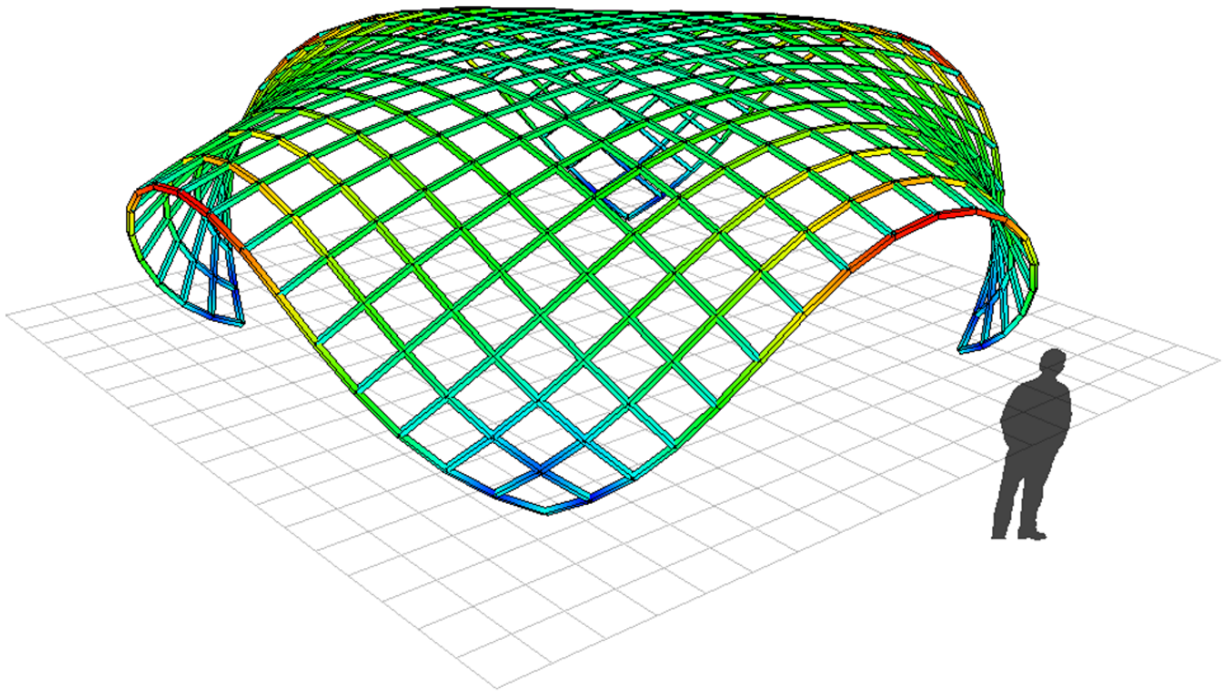
CHALMERS
UNIVERSITY OF TECHNOLOGY



Structural design and analysis of elastically bent gridshells

The development of a numerical simulation tool

Master's thesis in Structural Engineering and Building Technology

EMIL POULSEN

Department of Applied Mechanics
Chalmers University of Technology
Gothenburg, Sweden 2015

Master's thesis 2015:95

Structural design and analysis of elastically bent gridshells

The development of a numerical simulation tool

EMIL POULSEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Applied Mechanics
Division of Material and Computational Mechanics
Chalmers University of Technology
Gothenburg, Sweden 2015

Structural design and analysis of elastically bent gridshells
The development of a numerical simulation tool
EMIL POULSEN

Supervisor: Niklas Johansson, Ramböll Byggteknik
Examiner: Mats Ander, Department of Applied Mechanics

Master's Thesis 2015:95
ISSN 1652-8557
Department of Applied Mechanics
Division of Material and Computational Mechanics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: Computational model of a gridshell constructed in the developed software showing the bending process

Printed by Reposervice
Gothenburg, Sweden 2015

Structural design and analysis of elastically bent gridshells
The development of a numerical simulation tool
EMIL POULSEN
Department of Applied Mechanics
Chalmers University of Technology

Abstract

An elastically bent gridshell is a type of freeform shell structure composed by a network of continuous elements across its span. It is assembled by straight members into a two dimensional mat which subsequently is bent into a three dimensional doubly curved shell. In contrary to gridshells made of discrete members, the continuous elements can easily be connected using identical clamps or bolts. The material of the structure must exhibit a low Young's modulus - bending capacity ratio in order to facilitate the elastic formation process without breakage. Actively bent gridshells can cross large spans with a small amount of material due to its shell action. They can be considered a sustainable design option to achieve large and architecturally qualitative roof structures, especially with local timber as the material of choice. Even though the benefits of elastically bent gridshells seem apparent, only a handful large scale structures of this kind have been built so far. A reason for this is thought to be the involved design process and lack of intuitive tools.

The objective in this thesis is to formulate a design and analysis process of elastically bent gridshells. Included in this is to develop a codebase which can simulate the highly non-linear bending process with automatic supervision of the material capacity. The mechanics used is based on a nodal six degree of freedom formulation of the Dynamic Relaxation method. All equations and logics are compiled into a C# .NET class library which functions as an application programming interface (API). It is given the name EMU.dll and has been implemented as a plugin to the parametric 3D modelling software Grasshopper3d® for Rhinoceros®. In order to achieve good code design which easily can be maintained and extended, the concepts of object oriented design patterns are used.

The structural output of the developed numerical framework is compared against analytical and physical models. Through four test cases the code is benchmarked and proven to be performing accurately. As a real-time structural analysis plug-in in a parametric environment, the user can easily interact with the model during run time. Suggestions of how the developed tool fits into a bigger system of the gridshell design and analysis process are presented. The implementation of software design patterns makes EMU.dll straightforward to extend and modify to suit project specific needs.

Keywords: Postformed gridshell, active bending, non-linear finite element analysis, Dynamic Relaxation, software design patterns, object oriented programming.

Preface

This master thesis has been carried out during the fall of 2015 as part of the six year programme Architecture and Engineering at Chalmers University of Technology, Gothenburg, Sweden. As the main focus of the thesis has been to propose a workflow for the design and analysis of actively bent gridshell structures, a non-linear structural analysis software has been developed. Videos, samples and other related explanatory material regarding the software can be found on the following link:

https://vimeo.com/emlplsn

Acknowledgements

Firstly I would like to thank my examiner Dr. Mats Ander and my supervisor Niklas Johansson at Ramboll for their guidance and inspiration throughout the thesis. Special thanks to Martin Pettersson and Patrik Thorsson at Ramboll for lots of interesting conversations and feedback. I would also like to thank the following people: Jim Brouzoulis (assistant professor in material and computational mechanics), John Harding and Iain Sproat (former members of Ramboll Computational Design), Viktoria Henriksson and Maria Hult (fellow students and opponents of the thesis). Last but not least I would like to express my gratitude and appreciation to Professor Karl-Gunnar Olsson, the founder of the Architecture & Engineering program at Chalmers for the inspiration and for making this happen.

Emil Poulsen, Gothenburg, December 2015

Contents

1
Introduction

The term gridshell is often referred to as a type of freeform shell structure which is constructed of one dimensional elements instead of a continuous surface. As for ordinary shells, the stability and load bearing capacity is gained from the doubly curved geometry of the surface. Mechanically, gridshells can be perceived as shells with regular areas of void, which forces the stresses to be concentrated in the present material left. This makes a very light and efficient structure which can cross large spans with a very small amount of material. [7]

Apart from being an efficient way to achieve large roof structures, actively bent gridshells are a rational and sustainable design option to achieve elegant shells. Considering its benefits one would expect to see more of these structures. The reason for why they are rarely built is thought to be the involved design and simulation process associated with them. The formation of the initially flat grid into a three dimensional shell is a very sensitive process where the limits of the material must not be exceeded (bending capacity for instance).

The aim of this thesis is to explore and propose a suitable design process for elastically bent gridshells. A digital tool is written to simulate the highly non-linear bending process utilizing a nodal six degree of freedom (DOF) formulation of the dynamic relaxation (DR) method. The logics of the form-finding tool, which has been given the name EMU.dll, is written as an independent class library in the object oriented programming language C# .NET. The logics of EMU.dll is exposed as an application programming interface (API) which can be implemented in various contexts. In this thesis, Grasshopper3d® for Rhinoceros® has been chosen as host software for EMU.dll.

When developing software and larger sets of code, it is essential to have a strategy for the organization of it. Principles of documented software design patterns are evaluated and implemented to achieve maintainability and extensibility of the code.

Subsequent to the form-finding, a process of structural analysis in ultimate limit state (ULS) and serviceability limit state (SLS) must be executed to verify the gridshell. Design requirements concerning load carrying capacity are explored along with methods for exporting the form-found model into a commercial FEM package using COM interoperability.

1.1 Context

In this section, a contextual perspective of the subjects in the thesis is presented. Firstly, a short presentation of the concepts around actively bent gridshells is carried out to give the reader an insight of the logics and fundamentals of such structures. In the second section, three selected realized gridshells are presented to show examples of how they can be implemented. Subsequently, existing methods for the design and analysis of gridshells are presented. A short contextual introduction of software design patterns and principals is given before the chapter concludes with presenting the purpose and limitations of the thesis.

1.1.1 Gridshells and actively bent structures

As just explained, gridshells are shell structures consisting of one dimensional members instead of a continuous surface. These elements can mainly be of two kinds.[10] The first is when the structure consists of relatively short (discrete) beams or bars which start and end between the nodes of the grid. This is the most common strategy when steel is chosen as the structural material. The other way of constructing a gridshell is to use continuous members spanning between the supports. These are initially flat and bent into their three dimensional desired shape by utilizing the material's bending capacity.

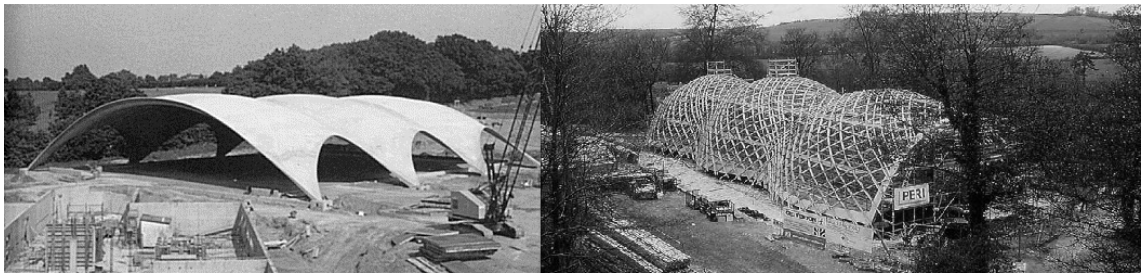


Figure 1.1: *To the left: example of a continuous concrete shell by Heinz Isler. [4] To the right: example of a gridshell (Weald and Downland gridshell, see section 1.1.1.2) [8]*

In terms of production, the elements of gridshell structures with discrete members between nodes (such as steel gridshells), must be fabricated in correct lengths and mounted between their two corresponding nodes. The geometry of these connections can be very complex, with various number of connecting members in different angles in 3D space making the production costs high. Also the assembly process usually requires some sort of false-work to support the shell while cantilevering until completion.

Gridshells with continuous members on the other hand are especially interesting if the manufacturing and construction process is considered. When the shell is constructed by slender continuous members that are elastically bent into position, a network of flat elements can be assembled on ground prior to the erection of the

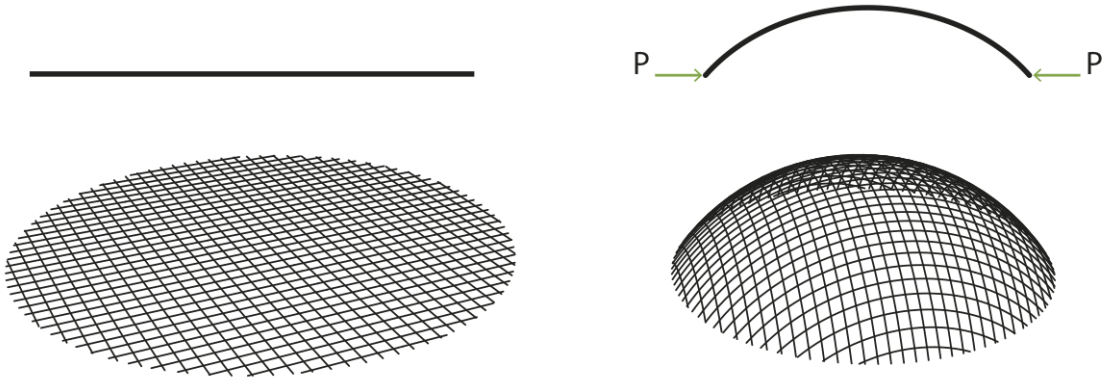


Figure 1.2: *Erection process of actively bent gridshell. The initial squares deforms into rhombuses when curvature is introduced. [16]*

structure. The connections do not need to be uniquely produced, but can be mass manufactured and added to the lattice. Mechanically, a number of criteria for the nodes connecting the members must be fulfilled to make the flat grid deformable: [7]

- The nodes lock the connecting members in translation.
- The continuous members must be able to rotate relative to each other (an extra rotational degree of freedom in each node is required).
- The connection must not substantially reduce the members load bearing capacity (by penetrating the material for instance).

To turn a directional flat grid into a doubly curved surface without changing the node position along the members is therefore only possible if the joints are rotatable. For a bidirectional grid with members arranged orthogonally, the initial flat configuration forms squares between the laths. When a Gaussian curvature $\kappa \neq 0$ is introduced these squares shear into rhombuses. The material of choice must have a low ratio between its Young's modulus and bending capacity. That means it must be possible to deform the material extensively without yielding or breaking. Since the laths seldom follow the geodesic curves on the surface, another requirement of the material is low torsional stiffness since they must rotate along their longitudinal axis. [8]

When the lattice has been erected to its desired shape, the freely rotating nodes must be constrained in order to make transmission of forces between the members possible. In its initial pinned condition, the grid can transmit forces in the direction of the laths and by out of plane bending, but not in-plane shear.[7] Diagonal stiffness must be added, which can be modelled and achieved in various ways:

- By stiffening the nodes by making them moment resistant.
- Provide diagonal bracing, either by adding cable elements or struts.
- Applying a continuous shear stiff cladding onto the grid.

The preferred bracing method is a choice influenced by a number of factors such as design intent, load conditions and given erection method.

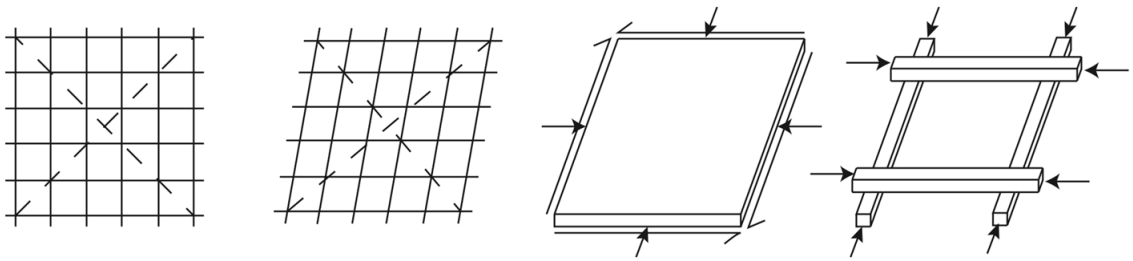


Figure 1.3: *To the left: lattice distortions. To the right: Shear stiffness is provided in a continuous shell element but not for a set of grid shell elements with rotatable joints. [7]*

1.1.1.1 Erection method

The method of raising the flat network of components into a three dimensional shell structure can be executed in different ways. A combination of external forces are applied to the structure which results in internal shaping forces within the material. As a consequence the members will deform. If the flat mat has been designed properly to match with the final shell geometry, the construction workers only need to put the beam ends to their corresponding support nodes after sufficient deformation. Four methods of achieving the deformation is presented in [12]:

- **Lift up**. To lift the structure upwards, a well anchored lifting device is needed above. This can be carried out using cranes. It is a safe method since no workers need to be below the structure during the procedure. However, a crane may become hard and expensive to transport if the building site is remotely situated. Many anchor points to the structure are needed to minimize the risk of over-stressing the material during the erection process. If the cables from the crane are not attached vertically to the grid, horizontal force components may cause extra stress in the members.
- **Push up**. Pushing the structure from the inside may be considered a method well suited for smaller gridshells. It can be carried out without any additional technology using pallets or similar objects. This method have been used for a number of small scale pavilions. On the contrary, The push up method was used for the Mannheim Multihalle gridshell using scaffolding towers (further discussed in section 1.1.1.2) - the largest actively bent timber gridshell to date. It is uncertain if this would have been accepted with today's safety regulations.
- **Ease down**. To avoid the process of lifting the gridshell upwards, the mat can instead be constructed or put on an elevated level relative its final support attachment. When this is done, the potential energy from the dead weight can

be utilized to successively remove the underlying scaffolding. Supervision of the node positions during the procedure is essential to avoid unexpected stress conditions. This was the method used in the Weald and Downland project discussed in section 1.1.1.2. As for the two previously presented methods, the ease down method may introduce large local stresses at the attachments of temporary support.

- **Inflate**. A method tested and discussed in [12] is to push up the gridshell from beneath by applying force from underlying inflated pneumatic cushions. This is an excellent method for distributing the pushing force evenly among the nodes, minimizing local stresses caused by the erection. The cushions may however become hard and expensive to produce with increasing complexity of the shell geometry. On the other hand the cushions could potentially be reused as cladding for the complete shell.

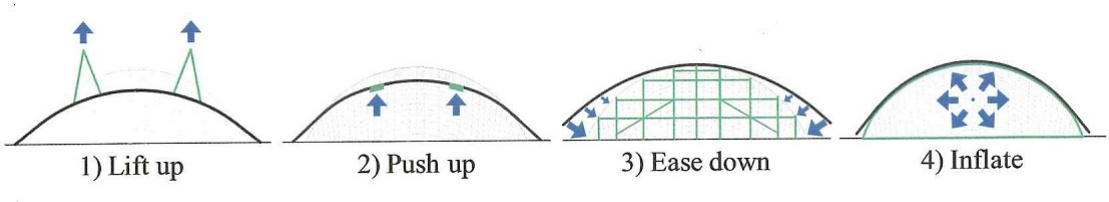


Figure 1.4: *Four different erection mehtods. [12]*

1.1.1.2 Precendent elastic gridshells

The concept of elastically bent gridshells made of identical straight members was first developed by professor Frei Otto in the 1960s. [7] For the German building exhibition at Essen 1962 professor Otto designed a small prototype which became the first of this type ever built. It had an elliptical base of 15 x 15 with a top height of 5 meter. The material of choice was Oregon pine. Since then, only a handful gridshells have been realized in fullscale. In the following subsection, three gridshells which are considered especially interesting for the thesis are presented.

