2D interaction models for finite element method on touch devices

Daniel Åkesson

14 januari 2013

Abstract

The computational power of hand held devices have increased significantly during the last years. Today it is possible to run advanced finite element simulations on these devices. This opens up new possibilities for creating software for the early design stage and for educational purposes. Touch devices such as the iPad has become increasingly popular, and the number of users are growing every day. The multi touch interface of the iPad has changed the interaction between computer and human. Some of the precision of using a mouse is lost, but is replaced by a more direct interaction with objects on the screen. The direct manipulation is what makes the multi-touch interface well suited for structural mechanics problems. The ability to give the user a feeling of being able to directly manipulate a model. The aim of this thesis is to develop an application to investigate how to take advantage of the new possibilities that the multi-touch interfaces creates in the field of structural mechanics.

There are today on the market some available software tools for the early stage design process, but they are mostly PC software. To investigate how the multi touch interface can be used for this type of applications an iPad application has been developed. The iPad application has been developed using Objective C and C++, where C++ is used for the FE computations for performance reasons.

The developed application is a finite element application using beam elements. A geometry can quickly be modeled using developed direct manipulation methods. The application does not have the conventional user interface, instead the result is continuously recomputed once the model is stable. Which is determined using an eigenvalue analysis.

The developed application is avalible in App Store under the name ”Sketch a Frame”. The created application uses the advantages of the multi touch interface. Creating new methods for the modeling process which has enabled the user to get a feeling of direct manipulation.

Sammanfattning

Beräkningskapaciteten på tablets har ökat markant de senaste åren. Idag är det möjligt att köra avancerade finita element simuleringar på dessa enheter. Detta öppnar upp nya möjligheter för att skapa mjukvara för det tidiga designskedet samt för utbildningssyfte. Tablets har blivit allt mer populärt och antalet användare växer varje dag. Multi-touch gränssnittet som iPad:en introducerade har förändrat interaktionen mellan användare och dator. En del av precisionen som fås med mus förloras, samtidigt fås en ökad känsla direkt manipulation, möjligheten att direkt kunna manipulera visualiserade objekt. Det är denna känsla av direkt manipulation som gör multi-touch gränssnittet intressant för strukturmekaniska applikationer. Möjligheten att ge användaren en känsla av att direkt kunna manipulera en modell. Målet med detta arbete är att undersöka möjligheterna som multi-touch gränssnittet öppnar upp inom strukturmekaniken.

Det finns idag några mjukvaruverktyg för det tidiga designskedet, de flesta är för PC. För att undersöka hur multi-touch gränssnittet kan användas för denna typ av applikationer har en iPad applikation utvecklats. Applikationen har utvecklats med Objective-C samt C++, där C++ har använts för beräkningsdelen av prestandaskäl.

Den utvecklade applikationen är en finita element applikation som använder sig av balkelement. En geometri kan enkelt skapas genom att utnyttja de framtagna manipulationsmetoderna. Applikationen har inte ett konventionellt gränssnitt, istället uppdateras det beräknade resultatet löpande då modellen är stabil. Detta bestämts med hjälp av en egenvärdesanalys.

Den utvecklade applikationen är tillgänglig i App Store under namnet Sketch a Frame". Applikationen använder möjligheterna från multi-touch gränssnittet. De framtagna metoderna för modelleringsprocessen har möjliggjort en känsla av direkt manipulation för användaren.

Contents

Acknowledgments

The work presented in this master thesis was carried out for the department of structural mechanics at Lunds institute of technology in collaboration with the software development company StruSoft. The idea for this thesis is Dr. Jonas Lindemann and also uses ideas and concepts from Senior Lecturer Karl-Gunnar Olsson.

I would like to thank Dr. Jonas Lindemann for giving me the opportunity to do this exciting project and for the guidance along the way. I would also like to thank StruSoft for giving me the opportunity to see how they work and exchange ideas. I would especially like to thank Dr. Pierre Olsson for his time and input in the development process. Finally I would like to thank my fellow students and friends for the support I have had during my education at LTH.

Lund, October 2012.

Daniel Åkesson

Chapter 1

Introduction

1.1 Background

Two decades ago only supercomputers could run FE-computations. The technology has since developed in a rapid rate and there is unused potential in tablets and laptops today, to run real-time FE-calculations. This opens up possibilities for creating software for the early design stage, where the user quickly can model and define load and boundary conditions and see the internal forces distribution as an tool in creating new design. A good design will often maximize the strength of the structure by utilise available materials in an efficient way.

Since the iPad was introduced in 2010 there has been a rapid development in tablets and the number of users are growing every day. The advantages of tablets are the small size and the direct touch-interface, enabling easy to use and direct applications. The multi-touch interface changes the interaction for the user compared to the regular mouse based applications. Some of the precision of using a mouse is lost instead the feeling of directly manipulating the model is increased. Direct manipulation is what makes the multi-touch interface well suited for structural mechanics problems; the ability to give the user a feeling of directly manipulat the model.

1.2 Aim of the thesis

The aim of the thesis is to develop an application to investigate how to take advantage of the new possibilities that multi-touch interfaces offers in structural mechanics software.

Chapter 2

Tablets

2.1 History

In 1968 a first concept called the Dynabook [1] was developed. It was never built but the idea was to use it for education. In 1983 Apple built a prototype of a tablet called Bashful [1]. This device had a keyboard and other add-ons such as floppy drive and phone but was also never built. Five years later, in 1988, the first tablet was released to the market and it was a portable device called the GRiDPad [1]. The device had handwriting recognition and sold for 2370 USD. Due to the high price it was mostly used in health care and law enforcement institutions.

Figure 2.1: The Dynabook, GRiDPad, Apple Newton and Microsofts tablet

In 1987 Apple started a big project to build a tablet PC called Newton [2]. As the project developed it went from being a tablet to a PDA (Personal Digital Assistant). The device needed a lot of processor power to be able to do handwriting recognition. At first, the device had three processors to supply enough processor power. This was expensive and resulted in a low battery time. Apple then switched to the ARM processor, which uses a different, more energy efficient architecture. The ARM processor architecture is today very common in all small devices such as phones, modems, iPads etc. due to the low energy consumption. The Newton PDA was finally released in 1993 after the deadline had been pushed back three times.

The next major event in the tablet market was when Microsoft in 2002 got involved with the Tablet PC [1]. The Tablet PC used a special version of the Windows XP operating system with special extensions for handwriting and stylus input. The Windows XP was a great desktop operating system but did not work as well on the tablet. The tablet was introduced at over 2000 USD and required a stylus for proper use. These were factors that never made it the success that Bill Gates thought it to be.

> *"The Tablet takes cutting-edge PC technology and makes it available wherever you want it, which is why I'm already using a Tablet as my everyday computer. It's a PC that is virtually without limits – and within five years I predict it will be the most popular form of PC sold in America."* - Bill Gates [3]

In 2007 the iPhone was released, and along with it the new iOS operating system. iOS took advantage of the new multitouch screen and this improved the interaction. The iPad was actually developed before the iPhone, but it was during this development Apple realised that touch and multi-touch could be used in a mobile device such as a phone [4]. The iPad was released in 2010.

2.2 Choice of tablet

The iPad was chosen as a tablet for several reasons. First, it is the most widely used tablet. Secondly, C++ can be easily integrated in the project, which enable easy reuse of existing matrix and FE libraries such as Newmat and Calfem++.

It is possible to reuse the C++ code to develop the application for Android or Windows 8. To visualize the result OpenGL ES would be beneficial to use as all three OS's supports it. The different OS's would still need custom written code as they all have their differences.

