regards to structural response.

1.1.1 Interaction model

The common interaction with structural analysis software can be summarized in a cyclic 3-step interaction model [10], see Figure 1.2. This 3-step interaction model is a characteristic which have not changed much since the introduction of such software. The majority of structure analysis software still use it today. When making more sophisticated models later in the design process, the steps are usually more distinctive.

Some of the difficulties conventional structure analysis applications face, if used in the conceptual design phase, is due to ill adapted interaction models. Jim Nieters defines an interaction model as *'a design model that binds an application together in a way that supports the conceptual models of its target users'* [11]. Users will over time create an *internal* image of the application. It makes actions predictable. Failure by the application to accurately follow the internalized image of a user can inaugurate a bad user experience.

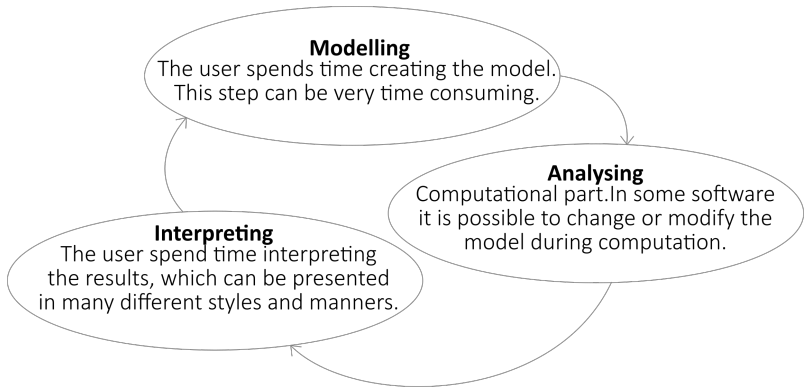


Figure 1.2: Interaction steps of conventional structural analysis software.

A metaphorical example are *Norman Doors* [12]. Doors are simple things which should not require a manual to use. Yet we encounter doors all the time which do not work the way we think they do. Such could be when there is a handle attached, indicating that the door is to be pulled, but the door does not open because it is a push door. In this case, the design did not conform to the internalized interaction model amongst users, causing irritation and confusion.

Like Norman doors, ill designed *user interfaces* are everywhere. The user interface of CAD tools is often their weakest link and least developed feature [13]. Features and functions can be buried deep within menus and toolbars [6]. A good interface allows users to discover and understand the interaction model of the application. Positive user experiences increase the incitement for further or more extensive use of an application. Good user experience design is seen as paramount by businesses and is considered vital to product success [14]. In [7] it is briefly mentioned how important it is to strive for consistency between applications to create a sense of familiarity.

1.2 Problem statement

Common structural analysis software fails to meet some of the demands of the conceptual design phase. Few tools are available which allows for structural analysis in the conceptual design phase. This attributes as a contributing factor to why structural demands are not incorporated into early design.

When using structural analysis software, it is necessary to create a digital model which can be analysed. Such software tend to focus on analytical capability over geometrical modelling tools. It can be difficult to produce complicated shapes and variations of these [13]. This can limit the expressive freedom needed in conceptual design, especially from the perspective of an architect. A way to effectively explore and compare digital models is also absent.

1.3 Aim of work

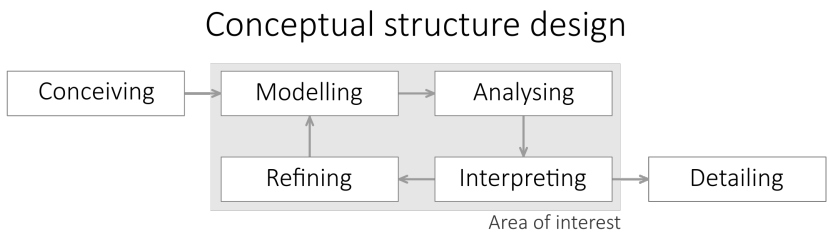


Figure 1.3: The cyclic design process this work aims to improve.

This work aim to improve the conceptual structure design process, see Figure 1.3. The aim is to:

- improve the conceptual structure design process as a whole. Should result in better preforming design leaving the conceptual design phase.
- enhance the interaction model between the modelling and analysing step. Should result in a more effective way of working with digital prototypes.
- establish a way that incite users to explore and compare designs. Should result in more informed design decisions with regard to structure demands in early design.

1.4 Approach and limitations

This work is conducted as a software development project. The goal is to create an application which implements an interaction model better adapted to the conceptual design phase. Developing intuitive and interactive tools is emphasised. In previous work, where tools were developed with more direct interaction models, yielded positive results. It resulted in an almost *playful* experience, improving the human computer interaction. A higher degree of

interactivity, such as real-time feedback, was considered positive aspects of developed applications. Implementing real-time feedback and direct interaction models can result in the user being more engaged in the design task [7, 8]

Which tools are used in design impact the final results [1]. A possible approach is to use a game engine as a development platform. It is a novel approach, but it is believed that its inherent emphasis on interactive content can positively affect the development towards creating a more interactive environment and more direct interaction models. However, using a game engine for non-game related development can potentially be a hindrance.

The game engine Unity was used in this work. This because it has built-in support for many different input devices as well as a compiler which can compile applications to many different platforms. A goal was to deploy the application online using WebGL *(Web Graphics Library)*. The benefits of developing applications for the web is the wide availability it entails.

This work focuses on the user interaction with the software. Emphasis is on the user experience and how it can be improved to incite design exploration - which entails a positive effect placed onto the conceptual structure design process.

Chapter 2

Existing conceptual design tools

2.1 pointSketch

The pointSketch name was derived from the idea *Fixed points in space* concept by Pierre Olsson [15]. The concept consists of creating points in space and later connect them together. With a very simple implementation of different modelling tools, simple two dimensional systems can relatively easily be *sketched*.

The application differentiates the modelling and analysing steps by using different application states. It enables the application to communicate with users and hint them what they are supposed to do. Only the tools relevant to the current task are displayed. This makes tools contextual. It eliminates unnecessary information to be displayed, attributing to a better user experience. A cluttered design can cause confusion [16].

The 3D version of pointSketch is less developed and in a beta stage. It demonstrates the pointSketch idea in 3D. The interaction model for the 3D version have the user move *painting planes* around with a set of buttons. Nodes can be placed onto these. The use of different working planes or the idea of setting up a workspace-grid is common in *Finite Element Analysis* software. From the perspective of conceptual design, it can be an ineffective solution as it requires many steps to complete a relatively simple task. Although not as developed as its two dimensional counterpart, it presents one solution to analyse simple three dimensional models.

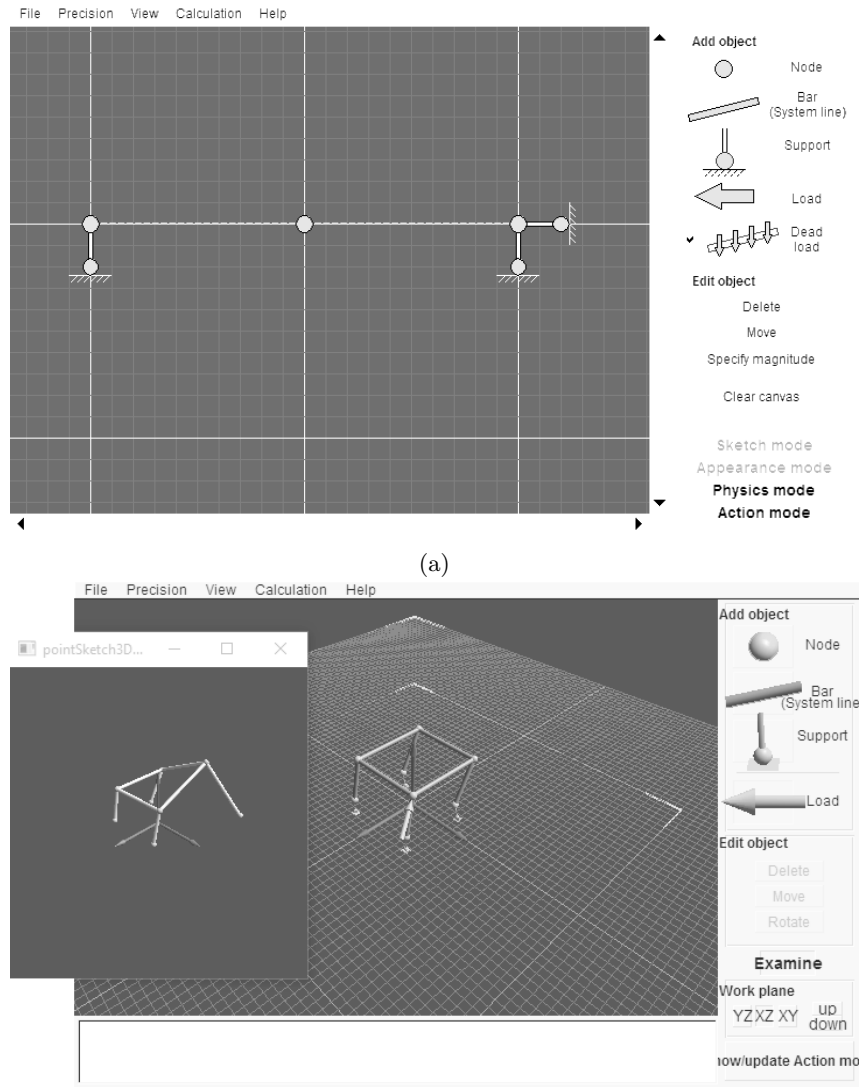


Figure 2.1: (a) The GUI of pointSketch2D [17]. (b) The GUI of pointSketch3D (beta) [18].

2.2 Sketch a Frame

Sketch a Frame is a complete design tool developed for tablets, created by Daniel Åkesson [19]. The possible advantages of a more direct interaction model using touch devices lead to the development of Sketch a frame. It implements the pointSketch idea and is very easy to use due to its very direct interaction model. Sketch a Frame conforms to touch-based application guidelines and behave in an expected predictable manner. The loss of accuracy compared to a mouse is combated with a snap functionality.

The level of interactivity is high. Real-time results are displayed on the same screen as the active model. The ease of creating, editing and interpreting feedback yields an almost *playful* experience. A temptation to move nodes to see the different responses is present, an experience which incites design exploration. The application allows for quick and direct drawing of two dimensional systems. Additionally, it does not use any numerical values. In some respect, actual values can be of little interest in conceptual design. The system response, such as deformation or rigid body motions, is clearly visualized and understood by the user.