Multihalle Mannheim

For the German garden festival in Mannheim 1975, a double layered timber gridshell was designed and built.[7] The concept of a membrane covering the exhibition area was first proposed by the architects but without any realistic concepts of how to solve the load-bearing system. It was initially proposed to lift the structure using balloons, but was declined by the building authorities. At this stage, Frei Otto was invited to do the structural design and the concept of an elastically bent gridshell was chosen as an appropriate solution.

The layout of the design includes two major domes; one spanning 60 meter and the other spanning 40 meter connected by pathways. Only a few small-scale pavilions of this type had been built before (including the Essen gridshell), so it was truly a pioneering project. The grid consist of a double layer system with 50 x 50 mm hemlock sections. The in-plane stiffness is achieved by pairs of 6 mm cables crossing every sixth node.

The Multihalle Mannheim roof is the largest and most complex elastically bent gridshell built to date.

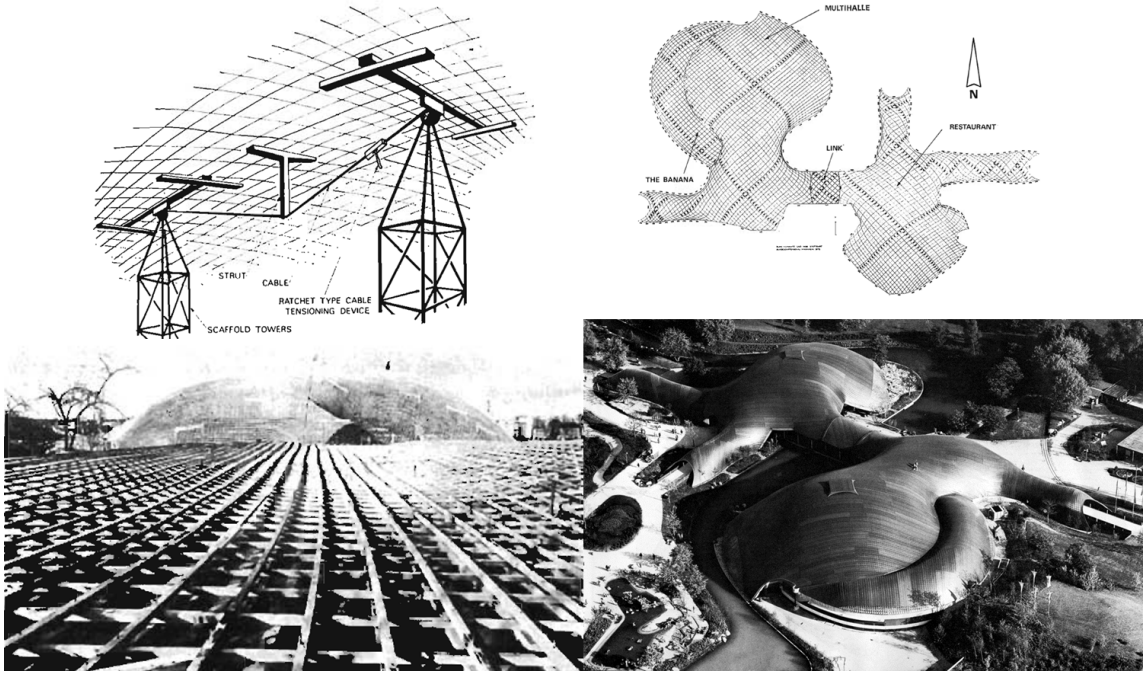


Figure 1.5: *Top left: the fork lifting system, top right: the plan, lower left: the flat lattice, lower right: birds eye perspective* [7]

Weald and Downland Gridshell

The Weald and Downland gridshell was built in 2002 for the Weald and Downland Open Air museum in West Sussex, UK.[8] The museum requested a modern timber building for the study and practice of building conservation, aligned with the timber-framing tradition in England.

The result became a triple-bulb hourglass shaped gridshell 48 meter long with a width varying between 11 and 16 meter. The internal height varies between 7 and 10 meter. The load-bearing lattice consists of 50 x 35 mm oak laths in a double layer system. These were assembled flat on an elevated height (as described in section 1.1.1.1) with 1000 mm spacing. In areas with high load concentration the spacing is decreased to 500 mm. Diagonal timber bracing were added to increase the stability which also functioned as support of the cladding. Red Cedar boards (in the lower part) and polycarbonate glazing (in the upper part) were chosen as cladding material.

The success of the project is considered to be a consequence of tight collaboration between the architects Edward Cullinan Architects, the engineers Buro Happold and the carpenters Green Oak Carpentry.

Figure 1.6: *The forming process of Weald and Downland gridshell* [14]

Soliday Pavilion

In 2011, a group of researchers from Université Paris est (including Olivier Baverel) and Ecole Nationale Supérieure d'Arcitecture de Grenoble designed a gridshell for the Soliday festival in Paris. [13] Its base is a half-peanut shape with a total area of 280 m^2. The dimensions of the structure is 26 meter in length, 15 meter wide and 7 m high. The material of choice is glass fibre composite in form of pultruded unidirectional tubes. Standard scaffolding joints were used to connect the orthogonal members. A couple of test pavilions in smaller scale with the same material had been built earlier by the team. It was lifted using two cranes and according to [13], the erection process took no longer than a few hours for 10 people.

Figure 1.7: *Erection process for the Soliday pavilion* [13]

1.1.1.3 Form-finding and design strategies

Since this thesis work is focused on the design process of elastically bent gridshells, it is necessary to study established and suggested methods in the past. These can in fact vary a lot - from fully analog to completely digital.

Physical modelling
When the Mannheim gridshell was built in the 1970s, computers were not a trivial part of neither the designers' nor the engineers' toolbox. A high level of physical modelling was therefore carried out to find the correct geometry of the shell. [7] Firstly, a hanging chain model was built to find the ideal node positions for the given boundary conditions and element lengths. The model was simplified by including every third lath. When the funicular form was found, the node positions were recorded by photogrammetric measurements. The data was registered in a computer where a program to eliminate some physical inaccuracies had been written for the project. From this initial model a more detail analysis model could be built.

Although the distance between the nodes in the hanging chain model was constant during the relaxation, no bending stiffness was accounted for. Due to the strain energy caused by the elastic deformation of the flat lattice, a redistribution of the nodes occurs. This was tested in an acrylic model using the information from the hanging-chain model. The acrylic model was also used for deformation and buckling testing.

Curvature analysis and compass method
For the Soliday Pavillion, a more geometrical approach was adopted. [13] In an initial step a NURBS surface describing a continuous representation of the shell was created. An equidistant grid was thereafter mapped onto the surface using the Chebychev net method (also known as compass method). This algorithm starts with creating two curves on the surface. These are usually oriented orthogonally relative each other but the algorithm does also work with other angles as long as they aren't oriented identically on the surface. Starting from the intersection point, the curves are successively divided in segments by a desired length. When this is done, three nodes of a rhombus are defined if the intersection point in the middle is included. From these, a fourth point can be found by creating two intersecting circles. This process is repeated for all four quadrants. See figure 1.8.

At this stage, an equidistant grid with acceptable curvature is found. The last step is to find the position of the nodes which represent the smallest bending energy of the grid. In the case of Soliday Pavilion, a dynamic relaxation engine in a commercial FEM package was used. The presented method is also described comprehensively in [14].

Pseudo pysical simulation
Previously presented methods represent form-finding strategies on two opposite ends - the first one almost completely based on information from physical modelling and

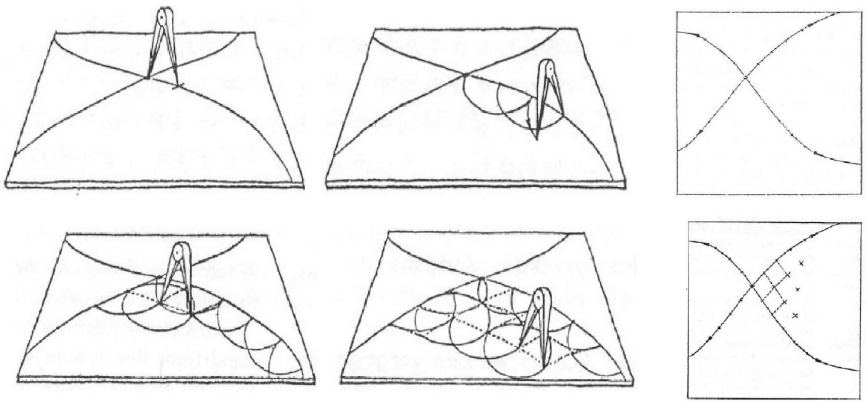


Figure 1.8: *A geometrical approach for generating equal distant grid on free form surfaces.* [17]

the second one achieved by digital means. Both have their pros and cons. With physical modelling the material properties present become an inevitable constraint in the form-finding (which is preferable). On the other hand the process of building the model can become time-consuming and the results may become hard to pass on to later design stages. Therefore, a digitalized process where real material properties are taken into account is perceived as "the best of both worlds" in the context of this thesis. This has been studied by many researchers, e.g. [9] and [3].

For elastically bent gridshells, the most critical stage for the structure is usually the erection. It can therefore be a good idea to mimic this process as accurate as possible when executing the form-finding to ensure that no overstress occurs. A method suggested in [3] is to make a total separation of the sought shell geometry and the grid itself. By doing this, the forming of the grid can start out from flat (just as it would in reality) and be bent into the three-dimensional shape by pulling it to the desired surface. The pulling force can be achieved by virtual springs between the grid and the surface. The spring stiffness which will dictate how hard the grid is attracted to the surface, can be adjusted continuously to avoid over-stressing.

After an equilibrium configuration has been found, post-processing of the geometry is needed (also described in [9]). Since the virtual springs do not exist in reality, these need to be removed and proper supports have to be inserted. The position of the supports can be defined by a clipping plane, which trims the grid in a desired height. This is necessary because the exact length of the members in the flat configuration is not known beforehand. This second step is sometimes referred to as "spring back analysis". See figure 1.10 for an illustration of the process.

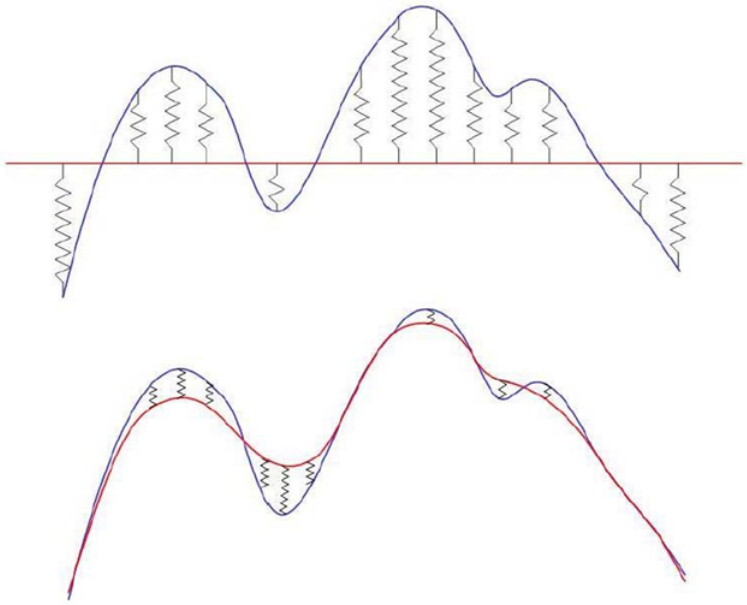


Figure 1.9: *Virtual springs from flat to target surface* [3]

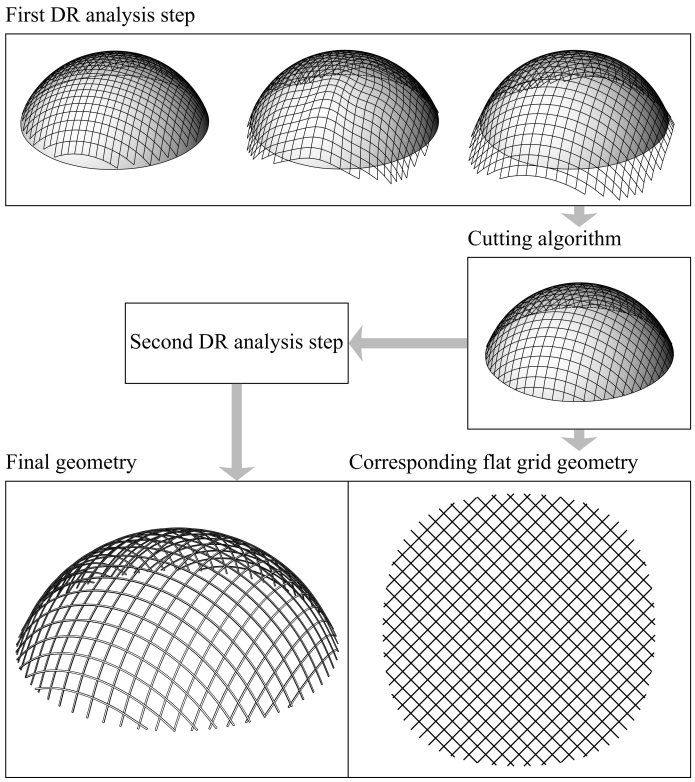


Figure 1.10: *Form-finding simulation suggested in* [9].

1.1.2 Organization of objected oriented code

Object oriented programming (OOP) is a programming concept where discrete snippets of code are encapsulated into smaller modules called *classes*. [6] These classes can represent real-life things (like a beam), and possess mainly two types of data; properties (like beam length, end nodes etc) and methods (like calculating dead load). A class defines a description or a "drawing" of something. To actually use the code defined in the class, it needs to be instantiated by creating an *object* of the class. A software developed using OOP is designed by making objects interact with each other. This concept started to developed in the 1950s and early 1960s at MIT. It is widely used today and is considered to be an efficient and powerful programming method, especially when developing larger software. Examples of languages supporting OOP is C++, Java and C#.

When building more complex software based on OOP, a major issue is the organization of the objects in order to make it as flexible, maintainable and extensible as possible. A strategy to accomplish this is the use of so called *software design patterns*, which is a set of solutions to recurring design problems in OOP. It was first popularized by Gamme et al [1]. The goal with design patterns is to facilitate the reuse of proven solutions and therefore gain following qualities:

- Improved software quality
- Reduced development time
- Provide a common vocabulary among developers.

Design patterns offer models and descriptions of how objects in a software can relate based on common design situations. However, the realization and implementation of these are entirely up to the designer him or herself.

1.2 Purpose

The purpose of this thesis is to formulate a process for the design and analysis of actively bent gridshells. Focus is put on the development of a numerical form finding tool written in the programming language C#. The following topics will be investigated:

- **Actively bent gridshells.**
  - Methods for finding the geometry of actively bent gridshells and proposals for workflow.
  - What requirements of the design are set by Eurocode 5 in the respect of timber gridshells?
  - How can these requirements be implemented in the form-finding process?
- **Geometrically non-linear finite element analysis.**
  - The limitations of linear analysis and when non-linear methods must be applied
  - Theory and implementation of 6DOF elements in dynamic relaxation
  - Numerical integration schemes

  - Verification and benchmarking
- **Organization of OOP code**
  - How can a structural analysis tool be designed to ensure flexibility, adaptability and extensibility?
  - How can the separation between geometry and structural properties take place?
  - What design patterns can be applied?

1.3 Limitations

The tool developed in conjunction with the thesis is not expected to be used for detailed structural analysis, but rather as an early stage form-finding tool. However, it should be sufficiently reliable to provide correct structural response, with output of real engineering quantities.

The code base will be developed as an independent direct linked library (dll) and is thought to work as an application programming interface (API) providing the business logic including the FE engine. Emphasize has been put on making the code base as independent as possible (i.e. minimizing third party code) hence making it possible to implement as a plugin on various CAD systems. Note that an independent graphics layer is not written. Instead the code is implemented and tested as a plugin for Grasshopper3d ®for Rhinoceros®.

The programming language used is C# .NET, which is a high level language based on automated memory allocation. Hence, manual memory allocation for code optimization (like in C++) is not possible. Also, code written using the .NET framework does not run on Mac iOS computers.

2
Theory

Since a new design tool is to be developed for the thesis, a rigid theoretical framework regarding the mathematics, mechanics and coding techniques involved needs to be established. A general description of the non-linear structural analysis is presented in the first section. This is followed by a presentation of the form-finding method implemented in the context, the so called Dynamic Relaxation algorithm including different schemes for numerical integration. Finally the concepts around how to improve code quality using software design patterns are explained.

2.1 Geometrical nonlinear analysis

2.1.1 General theory

The purpose of mechanical analysis of buildings is to understand their structural behaviour under different load conditions. With a given set of boundary conditions, external forces, elements and their connectivity, it is possible to compute the internal forces as well as the deformations of the equilibrium state. If the structure is modelled correctly, it can be simulated and tested before it has been built, which has enormous benefits. The key is to make the right assumptions. A very common assumption in structural analysis is that the structural model behaves linearly. This can be concluded in the following aspects [18]:

- The displacement-strain relationship is linear
- The stress-strain relationship is linear
- The equilibrium conditions are established at the original (non-deformed) geometry.

This implies that if the load is doubled, the deformation, strain and stress are also doubled. Geometrical linearity is valid only if the displacements and strains are relatively small. The result will be that the equilibrium equations are linear, the load-displacement relationships as well as the load-internal forces relationship are linear. In practice this means that the math is based on the starting geometry, load cases can be added together and partial factors can be applied after analysis.