Chapter 3

Previous work

3.1 ForcePAD

ForcePAD is an intuitive tool for visualizing how structures behave when subjected to loads and boundary conditions. [5]. The application is built up with three different modes. Sketch mode is used to draw a geometry using the brush tool. Physics mode where the user inputs constraints and forces. The final mode is the action mode where the result is visualized. No numerical values are used in the result as the application focuses on giving the user an understanding of how the forces will be distributed.

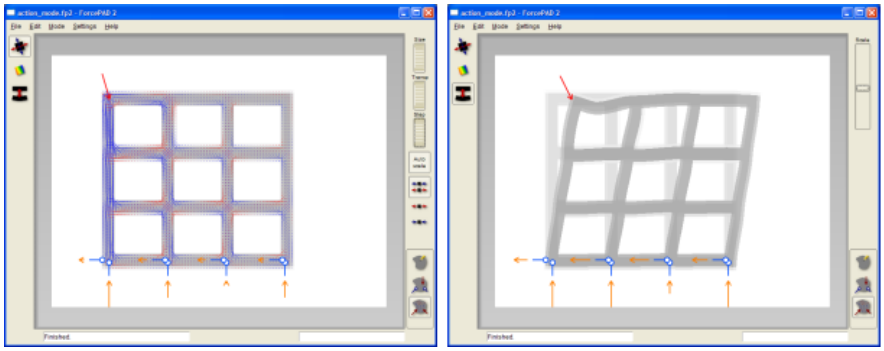

Figure 3.1: ForcePAD

Another version of the application was built for an exhibition at the Science Center at LTH [6]. In the exhibition a SmartBoard touch device is used. The SmartBoard is a large touch enabled computer screen. To be able to successfully run ForcePAD on a touch screen some modifications to the user interface was imperative [7]. The users are casual by passers which requires a simple user interface. To simplify the user interface, menus was removed and a floating tool box was introduced to better work with the touch interface. These modifications made the application more accessible and it has been used effectively in illustrating different mechanics phenomena for the audience. However, visitors still needed additional instructions to successfully use the application.

3.2 pointSketch

PointSketch is an application built for use in the early design process [8]. The interface is built for mouse input. To input a geometry nodes are set out and then bar elements can be drawn between these. Forces and constraints can then be applied to the nodes. When the geometry is complete an action mode can be entered where forces are visualized by coloring the elements and the deflections are visualized using an animation. There are two different versions of the application; one for two dimensions and the other for three. The pointSketch can also show if the model is stable or not. If it is a mechanism, the mode of the mechanism will be illustrated with an animation.

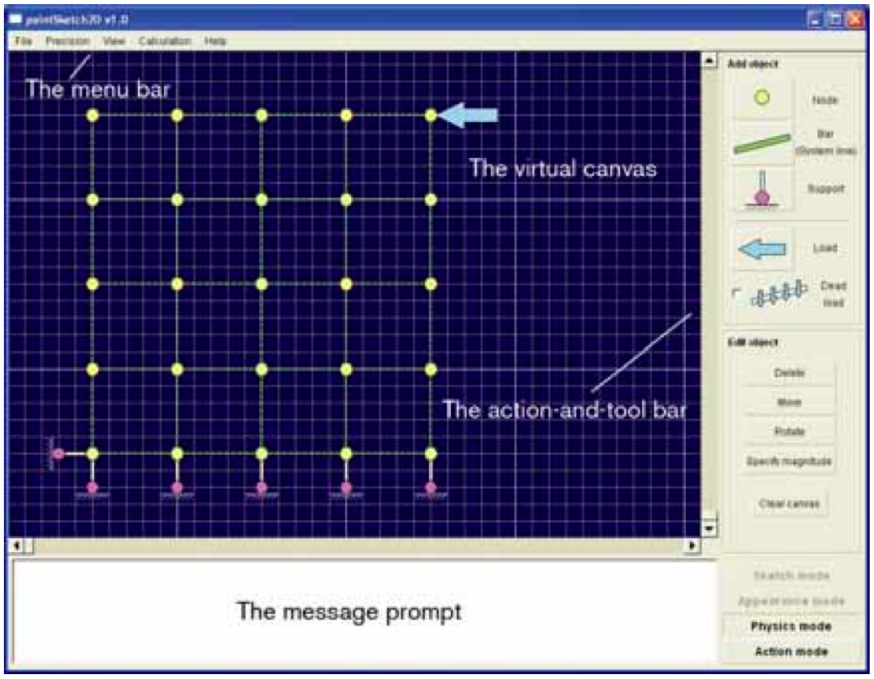

Figure 3.2: The GUI in pointSketch2D, currently in the physics mode.

3.3 Intesym CASA software suite

The company Intesym have created a software suite for the iPad with tools to analyze whole structures in both 2D and 3D [9]. The input is either entered numerical or by drawing. Material parameters can be chosen and the result will present if the structure has sufficient strength to the existing load and show how the structure will deform. The prices on the apps are ranging from 189 SEK to 1399 SEK depending on the type of element used and if it is 2D or 3D. The user interface is built up like a classic FE-software, with three stages to go through: geometry, load and last results. A tab bar is used to go through the different stages.

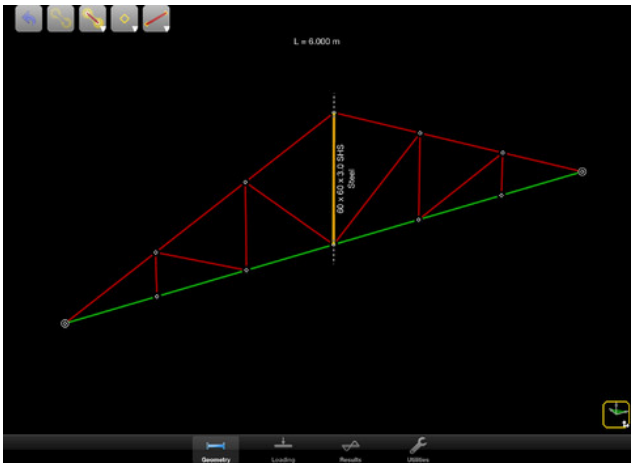

Figure 3.3: Screenshot of intesym CASA Beam 2D

Chapter 4

Developing for the iPad

Apple have created a tool to create apps for both Mac OS and iOS that is called Xcode. Even though most code is written in Objective C there is support for including C++ code and libraries.

4.1 Xcode

XCode is an advanced development environment that can handle simple as well as large projects with multiple sub-projects. In the storyboard mode elements are available to build up the interface such as views, scroll views, sliders, buttons etc.. Transitions between views can be created visually by using the storyboard. The storyboard allows for a good overview of a project and visualizes how the views are connected.