The application can yield an intuitive understanding of a systems response in the conceptual design phase. The direct manipulation and direct touch interface yields a feeling of strong manipulative control - enabling users to explore designs more freely and encourages design exploration [7].

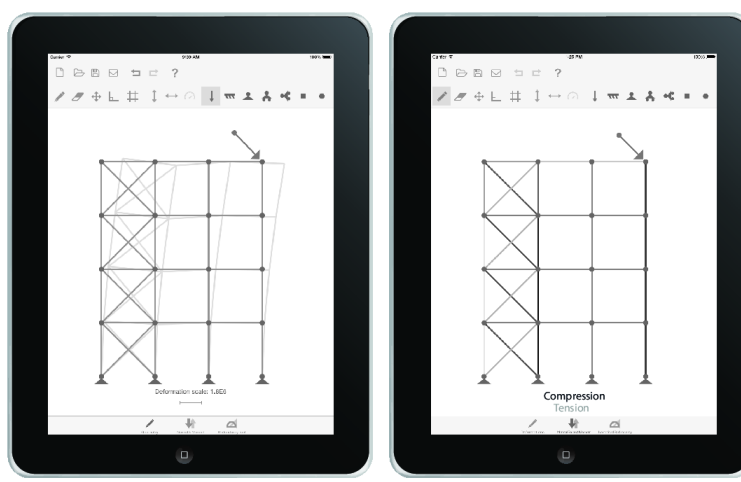


Figure 2.2: Sketch a Frame.

2.3 StructureFIT

StructureFIT emphasis on design exploration using a guidance approach [20, 21]. The application is created by Caitlin Mueller and uses genetic algorithms to present the user with well performing alternative design suggestions. The user selects a design of interest, evolve it, and receives a new set of designs. The process is then repeated.

The software enables the ability to define nodal positions, boundary conditions and other relationships between structural members. It can be done in a live editing window or a tabular option. It is also possible to model a system in a free-form style and play with the evolutionary parameters to yield different design. The different designs are presented side by side indexed by performance against the base design with a decimal fraction. A designer is thus able to compare designs using a single value. The visual representation of the systems helps create an intuitive understanding of how different design alternatives affect system performance.

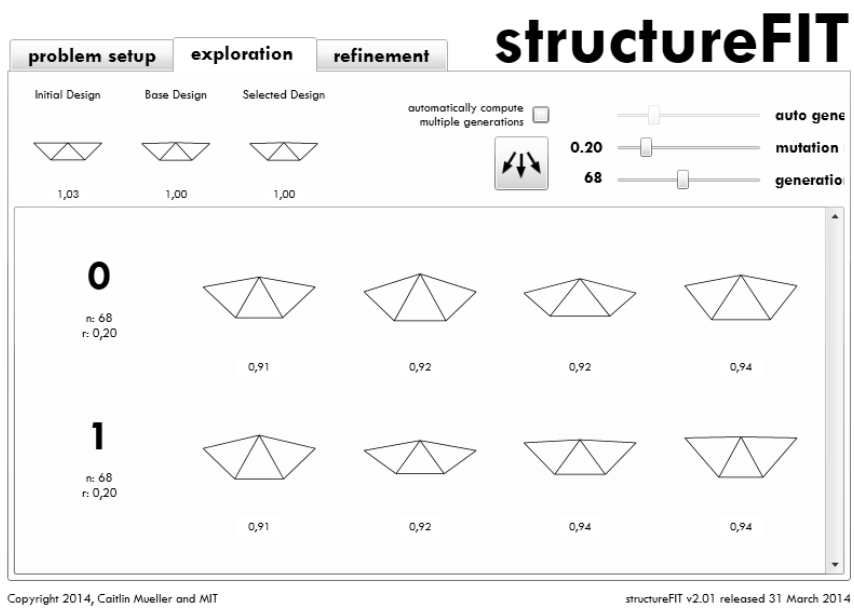


Figure 2.3: Design exploration using genetic algorithms.

2.4 Arcade

Arcade [22], created by Kirk Martini, took a different approach to structure analysis software. The most interesting feature is the use of a physics engine as a computational method [10]. The application presents a familiar 2D modelling space and is relatively easy to use. Modelling tools offers only limited functionality. The flow and playfulness found in Sketch a Frame is not found here. There is little to no incitement to alter a model, as it can be difficult to set it up properly.

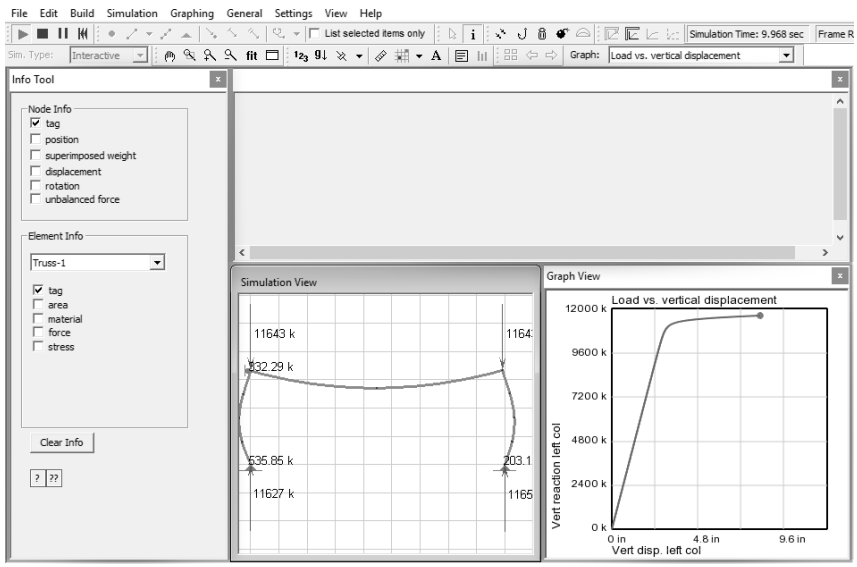


Figure 2.4: Arcade use several windows within a window design for the GUI.

Chapter 3

The Unity game engine

3.1 Environment

Unity is a *game engine*, a software framework. Game engines includes all the basic necessities a developer need to develop a video game, such as a rendering engine enabling graphics and a physics engine enabling physics. Unity supplies a *integrated development environment* in which the developed application can be executed together with debugging tools [23].

A game engines inherit focus on interactive content can be seen as a hindrance to non-game related software development. Its emphasis on interactive content could aid in moving away from the generally stiff and rigid nature of common structure analysis software. Choosing Unity is in part because Unity have its own integrated development environment. This simplifies the development process.

Using a game engine as a development environment for structural analysis software is a novel approach. It is not the first time Unity will be used to create non-game related software. It has previously been used to create other non-game related applications [24]. The architectural visualization application Nuovo [25] and land management application INSIGHT [26] are good examples of non-game related applications using Unity.

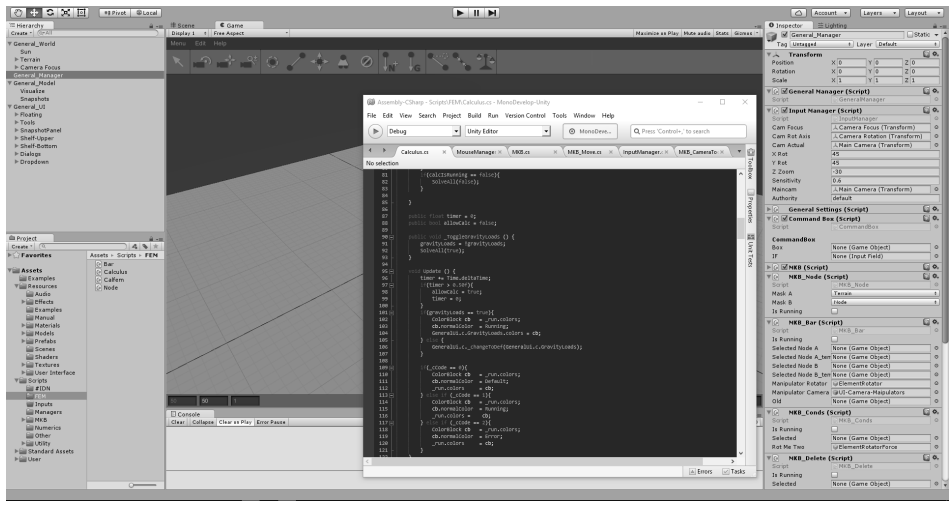


Figure 3.1: The integrated development environment supplied with Unity.

One advantage of using Unity is the ability to create applications with a high degree of interactivity while providing a good amount of control for developers. Unity come with some ready-made assets, such as panels and buttons for the user interface or different rendering components allowing meshes to be rendered with built-in or custom made shaders for enhanced visual fidelity.

Of particular interest is the built in real-time physics engine. K. Martini used a physics engine as a computational method when developing Arcade. Using a physics engine have several positive consequences. One such is the ability to work with unstable structures and large displacements. Additionally, non-linear behaviour is yielded practically for free [27].

3.1.1 C#, Mono and .Net

Unity implements Mono which is an open source UNIX implementation of the .Net framework. This enables access to most of the .Net API, limited primarily by the support of the currently implemented version of Mono [28,29]. It is possible to use common libraries such as system Input-Output, Collections or Diagnostics. There are compatibility issues, but most general purpose libraries can be used. Additionally, the Mono complier is very versatile, allowing for deployment to many different platforms [30].

3.1.2 MonoBehaviour

MonoBehaviour is a base class which scripts in Unity derive from and provides the basic interface to the Unity engine [31]. The class enables functions such as Update(), Start() or Awake(), called upon by the engine at different stages of execution.

The MonoBehaviour base class is not serializable, meaning it is not possible to serialize it to a stream of bytes and cast it back to an object type successfully. It causes problems with persistence as objects interfacing with Unity have to inherit from this base class. Objects such as *Nodes* and *Elements* will need secondary purposely created serializable classes for persistence as well for threaded operations, as the MonoBehaviour base class is in addition not thread-safe.

3.1.3 External libraries

Unitys implementation of C# does not provide a library that supports numerical matrix operations. An open source initiative, Math.Net, develops and maintain a .Net toolkit [32] enabling most common matrix operations. The *Numerics* library licence is MIT/X11, which means there is no restriction to edit or modify the library [33]. Additionally, it does not restrict the distribution of the software, as is or implemented. Numerical operations were implemented with, and enabled by, Math.NET Numerics.