Most structures behave linearly as the structural design is usually carried out to keep the deformations of a given configuration low. For instance, the vertical deflection is normally kept below L/300 for a simply supported beam (depending on the code). However, in the case of actively bent gridshells the deformations must be extensive

in order to transform a flat lattice into a three dimensional shell structure. It is therefore incorrect to assume geometrical linearity. To illustrate this, consider the following simplified case:

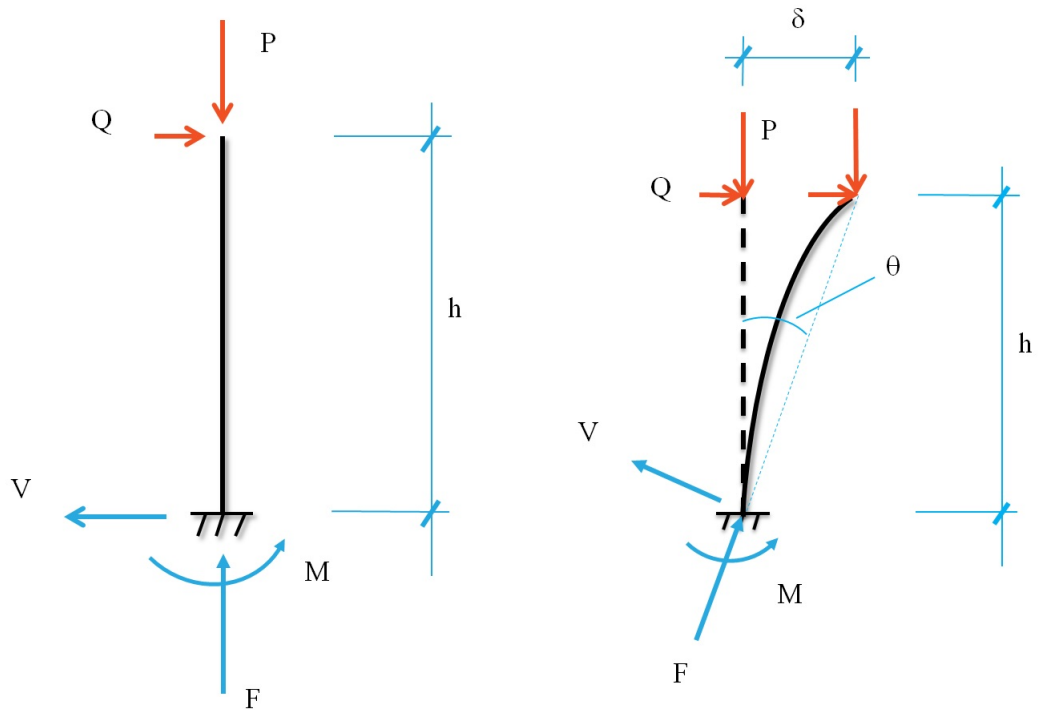


Figure 2.1: *To the left: geometrically linear structure. To the right: geometrically non-linear structure.*

For the structure on the left hand side where linearity can be assumed, the deformation due to P and Q is ignored when the equilibrium equations are established:

$$F = P \tag{2.1.1}$$
$$V = Q \tag{2.1.2}$$
$$M = Qh \tag{2.1.3}$$

If the mechanical properties and the external forces acting on the structure are given, the system can easily be solved to obtain the internal forces and deflections. But when the deformations start to get significant as in the structure on the right hand side, they will have impact on the force components. The axial load starts to increase the moment and the lateral load starts to reduce the vertical reaction.

$$F = P\cos(\theta) - Q\sin(\theta) \tag{2.1.4}$$
$$V = Q\cos(\theta) - P\sin(\theta) \tag{2.1.5}$$
$$M = Qh + P\delta \tag{2.1.6}$$

An angle θ is introduced which has to be considered when establishing the equilibrium equation. As a result, the structure does no longer behave linearly. To solve a

non-linear system like this, other methods need to be adopted.

A common method for geometrically non-linear FE analysis is incremental load stepping. The concept of this algorithm is to solve the same system of equations as in the ordinary Direct Stiffness Method (DSM); $\mathbf{Ka} = \mathbf{f}$, but with a modification on the the load component. $\mathbf{f}$ is scaled down and applied onto the structure in sufficiently small steps to make the deformations behave linearly. The procedure is iterative and can be described in the following pseudo code:

Algorithm 1 Incremental load step FE solver

```
1:  procedure SolveIncLoad                          ▷ Solve incremental load step
2:      coord ← nodal coordinates                    ▷ assign node coordinates
3:      K ← Global stiffness matrix                  ▷ assign K
4:      a ← initial displacement                     ▷ assign a
5:      f ← initial force vector                     ▷ assign f
6:      inc ← number of load increments              ▷ assign inc as number of steps
7:      iL ← f / inc                                 ▷ compute incremental load by division
8:      for ( i = 1; i < inc; i++ ) do               ▷ loop through load increments
9:          cL ← iL times i                          ▷ update current load
10:         <Solve a for K and cL>                   ▷ Ka = cL
11:         <Add a to coord>                         ▷ coord = coord + a
12:         <Recompute K based on updated coord>
13:     return coord                                 ▷ deformed model
```

The method is an extension of conventional FE analysis. Interaction with the model during the load step iteration is not possible and sufficient prescribed boundary conditions must be provided to form a solvable system of equations. These limitations constitute the demand of a more flexible method for geometrically nonlinear analysis.

2.1.2 Dynamic relaxation

An alternative approach of finding the equilibrium state of a structure considering the non-linear effects is the method of Dynamic Relaxation (DR). It is an iterative technique where the system is solved as a fictitious dynamic problem in discrete time steps. [11] By introducing the mass multiplied with the acceleration as an extra term in the static equilibrium equation, a residual force of a node can be computed at each time step. The node is subsequently moved in the direction of the acceleration. When the position of the node is updated, new elongations and rotations occur in the structure which introduces new forces and moments in the elements. These are computed and added to the adjacent nodes as residual forces which again can be translated into accelerations, velocities and finally new positions. The procedure continues until an equilibrium configuration is reached for the complete structure. A summary of the method is stated in Algorithm 2. The reader is referred to [4] for a more thorough explanation of the different steps in the procedure.

Algorithm 2 Dynamic relaxation

```
1: procedure SolveDynamicRelaxation(a, b)                    ▷ DR procedure
2:     TOL ← kinematic threshold                              ▷ assign TOL
3:     m ← FE model                                           ▷ assign m
4:     converged ← false                                      ▷ set converge flag to false
5:     while not converged do                                 ▷ loop until convergence
6:         for each node n in m do
7:             <Reset forces and moments>
8:         for each element n in m do
9:             <Compute internal forces in n>
10:            <Apply force to adjacent nodes>
11:        for each node n in m do
12:            <Compute acceleration (force, mass)>
13:            <Compute velocity (acceleration, time step)>
14:            <Compute new displacement (velocity, times step)>
15:        if kinetic energy < TOL then
16:            converged ← true                               ▷ set converge flag to true
17:        else
18:            converged ← false                              ▷ set converge flag to false
19:    return m                                               ▷ deformed model
```

An important difference between the DSM and DR is the absence of a global stiffness matrix in DR. [11] The equilibrium state of the nodes are sought locally, which indirect will result in the global equilibrium state when the solution has converged. This means that the lack of prescribed DOFs will not result in an unsolvable system. In fact, using DR even the structural behaviour of a mechanism can be simulated.

2.1.2.1 Rotation

In order to enable multi-axial analysis of beam elements, each node in a beam must possess a local coordinate system. The Z axis of these are aligned with the direction of the beam initially and gets updated at each iteration. As an initial step of calculating the internal forces and moments (line 9 in Algorithm 2), the internal rotations must be computed as a function of the two coordinate systems and the one dimensional line vector connecting the nodes. In figure 2.2 a diagram over the axes and angles present in the calculations is displayed. The rotations of a beam can be computed using equation 2.1.7 - 2.1.9.

$$\theta_{x,1} = \frac{\mathbf{y}_1 \cdot \mathbf{p}}{|\mathbf{p}|} \qquad \theta_{y,1} = -\frac{\mathbf{x}_1 \cdot \mathbf{p}}{|\mathbf{p}|} \tag{2.1.7}$$

$$\theta_{x,2} = \frac{\mathbf{y}_2 \cdot \mathbf{p}}{|\mathbf{p}|} \qquad \theta_{y,2} = -\frac{\mathbf{x}_2 \cdot \mathbf{p}}{|\mathbf{p}|} \tag{2.1.8}$$

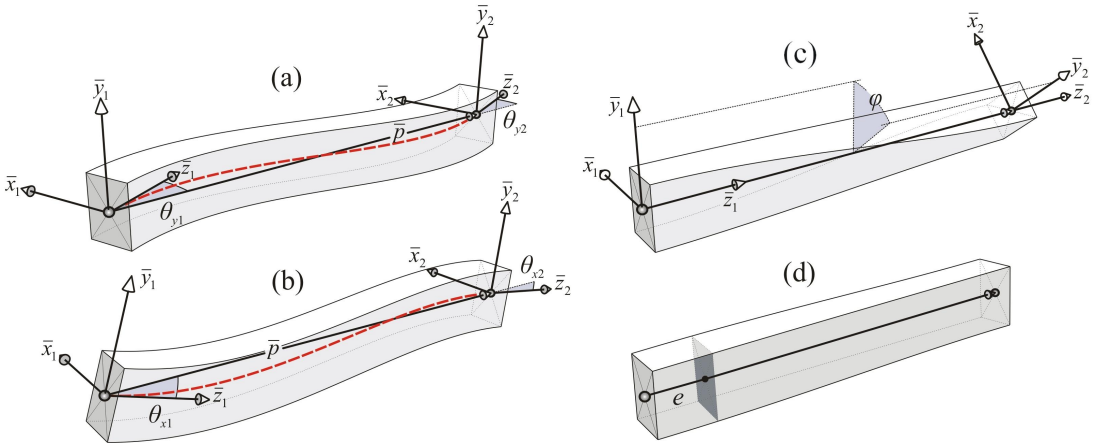


Figure 2.2: *Definition of beam coordinate systems* [9]

Where
$\theta_{x,1}$, $\theta_{y,1}$, $\theta_{x,2}$ and $\theta_{y,2}$ are the rotations around the local x_1, y_1, x_2 and y_2 axes.
$\mathbf{p}$ is a vector connecting the two end nodes expressed in coordinates of a global coordinate system. The twist angle φ is calculated in using the same arguments:

$$\varphi = \frac{\mathbf{x}_1 \cdot \mathbf{y}_2 - \mathbf{x}_2 \cdot \mathbf{y}_1}{2} \tag{2.1.9}$$

The elongation of a beam in bending is caused by a combination of elastic strain and bowing. Given the output from equation 2.1.7 - 2.1.9 and the initial length L_0, the elongation e is computed by

$$e = \frac{|\mathbf{p}|^2 - L_0^2}{2L_0} + \frac{L_0}{60}\left[4\left(\theta_{x,1}^2 + \theta_{y,1}^2\right) - 2(\theta_{x,1}\theta_{x,2} - \theta_{y,1}\theta_{y,2}) + 4\left(\theta_{x,2}^2 + \theta_{y,2}^2\right)\right] \tag{2.1.10}$$

For derivations of the presented formulas, the reader is referred to [2].

2.1.2.2 Element internal forces

When the rotations of the element coordinate systems have been computed, the normal force N, major end-moments $(M_{x,1}, M_{x,2})$, minor end-moments $(M_{y,1}, M_{y,2})$ and twisting moment (M_φ) can be calculated using the following equations [11]:

$$N = \frac{EA}{L_0} \cdot e \tag{2.1.11}$$

$$M_{x,1} = \frac{NL_0}{30}(4\theta_{x,1} - \theta_{x,2}) + \frac{2EI_x}{L_0}(2\theta_{x,1} - \theta_{x,2}) \tag{2.1.12}$$

$$M_{x,2} = \frac{NL_0}{30}(4\theta_{x,2} - \theta_{x,1}) + \frac{2EI_x}{L_0}(2\theta_{x,2} - \theta_{x,1}) \tag{2.1.13}$$

$$M_{y,1} = \frac{NL_0}{30}(4\theta_{y,1} - \theta_{y,2}) + \frac{2EI_y}{L_0}(2\theta_{y,1} - \theta_{y,2}) \tag{2.1.14}$$

$$M_{y,2} = \frac{NL_0}{30}(4\theta_{y,2} - \theta_{y,1}) + \frac{2EI_y}{L_0}(2\theta_{y,2} - \theta_{y,1}) \tag{2.1.15}$$

$$M_{\varphi} = \frac{GJ}{L_0}\varphi \tag{2.1.16}$$

Here E is the Young's modulus, A the cross sectional area, I_x the second moment of area about the local x axis, I_y the second moment of area about the local y axis, G the shear modulus and J the torsional constant.

2.1.2.3 Apply element forces to nodes

The element forces computed above shall now be applied to the nodes in the system. Since they were originally calculated based on the local coordinate system of the beams, a transformation to global axes needs to take place. Contributions from forces N_{pi} can be decomposed into global x, y, z components, while contribution from moments must be transformed explicitly using the local axes.

$$F_{1,i} = \frac{1}{L_0}\Big(Np_i + M_{x,1}y_{1,i} - M_{y,1}x_{1,i} + M_{x,2}y_{2,i} + M_{y,2}x_{2,i}\Big) \tag{2.1.17}$$

$$F_{2,i} = -\frac{1}{L_0}\Big(Np_i + M_{x,1}y_{1,i} - M_{y,1}x_{1,i} + M_{x,2}y_{2,i} + M_{y,2}x_{2,i}\Big) \tag{2.1.18}$$

$$M_{1,i} = -\epsilon_{ijk}\Big(M_{x1}\frac{p_k y_{1j}}{L_0} - M_{y1}\frac{p_k x_{1j}}{L_0} + M_{\varphi}\frac{x_{1j}y_{2k} - y_{1j}x_{2k}}{2}\Big) \tag{2.1.19}$$

$$M_{2,i} = -\epsilon_{ijk}\Big(M_{x2}\frac{p_k y_{2j}}{L_0} - M_{y2}\frac{p_k x_{2j}}{L_0} + M_{\varphi}\frac{x_{1j}y_{2k} - y_{1j}x_{2k}}{2}\Big) \tag{2.1.20}$$

Where
i, j, k is $1, 2, 3$ representing x, y, z,

$$\epsilon_{ijk} = \begin{cases} 1 \text{ if } i = 1, j = 2, k = 3 \text{ or a permutation of these values.} \\ -1 \text{ if } i = 3, j = 2, k = 1 \text{ or a permutation of these values.} \end{cases}$$

2.1.2.4 Rotations and translations

When the summation of all forces and moments from the adjacent elements relative the nodes has been performed, the calculation of new positions and rotations can be executed. The problem can be formulated as a second order differential equation which typically requires a numerical integration scheme for the general case. This involves the computation of the following steps:

- The translational acceleration based on the node's translational (fictive) mass and force
- The translational velocity based on the translational acceleration and time step
- The translations given the translational velocity and time step
- The rotational acceleration based on the node's rotational mass moment of inertia
- The rotational velocity based on the rotational acceleration and time step
- The rotations based on the rotational velocity and time step

In short converting the forces and moments in a node to translations and rotations. The process of numerical integration can be done in various ways. This is explored further in the next section.

2.1.3 Numerical Integration schemes

In structural dynamics, the spatial coordinates of a multiple degrees-of-freedom system with mass matrix $\mathbf{m}$ at a given time t can be obtained by solving the second order (ordinary) differential equation of motion [19]:

$$\mathbf{m}\ddot{\mathbf{u}}(t) + \mathbf{c}\dot{\mathbf{u}}(t) + \mathbf{k}\mathbf{u}(t) = \mathbf{p} \tag{2.1.21}$$

Where $\mathbf{u}(t)$, $\dot{\mathbf{u}}(t)$ and $\ddot{\mathbf{u}}(t)$ is the position vector and its first and second time derivatives (also analog with rotation), $\mathbf{p}$ is the external force vector and $\mathbf{c}$ is the damping.

For smaller systems in lower dimensions, an analytical solution can be found relatively easy. Either by direct integration or by adopting methods like Laplace transform. For larger systems in a three dimensional space, exact solution might be hard (or impossible) to find and numerical techniques have to be applied. This implies that $\mathbf{u}(t)$ is not found as a continuous function, but is instead solved by iterating through small time steps generating the solution as data points. For equation 2.1.21 in the application of the DR method, this is executed in two steps. First by calculating the velocity based on the acceleration;

$$v(t) = \int_{t_0}^{t1} a(t)\,\mathrm{d}t \tag{2.1.22}$$

and secondly by obtaining the position based on the velocity;

$$u(t) = \int_{t_0}^{t1} v(t)\,\mathrm{d}t \tag{2.1.23}$$

There are several ways to solve the time dependent integrals above numerically which are typically organized under two categories; explicit and implicit methods. Explicit methods calculate the next time step solely based on data from the current

position, while implicit methods calculate the next time step based on data both from the current state and the future state. [15] Normally implicit methods are numerically stable and more accurate but are computationally more expensive. In the following sections three numerical integration schemes are presented which have been implemented in the developed codebase.

2.1.3.1 Symplectic Euler

The most basic way of integrating the equation of motion is the adoption of the Forward Euler method. [20] The next position is calculated based on the current position and current velocity. Secondly the next velocity is calculated on the current velocity and the current acceleration. The scheme is here exemplified in the x direction, but is analogous to y and z (The same applies for the equations written in the upcoming two sections).

$$u_{xi}^{t+\Delta t} = u_{xi}^{t} + \dot{u}_{xi}^{t}\Delta t \tag{2.1.24}$$

$$\dot{u}_{xi}^{t+\Delta t} = c\dot{u}_{xi}^{t} + \ddot{u}_{xi}^{t}\Delta t \tag{2.1.25}$$

For the application of Dynamic Relaxation, this method usually generate poor results in terms of accuracy and stability. [20] A slight modification of Forward Euler, where the updated velocity is used for the computation of the new position, gives better results without extra cost.

$$\dot{u}_{xi}^{t+\Delta t} = c\dot{u}_{xi}^{t} + \ddot{u}_{xi}^{t}\Delta t \tag{2.1.26}$$

$$u_{xi}^{t+\Delta t} = u_{xi}^{t} + \dot{u}_{xi}^{t+\Delta t}\Delta t \tag{2.1.27}$$

This method is called Symplectic Euler (also known as Semi-implicit Euler).

2.1.3.2 Fourth order Runge-kutta

The fourth order Runge-kutta (RK4) is a method that results in very accurate approximations of any analytical ordinary differential equation. [15] The reason for this is that it takes into account the integration of the second and third derivatives. Exemplified in the x direction, a scalar version of the technique is performed as follows [9]:

$$k_1^a = \ddot{u}_{xi} \qquad\qquad k_1^b = \dot{u}_{xi}^{t+\Delta t} \tag{2.1.28}$$

$$k_2^a = \ddot{u}_{xi} + \Delta t\frac{k_1^a}{2} \qquad\qquad k_2^b = \dot{u}_{xi}^{t+\Delta t} + \Delta t\frac{k_1^b}{2} \tag{2.1.29}$$

$$k_3^a = \ddot{u}_{xi} + \Delta t \frac{k_2^a}{2} \qquad k_3^b = \dot{u}_{xi}^{t+\Delta t} + \Delta t \frac{k_2^b}{2} \tag{2.1.30}$$

$$k_4^a = \ddot{u}_{xi} + \Delta t k_3^a \qquad k_4^b = \dot{u}_{xi}^{t+\Delta t} + \Delta t k_3^b \tag{2.1.31}$$

$$\dot{u}_{xi}^{t+\Delta t} = \dot{u}_{xi} + \frac{\Delta t}{6}(k_1^a + 2k_2^a + 2k_3^a + k_4^a) \quad u_{xi}^{t+\Delta t} = u_{xi} + \frac{\Delta t}{6}(k_1^b + 2k_2^b + 2k_3^b + k_4^b) \tag{2.1.32}$$

The left column represents the integration of acceleration and the right column represents the integration of velocity. Critically speaking, this algorithm includes more steps to compute and is therefore relatively expensive compared to the previously presented integration methods.