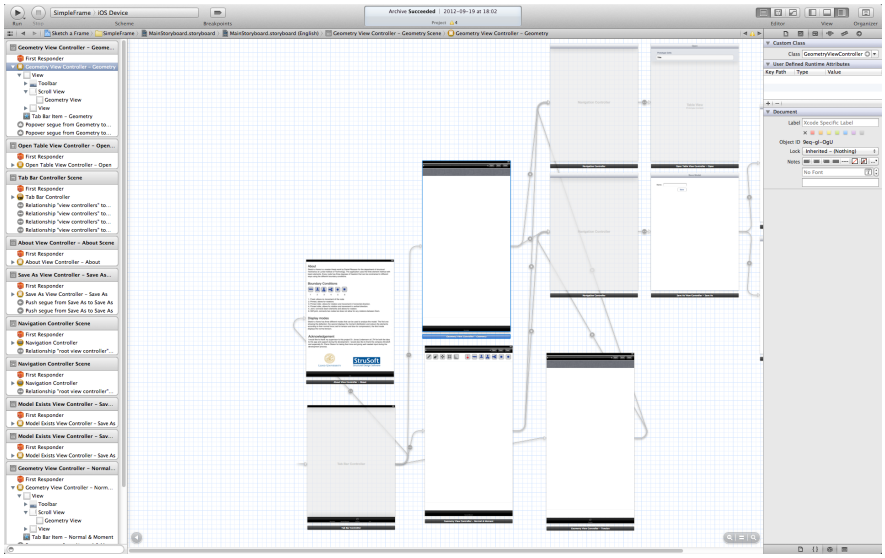

Figure 4.1: Xcode with a storyboard open

View elements are like blank sheets of paper, they are used to present something for the user e.g. draw a model. When building apps the Model View Controller design patters is often used, referred to as MVC.

The controller class is where the input from the user is handled, it can send commands to manipulate the model. The model is where the data is stored, when the model is manipulated it sends a notification to the view. This notification updates the content of the view accordingly. In the view generates the visual representation of the model that the user sees.

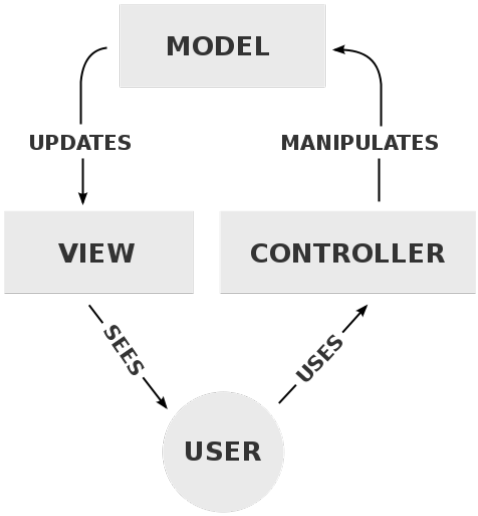

Figure 4.2: Model View Controller

4.2 Programming languages

The languages primarily used when developing in XCode is Objective-C and C++. They are both a superset of C but they have developed in different directions.

4.2.1 C++

C++ is an object-oriented, general purpose programming language that combines high-level and low-level language features [10]. It is today one of the most popular programming languages [11] and a few well known applications of the language are Adobe Photoshop, Google searchengine, Facebook and Autodesk AutoCAD [12]. It is also used in device drivers and hardware design which illustrates the versatility of the language. One of the strengths of C++ is that is supports a multi paradigm programming style in which it is possible to combine different styles of programming [13].

Background

In 1979 Bjarne Stroustrup worked at AT&T and he could not find a suitable programming language to solve a problem [13]. He wanted a programming language that was both efficient and elegant, but no existing languages at the time had these qualities. He then decided to create a new programming language that at first was called ”C with classes”. This first version of the language was used internally at AT&T in august 1983 [13]. Later that year the name was changed from ”C with classes” to C++ as a pun involving the C increment operator [10].

The first commercial version was released two years later in 1985 and has since then continued to evolve. In 1998 C++ got standardized and published as ISO/IEC 14882:1998 and the standard has since then been updated in 2003, 2007 and the latest in 2011.

Influences

C was a big influence as it still today is retained as a subset and C owed much to its predecessor BCPL (Basic Programming Language). In fact BCPL's commenting system // was reintroduced in C++ [10]. Another

main inspiration source was Simulia67. This language was slow but had good features for large software development. One of the features was the class concept which was borrowed and used in C++.

4.2.2 Objective C

Objective-C is a object-oriented high-level programing language that uses C with Smalltalk syntax. One major difference from C++ is that Objective-C does not call a function, it sends it a message. This makes Objective-C slower as the compiler does not make the same type of direct bindings as in C++[14]. Objective-C has backwards compatibility and all C code can be run without modifications.

An advantage of Objective C is that the syntax does not intersect with the syntax from C++. This has lead to the creation of Objective C++, which consolidates the features of all three languages [15]. Objective C++ is a pure superset of C++, so existing C++ code can be implemented without modifications.

```
obj->method(argument); // C++ syntax
[obj method:argument]; // Objective-C syntax
```

History

Alan Kay coined the term object-oriented in the 1960s and in the 1970s he developed a language called Smalltalk to demonstrate the style of programing [14]. The most common version of the language was released in the early 1980s. However Smalltalk was a high level language and could at the time only be run at expensive computers. Brad Cox liked the idea but wanted a language that could be run at all computers. His idea was to couple the high level Smalltalk with the much simpler C. In 1986 he together with Tom Love started the company StepStone. In 1988 NeXT bought a license to Objective-C and it was the main language used when they developed their NeXTSTEP operating system. The founder of NeXT was Steve Jobs. When he later went back to work for Apple again they purchased NeXT in 1997. Apple used Objective-C in their new operating system Mac OS X. In 2007 a new major release came called Objective-C 2.0.

4.3 External libraries

There is no built in support for matrix computation in C++. To support matrices external libraries must be used. One such library is the Newmat matrix library [16]. Newmat is written for scientists and engineers and enables the common matrix operations such as transpose, inverse, eigenvalues, linear equation solve etc. Newmat introduces a few new data types, different types of two-dimensional matrices and the row vector and column vector [17]. The row- and column vector is similar to the array but with the difference that matrix operations can be performed on them.

The CALFEM C++ [18] implements a subset of CALFEM subroutines using the Newmat library. The library contains functions for most of the common element types and also functions for assembling stiffness matrices. However, the library did not include a function for 2D beam elements, stiffness matrices for these elements where implemented in the same way as in the CALFEM toolbox [19].

Chapter 5

User interface development

Initially the idea was to create a simple user interface that was intuitive to the user. A manual should not be necessary for using and understanding the application. Focus was on creating an application where the user can get an understanding of the structural behavior.

5.1 Guidelines

To make it easier for a user to quickly understand how different applications work it is important to have a consistency between applications. Xcode contains several of these user interface elements and guidelines to aid in the design process. Apple has also written guidelines on how to design easy to use user interfaces [20]. How the application works should not be based on the capabilities of the device, it should be based on the way people think and work. The application should be optimized and specifically built for the device that it is running on. Aesthetic integrity is important, its not just about creating something beautiful it is about how the design integrates with the functionality of the application. The more productive type of applications generally have more subtle decorative elements compared to games.

Users enjoy direct manipulation on the screen, this keeps them engaged in the task and it makes it easier for them to understand what is happening and it gives them a sense of control [20]. Direct manipulation is a human-computer interaction style with continuous representation of objects with rapid, reversible, and incremental feedback & actions [21]. The idea is to allow the user to directly manipulate objects presented to them using actions that represent real world actions. An example of direct manipulation on the iPad is when the user pinches to zoom instead of for example tapping, as the zoom follows the finger movement.

Metaphors from the real world helps the user to quickly understand how software works [20]. A classic example is the folder. People use it in the real world to put stuff in it and it works the same way on a computer. This is used a lot in iOS, e.g. when flicking through photos or when sliding on/off sliders.

5.2 Development process

The development process of Sketch a Frame have been an iterative process. Users have been observed using the application. Observing users is a valuable input in the development process. The test users have been engineering students and employees of StruSoft, all with experience in structural mechanics.