3.2 Built-in physics

The Unity game engine use PhysX as a computational method for its physics engine [34]. PhysX was originally created by AGEIA technologies but is currently owned and maintained by NVIDIA [35].

Unitys implementation of PhysX elevates the API interface to higher level C#, linked through the MonoBehaviour class. It enables real-time computations and results, but the method is limited to time-step simulation. The idea was to use this as an approach for computation in the developed application. Core access to the physics engine is not possible and the software can only be used as is.

The developers of Unity provide developers with a number of different object-physical components. Among these are springs, joints, rigidbodies, collides and much more. The approach in the developed application was to use them to enable direct computation and real-time results. Early testing indicated, despite the low level of access to the physics engine, that the usage of the

integrated Physics engine was possible. However, it was eventually dismissed for a number of reasons, see section 4.4.

Chapter 4

Present work - Developing StructSTUDIO

4.1 Application interaction

4.1.1 Interactive modelling tools

An obstacle to developing a more direct interaction model is to find a good method that enables users to work effortlessly in three dimensions. A proposed approach by D. Åkesson was to use the *head mounted display* Oculus Rift together with the Leap Motion controller. The Leap Motion controller is a hand tracking device which D. Åkesson has successfully used in a previous project [8].

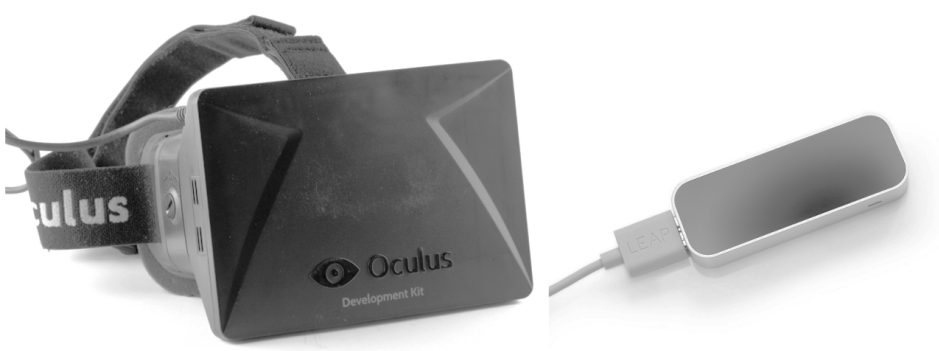


Figure 4.1: Left: The Oculus Rift, Right: The Leap Motion controller

An important difference between how the Leap Motion device was implemented in this work versus that developed in [8], was which tracking mode was used. The device have two modes, front hand and back hand tracking, see Figure 4.2. Åkesson utilized the front hand tracking mode and this work used the back hand tracking mode.

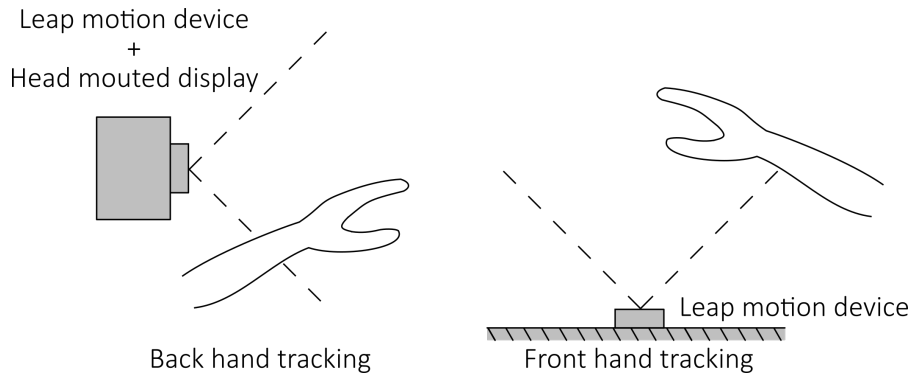


Figure 4.2: Back hand tracking vs. front hand tracking as in [8]

The Leap Motions back hand tracking algorithm could not accurately track hands from their back with the firmware version of the time. This resulted in a behaviour where *virtual hands* inside the game environment could disappear off screen or break completely. Gesture recognition was poor, and critical gestures such as grip and pinch only executed a fraction of the time. It made it difficult to develop good geometric modelling tools. The tracking algorithm have since been improved significantly [36].

Due to the technical difficulties with the Leap Motion controller, the development project opted to revert back to a mouse and a keyboard as *human interface devices*. It presented a different challenge. As mentioned in previous chapters, modelling tools in common structural analysis software is not agile enough to sufficiently enable users the control they need in conceptual design. A different approach is needed.

A proposed interaction model for the modelling step is that of common three dimensional modelling software. The vertexes of a mesh can represent the nodes and the edges the elements, refer to Figure 4.3. Three dimensional modelling tools have been in development for a long time and the interaction model have been refined over decades. The model does additionally have established ways of navigating a scene using camera tools.

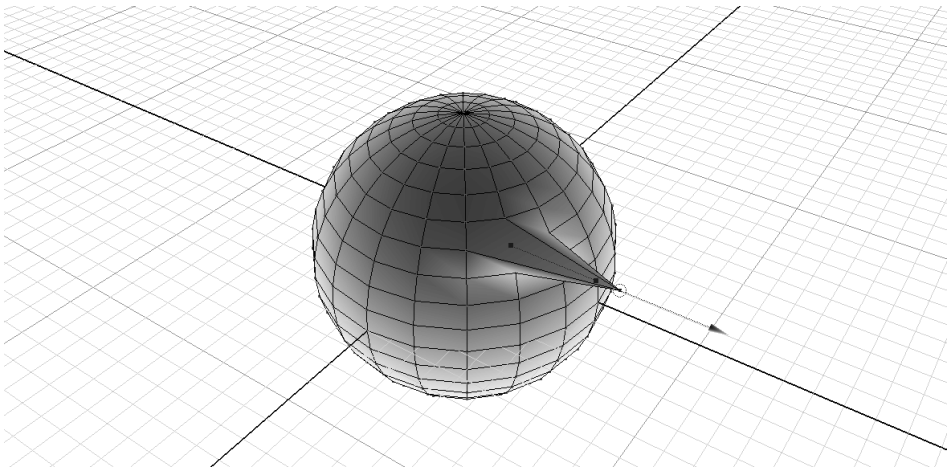


Figure 4.3: Vertex being moved in a 3D modelling software using a manipulator-tool.

The paradigm enables good geometrical modelling tools, but software using the common three dimensional modelling tool paradigm tend to be shortcut heavy. To effectively use these applications, a user must learn many key shortcuts or commands. It is not uncommon that shortcut heavy programs can feel a bit daunting to new users. It is important that the *user interface design* can explain what each tool does and how it works to speed up the learning process.

Using the common three dimensional tool paradigm goes well with research suggesting that such more direct implementations of the human-computer interaction model further engage users interaction with a software as well encourages exploration [9]. To promote users to explore and create alternative designs, the process of creating or modifying a 3D-model was developed in such a way that it does not incite fear of altering a model. Exploring design should be rewarding and fun, like Sketch a Frame.

4.1.2 Structural analysis

Results shall if possible be presented in real or near real-time. Real-time feedback can create a temptation to alter the design to *see what happens next*. As prior explained, a possibility was to utilize Unitys built-in physics engine. It was however difficult to fully utilize, refer to section 4.4.1.

After the difficulties with the built-in physics engine, a numerical solver was used instead. It was implemented in a way that enabled multi-threaded computations, enabling almost real-time feedback - lagging behind by only a few cycles depending on system size and computer hardware configuration. A benefit of multi-threaded computations is the ability to solve larger systems without blocking the main thread so it cannot continue executing. It was possible to push almost all computations to separate threads, minimizing the impact on the main thread execution time.

Enabling modelling tools to be functional during computations regardless of method was considered a priority to retain as high level of direct manipulation as possible. The solver and associated scripts have gone through several iterations during the development in an effort to improve computational times and the associated performance impact.

4.1.3 Interpreting results

Results can be difficult to interpret. The magnitudes of stresses and strains can be hard to relate to without context. Numerical values need context to be understood, such as element section geometry or material properties. Considering the conceptual design scenario, it is not unthinkable that numerical values are of less interest. In both Sketch a Frame and StructureFIT it is possible to obtain valuable information about structure response and performance without presenting actual numerical values.

A visual interface similar to that of StructureFIT, where designs are presented side by side indexed by performance, was implemented as it aided the user to interpret their design alternatives visually. The concept was a slot system, where a user can save or load designs into different slots. Slotted designs can be compared against a chosen base design.

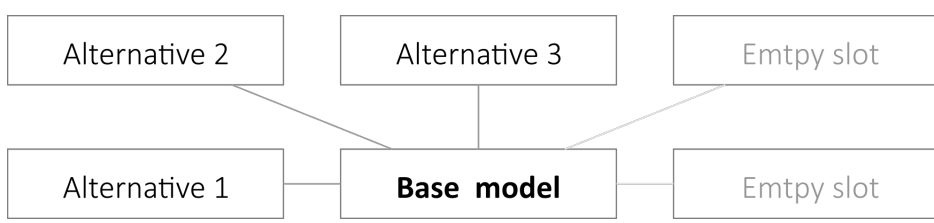


Figure 4.4: A slot system for system performance comparison.

4.1.4 Refining

Refining is an iterative process where changes are presumably made based on the interpretation of analytical results. This step is omitted as a separate process, as results presented in real-time enables refinement to occur simultaneously as modelling.

4.2 User interface

4.2.1 Principles

This works emphasis is on an interactive and intuitive design environment. The intuitive aspect resides in the user interface and interactivity enabled by software design. The application need a user interface which is intuitive to the user and consistent with other applications. Common user interface principles were applied to this part of the development process [37].