2.1.3.3 Velocity Verlet method

Verlet integration is a family of algorithms which is very popular in molecular dynamics. [15] The concept of Velocity Verlet is to compute the current velocity based on the previous and current acceleration. Here, examplified in the x-direction:

$$\dot{u}_{xi}^{t} = c\dot{u}_{xi}^{t-\Delta t} + \frac{\Delta t}{2}(\ddot{u}_{xi}^{t-\Delta t} + \ddot{u}_{xi}^{t}) \tag{2.1.33}$$

$$u_{xi}^{t+\Delta t} = u_{xi}^{t} + \Delta t \ \dot{u}_{xi}^{t} + \frac{\Delta t^2}{2}\ddot{u}_{ix}^{t} \tag{2.1.34}$$

This method has many positive aspects in terms of long-time accuracy and speed. It does however require extra memory since the values from $t - \Delta t$ must be stored. Memory efficiency is important to consider when many particles (nodes) are present. In physics engines for computes games this becomes especially relevant.

2.2 Design requirements

In order to confirm the load-bearing capacity of a structure subjected to predefined loads, it must be checked against a set of harmonized technical rules. Eurocode is used in this report. Even though elastically bent gridshells can be implemented in many materials, the requirements for timber structures (Eurocode 5) are presented in this section.

2.2.0.4 General design values

Based on empirical tests, the characteristic material strength (which normally represents a 5 percentile ratio) of a material can be determined. Based on this value, a design value can be derived and used for calculations in the ultimate limit state (ULS) and the serviceability limit state (SLS). The general formula to calculate the design value X_d of a strength property can be calculated as

$$X_{t,0,d} = k_{mod}\frac{X_k}{\gamma_M} \tag{2.2.1}$$

where:
X_k is the characteristic value of a strength property;
γ_M is the partial factor for a material property;
k_{mod} is a modification factor taking into account the effect of the duration of load and moisture content.

2.2.0.5 Combined bending and axial compression

When timber is subjected to bending, compression and tension stresses are introduced in the cross section. These must not exceed the bending capacity of the the material. This is confirmed by equation 2.2.2 and 2.2.3;

$$\left(\frac{\sigma_{c,0,d}}{f_{c,0,d}}\right)^2 + \frac{\sigma_{m,y,d}}{f_{m,y,d}} + k_m\frac{\sigma_{m,z,d}}{f_{m,z,d}} \leq 1 \tag{2.2.2}$$

$$\left(\frac{\sigma_{c,0,d}}{f_{c,0,d}}\right)^2 + k_m\frac{\sigma_{m,y,d}}{f_{m,y,d}} + \frac{\sigma_{m,z,d}}{f_{m,z,d}} \leq 1 \tag{2.2.3}$$

where:
$\sigma_{m,y,d}$, $\sigma_{m,z,d}$ are the design normal stresses due to bending about the principal axes y and z respectively.
$f_{m,y,d}$, $f_{m,z,d}$ are the corresponding design bending strength due to bending about the principal axes.

The factor k_m makes allowance for re-distribution of stresses and the effect of inhomogeneities of the material in a cross-section. For solid timber, glued laminated timber and LVL (laminated veneer lumber) the value is determined by the members cross section. For rectangular cross-sections, $k_m = 0.7$ and of other cross-sections $k_m = 1.0$. For all other wood-based structural products, $k_m = 1.0$.

2.2.0.6 Tension

For member in tension, the following condition must be satisfied:

$$\sigma_{t,0,d} \leq f_{t,0,d} \tag{2.2.4}$$

where:
$\sigma_{t,0,d}$ is the design tensile stress along the grain;
$f_{t,0,d}$ is the design tensile strength along the grain.

For solid timber in tension or bending the effect of member size on strength may be taken into account. Since larger cross sections are more likely to contain imperfection, an increase of capacity $f_{t,0,k}$ with the factor k_h can be applied for cross section with its largest dimension less than 150 mm using the following logic:

$$k_h = min \begin{cases} \left(\frac{150}{h}\right)^{0.2} \\ 1.3 \end{cases} \tag{2.2.5}$$

where:
h is the depth for bending members or width for tension members, in mm.

2.2.0.7 Torsion

The twist of a beam calculated by 2.1.9 will induce torque (or twisting moment) calculated by 2.1.16. These will induce shear stresses in the cross section and must not exceed the material's shear limits which can be checked by equation 2.2.6.

$$\tau_{tor,d} \leq k_{shape} f_{v,d} \tag{2.2.6}$$

$$k_{shape} = \begin{cases} 1.2 & \text{for a circular cross-section} \\ min \begin{cases} 1 + 0.15\frac{h}{b} \\ 2.0 \end{cases} & \text{for a rectangular cross-section} \end{cases} \tag{2.2.7}$$

where:
$\tau_{tor,d}$ is the design torsional stress
$f_{v,d}$ is the design shear strength
k_{shape} is a factor depending on the shape of the cross-section
h is the larger cross-sectional dimension
b is the smaller cross-sectional dimension

2.3 The design of object oriented code

As presented in section 1.1.2, software design patterns were invented to offer solutions to recurring problems. These patterns are often based on more general design principles applied when building object oriented programming (OOP) applications. [1] A few of these are considered especially relevant to the thesis and are presented in the next section.

2.3.1 Design principles

Software design principles represent a set of guidelines for OOP designers to follow in the pursuit of good coding practice. Some of them are here presented in an ab-

stract fashion, without any direct relation to its implementation or language.

- **Encapsulate what varies** - *Identify the aspects of the application that vary and separate them from what stays the same.* This principle forms the foundation for many design patterns. It means that if there are parts of the code that are prone to change in the future, they should be encapsulated from other parts which stay the same. The driving achievement of this principle is that when something is modified, as little code as possible should notice. In the context of this report it can become relevant in many ways, for instance the integration scheme of the motion equations which can be implemented using various methods.

- **Program to interfaces, not implementations.** - *Interfaces are contracts and they don't know anything about implementations.* This principle indicates that code operating on an object-level should not communicate with the concrete class itself, but rather with its interface or abstract class. This makes it much easier to substitute the implementation at a later stage or even at runtime. The client code (operating code) doesn't know how the class performs its duties, it only knows what to tell it to do. This can come in handy when the relax engine operates with the finite elements - it doesn't know how it calculates e.g. its internal forces, but only knows that it *does*. A related subject is the use of abstract methods - which is provided by the super class or the interfaces and obligates the sub class to implement a function.

- **Composition over inheritance.** - *HAS-A can be better than IS-A.* By the use of inheritance a class is tied to be in a certain way, which may become disadvantageous in the long run. Another more flexible approach is the use of composition - i.e. a class is aggregated (or an instance) in another class. The relation between the geometrical object `Edge` and the structural object `FeBar` can be carried out both using inheritance and composition. To be preferred is composition since it opens up for more flexible solutions. This also avoids a lot of object casting between the sub class and super class.

- **Loose coupling** - *Strive for loosely couple designs between objects that interact.* In a project where the concepts of OOP has been adopted, objects must communicate with other. The knowledge these objects which interact have about each other is an essential topic. In general - the lesser the better. The only thing an object needs to know about another object is that it implements a certain interface. It should not care about the concrete class behind it or details of its implementation.

- **Open closed principle** - *Classes should be open for extension but closed for modification.* As discussed in [1], an active application must grow and change over time or it will *die*. Therefore, the possibility to extend the code base will be fundamental. This shall preferably be done without changing the existing classes. Much time can be spent on making the classes stable and bug-free. If

new functionality is embedded in the existing classes, there is a probability of introducing new bugs. This design principle states that it is better to extend the existing classes by making new ones. The relation between the new and old object is however up to the designer to decide. An interesting paradoxical fact on this theme (also stated in [1]) is that *"change is the one constant in software development"*.

- **Single responsibility** - *a class should only have one reason to change.* This principle indirectly states that classes must not be too large, and if they are, they must be separated properly. The way code is divided between classes is often based on intuitive decisions. Sometimes it is easy to map programming objects to real-life objects. The functionality and properties of a node and a bar would not likely be mixed in a common class in a FEM OO application. When there is no such reality-based matching, the dividing line becomes harder to draw. The single responsibility principle puts emphasis to the classes' responsibilities - if a class has two reasons to change, it should be split up into two classes.

2.3.2 Software design patterns

The art of using the principles described above in the pursuit of flexible, reusable and maintainable systems is not always straightforward. In some contexts, these principles might even strive for solutions where they contradict each other. With this background, suggested OOP systems which do not violate the principles have been developed to cope with reoccurring problems. These are refereed to as design patterns and there were originally 23 of them. [1] Presented below are two which are considered especially relevant to the thesis work.

2.3.2.1 Strategy pattern

The strategy patterns defines a family of algorithms, encapsulates them by putting them in different classes with a common interface, and makes them interchangeable. [1] One of the biggest qualities achieved by this pattern is the encapsulation of behaviours that might vary. In figure 2.3, the context class has two strategies (or behaviours) that may vary and are therefore encapsulated. In a structural analysis application, the context might be a finite element. The two strategies could represent two properties of the element - cross section and material. These can vary and should therefore be abstracted and encapsulated behind the interfaces `IMaterial` and `IXSection`. So the element aggregates the interfaces but doesn't know its details; i.e. it doesn't know what kind of material or cross section hiding behind the interfaces. This makes it very easy to change the properties of the finite element (even during runtime).

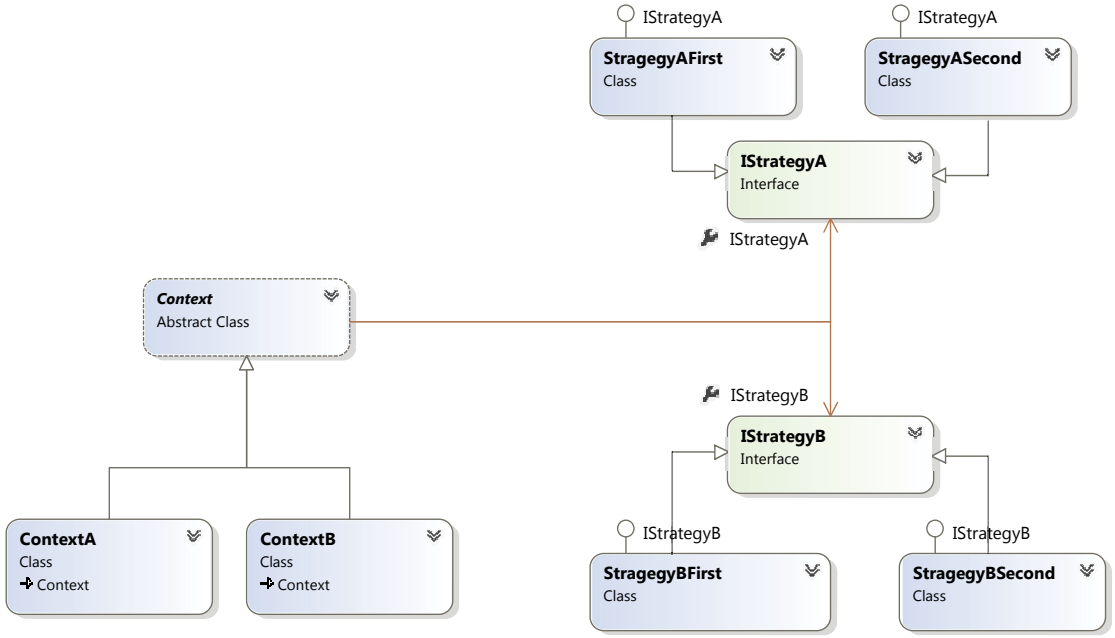


Figure 2.3: *UML Class diagram of the stategy pattern*

2.3.2.2 Abstract factory pattern

The concept of the abstract factory pattern is to define a resilient way of creating new objects. [1] Many times the instantiation of an object depends on a user choice, which often leads to floating hard-to-maintain code using conditional statements (like if or switch blocks). For the case of instantiating a finite element object, it could look like this:

```
FeElement newElem;
switch(elemType){
    case Elements.FeBar:
        newElem = new FeBar();
    case Elements.FeBeam:
        newElem = new FeBeam();
    case Elements.FeSpring:
        newElem = new FeSpring();
    default:
        newElem = null;
}
```

Serveral concrete classes can be instantiated depending on the the input enum `elemType`. Often this kind of conditional block occurs on several locations in the source code, making the process of adding more elements cumbersome. It is prone to change and should therefore be encapsulated, which is what the abstract factory pattern addresses. The code snippet above is moved to a factory class `RelaxElementFactory` or `MatrixElementFactory` (which both are derived from (`ElementFactory`) which holds the method `CreateElement(Element elementType)`;

```
ElementFactory elemFacRelax = new RelaxElementFactory();
ElementFactory elemFacMatrix = new MatrixElementFactory();

RelaxBeam beamRelax = elemFacRelax.CreateElement(Element.Beam);
MatrixBeam beamMatrix = elemFacMatrix.CreateElement(Element.Beam);
```

The code above will create two different elements, one of the type `RelaxBeam` and one of the type `MatrixBeam`.

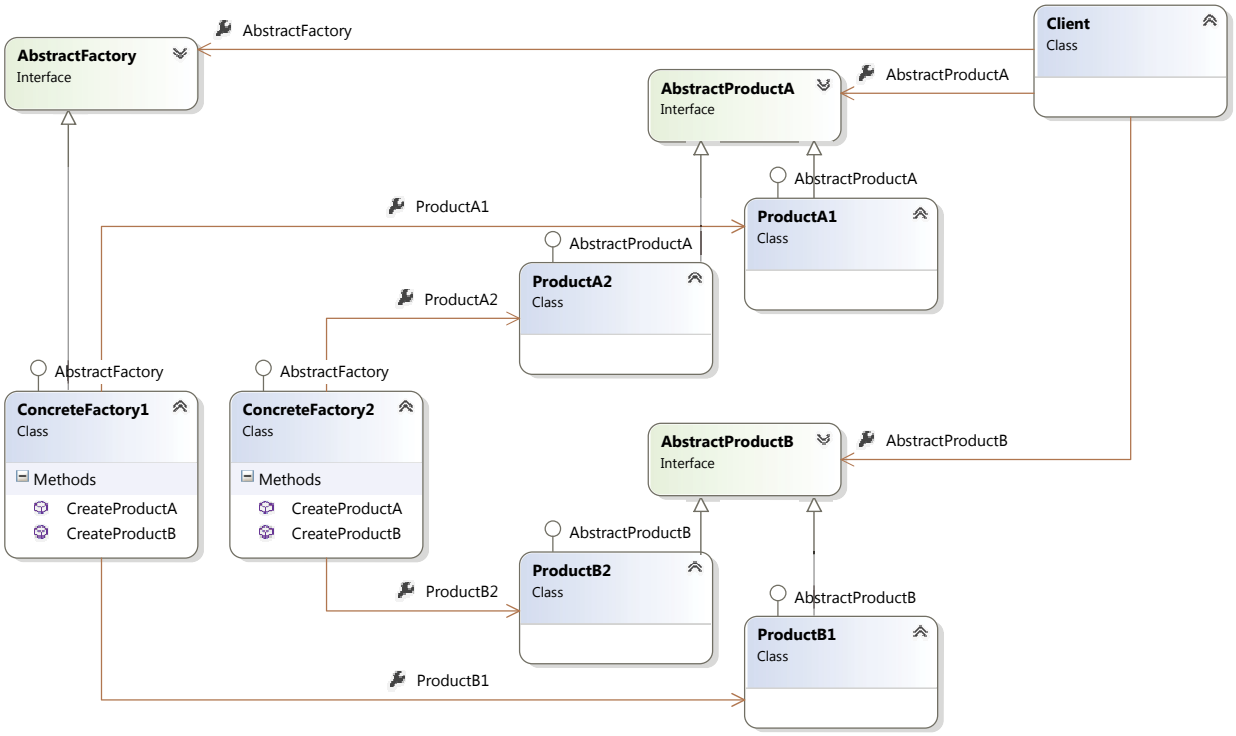


Figure 2.4: *UML Class diagram of the abstract factory pattern*

In figure 2.4 a class diagram of the abstract factory pattern is provided and the following analogies can be made:

- `AbstractFactory` represents `ElementFactory`
- `ConcreteFactor1` and `ConcreteFactory2` represent `RelaxElementFactory` and `MatrixElementFactory`
- `AbstractProductA` represents `FeElement`
- `ProductA1` and `ProductA2` represent `RelaxBeam` and `MatrixBeam`.

In short, the Abstract Factory Pattern simplifies the process of extending the application in terms of adding more elements.

3
Methodology

In this chapter the methodology of the proposed design and analysis process is explained. An overview of the approach taken is firstly presented and then followed by a more in-depth description of the different steps the method is composed of.

3.1 General approach

The method for gridshell design in this thesis starts with an accurate simulation of the bending process. An initial lattice with realistic properties and an optional target surface is defined geometrically. Boundary conditions and shaping forces are applied to the system. The equilibrium state of the geometry is then computed using the six degree of freedom Dynamic Relaxation engine written based on the theory in section 2.1.2. The nature of the algorithms allows the user to interact with the model during the simulation process.

Taking the form-found model as a point of departure, the structural verification procedure is executed through analysis in an external FEM software. A link between the form-finding software and the FEM application allows data to be exchanged automatically, without having to save and open files from the hard-drive. Using the described link, the output from the structural analysis is fed back to the form-finding application where detailing design can be made.

3.2 Workflow

In this section a complete methodology of the workflow is presented for the design and analysis of actively bent gridshells. The process is composed by an initial model generation, a form-finding simulation and global structural analysis.

3.2.1 Model generation

The first step towards the simulation is to build a representative centreline structural model for the problem at hand. The initial geometry in this example is defined by a bidirectional flat lattice which is generated using native Grasshopper3d® components. The materials and the cross sections used in the model must be defined in parallel.