The initial concept of the user interface was inspired by the current way of designing FE user interfaces. The interface has a tab bar at the bottom where the user defines the geometry in the first tab, and applies the forces and boundary conditions in the second and the third visualizes the results. After some design iterations, the need for different tabs for geometry, constraints and visualization felt unnecessary. A single view could handle all these tasks.

The single view approach had other problems, see figure 5.2 (1). When the result views where combined the view quickly became cluttered and complex. Visualizing the tension by coloring the elements created a

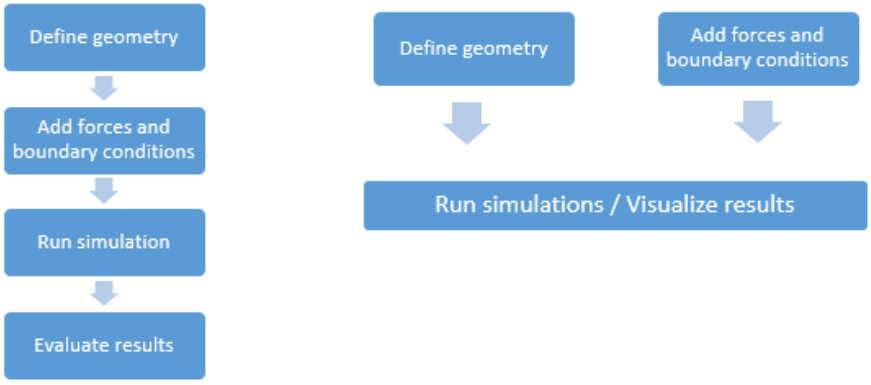

Figure 5.1: 1. Classic simulation cycle, 2. Direct manipulation cycle

problem since the elements were already colored according to their normal force. To solve this there was a need to be able to turn on and off the different result views.

A menu was created where the user could turn on and off different results using sliders, see figure 5.2 (2). The menu also created a space where other settings could be set. It created good customization for the user but it was not intuitive and when users tried the application it was not obvious that the options could be found in the popover menu. Having menu bars with options is great on a computer application, but for a touch based application the user expects the important options to be in plain view.

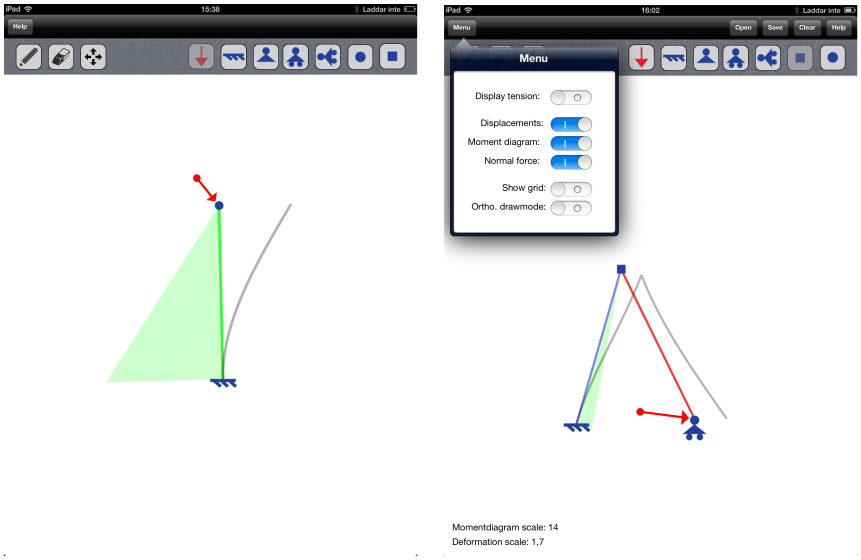

Figure 5.2: 1. Early one view version, 2. Version with a popover menu

The whole menu concept was removed for a much less complex user interface. Although it does not offer the same degree of customization it makes the application much easier to use. New users to an application expects to get started straight away and do not have a lot of patience.

5.2.1 Geometry modeling

The first tool that was developed was the pen tool. In the first version the user would touch the screen to input nodes and then swipe between them to create lines. After observing users this was quickly improved to allow users to draw lines and the application would create a start and end nodes automatically, see figure 5.3.

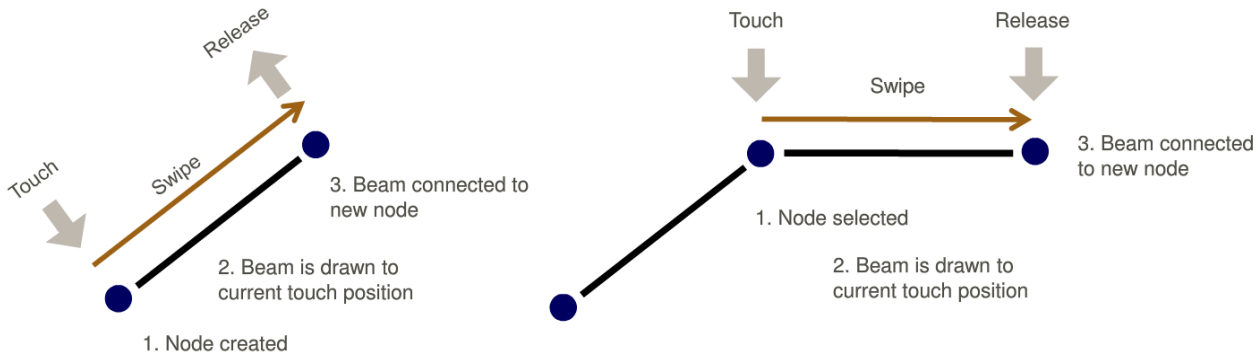

Figure 5.3: Creating an element with start and end node, and continuing to draw [22]

Most users tries to draw lines when they use the pen tool, which is not so surprising considering how the pen metaphor works in the real world. In the real world a pen is used to draw lines and not to draw nodes. Users expected to be able to draw and connect lines from and to the middle of other lines, see figure 5.4. Without this ability, users got confused as they had nodes on top of a line without connection.

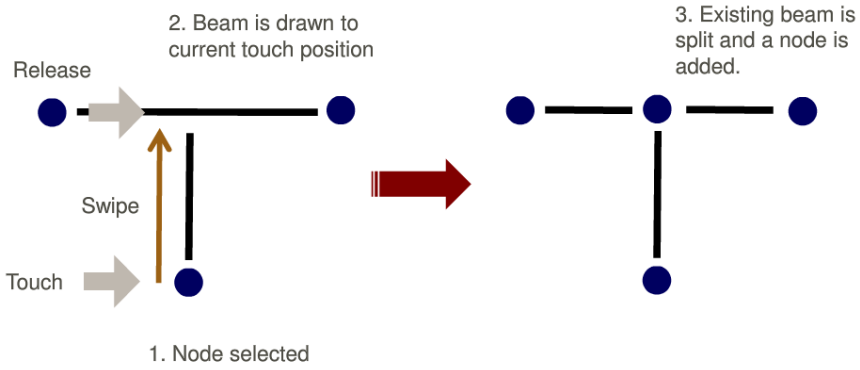

Figure 5.4: Drawing to an existing element [22]

Users also wanted to be able to draw straight lines to create geometries so a grid and an orthogonal draw mode was added. At first these options were in the popover menu. It was not obvious to users what the menu option ”ortho draw” meant. Instead of menu options, icons for these options were added to the top icon bar. Users understood the new icons of the grid and the orthogonal draw mode from start.