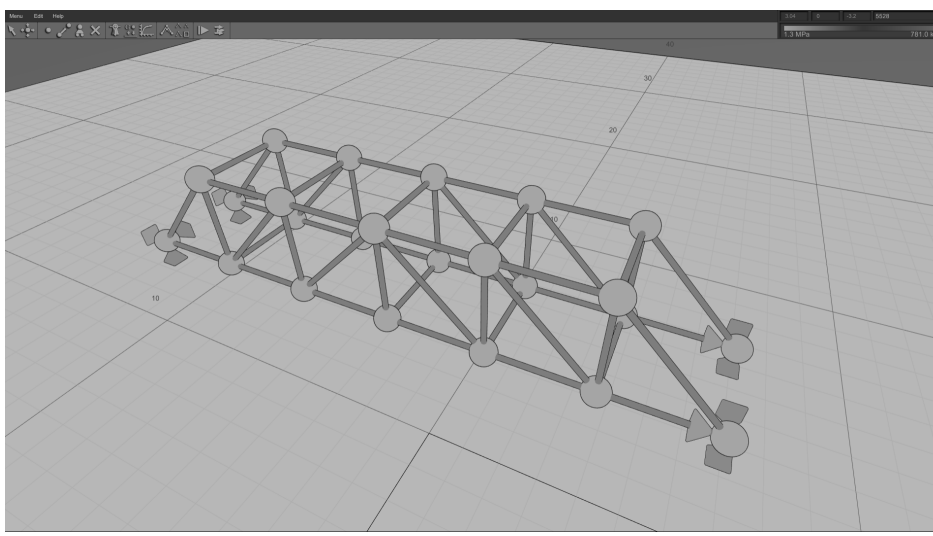


Figure 4.5: The interface of Interactive Structure Design

To ensure that users stay focused to the design task as possible, a clean user interface design was opted for. Throughout development, unnecessary windows, buttons and dialogue boxes were continuously removed. Other non-intrusive ways were developed to communicate with the user without disturbing, such as turning the model grey when it cannot be solved instead of a dialogue box.

Tools were made contextual, thus only visible during the active execution of a certain tool. Figure 4.6 shows the move tool and expose the tool parameters for that tool only at the bottom of the screen. The different options always appear at the same place on the screen, which is intuitive for the user.

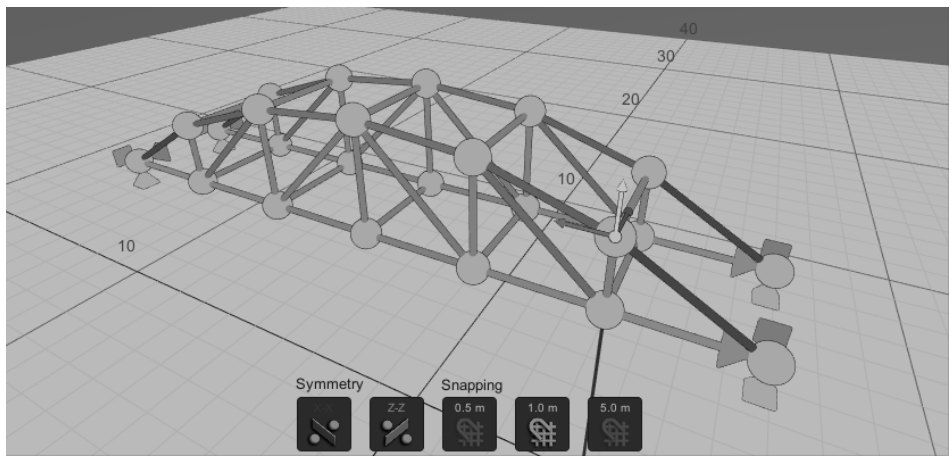


Figure 4.6: The move tool with its tool parameters.

The user interface consists mainly of a few buttons. Making icons for these was difficult. Some icons became more intuitive than others, but most make sense after using the software. To combat potential confusion, tool-tip functionality was added, see Figure 4.7.

Figure 4.7: Button graphic have seen a few iterations through the project.

The tool-tip also shows the shortcut key. The idea is that the user quickly finds the logical shortcut key layout on the keyboard, enabling them to use the modelling tools more effectively. The shortcut key layout consists of keys from left to right on a keyboard. Each step to the right yields a tool with functionality closer to analysing and left tools closer to modelling. All keys are within reach of the

left hand, enabling an experienced user to work very effectively. The figure also reveals the absence of icons for camera manipulation. In addition, there are no visual cues when manipulating the camera. This is an oversight on my part. Thankfully, users commonly try different key combinations, such as ctrl + mouse or alt + mouse almost immediately when using 3D applications, quickly finding the familiar orbiting controls and such alike. Camera tools and corresponding icons was later implemented after a user test, which revealed the issue among others, see chapter 5. Other icons were also reviewed and remade, as they were deemed unintuitive.

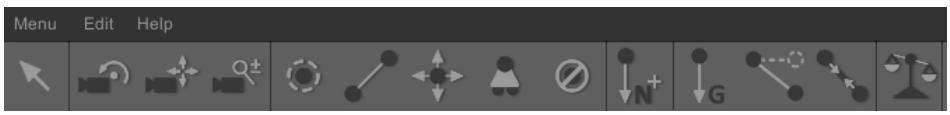


Figure 4.8: The new menu bar with more intuitive icons as well access to camera tools.

The user interface of the developed application utilized a coherent colour theme and icons change colour depending on state. Visual cues are a good way for the application to communicate with the user. Different information, such as hovering, active selection or a disabled state, make use of different but consistent colours.

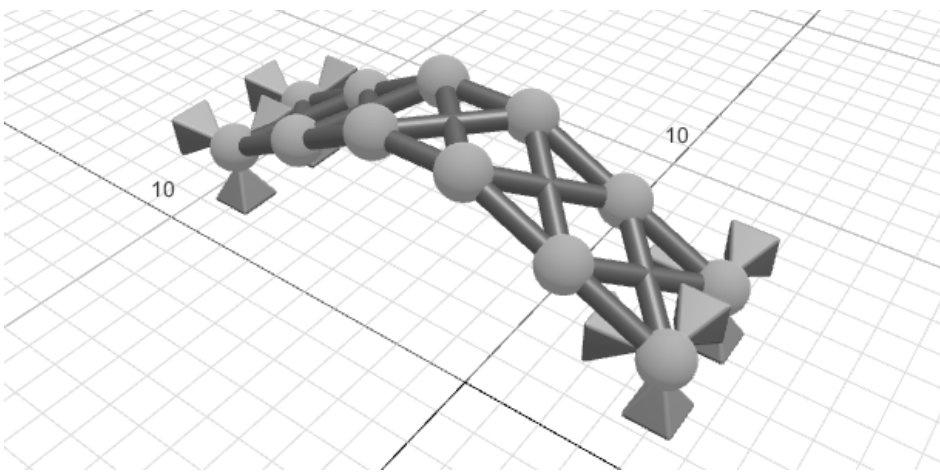


Figure 4.9: Old object design were elements were disproportionated.

It was surprisingly difficult to establish clear and comfortable visuals of the active model. Many different shaders *(a function containing the visual behaviour of a mesh surface, such as reflections, colour, light emittance, etc.)* and colours as well as sizes of the geometry representing the different system members, were changed, and evaluated continuously. This part of the development encountered many difficulties, especially when compiling the application to the web. WebGL have compatibility issues when using modern shaders.

During the development of the application, a few simple toon-like shaders was produced. These could render a dark rim at the edge of objects. It made it easier to see which objects were currently selected or being hovered. The purpose of their creation was to create compatibility with WebGL.

4.2.2 Shapshots

Snapshots, a feature inspired by the system comparison visualization from StructureFIT, is a system where a user can save or load designs to and from different *slots*. Snapshots are a *photography* of the active model at any given time, thus the name Snapshot. The idea is to enable the user to explore design alternatives and compare them using the indexed performance like that of StructureFIT.

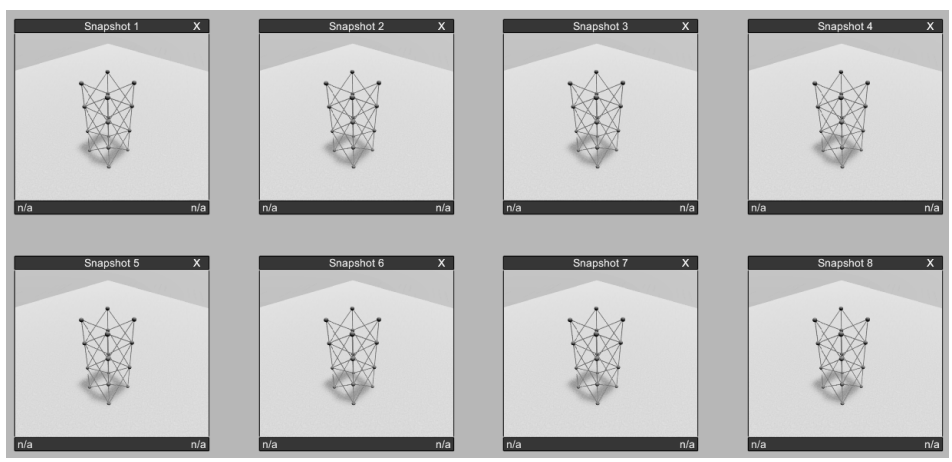


Figure 4.10: Early snapshot user interface design with the same model loaded into all available slots.

The first iteration of the tool enabled users to save snapshots using buttons which added entities to something similar to a spreadsheet. This concept proved to be very ineffective. With increasing number of snapshots it became increasingly difficult to recognise and differentiate entities. A second iteration introduced a separate window. Picture frames where added instead of text and entities could be recognised and compared visually more easily, see Figure 4.10.

But the separate window disconnected the user from the active model. The snapshots where moved to the main screen instead with transparent *fold-in*, *fold-out* windows, hiding them when not in use. The windows received add, remove and load-into-active-scene buttons, enabling the user to easily swap between designs.