The centrelines are separated into two different lists, one aligning with the X axis and one aligning with the Y axis. These can then be plugged into two different *Lath* components and marked with an index for each corresponding directions. The Lath component works like the 12 DOF beam element with the exception that it doesn't transfer moments between the lath groups. This enables the node with connecting laths in different direction to work like a pair of scissors - moments will be transferred within the laths but not between. The method adopted to achieve this is based on separating the contributing moments from the laths into different sets of lists according to [11]. As an example, moment contributions from lath group 1 will not be mixed with contributions from group 2. This results in two different residual forces which subsequently get integrated separately.

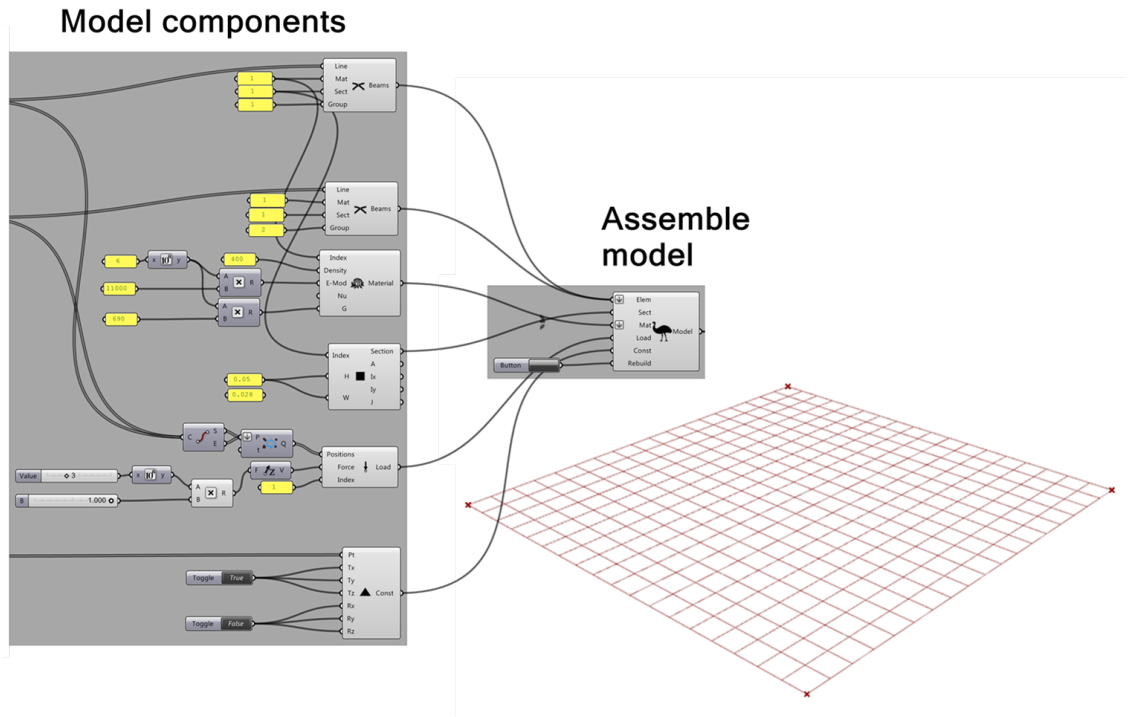


Figure 3.1: *Part of the GH definition for the model generation.*

Note that the material and cross section indices defined in the lath components must correspond to defined properties in order to map them correctly. The supports in the model in this example are defined as pinned joints (locked translation and free to rotate) located at the corners of the rectangular grid. The output from the lath element, cross section, material, forces and support components are plugged into the model component. This will assemble the pieces into a FE model and sort the topology.

The model generation procedure can be diverse depending on the topology and geometry of the desired shape. Here a simple two dimensional mat is studied. If a certain target shape is desired explained in section 1.1.1.3, the user is provided with a *SpringTargetSurf* component.

3.2.2 Form-finding

Due to the interactive nature of Grasshopper3d® and the plug-in developed for the thesis, the user is able to update the position of the support points during runtime. The driving form-finding mechanism in this example will be moving the support points, in the plane of the grid, towards the centre of the initial grid. If the grid is subjected to a triggering transversal load, imperfections will cause the grid to buckle due to emerging compression forces when the support nodes are moved. When the structure has started to buckle, the triggering imperfection load can be set to null. This is illustrated in figure 3.2.

As mentioned in section 1.1.1.3, a strategy to form-find the geometry for an elastically bent gridshell is to pull a flat grid to a predefined surface. A component for this has been developed with an optional dynamic adjustment of the pulling force taking the utilization ratio of the laths into account. When the utilization ratio is low, the pulling force is large and vice versa. This ensures the curvature of the mat to never exceed its corresponding stress limit of the material. Material data such as Young's modulus and bending strength can be imported from EC5 if timber is the material of choice.

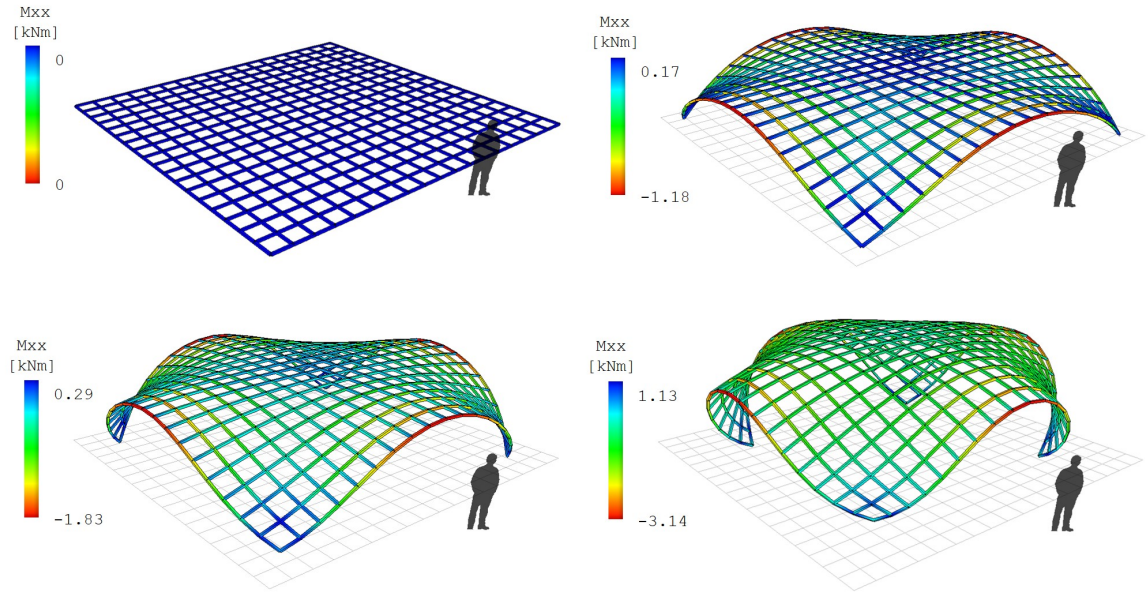


Figure 3.2: *The form-finding process of the lattice driven by pushing the end nodes towards the centre. Bending moments are supervised realtime.*

3.2.3 Global structural analysis

So far there are no external forces applied on the structure apart from the forces keeping the supports focused towards the middle which causes the deformation. To verify the safety of the structure, it must be analysed with respect to the actions subjecting the structure during its lifetime according to appropriate building codes.

On element level stresses must not exceed the limits of the material. Global stability and dynamic properties are also important to check.

3.2.3.1 Ultimate limit state analysis

As previously mentioned, the ratio between the bending stiffness and the span must be small during the erection process of the structure. This is achieved by having cross-sections with a relatively small height. This may become a disadvantage during the buildings life-time when the elements must carry load without risking global buckling or becoming overstressed. If timber is used, the introduction of a second layer of laths is a solution. They are connected to the primary laths without coupling the two cross-sections in bending making the height of only one lath being structurally active during the erection process. Upon the completion of the formation, these two layers can be coupled to get a composite cross section with a substantially larger second moment of area. [9] [8] This is made possible by tightening the node connection and by the installation of shear blocks which enable the transfer of shear between the parallel layers. During the forming process, rotations are allowed in the nodes which structurally makes the grid a mechanism. In order to achieve full shell action, these joints must be tightened and additional diagonal stiffness (like bracing) must be added.

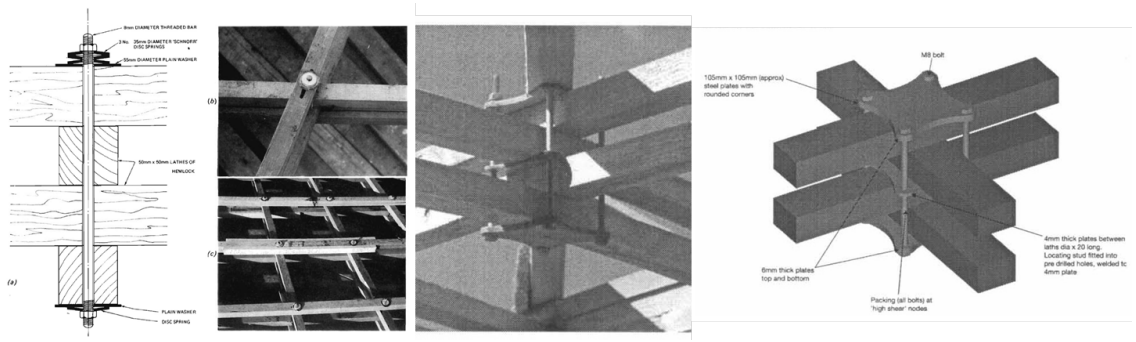


Figure 3.3: *Connection details of double layer gridshells.* [7]

Appropriate load case combinations based on prestress, dead load, imposed load, environmental loads etc. must subsequently be identified and analyzed in relation to the structure. The resulting stresses can be compared against the criteria presented in section 2.2 to verify the elements' load bearing capacity. Notable is that relaxation in the bent material due to creep may reduce the internal prestress over time. For timber, a reduction up to 0.5 can occur. [14]

3.2.3.2 Buckling analysis

As for all thin shell structures with a low cross-section height compared to its span, prevention of global buckling due to asymmetrical loads is a fundamental design criteria. As further described in section 4.2.3, buckling loads and buckling modes can either be found using eigenvalue analysis (like linearized prebuckling) or by adopting iterative techniques. In the case of the current version of EMU.dll, eigenvalue analysis of the structure is not possible since there is no global stiffness matrix.

On the other hand, buckling behaviour of an overloaded structure analyzed using the 6DOF DR implementation is solved naturally. This can be explained by the minimum energy state buckling modes represent which is the driving goal of DR.

3.2.3.3 Global structural analysis using Autodesk Robot® using COM interoperability

For applications developed by large software companies like Autodesk®, it is fairly common to find an associated application programming interface (API). An API is usually a thin layer of high-level code that allows programmatic communication with the application core for a given software. This allows users to automate certain aspects of the application, i.e. control the program from a script instead of clicking on buttons. APIs are essential in terms of plugin development.

Using the API developed by Autodesk for their FEM application Robot®, a W.I.P. link (COM interoperability) from Grasshopper3d® has been developed by engineers at Ramboll®. The link opens up possibilities to automate the procedure of structural analysis of a form-found gridshell. The entire Grasshoppr3d®model can be automatically be generated within seconds. The analysis results from Robot®can be fed back to Rhinoceros®as a foundation for detailed structural design. The usage of the link in action is illustrated in figure 3.4.

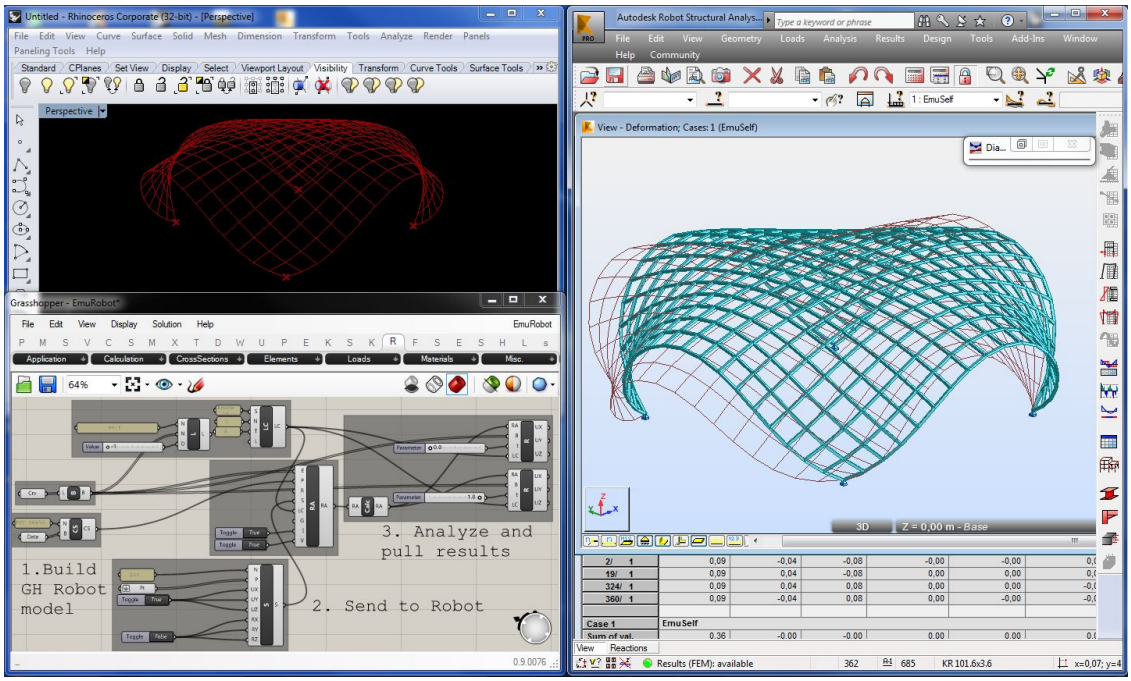


Figure 3.4: *Generation of FE model in Autodesk Robot through COM interoperability.*

4
Results

In this section the results of the proposed method is presented. An overview of the code base written will be explained including design philosophies and class diagrams. The implementation of the code base in Grasshopper3d®will be explained subsequently.

Three test cases are set up to benchmark the validity of the programs output by comparing with analytical solutions from literature. Subsequently, a design case study of more geometrically complex nature is performed. It aims to verify the proposed workflow presented in the report, including the design phase, geometry generation, form-finding and lastly structural analysis. Also, a physical scale model is built for comparison and verification.

4.1 EMU.dll

All code written throughout the thesis work has been collected and stored in a single dynamic linked library (dll), which has been given the name EMU. The programming language chosen is C# .NET; a modern high-level language well suited for the application in this context. The integrated development environment (IDE) chosen is Visual Studio Express, which is Microsoft's own platform free of charge.

Considering the vast amount of logic required to achieve the desired functionality of EMU.dll, code management in terms of organization and structure has been essential. As presented in section 2.3, the organization of object oriented code is a well studied topic and solutions to reoccurring problems have already been established many years ago, so called software design patterns. These have been studied and implemented when considered appropriate in EMU.dll. Apart from the relation between objects, the codebase has been categorized into a number of main branches (so called namespaces in C#). Most relevant to present in this chapter are the three following:

- Geometry namespace
- Structural namespace
- Structural.Relax namespace

These will be presented in the following sections.

4.1.1 Geometry namespace

The geometry namespace contains classes which solely deal with geometrical operations. They have no intelligence about FEM nor Dynamic Relaxation. It has been written in such a way that it can be used in other non-structural contexts as well. The mesh is the central top-level object which aggregates collections of its components - the faces, edges and vertices (See figure 4.1 for a class diagram). An important responsibility of the geometrical components is storage of topology information; an arbitrary vertex in the mesh knows its connecting edges, adjacent faces and surrounding vertices. The same intelligence holds for faces and edges. This makes lookups and queries fast. The actual geometrical properties of the objects (such as Cartesian coordinates) are encapsulated in custom created value types, so called *structs* in C#, which don't hold any information about the topology. `EmuVector`, `EmuPlane` and `EmuPoint` are examples of structs.

All geometrical objects have a corresponding interface to separate the out-facing abstraction from the actual implementation. This makes the process of exchanging the implementations smooth. For more complicated geometrical objects such as NURBS surfaces, logic from third party assemblies can be used in EMU.dll without being dependent on them using the Adapter pattern. In this case an interface has been created in EMU.dll (such as `IEmuSurface` representing the *target* in the Adapter pattern) which offers rather sophisticated methods such as computing normals and closest points on surfaces. This interface can later be implemented as a concrete class (the *adapter*) in a context where EMU.dll is used. If the context happens to be Rhino (as in this case) all requests of `IEmuSurface` are delageted to a Rhino surface object (the adaptee) which is aggregated in the *adapter*. A goal has been to make EMU.dll completely independent of any third party code (not including the .NET framework).