The first version of the eraser tool worked just as the first pen tool, to erase simply tap. But again the metaphor of an eraser made most of the users swipe with it as one would do in real world using an eraser, so the tool was rebuilt to support that.

The move tool is one of the few tools that have been successful from the initial version. A node or a force is moved by dragging it on the screen. There has been a few minor updates where the first version needed the user to start dragging the force from the tail where as it now works as long as it is close enough to the line that the force creates. Again, trying to support as many different ways of using the tools as possible, to maximize the success rate for the user.

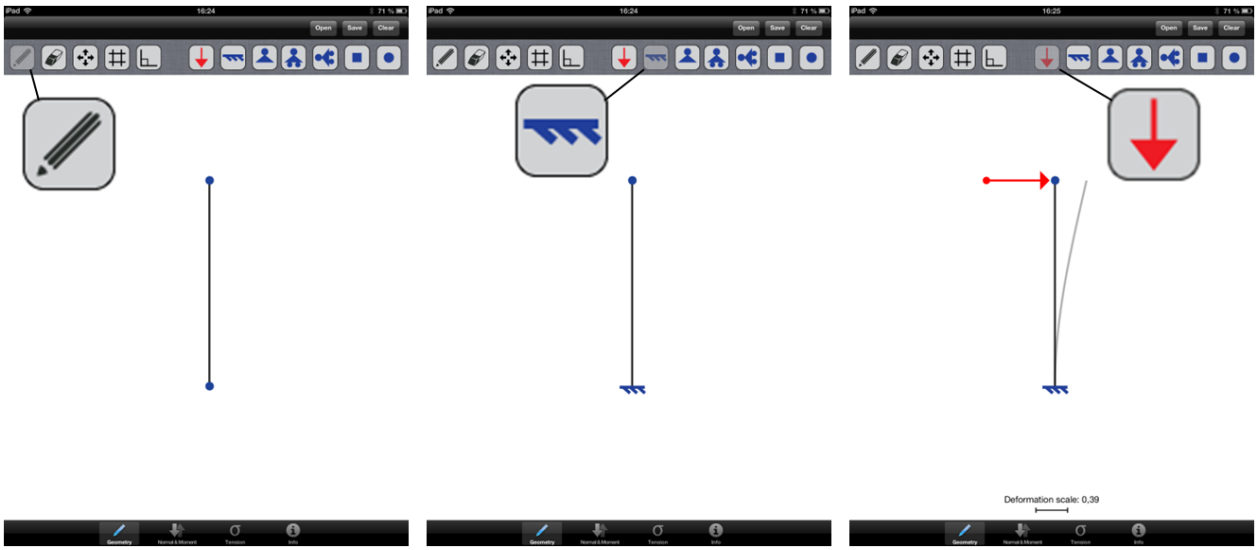

Figure 5.5: Drawing a geometry, adding boundary condition and force

Adding a force was at initially accomplished by tapping a node and a standard force was applied, but again users tried to swipe to add the force, which makes sense since after adding the force it needs to be adjusted to the users preference. Some users try to apply a force by going from the tail of the force to the node they want to apply it to. This is probably due to the fact that it is intuitive to follow the arrows direction. However there is no support for this as it makes more sense to first choose the node and then it is possible to adjust the force and see how the geometry responds. When the user have failed to add a force by swiping from the tail of the force to the node, the next thing the they try is usually the other way around. Support has also been added for adding a force in the middle of a line and it creates a moment stiff node keeping the beam continuous.

5.2.2 Boundary conditions

It was first decided how users would set the boundary conditions. Since it is beam elements every node has three degrees of freedom which makes up for eight different combinations. Two different ideas were contemplated, either the user would tap on the degrees of freedom to lock or unlock them, or to have standard boundary conditions that the user would choose from. The latter option was chosen as it results in the less complex user interface. Although, only six of the eight possible eight boundary conditions were implemented in the application as two of them (locked in moment and horizontal or vertical) do not have a lot of real world applications. Boundary conditions are set by choosing a boundary condition in the icon menu and then tap at the preferred node.

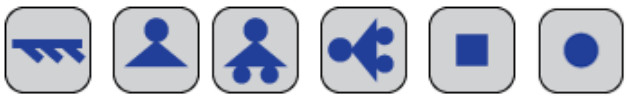

Figure 5.6: Boundary condition tools

The application do not have support for individual joint connections to a moment stiff node. A version with this function was built but it slowed down the performance of the computation as well as made the user interface more complicated and was therefore not implemented.

5.2.3 Scale

A challenging aspect in the user interface design was how to scale the deformations and the moment distribution. The initial version did not have any options for this but this lead to problems with stiff models such as trusses as the deformations were to small to be visualized. An initial version of a scale gave the users the possibility to change the scale of the force to get appropriate deformations. However, on certain models this resulted in very large moment distribution when the deformation was scaled appropriate, and on other models the opposite problem would occur. To solve this problem two different scales were introduced, one for the moment distribution and one for the deformations. This was all implemented in the, at the time existing, menu pop over. Two auto buttons were implemented that set the scales on the deformation and the moment distribution to a fixed size. However, this got all too complex for the users, even if the users understood the concept it was still needed to go into the menu and set the scales on every new model. A new button was introduced called ”Auto” that rescaled both moment distribution and the deformation. This made for a better user experience, the existentance of the menu was questioned.

Different ideas were considered, contemplating having a fixed size of the max deformation and the moment distribution. The problem with this would be that the user would lose some of the important direct manipulation feeling of how the model behaves when a force is increased or decreased. Another idea was to let the rescaling occur when the swipe motion of moving a force is finished. The pros with this is that the rescaling is automatic but still lets the user feel how the model behaves when a force is increased or decreased. The problem with this solution is to get the users to understand that the application automatically rescales.

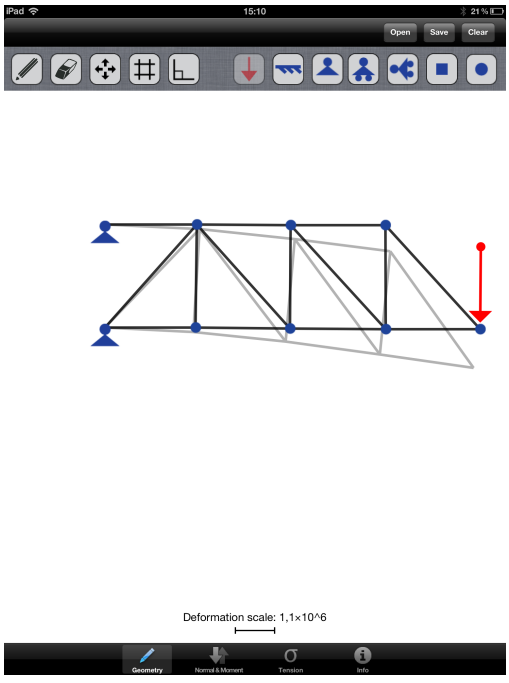

Figure 5.7: The initial version of the scale

The application is developed without use of any numbers such as material parameters. However, the need to be able to compare to different models was imperative. Two models can be drawn in the same view and compared, but if the difference between the deformations is as large as 10^9, which they can easily be, that does not work. A numeric scale label was introduced that presents the current scale for the users.

Extensive user trials were performed and it was clear that users struggled to understand how to compare different models. Users did not observe that the scale updated as the force was released. An updated version of the scale label was introduced, where it would bounce as the scale was updated. This helped getting the users attention to the scale, but they still did not understand the concept.