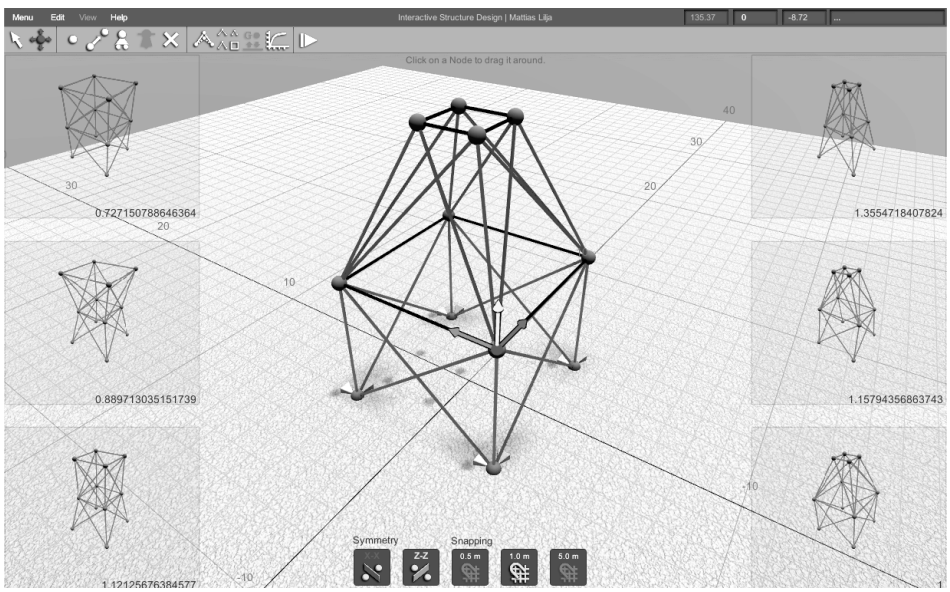


Figure 4.11: Improved snapshot UI design.

Using several geometrical systems and cameras, live streams of several snapshots could be visualized at the same time. Using many cameras and object instances vastly increase draw calls. Draw calls are expensive operations performance wise, and in general should be kept as low as possible. To combat lag and poor performance, an iterative rendering cycle was implemented. It added manual control to camera rendering calls. Rendering time went from being in the region of 10 ms down to 2< ms on a decent computer.

4.3 Implementation

4.3.1 Class structure

The strategy was to create a few heavyweight classes that communicated with each other. Each class was given methods which was relevant to its purpose. The classes are assigned to public statics identifiers, meaning they can be called and accessed by any script in the application. The advantage with this approach is a structure which can be well organized, something which in the long term proved valuable as the code base grew. The most important classes are described in this section.

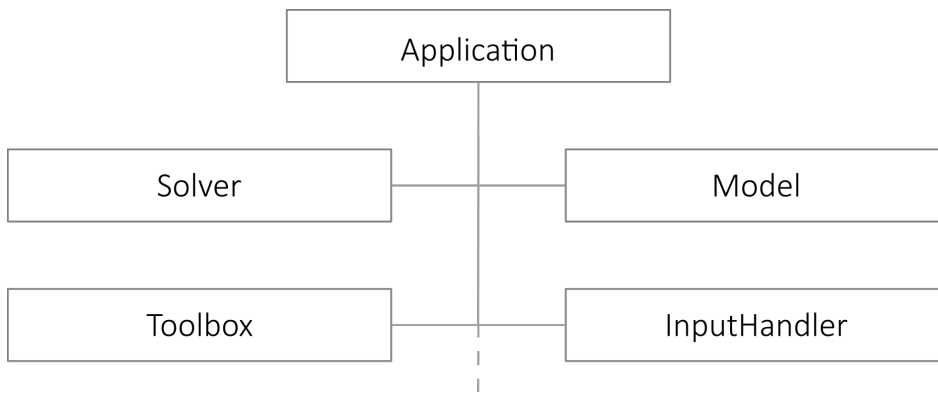


Figure 4.12: Software overview

4.3.1.1 InputHandler

The initial approach was to use alternative input devices such as the Leap Motion hand tracking controller. A number of other input devices was also considered. The input handlers main task was to sort I/O and pair human intention with appropriate action.

Figure 4.13: The input handler yields different authority to tools and functions depending on the current application state.

The discontinued development with support for the Leap Motion controller deprecated the main purpose of the input handler. However, it still served as a

class which yields layered authority. It serves as one of many safety features which prevents multiple execution of tools. It also checks for super actions - which can suspend and execute above normal tools, such as camera manipulation whilst freezing execution of a current tool preventing mishaps.

4.3.1.2 Toolbox

The toolbox is a class, which utilizes a standard tool structure where tools functions as a plug-in. The core structure was made to be modular so that tools could be added continuously. Highly interactive software generally entails a high probability of the user inadvertently glitching the system. This application, which implements 3D modelling and structural analysis, will require a good selection of tools and functions. It was important that these were isolated from each other to prevent the code from breaking, even if cross execution was to occur. This was combated trough establishing a routine scheme and structure, which tools and functions had to conform to.

Among the many tasks delegated to the toolbox class, the most important is to make only one tool execute at any given time. It is the only class with authority to start execution of different tools. Upon a call to start execution of a tool, the class does in addition have to receive appropriate authority from the input handler, making it even harder for tools to execute unexpectedly.

```
public void _startTool(){
        reset();
        StartCoroutine(Tool());
}
public IEnumerator Tool(){
        if (InputHandler.c.Authority != "default"){
                yield break;
        } else if (InputHandler.c.Authority == "default"){
                InputHandler.c.Authority = "editing";
                myTool.c.StartTool();
                while(myTool.c.IsRunning == true){
                        yield return null;
                }
        }
}
```

The above code block is a stripped down version of the tool execution template in C#. Firstly, a check is made to ensure the tool can be given authority by the input handler. The toolbox class checks whether the tool is running in a coroutine. A coroutine can pause execution and return control to Unity using yield commands [38]. This way a script can loop a check, such as whether a tool

is running or not, without entering an infinite loop. Coroutines are extensively used in the tool code.

To achieve a direct interaction model, tool execution should not hinder users intent. Tools were made to execute with only one action and with the ability to start execution from any point in the program. Whenever a call is made to activate a tool, the toolbox class resets every tool through a global reset function. It calls an abort routine which exists in every tool. This effectively resets the entire application to a known state. This enabled tools to be built from a known application state. The reset happens very quickly and is unnoticeable to the user.

4.3.1.3 The Model class

The model class handles request to alter or to edit a model. It contains routines to add or remove nodes, seeding the system members with unique identities or joining members together using topology information stored in the class. In addition, it keeps snapshot information and provides routines used by other scripts to save or load snapshots to and from memory as well as methods used for persistence, such as loading and saving models to and from a disk.

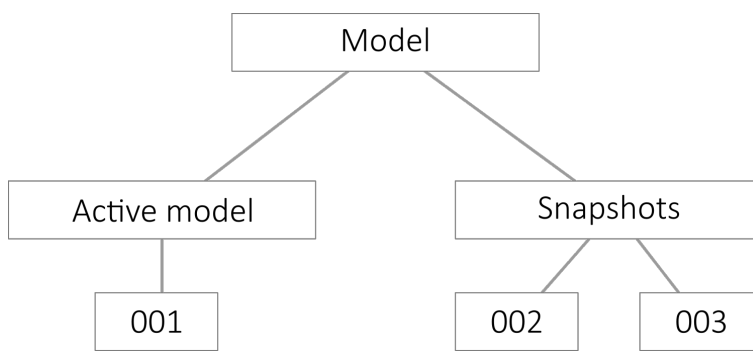


Figure 4.14: The Model class data structure.

4.3.1.4 The Solver class

The solver class acts as a translator. It translates the object scene to numerical code which can be sent to the numerical solver. The results from the solver are later interpreted and translated back to the object scene.

It is possible for the user to build systems which are unsolvable by the solver. This could happen when there is not an adequate amount of boundary conditions. To prevent crashes or code from stop executing, the solver has built in functions which checks systems to determine whether the system code can be sent to the solver or not. Invalid results returned by the solver are immediately dismissed. This can occur if the user modifies the structural topology during computations. It is only a possible for this to occur if threaded computations are used. The main thread, where modelling happens, would otherwise have to wait for computations to finish at which point the system topology will still be the same as it cannot be changed during computation.

4.3.2 Modelling

Almost all interactions with objects are through the physics engine - namely a feature called raycast. The users mouse position is placed into a virtual camera and a ray is shot at a base plane. The hit point becomes the x-y coordinates for the temporary node. Node height is determined by using the mouse scroll, changing the working plane which is kept between interactions. This makes it easy to work on *the same floor* continuously. The height of the working plane is displayed at all time, as well the temporary coordinates of the node.

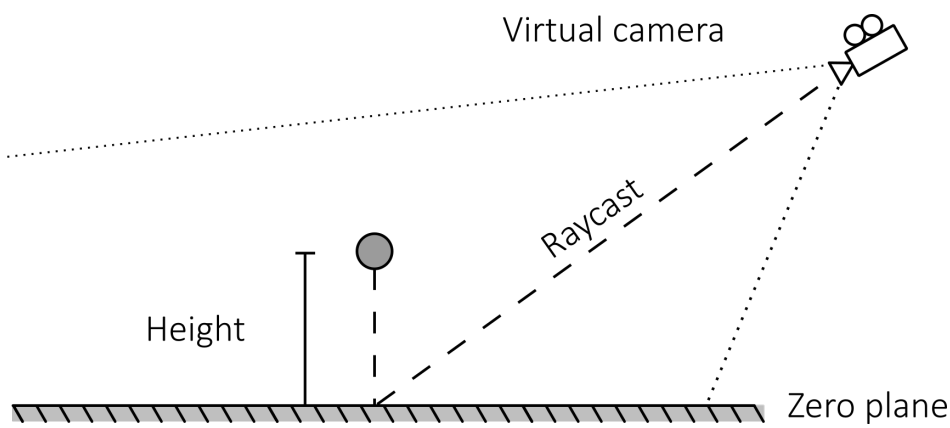


Figure 4.15: Node placement using raycast.

Manual entry of node position is possible after placement. This is done by selecting a node and enter specific coordinates in the coordinate window. Inspired by the element creation process in [7], elements are created between nodes by dragging (click + drag) or linking (click + click), see figure 4.16. In addition, there is also a delete tool and a move tool, where the latter can be seen in Figure 4.6.

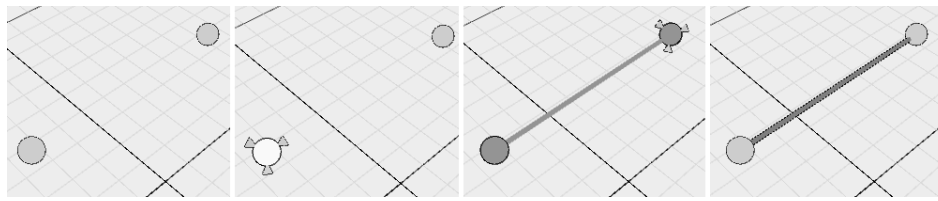


Figure 4.16: Connecting nodes with elements

The modelling process needs to be easy and intuitive to yield a perceived experience of direct manipulation. To achieve direct manipulation, real time feedback of the temporary position of a node is displayed, together with its symmetry entities if symmetry is enabled. In addition, an audible *pop* effect is triggered at the moment of placement as an action feedback notification.