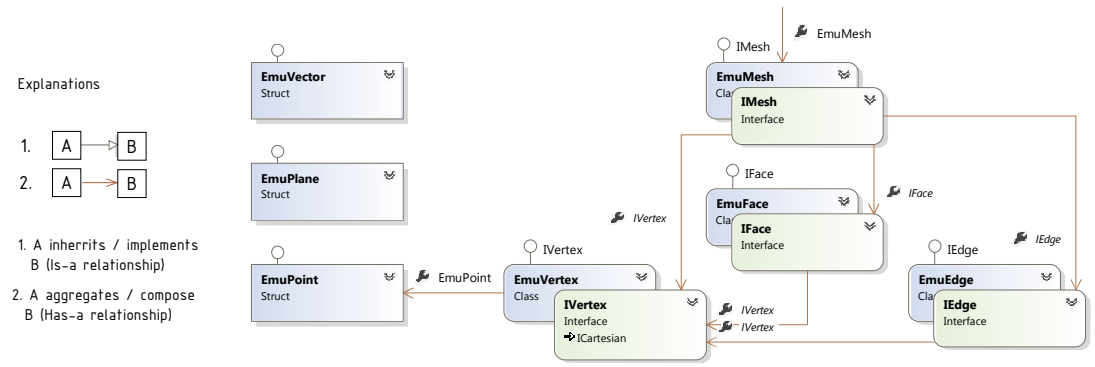


Figure 4.1: *Class diagram of the geometry namespace. A central mesh is the top-level object which aggregates face, vertex and edges. See appendix 1 for the complete class diagram.*

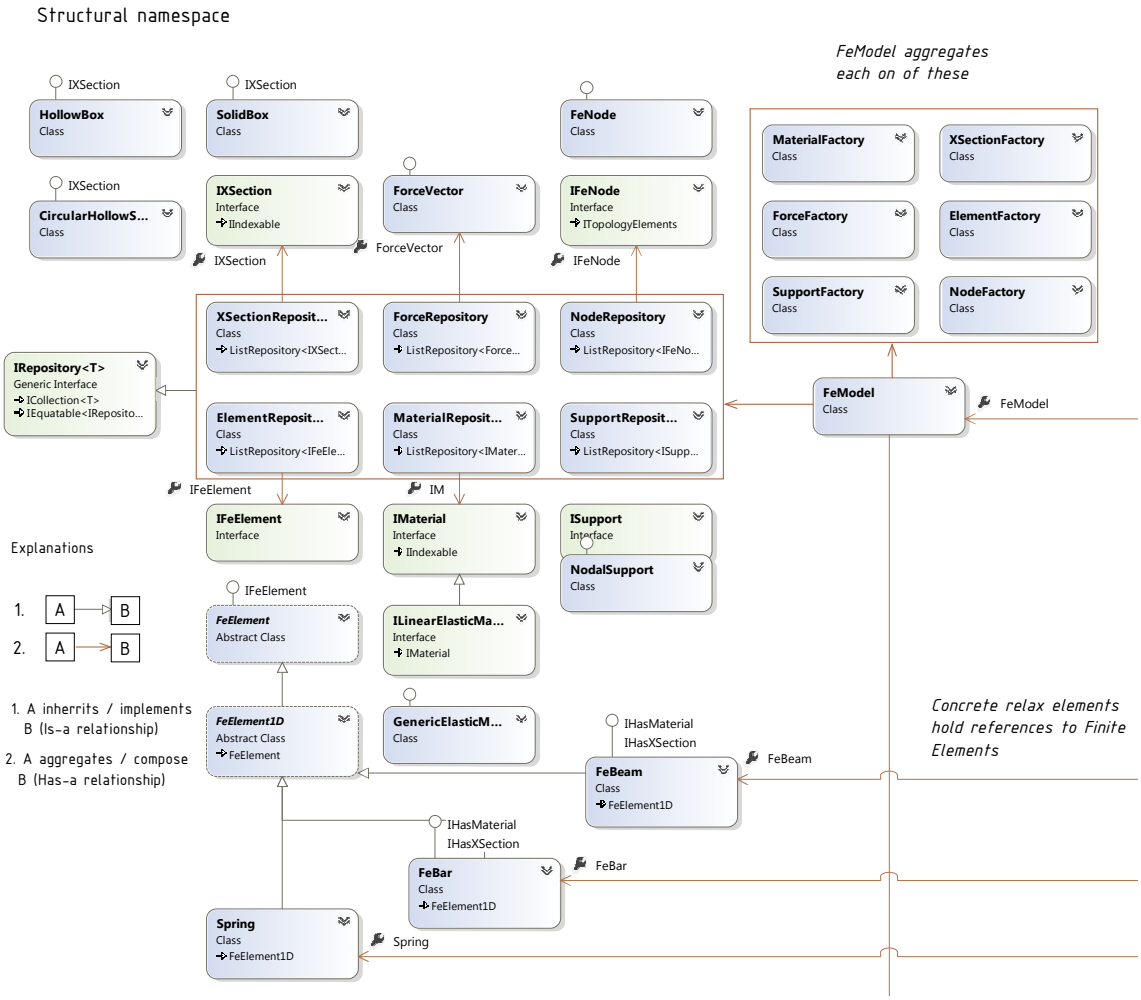


Figure 4.2: *Class diagram of the structural namespace. See appendix 1 for the complete class diagram.*

4.1.2 Structural namespace

In the structural namespace, all classes related to the modelling of a structure are collected. As seen in the class diagram in figure 4.2, the central object is the `FeModel` which holds all information needed to perform an analysis. The components of the model (elements, nodes, materials, cross sections, external forces and supports) are collected in repository classes which don't expose their underlying data structure. By communicating with the repositories through the interface `IRepository`, the idea is being able to switch its implementation and not fix arrays/lists as the only option. For instance, searching for the closest node in a structure given a single point can be made much faster if the nodes are stored in a space partitioning data structure (such as k-dimensional tree) instead of a single array. In that way, the search algorithm can discard large chunks of data and find the sought node with $O(\log n)$ complexity instead of $O(n^2)$ complexity. [5]

The elements themselves store a reference to their geometrical representations. A

`FeBeam` has a corresponding `EmuEdge` and can access the topological information through it. Other element specific properties such as material and cross section are aggregated on an abstract level in the element class. Following the concept of Strategy pattern (see section 2.3), this makes it possible to exchange properties of the elements at runtime (changing materials and cross sections for instance).

The process of creating structural elements has been encapsulated in specific classes according to the abstract Factory pattern (see section 2.3.2.2). All factories inherit from the same supertype and are aggregated in the `FeModel` class.

4.1.3 Structural.Relax namespace

An important matter during the development of EMU.dll has been to make the relation between the structural model and the solver loosely coupled, meaning that the method of finding the equilibrium configuration of a given structure is not specified within the model itself. The Relax namespace, which actually lives inside of the Structural namespace, provides all classes needed to perform dynamic relaxation of the model. The relax specific data for the elements (velocities, accelerations etc.) is stored in separate relax objects. Note that these are not extensions of their corresponding elements in the Structural namespace. Instead an aggregation relationship is chosen. For instance the `RelaxBeam` does not inherit its properties and methods from `FeBeam`, but encapsulates an instance of `FeBeam` and can in that way reference its data and functionality. The reasons for why a *HAS-A* relationship is preferred over an *IS-A* relationship are many, but central is the avoidance of copying data across the namespaces by typecasting. See figure 4.3 for a class diagram.

Since a separation between the analysis results and the model components are desired, the relax objects do also store the results from the relaxation process. The relaxation engine is written in a way that it communicates with the elements on an abstract level. As a consequence, adding new element types is possible without having to rewrite the relaxation engine. Also, the integration scheme is encapsulated in an integrator object. In the `Step()` method, which is called from the `RelaxSolver` class, the `IntegrateTimeStep(IIntegrator integrator)` method is invoked in the nodes individually:

```
public override void IntegrateTimeStep(IIntegrator integrator)
{
    integrator.IntegrateTranslationalDofs(this);
    integrator.IntegrateRotationalDofs(this);
}
```

Here, `this` refers to the node object because the function is stored in the node class. As an input argument the chosen integration object is delivered behind the interface `IIntegrator`, which makes it possible to easily change or implement new integration schemes.

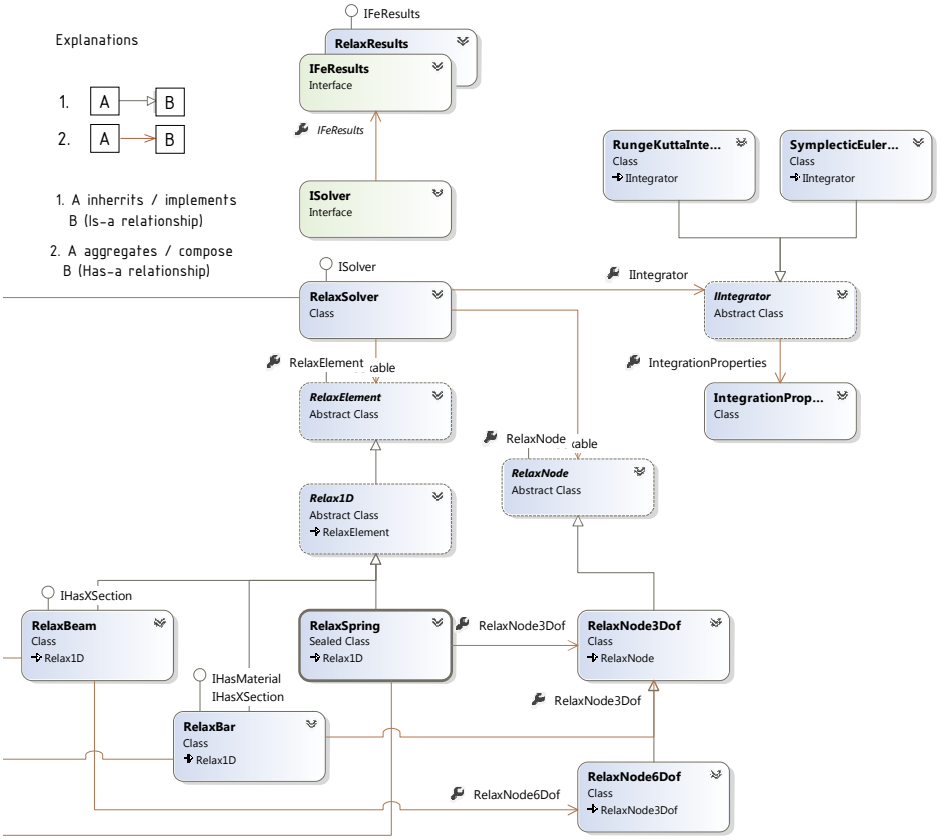


Figure 4.3: *Class diagram over the Relax namespace. See appendix 1 for the complete class diagram.*

4.1.4 Implementation in Grasshopper3d®

As stated before, EMU.dll is a class library which holds algorithms and logic for geometry and structural analysis. It is not a standalone executable software which facilitates a graphical user interface (GUI) to preview models in, but rather an application programming interface (API) developed to be implemented in other software. For this thesis, the graphical algorithm editor Grasshopper3d® (GH) for Rhinoceros® has been chosen as host application. GH is a suitable program for parametric design, which lets the user generate geometries rather than drawing them as in ordinary computer aided design (CAD) software. A model is built by connecting components which encapsulate functionality or objects. By using the Rhino software development kit (SDK) *Rhinocommon* and the Grasshopper SDK, it is possible to program custom components collected in a Grasshopper Assembly (gha). In this way, it was possible to implement EMU.dll as a plugin for Grasshopper (EMU.gha) which essentially is composed of a number of components.

In the current version of EMU.gha (v.1.0.0.0), the components are organized in 7 categories. They can be seen in figure 4.4 under the EMU tab:

- **Elements:** Finite elements such as beams, springs and bars.
- **Load:** A set of load components - point loads, transversal line loads etc.

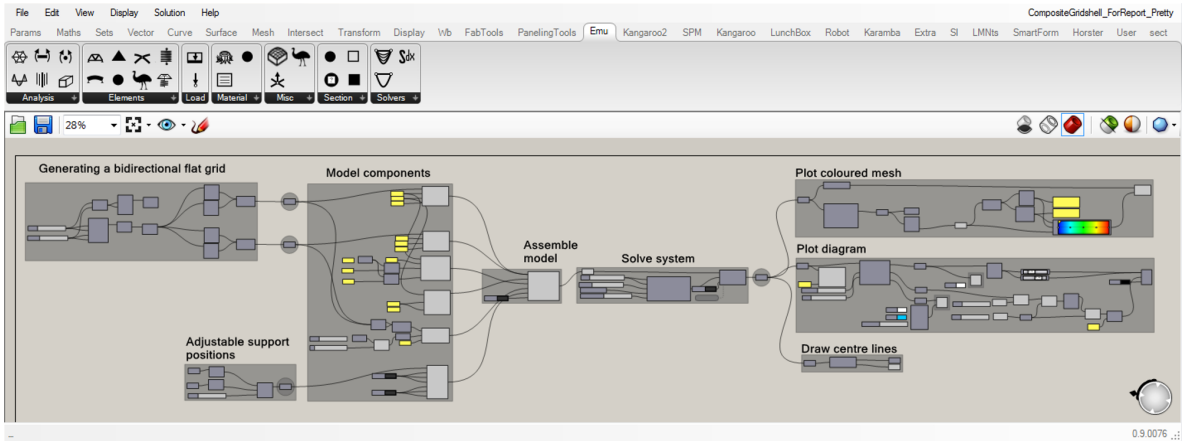


Figure 4.4: *A typical setup of Grasshopper components. The geometry is generated using native GH components, which are converted into model components. These are assembled, solved and subsequently analyzed.*

- **Material:** Currently available is generic linear elastic material and timber.
- **Section:** Components for creation of cross section objects.
- **Solvers:** To solve the structural system.
- **Analysis:** Components to extract data from the analysis.
- **Misc:** A range of utilities, such as a Chebyshev net generator.

The strength of Grasshopper as a host for the code written in the thesis is without a doubt the parametric environment it exhibits. Grids and geometries can easily be generated by connecting native components which later are used as input to EMU.gh. The plugin takes the geometrical information and sends it straight into EMU.dll where the structural model is built and analyzed. The logic and algorithms of EMU.gha is entirely referenced to the EMU.dll.

4.2 Benchmarking

In order to verify the correctness of the algorithms written to perform the structural analysis, three test cases have been set up, analyzed and compared with their corresponding analytical solutions. The first two test models deal with linear analysis and the last system aims to test the nonlinear behaviour of the structure.

4.2.1 Simply supported beam

The bending moment diagram for a simply supported beam with a uniformly distributed load follows the shape of a parabola (see figure 4.5) with its maximum value in the middle ($M(L/2) = M_{max}$) and zero at the ends ($M(0) = 0$, $M(L) = 0$). [21] The maximum moment M_{max} and the maximum deflection w_{max} can be calculated using the two following trivial formulas:

$$M_{max} = \frac{WL^2}{8}, \tag{4.2.1}$$

$$w_{max} = \frac{5WL^4}{384EI} \tag{4.2.2}$$

where
W is the uniformly distributed load [N/m], L is the span of the beam [m], E is the Youngs modulus in the material and I is the second moment of area for the cross section used.

In this test, the following properties of the beam will be used:

$$E = 210\ GPa \qquad I = 4.27 \times 10^{-6} \qquad L = 5 \qquad W = 2\ kN/m$$

Using the formulas 4.2.1 and 4.2.2, the maximum bending moment and deflections are calculated:

$$M_{max}^{exact} = \frac{WL^2}{8} = \frac{2 \times 10^3 \cdot 5^2}{8} = 6250\ Nm \tag{4.2.3}$$

$$w_{max}^{exact} = \frac{5WL^4}{384EI} = \frac{5 \cdot 2 \times 10^3 \cdot 5^4}{384 \cdot 210 \times 10^9 \cdot 4.27 \times 10^{-6}} = 0.01814\ m \tag{4.2.4}$$

The maximum moment and deflection calculated by EMU.dll is:

$$M_{max}^{emu} = 6249.724\ Nm \tag{4.2.5}$$

$$w_{max}^{emu} = 0.018038\ m \tag{4.2.6}$$

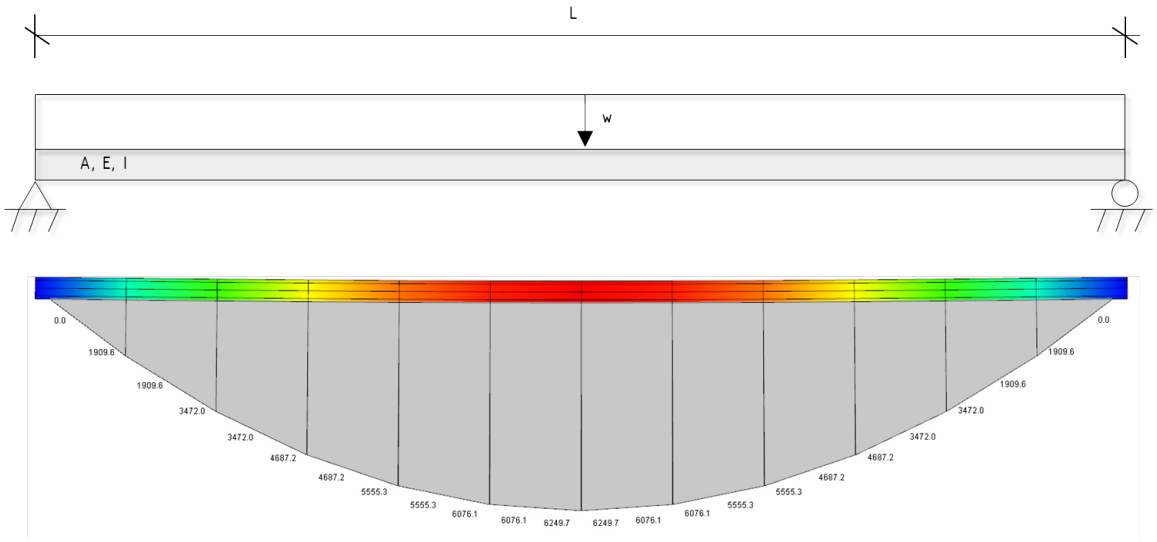


Figure 4.5: *Analysis output from EMU. A moment diagram as a result of a simply supported beam subjected to a uniformly distributed load. The values and colours correspond to bending moments.*

Comparing the output of EMU.dll with the analytical calculation gives the ratio:

$$ratio_M = \frac{M_{max}^{exact}}{M_{max}^{emu}} = \frac{6250.000}{6249.724} = 1.000044 \tag{4.2.7}$$

$$ratio_w = \frac{w_{max}^{exact}}{w_{max}^{emu}} = \frac{0.01814}{0.018038} = 1.005655 \tag{4.2.8}$$

Which is equivalent to a discrepancy of 0.044 ‰ and 5.655 ‰.