The problem was that the difference of a stiff model compared to a flexible could be as large as 10^9, which makes it hard to visualize. An idea was to have a textured background that would change with the scale, however, the level of the rescaling made this impossible to implement. Another idea was a logarithmic scale but that would be hard to implement due to the huge rescaling factor, and would be challenging to get the users to understand that the scale is logarithmic. The next idea was to display a colored line on the screen when the force was released, this line would illustrate where the largest deformation / moment was. Then the scale would have the same color, creating a color connection to the scale.

The winning concept was to have a scale at the bottom of the screen just as there is on an ordinary map, showing the length of the current scale. This scale is stretched and shrunken just as the maximal deformation and moment is. At this point the new tab view had been introduced which also helped as deformation scale and moment scale is never shown at the same time. Scientific numbering was also introduced for small or large numbers.

Chapter 6

Implementing Sketch a Frame

6.1 Common structure of the code

The application is created around a C++ data model that contains all the data for the current model. The data model contains functions to retrieve data for the nodes, lines, forces and boundary conditions. It also contains a function to do all the computations on the model once the model is stable. The computations are then performed in the CALFEM C++ [18] library and the result is stored in the data model. It is preferable to use C++ for the heavier operations as it performs computations faster than Objective-C [23].

The user interface in written in objective C and it can access the C++ data model functions to retrieve data. A singleton is used to pass the data around in the user interface so that all views and controllers use the same instance of the data model. A singleton is a function that makes sure an object is instantiated and then returns this object.

6.2 Drawing

The view where the model is drawn is connected to a custom written class. This custom class iterates through the data from the C++ data model and draws it accordingly. Three different coordinate systems are used throughout the process. To compute normal forces etc. a local coordinate system for the current element is used. This is transformed into a global cartesian coordinate system and stored in the data model. In the quartz [24] drawing process a flipped coordinate system is used as standard, so when drawing the data is transformed into the flipped coordinate system.

As beam elements are used the rotation, x-, y-displacements and rotation can be obtained. The initial idea was to use a built in function in quartz drawing, to visualize the deformations with perfect curves. The cubic beizer curve is a built in function in quartz drawing and uses four points to describe a curve, see figure 6.2.

However, while the rotation was computed the length of the P0->P1 and P2->P3 vectors was not. A different approach was used, dividing every line into smaller elements and calculate these nodes using the FE-model. This approach worked great and experimenting with the number of elements needed to get visually correct curves resulted in every line being divided into 20 smaller pieces.

To display the tensions the elements gradient colors are used. There is no built in support for gradient colored lines in the Xcode environment, however, there is support for gradient rectangles. The first approach was drawing rectangles instead of lines, using trigonometry to calculate the four points of the rectangle and color it with the gradient function. Drawing the gradient was a heavy operation and the iPad could not keep up with the real time calculations. Same approach was used here as with the deflections, dividing up every line into pieces and then color the smaller lines into solid colors. Dividing up every line into 40 pieces gave the same gradient effect and was more efficient.

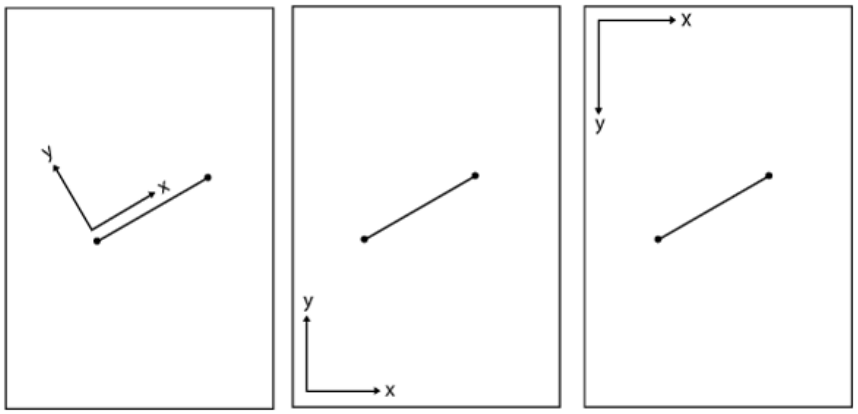

Figure 6.1: From local to cartesian to flipped coordinate system

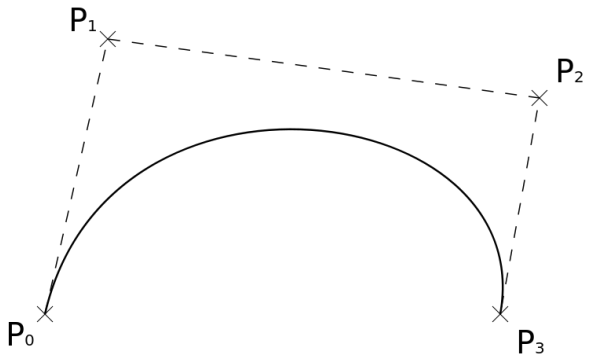

Figure 6.2: Cubic beizer curve

6.3 Data models

Two different sets of data models are used in the application. The first C++ data model holds all the current model data, but to be able to save models another data model was needed. There is a built in framework in Xcode called coredata [25]. This allows for creation of data models that stores the data on the device. Creation of these data models can be done visually and then exported to code. Connections between the different objects in the model can be created visually as well.

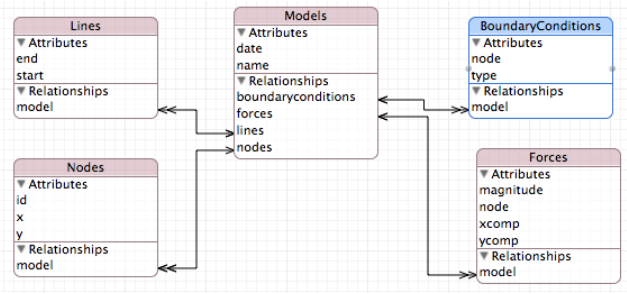

Figure 6.3: XCode data model

When a model is opened the saved model is parsed and stored in the current C++ data model, and when a model is saved the opposite happens.

Objective-C example of how the data is parsed when a model is opened:

```
+(void)readModel:(Models *) model:(UIManagedDocument *)document
{
...
    femModel->setName([model.name UTF8String]);
    NSArray *nodesInModel = [document.managedObjectContext
        executeFetchRequest:requestNodes error:nil];

    for (int i=0; i<[nodesInModel count]; i++)
    {
        Nodes *myNode = [nodesInModel objectAtIndex:i];
        femModel->addNode([myNode.x doubleValue], [myNode.y doubleValue]);
    }
...
}
```

6.4 Finite element model

As beam elements are used every node has three degrees of freedom [19]. Every degree of freedom has a corresponding displacement, to constrain a degree of freedom its displacement is set to zero. However, to create moment free nodes, hinges, a different approach is required. If two elements meet in a node and they are hinged together, a new rotational degree of freedom needs to be introduced, the node now has four degrees of freedom vertical, horizontal and two rotational, see figure 6.4. Both of the rotational nodes are unconnected but assembled into the stiffness matrix. The free rotational degrees of freedom needs to be assembled into the stiffness matrix for the eigenvalue analysis to work.

When boundary conditions are set, different degrees of freedom are hindered to move. This is done by setting the corresponding rows in the displacement vector to zero.