4.4 Computations

4.4.1 Integrated physics

To enable dynamic analysis and object recognition by the integrated physics engine, node and elements need to have a rigidbody component attached to them. The added rigidbody component gives objects properties such as mass and the ability to detect and react to collisions. The rigidbody component is essential for the object to be recognised by the physics engine.

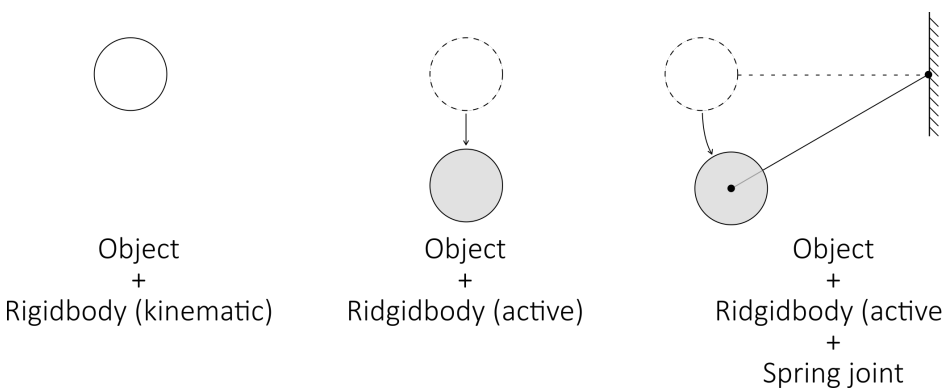


Figure 4.17: Effects of rigidbody and spring joint components added to an object

Two physical components of interest. The spring joint [39] and the configurable joint [40]. The latter is the most configurable of the two. Joints are a collective name of components which in one way or another ties two or more rigidbodies together. Spring joints are linear spring connections where the rigidbodies can be considered the nodes. The joint, in this context, refer to the actual spring with the property to push or pull a rigidbody. The rigidbodies are attached to the spring joint, which is the opposite of normal spring which connect to objects.

Extensive testing suggested that the springs behaviour is different from a normal spring, which can from the perspective of finite element analysis be considered to *push or pull an equal amount to connected bodies*. The first observation was a preferred direction of movement where the joints seemed to want to move one node over the other. Additionally, the joints have a base connection. The base connection is rigidly attached to a rigidbody whilst the other end is attached in a way that allows the rigidbody or the joint to pivot, see Figure 4.18. The conclusion was that the spring joint does not behave homogeneously.

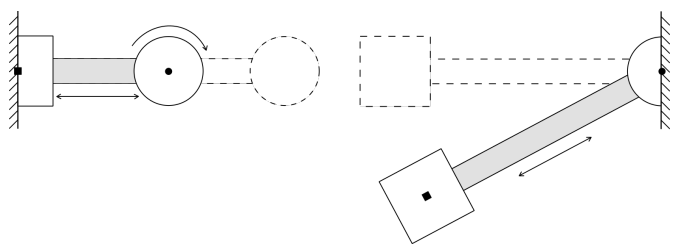


Figure 4.18: Effect of the different connections

Observe Figure 4.19. The base connection A is locked to the rigidbody it is attached to. It can not rotate around this connection. It rotates with the rigidbody. The other connection B, is locked to the second rigidboy by translation. That rigidbody will push or pull the connection, resulting in the spring changing length. The physics engine will the try to compensate by pushing or pulling the ridigbody with the spring. Several factor determine the amount of which it is moved, such as mass and drag.

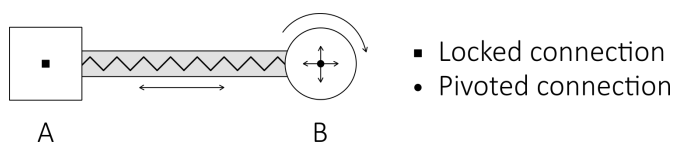


Figure 4.19: A spring joint with two connections.

The behaviour of the joints results in a preferred direction of movement and preferred points where rotation could occur. During early testing it was evident that this was not going to work. Solving this was difficult. A more true-like spring behaviour was obtained by using two springs in opposite direction with half the spring stiffness each, see Figure 4.20. The two springs made up what was used as a bar element.

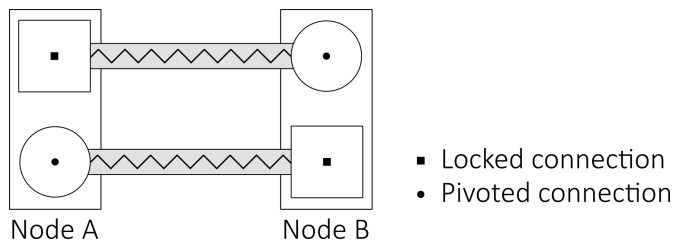


Figure 4.20: Spring element with no preferred direction, but more expensive on performance.

The combined dual spring joint element was implemented but indicated yet another unwanted behaviour, rotational stiffness. To proceed, an additional dummy-joint was added which gave the element rotational freedom at each connection. The added joint caused severe stability issues and decreased performance significantly. In addition, it yielded unsatisfactory deformations, see Figure 4.22a.

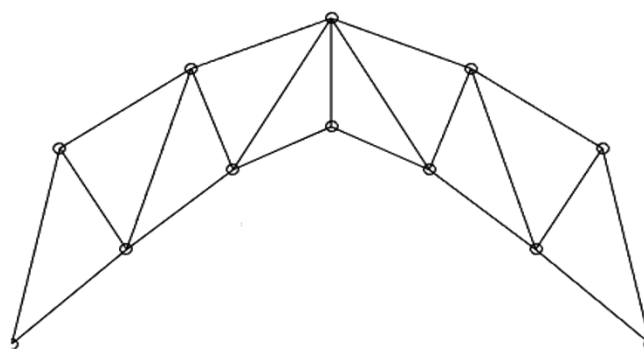
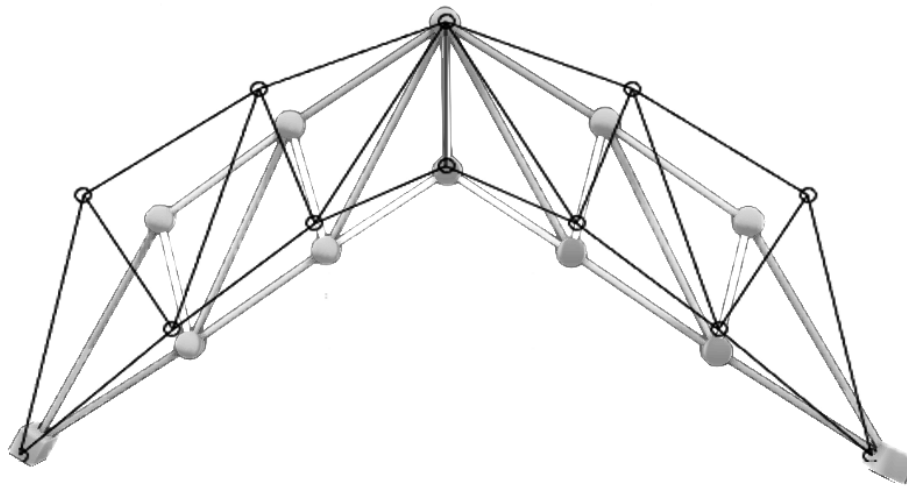


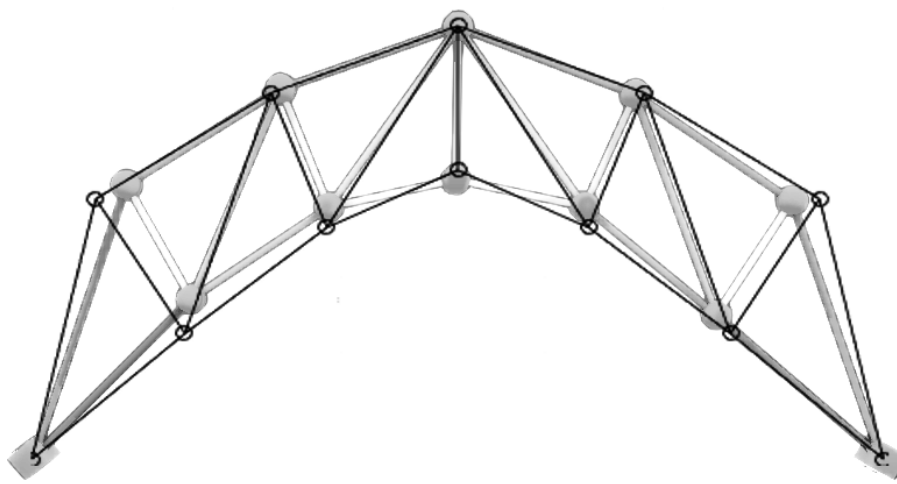
Figure 4.21: 2D bar elements - top middle node displaced 10m vertically

Observe the system in Figure 4.21 with deformations that of linear finite bar elements. Compare the deformations yielded by the spring elements in Figure 4.22a and 4.22b where the finite system is superimposed into the figures. Two additional examples in 3D illustrates the different behaviours, see Figure 4.23a and 4.23b. These systems have a central node displaced 10m as in Figure 4.21 but in the lateral plane instead. They illustrate the different spring joint elements behaviour.

Further experimentation yielded inconsistent results. It was not possible to establish a more correct physical behaviour of the elements and make results predictable. The method which yielded best results was computationally heavy and could not be improved, in part because the physic core could not be accessed.