4.2.2 Beam overhanging both supports

The beam from the previous test is analyzed again, but with different positions of the supports (see figure 4.6). They are moved towards the centre of the beam with a distance a and c, creating a cantilever effect at the ends. Here three positions are of interest to study; over the supports (M_1 and M_2) and the location in the span where the moment becomes largest (M_3). The moment for these positions can be calculated using the three formulas [21]:

$$M_1 = -\frac{Wa^2}{2} \tag{4.2.9}$$

$$M_2 = -\frac{Wc^2}{2} \tag{4.2.10}$$

$$M_3 = R_1\Big(\frac{R_1}{2W} - a\Big) \tag{4.2.11}$$

where

$$R_1 = \frac{WL(L-2c)}{2b} \quad R_2 = \frac{WL(L-2a)}{2b} \tag{4.2.12}$$

The variables have the same meaning and quantity as the previous example with the extension of a, b and c which are lengths along the beam according to figure 4.6. The values used for these in the test are:

$$a = 0.9375\ m \qquad b = 3.125\ m \qquad c = 0.9375\ m$$

Using the formulas 4.2.9 and 4.2.10 and 4.2.11 the reaction forces, maximum bending moment and deflections are calculated:

$$R_1 = \frac{WL(L-2c)}{2b} = \frac{2\times10^3 \cdot 5(5-2\cdot0.9375)}{2\cdot0.9375} = 5000\ N \tag{4.2.13}$$

$$R_2 = \frac{WL(L-2a)}{2b} = \frac{2\times10^3 \cdot 5(5-2\cdot0.9375)}{2\cdot0.9375} = 5000\ N \tag{4.2.14}$$

$$M_1^{exact} = \frac{Wa^2}{2} = \frac{2\times10^3 \cdot 0.9375^2}{2} = -878.90\ Nm \tag{4.2.15}$$

$$M_2^{exact} = -\frac{Wc^2}{2} = \frac{2\times10^3 \cdot 0.9375^2}{2} = -878.90\ Nm \tag{4.2.16}$$

$$M_3^{exact} = R_1\Big(\frac{R_1}{2W} - a\Big) = 5000\Big(\frac{5000}{2 \cdot 2 \times 10^3} - 0.9375\Big) = 1562.5\ Nm \tag{4.2.17}$$

The maximum moment and deflection calculated by EMU.dll is:

$$M_1^{emu} = -878.9\ Nm \tag{4.2.18}$$

$$M_2^{emu} = -878.9\ Nm \tag{4.2.19}$$

$$M_3^{emu} = 1562.4973\ Nm \tag{4.2.20}$$

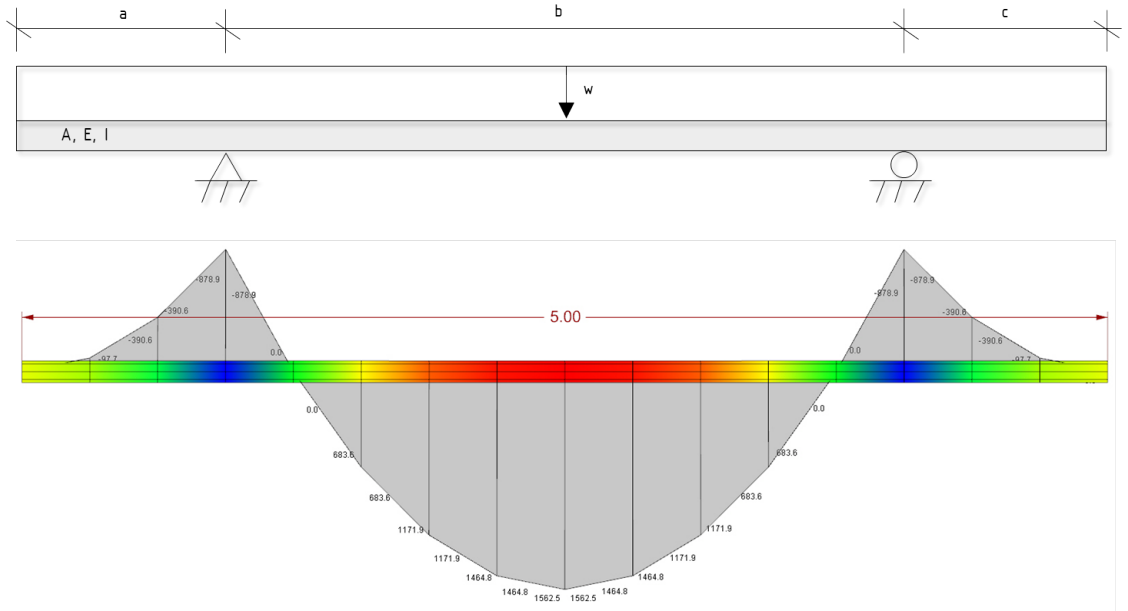


Figure 4.6: *Analysis output from EMU. A moment diagram as a result of a beam with cantilevers at both supports subjected to a uniformly distributed load. The values and colours correspond to bending moments.*

Comparing the output of EMU.dll with the analytical calculation gives the ratio:

$$ratio_{M1} = \frac{M_1^{exact}}{M_1^{emu}} = \frac{-878.900000}{-878.906095} = 1.000000 \tag{4.2.21}$$

$$ratio_{M2} = \frac{M_2^{exact}}{M_2^{emu}} = \frac{-878.900000}{-878.906095} = 1.000000 \tag{4.2.22}$$

$$ratio_{M3} = \frac{M_3^{exact}}{M_3^{emu}} = \frac{1562.5000}{1562.4973} = 1.000002 \tag{4.2.23}$$

which is equivalent or less than a discrepancy of 2 millionth.

4.2.3 Buckling of slender pin-ended column

To test how the program deals with geometrical non-linearity, a simple buckling problem is formulated according to the second Euler buckling case. The analytical buckling load P_{cr} for a structure with pinned ends subjected to axial load can be calculated with the following formula [18]:

$$P_{cr} = \frac{\pi^2 EI}{L^2} \tag{4.2.24}$$

Where
L is the length of the column [m], E is the Youngs modulus in the material $[Pa]$ and I is the second moment of area of the cross section $[m^4]$. Using the values

$$E = 210\ GPa \qquad I = 20.44 \times 10^{-3}\ m^4 \qquad L = 32.5\ m$$

the value of the analytical buckling load can be computed as

$$P_{cr}^{exact} = \frac{\pi^2 \cdot 210 \times 10^9 \cdot 20.44 \times 10^{-3}}{32.5^2} = 40.12 \times 10^6\ N \tag{4.2.25}$$

A common approach for linear buckling analysis in FEM is to utilize the property of singularity in the global stiffness matrix. [22] When a buckling load is reached for a certain structure the displacements of the nodes become indeterminate and the stiffness matrix is singular: $det(\mathbf{K}) = 0$. This state can be found by iteratively adding more load and examining the properties of $\mathbf{K}$ continuously described by Algorithm 1 in section 2.1.1. Another approach is to solve the eigenvalue problem of the structure, and as an outcome obtain the critical load factor and the buckling shape.

In Dynamic Relaxation, there is no such thing as a global stiffness matrix to examine. A similar method by adding load and evaluate the structure can still be used. In this example, the analytical Euler buckling load is calculated and multiplied by a factor λ. The load is applied on the structure and λ starts with the value 0 and is being increased by 0.005 until it reaches 1.01. The vertical displacement of the top node is recorded after each increment. Note that a static equilibrium must be found before incrementing λ. Therefore the program is set to iterate 500 times before additional load is applied to ensure static equilibrium is reached.

Following the procedure described above, the buckling load can be found by studying the diagram produced by comparing the load against the vertical displacement of the top node (See figure 4.7). The model behaves linearly until the buckling load is reached. The displacement will then keep growing and become very large without adding more load.

To get a quantitative comparison between the analytical and numerical results, the vertical position on the load-displacement curve where the second derivative is the

largest (proportional to the curvature) is found. This position is considered being a representative turning-point where linear elasticity no longer applies for the structure and buckling occurs.

$$P_{cr}^{emu}\left(\frac{\partial^2 P}{\partial w^2}\Big|_{max}\right) = 40.04 \times 10^6 \ N \tag{4.2.26}$$

Comparing the exact solution against the numerical gives

$$\frac{P_{cr}^{exact}}{P_{cr}^{emu}} = 1.002 \tag{4.2.27}$$

Which is equivalent to a discrepancy of 2 ‰.

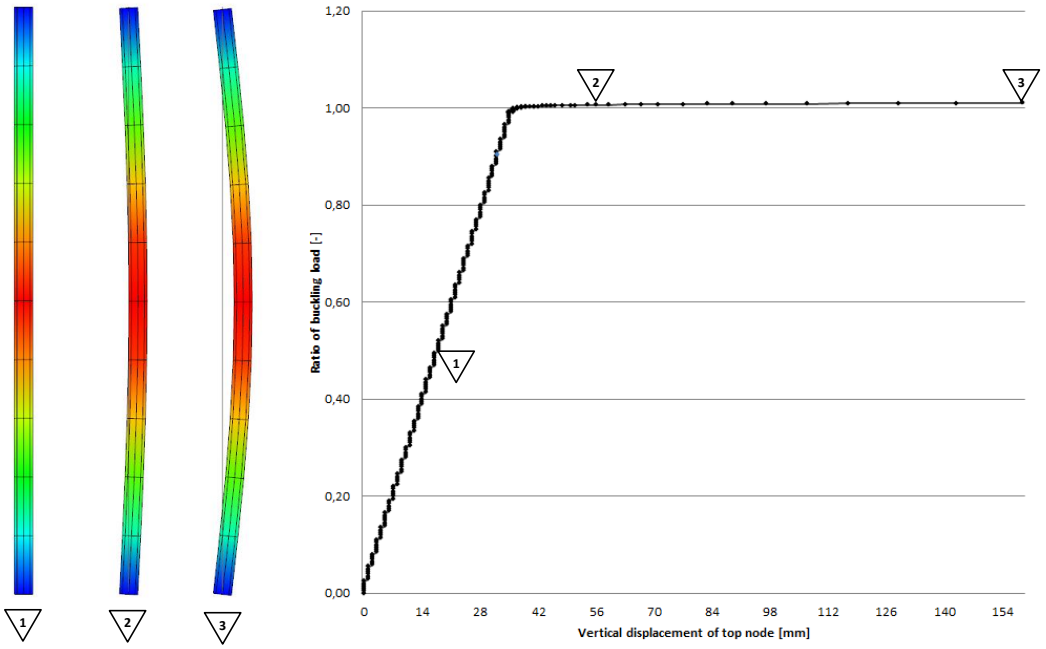


Figure 4.7: *Analysis output from EMU: Euler 2 buckling test*

4.3 Case study

In order to demonstrate the output of EMU.dll for more geometrically complex models, the gridshell from section 3.2.2 is going to be further analyzed. The form-found model will be compared to a representative physical modell.

4.3.1 Computational model

The results presented in the following four sections concern the analysis of a form-found square grid with 18×18 cells, each cell with 500 mm side length. The grid is composed of timber C24 laths with 50×50 mm cross sections. There are no additional external forces applied on the structure apart from the forces needed to

create the shape. A multidimensional analysis of the internal stresses is performed, taking into account bending about the major axis (M_{xx}), minor axis (M_{yy}) and torsion (M_{zz}).

The moments are presented in two diagrams each for every analysis; one showing the data visualized as a gradient mesh, the other one as a plotted moment diagram along one direction of the grids.

4.3.1.1 Moment about major axis (element local M_{xx})

The analysis of the major axis bending naturally produces the largest stresses in the grid. The areas with largest values are at the middle of the edges where the maximum curvature is found. Largest positive bending is found near the supports where the moment also shifts to negative.

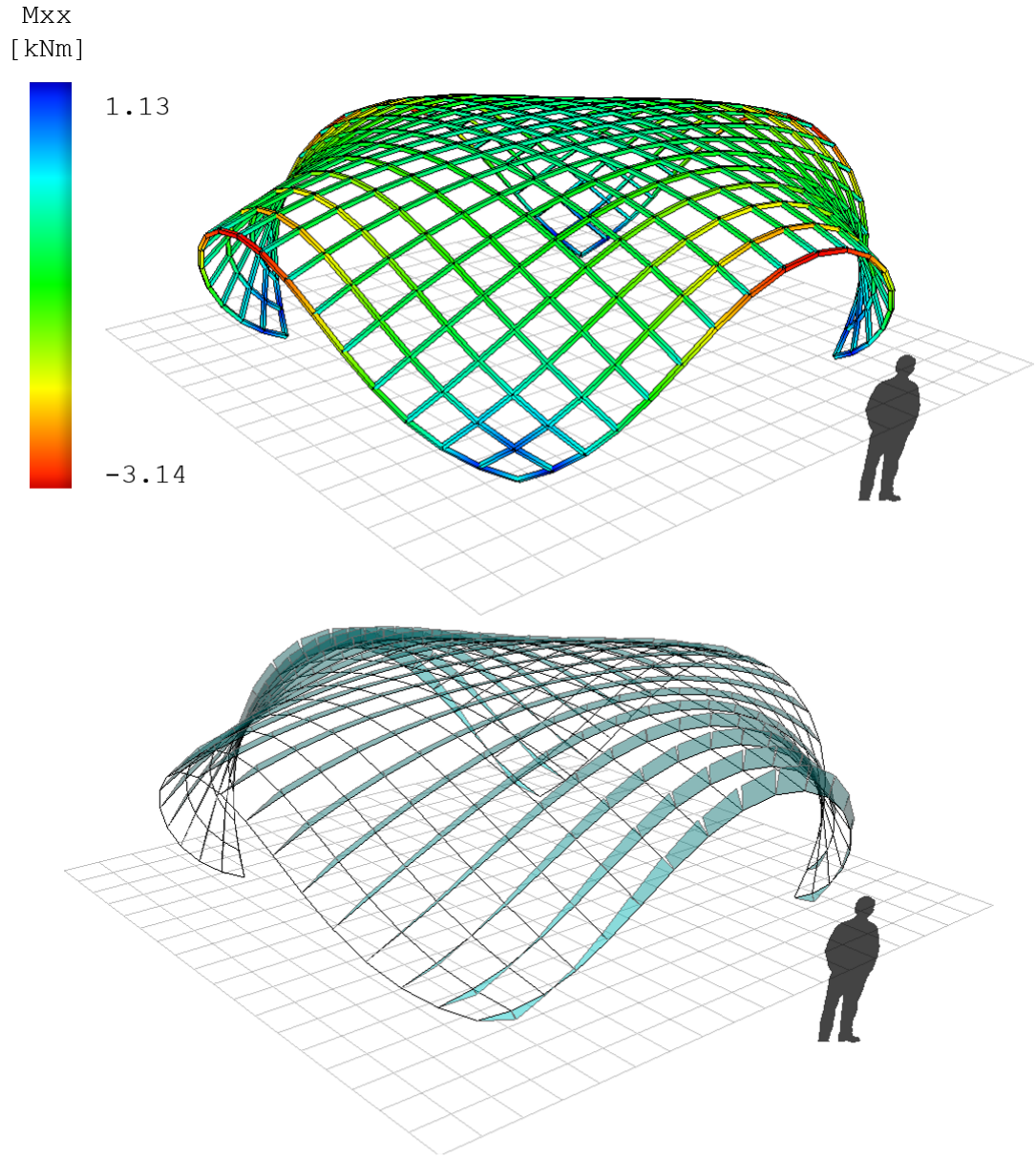


Figure 4.8: *Bending moment about major axis analyzed in EMU.dll. On the top visualized as a colour gradient, on the bottom as a moment diagram in one direction of the grid.*

4.3.1.2 Moment about minor axis (element local M_{yy})

With a maximum value of 1.92 kNm the largest minor axis moments is about 60% of the largest major axis bending moment. These occur close to the support nodes. The normal force of the laths in the orthogonal direction (aligned with the global Y axis) are largest there, making the shear force in the in the X axis elements extensive at this position. On the other hand, the areas on the middle of the edge elements (where the maximum major axis bending is found), the minor axis bending is relatively small.

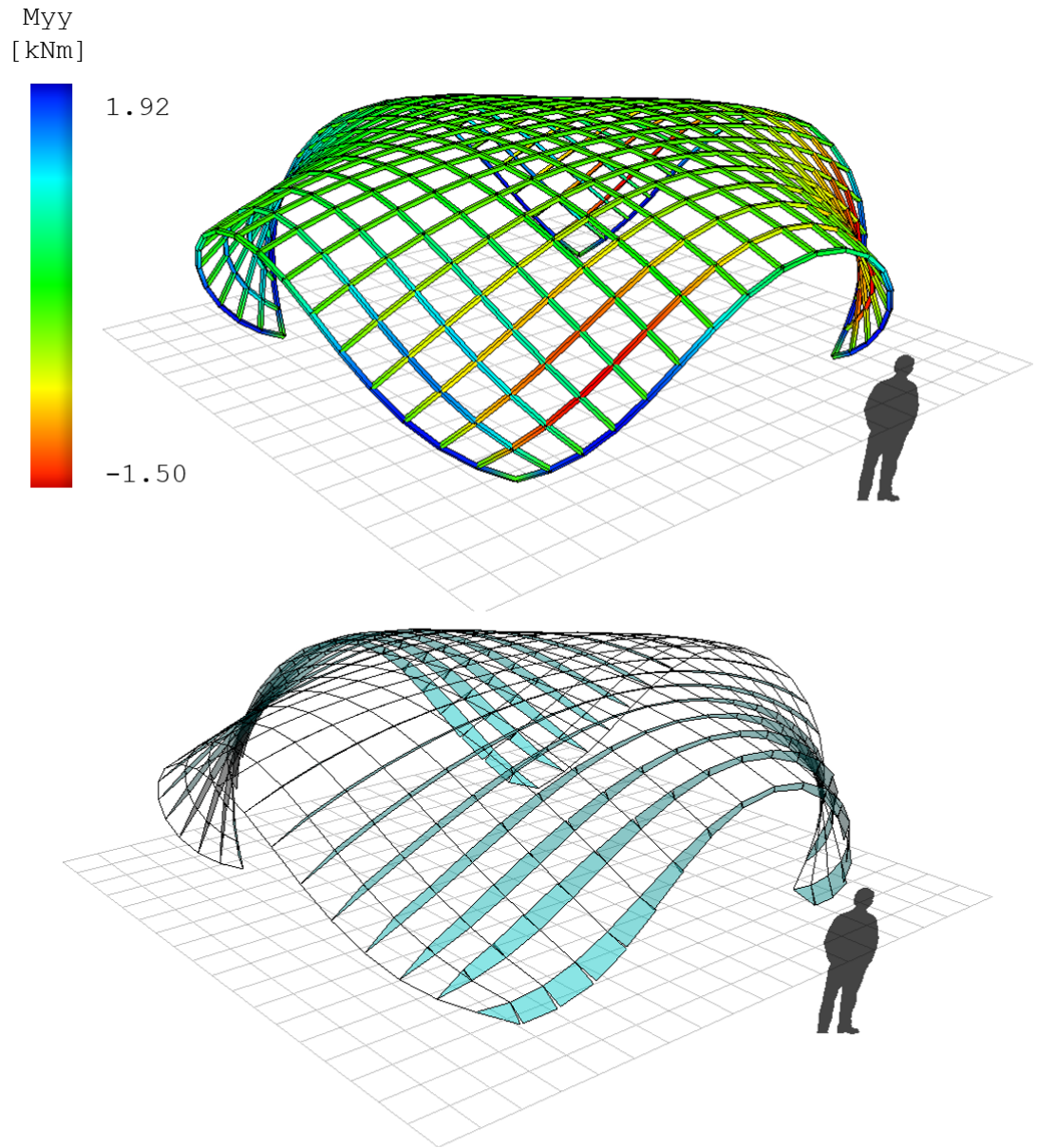


Figure 4.9: *Bending moment about minor axis analyzed in EMU.dll. On the top visualized as a colour gradient, on the bottom as a moment diagram in one direction of the grid.*

4.3.1.3 Torsion (element local M_{zz})

Torsions in the elements are very small compared to the major and minor axis bending - less than 1%. The torsions are constant over an element, and changes direction in the middle of the laths. The largest values are found where the elements twist the most, i.e. in the corners.