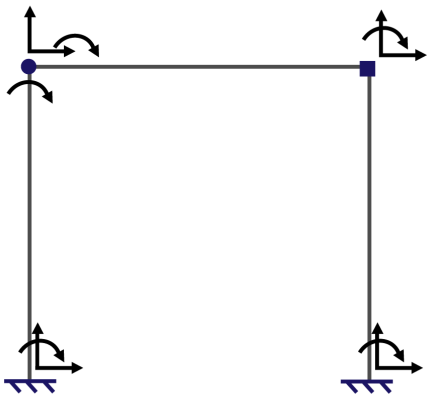

Figure 6.4: Degrees of freedom

The result in the application is only displayed once the model is stable. If the model is stable is determined using an eigenvalue analysis [26].

$$det(K - \lambda I) = 0 \ .$$

The eigenvalue vector contains the eigenvalues. Every value in the eigenvalue vector corresponds to a deformation shape. The value can be seen as the relative energy that is needed to deform into the corresponding deformation shape. A stable model built with beam elements have three zero values. As they are zero that means it takes no energy to move or deform. These three values are the rigid body motions of the whole structure, vertical, horizontal and rotation. If there are more then three zero values that means that the structure is a mechanism, and the number of zero values subtracted by three gives the degree of mechanism. In Sketch a Frame this eigenvalue analysis is performed on the reduced K matrix for better efficiency. In that case there is no zero values if the model is stable.

6.5 Element used

The following element stiffness matrix is used for the beam elements [19].

$$\mathbf{K}^e = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix}$$

6.6 Finite element solver

The element stiffness matrices are calculated in CALFEM C++ and then assembled into a global stiffness matrix using the assem command in CALFEM C++. The force vector is assembled by dividing up the forces into vertical and horizontal composants, and inserted into the corresponding row in the force vector. The global stiffness matrix is then reduced with the help of the known displacements. The following example could for example be a beam that is fixed in one end.

$$\begin{bmatrix} K_{1,1} & K_{1,2} & K_{1,3} & K_{1,4} & K_{1,5} & K_{1,6} \\ K_{2,1} & K_{2,2} & K_{2,3} & K_{2,4} & K_{2,5} & K_{2,6} \\ K_{3,1} & K_{3,2} & K_{3,3} & K_{3,4} & K_{3,5} & K_{3,6} \\ K_{4,1} & K_{4,2} & K_{4,3} & K_{4,4} & K_{4,5} & K_{4,6} \\ K_{5,1} & K_{5,2} & K_{5,3} & K_{5,4} & K_{5,5} & K_{5,6} \\ K_{6,1} & K_{6,2} & K_{6,3} & K_{6,4} & K_{6,5} & K_{6,6} \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix} \Longrightarrow$$

$$\begin{bmatrix} K_{4,4} & K_{4,5} & K_{4,6} \\ K_{5,4} & K_{5,5} & K_{5,6} \\ K_{6,4} & K_{6,5} & K_{6,6} \end{bmatrix} \times \begin{bmatrix} a_4 \\ a_5 \\ a_6 \end{bmatrix} = \begin{bmatrix} f_4 \\ f_5 \\ f_6 \end{bmatrix}$$

The Newmat library is then used to solve the equation system and the unknown displacements are obtained. The following code is used:

```
ColumnVector a = K.i() * f;   //Newmat C++ syntax
```

Then the force vector is obtained by:

$$\mathbf{K} \times \mathbf{a} = \mathbf{f}$$

The force vector is in the global cartesian system, see figure 6.1 (2), with the help of the rotational matrix it is transformed to local coordinate system, see figure 6.1 (1). The three forces obtained in every node now corresponds to the moment, normal- and shear force.

The tension is calculated by using:

$$\sigma_n = \frac{N}{A} + \frac{M}{W}$$

The material parameters used are from a IPE-80 beam. The reason for choosing this type of beam is because it gave a good balance between the tension from the normal force and the tension from the moment.

From the data model an instance of the calfemBrain is created which contains all the FE-functions. It is instantiated as a bool to return whether the model was stable or not.

```
bool CFemModel::calculate(bool geometryUpdated, bool doStaticAnalysis)
{
    this->getCalfemBrain(this);
    return calfemBrain->femCalculations(geometryUpdated, doStaticAnalysis);

}
```

6.7 Optimizing performance

To optimize the computations and minimize lag a few measures have been taken. The most critical point is when the geometry or a force is moved as real time computations are then performed. The eigenvalue analysis is one of the heaviest operations, but only needs to be performed when there has been major changes in the geometry, more than moving a node. It is only performed at these points and never during the real time computations. The stiffness matrix only needs to be assembled when there has been changes in the geometry, not when changing the direction or magnitude of a force.

Even though the iPad has limited computational power the real time calculations work well. The total time it takes to compute and draw the result grows with the number of nodes, as seen in 6.5. The computation part takes 30% to 40% of the total time, which shows that it is important to draw in an efficient way.

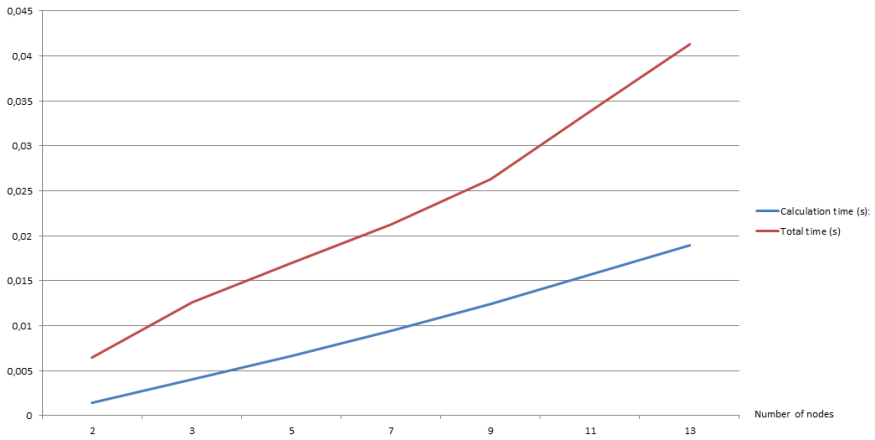

Figure 6.5: How the total time to compute and draw grows with the number of nodes

The total update time grows almost linearly with the number of nodes, there is some variance in the drawing part of the total time. This is because it depends on how the model is drawn and not only the number of nodes.

Apple's core graphics framework is used in the application to perform the drawing. This is easy to use and has all the elementary functions needed. An alternative would be to use OpenGL ES which has better performance but harder to implement [24]. Since the drawing takes up 60-70% of the time of a iteration this could help to further reduce lag.

Chapter 7

Future work

There is still a few functions that users have been requesting that has not been implemented due to time restrictions. The most requested is an undo button which is a part of direct manipulations, reversible actions. Another is if the model is mechanism then to visualize it using an animation. The computational part of this is already in place with the eigenvalue analysis but not the visual presentation.

There is a lot of potential in FE-software for touch devices as the development is this is a new area. For future work it would be interesting to try to integrate more advanced models with the touch interface, using more advanced FE-elements and examine how this would work with 3D. Continue to develop the direct manipulation methods for the user interface and explore other touch enabled devices. Future work would also need to focus on fields of application, working together with designers and architects to examine how this type of applications can be used.

Chapter 8

Discussion

When starting this project the idea was to create an application that gave the user the feeling of being able to directly manipulate a geometry to see how it behaves. It is hard to define what an application feels like but the developed application has achieved a good feeling of direct manipulation, almost making it feel playful as it is easy moving elements and forces. During the development process of the user interface a lot was learned of what works well and what does not. Some ideas that was thought to work well did work, or give the desired feeling when they were implemented, and had to be removed.