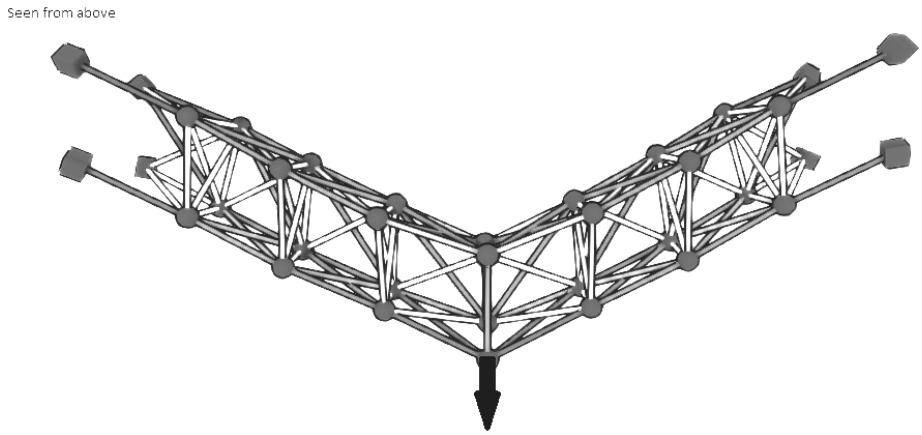


(a)

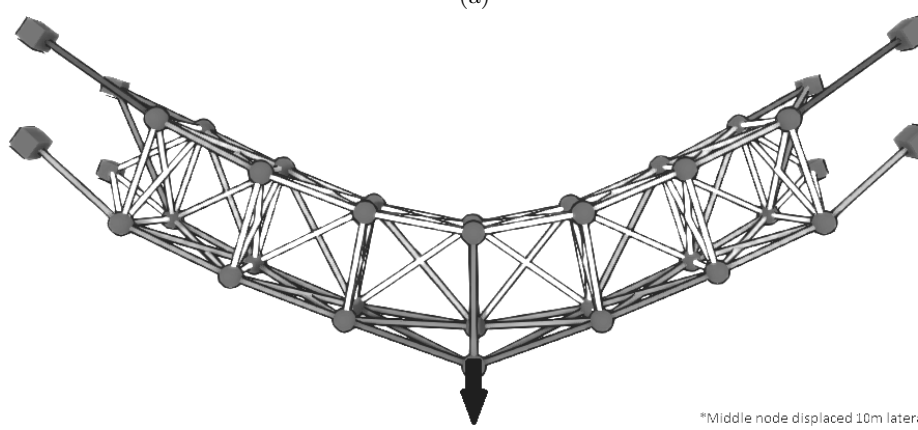


(b)

Figure 4.22: (a) Deformation in 2D with no rotational stiffness. (b) Deformation in 2D with unknown amount of rotational stiffness.



(a)



(b)

Figure 4.23: (a) Deformation with no rotational stiffness of a 3D structure. (b) Deformation with unknown amount of rotational stiffness of a 3D structure.

4.4.2 Numerical analysis

Numerical systems used in structural design are usually created using finite elements. In the developed application, three-dimensional bar elements are used. The element have 6 degrees of freedom with the element properties surface area and modulus of elasticity [41].

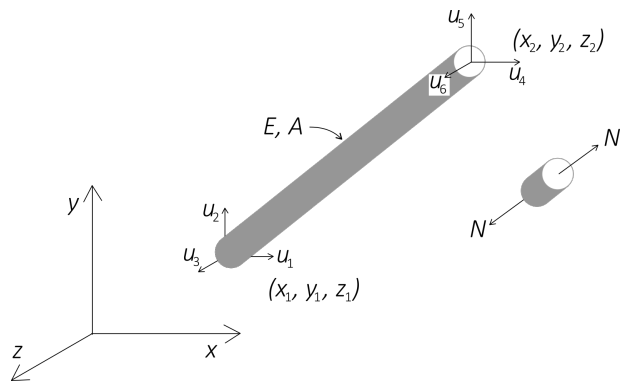


Figure 4.24: A three-dimensional bar element.

The element is used to construct a stiffness matrix with a linear elastic material model. The equation system becomes:

$$\mathbf{Ka} = \mathbf{f} \tag{4.1}$$

where,

$$\mathbf{K} = \frac{EA}{L}\begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} \quad \mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix} \tag{4.2}$$

K is the element stiffness matrix, derived from the element properties and material model. The **a** matrix describes displacements and **f** the forces. The above system represents one element with one set of degrees of freedom. Every element is assembled into a global equation system which is solved in the *numerical solver* class.

Internal forces are calculated using strain:

$$\sigma = \epsilon E = \left(\frac{l_n}{l_0} - 1\right) E \tag{4.3}$$

ϵ is the element strain, l_0 is the initial element length and l_n the new element length. Positive stress represents tension.

4.4.2.1 Performance

Using a numerical approach provides the opportunity to push computations to another thread on the CPU. This is beneficial for the user interaction with the application, as threading can result in improved performance. It additionally reduces the risk of manipulative tools not responding to input if the solver does not finish fast enough.

Threading computations means that the results can lag behind any number of update cycles. In general, this is not an issues as the computational time usually corresponds to less than one or two render cycles depending on hardware. The lag is barely distinguishable if at all if computation does not limit the manipulation of objects, if which lag is very noticeable. This is the conclusion for the first implementation, where manipulation of objects had to wait for the solver to finish when working with larger systems. This reduced the perceived interactivity of the application.

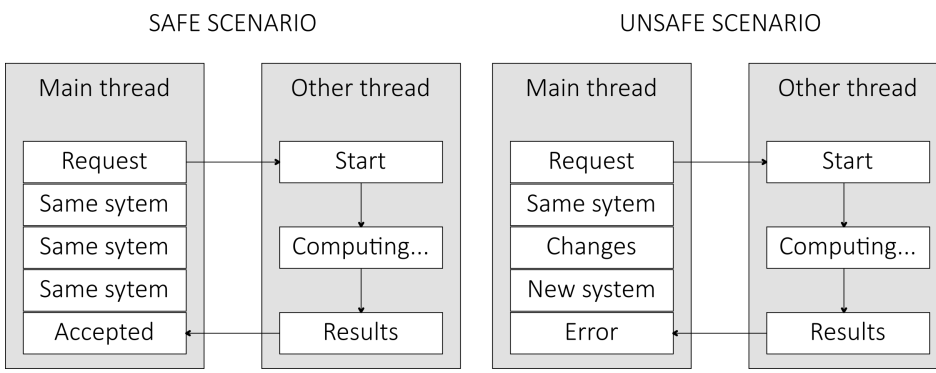


Figure 4.25: Potential scenario using threaded computations.

Although unlikely with small systems, but working with bigger systems where the computational time is greater than a few cycles can yield a solution to a system which have changed, making the result of that computation invalid. Results may in addition be incompatible with the new system. This could result

in invalid array accesses, freezing the main thread. The potential destructive behaviour was prevented using exception handlers, which can allow a code to continue even if an exception was thrown [42]. Catch handlers was used to revert the application back to a runnable state if this scenario occurs.

4.4.2.2 Solver

Calfem is a finite element toolbox [41] with common numerical operations for numerical computation. It was originally created for Matlab, but a C# adaptation of its most common methods is available [43], among whose is the solver.

The solver can only solve numerical systems, making it incompatible with objects in Unity. This is why the solver need to translate the scene from the *object domain* to the *numerical domain* and back. The solver class distributes the results to the different objects in the active scene. It allows other routines to immediately colour objects in colours representing the stresses in the different elements or such alike.

4.4.3 Snapshots

Snapshots use an indexed parameter for system performance comparison. The performance parameter is the strain energy. Strain energy is potential work stored in a system whilst undergoing deformation. This parameter is easily computed from the finite element system in eq. 4.1. For uniaxial stress, the incremental strain energy is defined as [44]

$$dW = \sigma d\epsilon;\ W(\epsilon) = \int_0^\epsilon \sigma(\epsilon) d\epsilon \tag{4.4}$$

Using *Hooke's law*

$$\sigma = E\epsilon \tag{4.5}$$

yields

$$W = \int_0^\epsilon \sigma(\epsilon) d\epsilon\ = \frac{1}{2} E\epsilon^2 \tag{4.6}$$

Applying 4.6 to the system in 4.1, where forces are determined by the stiffness matrix $\mathbf{K}$, which is derived from the constitutive relationship between stress and strain, it is possible to use the same approach to yield [44]

$$W(a) = \frac{1}{2}\mathbf{a}^\top \mathbf{K}\mathbf{a} \tag{4.7}$$

Eq. 4.7 is the expression used to obtain the strain energy. Each snapshot class stores the last valid strain energy yielded by the solver. The UIs visual

representation of a snapshots use this parameter to calculate and present an indexed number, which is normalized against a user chosen design, enabling the user to compare designs by performance indexes.

Chapter 5

User testing

5.1 Method

There have been opportunities to observe users as they use the application throughout the development. Users have mainly been students in programmes relevant to construction. Some users tested the application remotely where direct observation was not possible and provided feedback in written form. This chapter summarizes some of the most interesting interactions observed and remarks given.

5.2 Application remarks

Comments regarding the application have been mostly positive. Most comments related to the clean interface and well-chosen colours which made the different 3D elements easy to tell apart. Additional positive remarks were given to the representation of boundary conditions. Most of the icons felt well-made and intuitive.

Modelling usually went without much trouble and users could be seen playing around with nodes. Their goal was usually to achieve a satisfactory and even *stress distribution* by trying to move the nodes in such a way that the colours representing the element stresses became evenly distributed in *nice looking gradients*. This was most noticeable when users interacted with one of the examples shipped with the software, specifically a simple 3D bridge.

The modelling tools were conceived as good. Using the scroll wheel to move nodes up and down was conceived as slightly alien at first. However, it quickly

became intuitive after some use. A negative consequence of the interaction model made it difficult to work with nodes which had a high height value. This as it was difficult to see the base-plane and node at the same time, making placement difficult. Refer to Figure 4.15. The implementation did in addition make it hard to work on a laptop, which in general do not have a scroll wheel built in.

Almost none of the observed users discovered the snapshot function by themselves. The snapshot feature had to be manually introduced and sometimes explained to make sense to users. This was unfortunate. It became evident that the snapshot feature does a poor job of explaining itself - even when exposed to the user. The aim for a *super-clean* interface might have hurt the development of the tool, as the feature was *tucked away* to not clutter the UI. Additionally, the icon for the snapshot feature is also highly non-intuitive, and really only made sense to a user after the snapshot feature was introduced. Users expressed positive remarks regarding the potential power in such a tool but wanted to see a more integrated solution.

The most positive remarks were given to the symmetry tools, which significantly speeded up the modelling process, which in most cases was a box-like structures or a simpler bridge structure. Negative remarks were given to the camera tools, which missed buttons and did not work well with laptop users where a *middle-click-hold* was hard to execute. The application environment did in general feel intuitive and familiar to users. The aim to develop an application with an interactive environment with a more direct interaction model than common structural design software is considered to be achieved.

5.3 Summary

The interaction model of the application failed to significantly improve or incite users towards design exploration beyond playing with their models in a behaviour similar to that of Sketch a Frame [7]. The snapshot feature was thought to be the main feature of the application. However, it was not discovered or used by users in any extensive observable manner. The snapshot feature was however seen as a potential powerfully tool, but needs a different implementation.

Chapter 6

Results and discussion

6.1 Discussion

The development of StructSTUDIO have been a learning process. Using a game engine proved no real hindrance to development. The issues were mainly related to the online version using WebGL. Due to issues with WebGL, less time could be spent implementing tools. Apart from adapting the application to the web, the development process did for the most part proceed without much hindrance.

The Unity integrated development environment and editor provided good tools and components to design a UI consistent with conventional applications. The improved control over the UIs shape, form and functionality made up for the extra development work.

6.2 Application usages

The application can calculate stresses and strains of structural members and visualize the results to the user. The snapshot tool can be used to compare designs. It does not incite to it, but the tool is fully functional non the less.

There is great value in design exploration, and using the snapshot feature yields intuitive understanding of how structural members affect system performance. Its ability to compare alternatives enables users to learn about structural performance. This leads to more informed decisions and entails attributes which makes the application applicable to education.

The application is not limited to analysis of *structural objects*. With more versatile 3D modelling tools, which were easier to use than that found in common structural analysis software, the 3rd dimension became more accessible. It enabled many different objects, not necessarily related to buildings, to be designed and compared, see Figure 6.1.

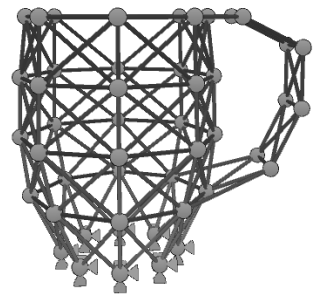


Figure 6.1: A coffee cup modelled in StructSTUDIO.

6.3 Conclusions

According to user feedback, the modelling and analysing aspects of the application was enhanced by the more direct interaction model as well as the implementation of real-time results. This further indicate and support conclusions made in previous work, that more direct interaction models can enhance the users engagement with the design task [7–9].

Inciting users to design exploration was in part achieved by implementing a more direct interaction model. The main feature of the software, Snapshots, did not seem to incite further design exploration - even after being introduced to the user. If the tool can be successfully developed and integrated to the design process, it can be used in both practical and educational scenarios. It can also provide a good middle ground for architects and engineers, where specifically an engineer has a tool which can in a visual and intuitive way help the engineer explain how the architects design choices effects structural performance.

Developing for the web also entails an extended availability over different operating systems as well as web browsers. The absence of an installation process made it easy for users to start using the application and the feedback they gave was valid for the same deployment of the application regardless of operating system or web browsers with only a few exceptions.

6.4 Suggested future work

With the snapshot feature being regarded as a potentially powerful tool, its true benefit to conceptual structure design should be further investigated using a different implementation than that used in this project.

The development with virtual reality and hand tracking devices was discontinued due to technical difficulties. The small amount of experience earned with such devices strongly suggested a potentially powerful ability to unlock users from the normal 2D-interface paradigm. The devices possible benefit to structural design should be revisited when the technology has matured.

A limiting factor with regards to using Unitys built-in physics is the inability to access the core physics engine. There are numerous open source engines available, which might prove to be a better choice for a dynamic computational approach.

It has throughout this thesis been mentioned that versatile modelling tools are important to the conceptual design phase. Applications mentioned in chapter 2, as well as StructSTUDIO, is limited to relatively basic geometry. It could be beneficial to explore possibilities with expanding applications like these with more powerful geometric modelling tools while keeping analytical capabilities and a high level of interactivity. Approaches could involve programmatic and parametric modelling, both of which could aid in creating or generating more complex geometries. In addition, applications like these could also benefit from the capability to generate geometry or suggest possible improvements based on analytical results like that of StructureFIT.

Bibliography

[1] Anders Häggman, Geoff Tsai, Catherine Elsen, Tomonori Honda, and Maria C Yang. Connections between the design tool, design attributes, and user preferences in early stage design. *Journal of Mechanical Design*, 137(7):071408, 2015.

[2] Mike Schlaich. Challenges in Education–Conceptual and Structural Design. In *IABSE Symposium Report*, volume 92, pages 20–26. International Association for Bridge and Structural Engineering, 2006.

[3] Martin Fröderberg and Roberto Crocetti. Engineers in need of an improved conceptual design toolbox. In *IABSE Symposium Report*, volume 102, pages 515–521. International Association for Bridge and Structural Engineering, 2014.

[4] Wynne Hsu and Bing Liu. Conceptual design: issues and challenges. *Computer-Aided Design*, 32(14):849–850, 2000.

[5] Caitlin Mueller and John Ochsendorf. An integrated computational approach for creative conceptual structural design. In *International Association of Shell and Spatial Structures (IASS) Symposium 2013 "Beyond the Limits of Man*, 2013.

[6] J. Lindemann and Lars Damkilde. *ForcePAD: a new User Interface Concept for Design and Optimisation*, pages 225–227. Department of Civil Engineering, Aalborg University, Denmark, 2009.

[7] Daniel Akesson and Jonas Lindemann. A tablet computer application for conceptual design. *Proceedings of the ICE - Engineering and Computational Mechanics*, pages 1–8, 2015.

[8] Daniel Åkesson and Jonas Lindemann. Using 3d gesture controls for interacting with mechanical models. In *Nordic Seminar on Computational Mechanics*, 2013.

[9] Daniel Åkesson. Using direct manipulation for real-time structural design exploration, 2016.

[10] Kirk Martini. A new kind of software for teaching structural behavior and design. In *2006 Building Technology Educators' Symposium Proceedings*, page 279. Lulu. com, 2008.

[11] Jim Nieters. Defining an interaction model: The cornerstone of application design, Accessed 2016. http://www.uxmatters.com/mt/archives/2012/01/defining-an-interaction-model-the-cornerstone-of-application-design.php.

[12] Donald A Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[13] Carlo H Séquin. Cad tools for aesthetic engineering. *Computer-Aided Design*, 37(7):737–750, 2005.

[14] Sari Kujala, Virpi Roto, Kaisa Väänänen-Vainio-Mattila, Evangelos Karapanos, and Arto Sinnelä. Ux curve: A method for evaluating long-term user experience. *Interacting with Computers*, 23(5):473–483, 2011.

[15] Pierre Olsson, Publikation Chalmers, Tekniska Högskola, and Majornas Copyprint. Conceptual studies in structural design. 2006.

[16] Edward R. Tufte. *Envisioning Information*. GRAPHICS PRESS LLC, 2005.

[17] Pierre Olsson. Pointsketch2d, Accessed 2016. https://www.chalmers.se/arch/SV/forskning/publikationer8817/datorprogram/pointsketch2d.

[18] Pierre Olsson. Pointsketch3d, Accessed 2016. https://www.student.chalmers.se/sys/FileHandling/downloadFile?folderName=4417hout&path=&fileId=pointSketch3D.exe.

[19] Daniel Åkesson. Sketch a frame, Accessed 2016. https://itunes.apple.com/se/app/sketch-a-frame/id563527046?mt=8.

[20] Caitlin T Mueller. *Computational exploration of the structural design space*. PhD thesis, Massachusetts Institute of Technology, 2014.

[21] Caitlin Mueller. Structurefit, Accessed 2016. http://www.caitlinmueller.com/structurefit/.

[22] Kirk Martini. Arcade, Accessed 2016. http://www.arch.virginia.edu/arcade/.

[23] Unity technologies. The unity game engine, Accessed 2016. https://unity3d.com/.

[24] Unity. Made with unity - non games, Accessed 2016. https://unity3d.com/showcase/gallery/non-games.

[25] NVYVE. Nuovo – condo visualization, Accessed 2016. http://nvyve.com/portfolio-items/domicile-nuovo/.

[26] Cloverpoint. Insight, Accessed 2016. http://cloverpoint.com/insight/.

[27] Kirk Martini. Real-time, non-linear, dynamic simulation in teaching structures: elementary to advanced. In *Proceedings of the 2005 American Society for Engineering Education Annual Conference*, 2005.

[28] Unity Technologies. Unity and mono compatibility, Accessed 2016. https://docs.unity3d.com/412/Documentation/ScriptReference/index.Unity-and-Mono-compatibility.html.

[29] Mono-Project. About mono, Accessed 2016. http://www.mono-project.com/docs/about-mono/.

[30] Mono. Mono (c# compiler), Accessed 2016. http://www.mono-project.com/docs/about-mono/languages/csharp/.

[31] Microsoft. base class(c# reference), Accessed 2016. https://msdn.microsoft.com/en-us/library/hfw7t1ce.aspx.

[32] Math.NET. Math.net numerics, Accessed 2016. http://numerics.mathdotnet.com/.

[33] Math.NET. Math.net numerics license (mit/x11), Accessed 2016. http://numerics.mathdotnet.com/License.html.

[34] Unity Technologies. Physics in unity 5.0, Accessed 2016. https://docs.unity3d.com/Manual/UpgradeGuide5-Physics.html.

[35] NVIDIA. Nvidia completes acquisition of ageia technologies, Accessed 2016. https://www.nvidia.com/object/io_1202895129984.

[36] Leap Motion. Leap motion developers - orion beta, Accessed 2016. https://developer.leapmotion.com/orion.

[37] Microsoft. User interface principles, Accessed 2016. https://msdn.microsoft.com/en-us/library/windows/desktop/ff728831(v=vs.85).aspx.

[38] Unity Technologies. Coroutines, Accessed 2016. https://docs.unity3d.com/Manual/Coroutines.html.

[39] Unity Technologies. Configurable joint, Accessed 2016. https://docs.unity3d.com/Manual/class-ConfigurableJoint.html.

[40] Unity Technologies. Spring joint, Accessed 2016. https://docs.unity3d.com/Manual/class-SpringJoint.html.

[41] P-E. Austrell et al. *CALFEM, a finite element toolbox*. KFS Lund AB, 2004.

[42] Microsoft. try-catch (c# reference), Accessed 2016. https://msdn.microsoft.com/en-us/library/0yd65esw.aspx.

[43] Daniel Åkesson and Vedad Alic. Calfem for c, Accessed 2016. https://github.com/CALFEM/calfem-csharp.

[44] Niels Ottosen et al. *Introduction to the finite element method*. Prentice Hall, 1992.

List of Figures