The most decisive point in the development was leaving the classical FE-interface and create a user interface that did real time computations when the model was stable. This has largely contributed to the feeling of direct manipulation as there is no need for going through different tabs to see the result.

After the launch into appstore there has been around 20 downloads daily from all over the world. It is one of the highly ranked applications when searching for ”mechanics”.

Top Markets — 83 Total

Market	Units	Change
1. USA	16	16 (100)%
2. Sweden	13	13 (100)%
3. Italy	7	7 (100)%
4. Spain	6	6 (100)%
5. China	5	5 (100)%
6. UK	4	4 (100)%
7. France	3	3 (100)%
8. Canada	3	3 (100)%
9. Thailand	2	2 (100)%
10. United Arab Emirates	2	2 (100)%

Figure 8.1: Downloads the first week

A student that of the civil engineer program got to try the application, he immediately had an idea of what he wanted to draw and he quickly sketched it even though it was his first time using the application. The application displayed ”mechanism of 1 degree” and he played around to see what he could do to solve

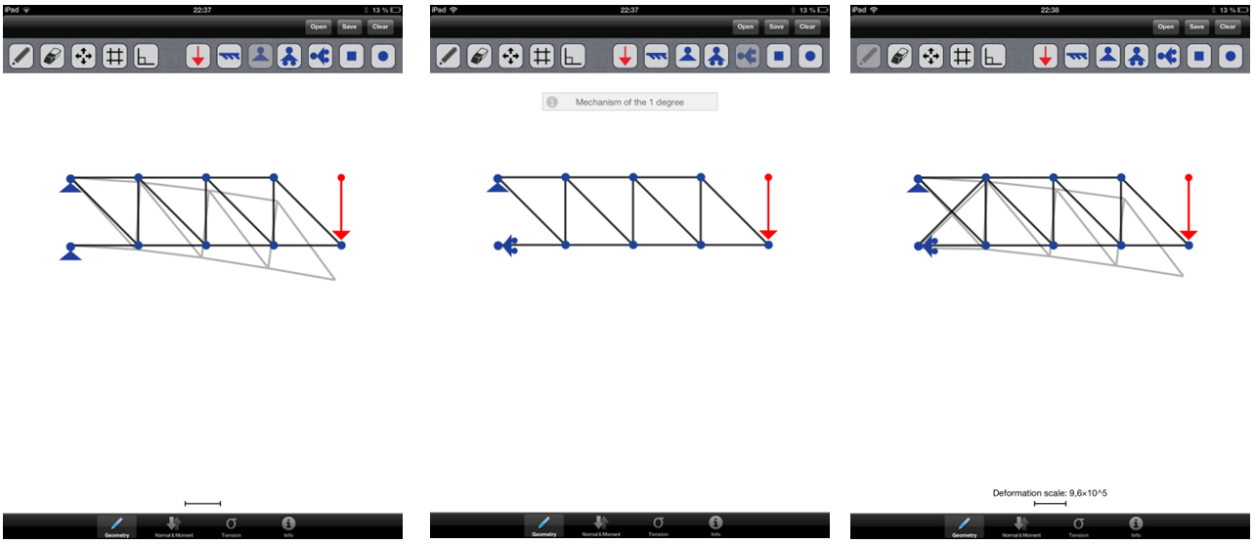

Figure 8.2: The solved hand-in assignment

the problem and he came up with a few alternatives. It turned out that he had just solved his first hand-in assignment in the finite element course, see figure 8.2 for the result. Not all interactions have run this smoothly but this shows the power in educational purposes.

Bibliography

[1] C. Steele. History of the tablet. http://www.pcmag.com/slideshow/story/285757/history-of-the-tablet/, 2011. [Online; accessed 25-July-2012].

[2] T. Hormby. The story behind apple's newton. http://lowendmac.com/orchard/06/john-sculley-newton-origin.html, 2006. [Online; accessed 25-July-2012].

[3] P. Gralla. Microsoft released its first tablet 10 years ago. so why did apple win with the ipad? http://blogs.computerworld.com/19251/microsoft_released_its_first_tablet_10_years_ago_so_why_did_apple_win_with_the_ipad, 2011. [Online; accessed 25-July-2012].

[4] S. Jobs. Apple keynote, 2007.

[5] J. Linemann *et al.* An approach to teaching architectural and engineering students utilizing computational mechanics software forcepad. *ITcon Special Issue ICT Supported Learning in Architecture and Civil Engineering*, 9:219–228, 2004.

[6] Vattenhallen science center, lth. http://www.vattenhallen.lth.se/, 2012.

[7] J. Lindemann and D. Åkesson. Using touch and multi-touch interfaces in structural mechanics software. Division of Structural Mechanics, Lunds tekniska högskola, 2012.

[8] P. Olsson. *Conceptual studies in structural design : pointSketch - a based approach for use in early stages of the architectural process.* PhD thesis, Chalmers tekniska högskola, 2006. ISSN 0346-718X; nr 2435.

[9] Intesym. Mobile apps. http://www.intesym.com/mobile.php, 2012. [Online; accessed 11-October-2012].

[10] B. Strorup. *The C++ Programming Language: Special Edition.* Addison-Wesley, 2000.

[11] Langpop. Programming language popularity. http://www.langpop.com, 2012. [Online; accessed 25-July-2012].

[12] B. Stroustrup. C++ applications. http://www2.research.att.com/~bs/applications.html, 2012. [Online; accessed 25-July-2012].

[13] B. Stroustrup. Interview bjarne strostrup. http://www2.research.att.com/~bs/bs_faq.html, 2012. [Online; accessed 25-July-2012].

[14] D. Chisnall. *Objective-C*, pages 1–9. RR Donnelly, 2011. ISBN-10: 0-321-74362-8.

[15] Cocoa dev. Objectivecplusplus. http://cocoadev.com/wiki/ObjectiveCPlusPlus, 2012. [Online; accessed 10-November-2012].

[16] R. Davies. Newmat c++ matrix library - short introduction. http://www.robertnz.net/nm_intro.htm, 2012. [Online; accessed 25-July-2012].

[17] R. Davies. Documentation for newmat10d, a matrix library in c++. http://www.robertnz.net/nm_intro.htm, 2006. [Online; accessed 25-July-2012].

[18] J. Lindemann. Calfem c++. http://www.byggmek.lth.se/personal/lindemann_jonas/programvara/, 2009. Division of structrural dynamics, Lund University.

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

[20] Apple inc. ios human interface guidelines. https://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html, 2012. [Online; accessed 1-September-2012].

[21] Ben Shneiderman. Direct manipulation: A step beyond programming languages. *SIGSOC Bull.*, 13(2-3):143–, May 1981.

[22] Jonas Lindemann. Using touch and multi-touch interfaces in structural mechanics software. Seminar NSCM25, 2012.

[23] M. Ash. Performance comparisions of common operations. http://www.mikeash.com/pyblog/performance-comparisons-of-common-operations.html, 2007. [Online; accessed 24-October-2012].

[24] Apple Inc. Graphics & animation starting point. https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/GS_Graphics_iPhone/_index.html, 2011. [Online; accessed 17-October-2012].

[25] Apple Inc. Introduction to core data programming guide. http://developer.apple.com/library/mac/#documentation/cocoa/Conceptual/CoreData/cdProgrammingGuide.html, 2012. [Online; accessed 17-October-2012].

[26] R. Cook *et al. Concepts and Applications of Finite Element Analysis*, page 564. Wiley, 2001. ISBN-10: 0471356050.