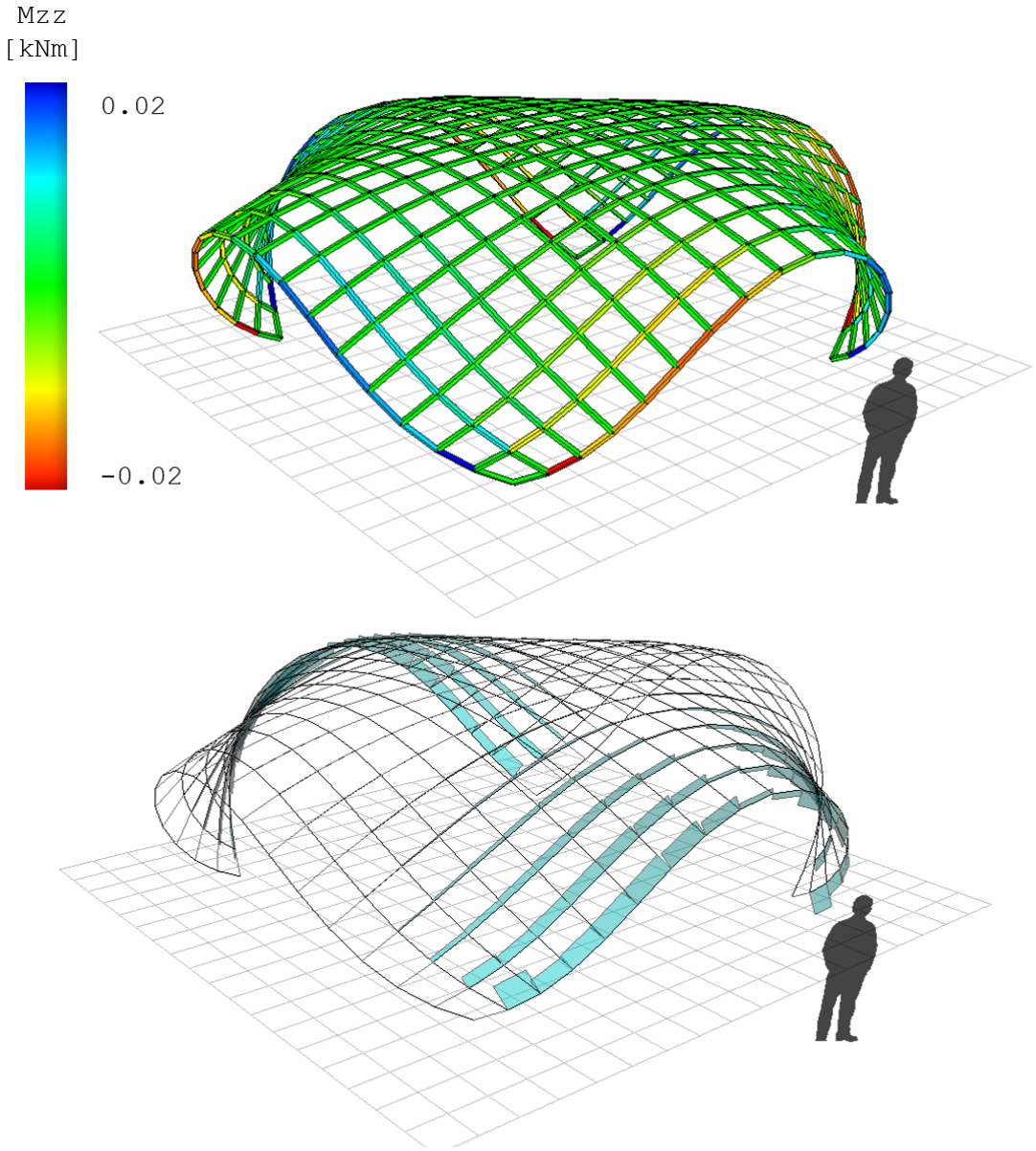


Figure 4.10: *Torsion analyzed in EMU.dll. On the top visualized as a colour gradient, on the bottom as a moment diagram in one direction of the grid.*

4.3.2 Physical model

The method of benchmarking against analytical solutions is highly preferable when testing the correctness of custom FEA implementations. For simple cases, like the ones presented in section 4.2.1 - 4.2.3, analytical solutions can easily be found. On the other hand, when the structural model becomes complex in its geometrical layout, finding an exact solution is hard. An alternative approach is to benchmark against physical models, hence why one was built in conjunction to the thesis work. It was constructed by fibre composite rods of 2 mm diameter in a bidirectional grid. The structure consists of 19 x 19 rods which define cells of 50 x 50 mm. The connections were made of firmly tightened rubber band, which allows rotation but not translation.

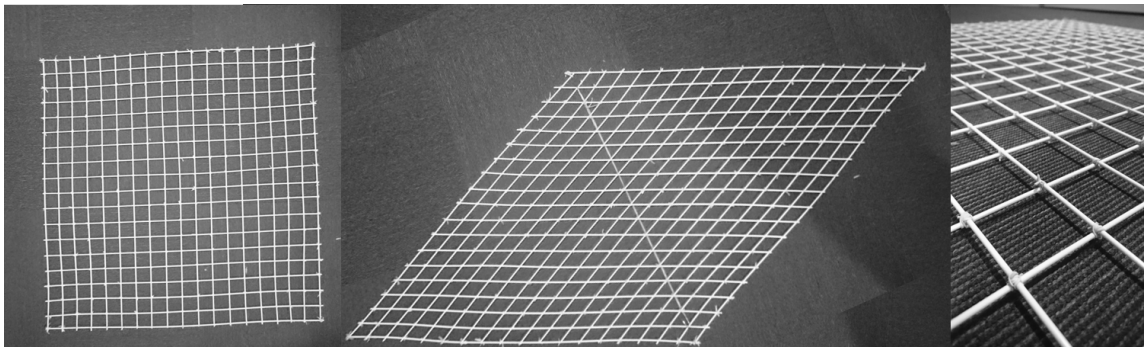


Figure 4.11: *Grid of fibre composite rods configuration*

To quantify the geometrical similarities between the digital and the physical model, the height from the floor to two characterizing nodes were measured. The first node is the one in the middle of the the outermost rod, corresponding to the height z_{edge}. Since there are four of this type of node, a mean value was calculated. The second node is the one representing the geometrical centre of the grid, corresponding to the height z_{mid}. Measured from the model, the following values were recorded:

$$z_{edge}^{phys} = 270\ mm \qquad\qquad z_{mid}^{phys} = 357\ mm$$

Measuring the same nodes in the computational model gives:

$$z_{edge}^{emu} = 266\ mm \qquad\qquad z_{mid}^{emu} = 371\ mm$$

Comparing the physical and digital data, ratios can be computed:

$$\frac{z_{edge}^{phys}}{z_{edge}^{emu}} = 1.015 \qquad\qquad \frac{z_{mid}^{phys}}{z_{mid}^{emu}} = 0.962$$

This is equal to a discrepancy of 1.5% and 3.8%.

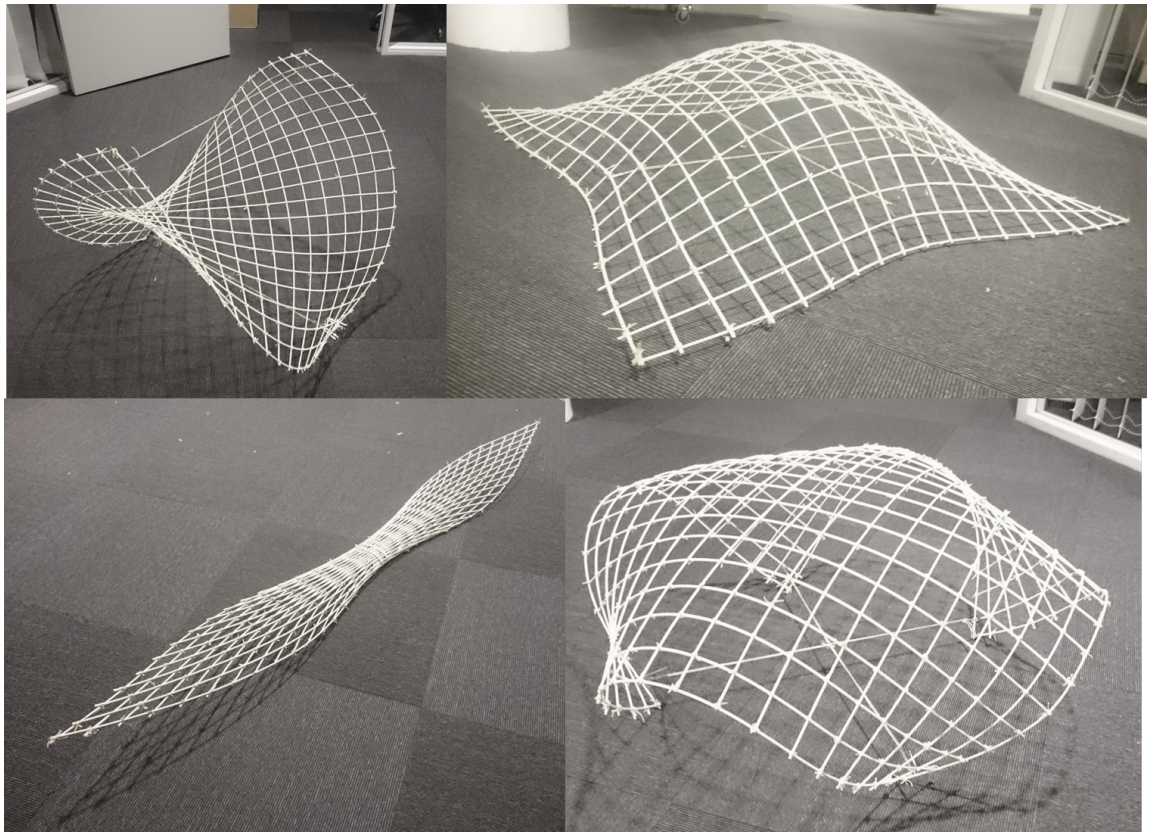


Figure 4.12: *Different connection nodes between a 670 mm string results in different buckling shapes.*

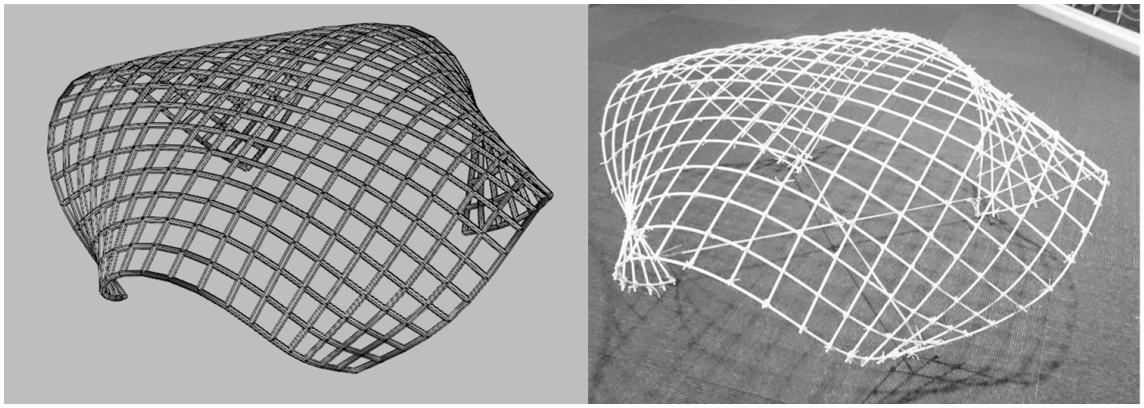


Figure 4.13: *Visual comparison between the digital and physical model.*

5
Discussion

In the following chapter a discussion concerning the results and methods presented in the thesis is held. Apart from purely technical aspects, a more general discussion regarding the feasibility of elastically bent gridshell structures is included. Finally some recommendations for further work is presented.

5.1 Reflections

5.1.1 The feasibility of elastically bent gridshells

The technology of bending material to achieve efficient doubly curved roof structures has, as previously presented, many advantages; connection repeatability, rational construction process, low mass-span ratio, aesthetics and sustainability to name a few. The complications concerning the design and analysis process was thought to be one of the main reasons why actively bent gridshells are not built more often. With the developed toolbox it is hoped to respond to this issue. Other potential obstructions for the realization of a gridshell have been discussed throughout the thesis work.

A major question is the financial aspects of such structure. Is it more expensive compared to other buildings due to its non-standard properties? Are bending active gridshells cheaper than discrete gridshells? Exemplified in [10], the total cost of the Weald & Downland gridshell was £1097/m^2, wheras the price of the roof constitute 28%. This is slightly below the average price of a typical visitor centre building which makes the argument that timber gridshells always end up being more expensive than less non-standard buildings invalid. A comparison between a number of gridshell projects is also carried out in [10]. Notable is the difference in cost between a typical steel gridshell made up of discrete members (Palacio de Comunicaciones) and typical timber gridshells constructed using elastically bent laths (Mannheim and Weald & Downland). The cost of the steel gridshell is more than three times as big compared to the Mannheim and Weald & Downland gridshells. (See figure 5.1). Further more, the relationship and trust between the parties involved in a building project is of great importance. Since a very few gridshells have been constructed in full scale, the technology might become considered unproven among clients. With increased experience of consultants and contractors comes increased confidence of clients. Therefore, a tight project team with trust is a necessity to successfully conduct a building of this kind, especially in the early stages.

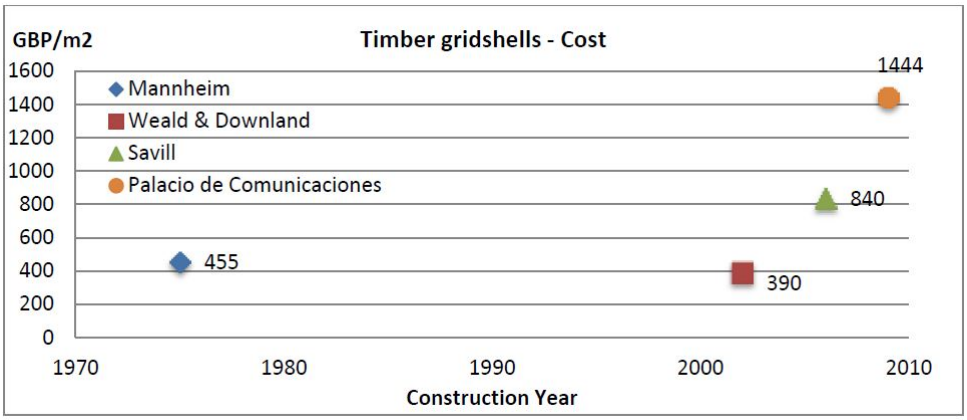


Figure 5.1: *Cost comparison between a steel gridshell (Palacio de Comunicaciones) and a three timber gridshells (Mannheim, Weald & Downland and Savill).* [10]

5.1.2 Structure of EMU.dll

The initial concept with EMU.dll was to produce a tool to use throughout the whole design process. Despite that there is nothing preventing functionalities aimed for the later design phases to be implemented, focus naturally was put on the initial form-finding stage. This is also considered being the stage which it is hard to find suitable tools for in existing software, hence why it was prioritized.

Finding a good OOP design for the codebase took time and effort. In fact, the current class structure was written entirely from scratch after only 4 weeks of development. The key was to evaluate the parts which are likely to change in the future and encapsulate these by separating implementations from interfaces. Examples of code prone to be extended are solver algorithm (potentially matrix solver and eigen solver), element types (potentially plates and tetrahedral elements), integration methods (potentially Implicit Euler and Crank Nicholson).

Although the numerical framework was programmed in the pursuit of simplifying the design and analysis of elastically bent gridshells, its design allows more types of geometrically non-linear analysis to take place. Possible scenarios could include:

- Form-finding of compression-only structures
- Form-finding of tensile membrane structures
- Instability analysis and post-buckling
- Progressive collapse analysis

Putting extra effort into the design and structure of the codebase required more development time in the beginning of project. The first steps consisted of sketching out class diagrams on paper, which made it easy to draw parallels to the way designing architects work. By many means the design process of software is very similar to the design process of buildings. The time and effort spent on thinking through the project in an early stage to achieve an overall good design, is very often paid back in later stages of the project with greater magnitude.

5.1.3 Performance of EMU.dll

When the output of EMU.dll in terms of geometry and structural data is compared to analytical solutions in section 4.2.1-4.2.3, the discrepancy is proven to be very small. The presented bending moment show a ratio less than 0.1 ‰, which could be considered negligible in the context. The ratio between the numerical and analytical buckling load tested in 4.2.3 is also very small, even though the point of maximum second derivative of the load-displacement curve (which was defined as the point of buckling) is highly dependent on the initial imperfection. The smaller the initial eccentricity is, the less deviation from the analytical solution. The dimensional measurements taken on the physical models corresponded well to the digital form-finding. The deviation of 3.8% can be explained by sliding rods and eccentricity between the two layers. It was hard to find a suitable method for joining the fibre composite rods in a way which would represent the real mechanical properties. The rubber bands made the joints rotatable, but the resulting friction was found to be too low which occasionally caused the rods to slide relative each other.

The computation speed works satisfactory for structures up to a certain scale. Also the load and support conditions have great influence of the simulation time. When too many elements are involved and load is concentrated to few areas, more computation is needed for each iteration which slows down the form-finding procedure. The easier the load can "propagate" through the structure, the faster the analysis is going to take.

As explained and presented in [2], incorporating bending stiffness between elements is also possible with a 3 DOF formulation of the dynamic relaxation mehtod. The moments of an element are based on the angles to adjacent elements, converted to force-couples and added to the connecting nodes. In this way a local coordinate system is not needed for each node which makes the computations much faster. On the other hand multi-axial bending analysis including torsion is hard (if not impossible) to achieve without coordinate systems. Multiaxial analysis was a criteria established early in the development.

EMU.dll is considered to be a useful tool which fits the gap of what existing software cannot offer in terms of design and analysis of elastically bent gridshells. There are however potentials for many improvements and areas of further development in order to make the tool more complete.

5.2 Recommendations for further work

- **Implementation on other platforms.** The EMU.dll API has only been implemented as a plugin to Grasshopper3d® so far. Yet there is nothing prohibiting the API to be implemented in other CAD packages, for instance Autodesk Revit/Dynamo®, Google Sketchup®. Even an independent user interface could be developed as a desktop application to prove its interdependency towards commercial host software. As a suggestion, OpenTK is a

graphics library which can be used to facilitate the geometrical representations. Microsoft WPF can be used for building the UI.

- **Test more integration schemes.** Only three integration algorithms have been implemented so far in the developed tool. Adding more, especially ones with implicit nature would be interesting. A full comparison between different integration schemes would also be good to carry out. A suggested format for this is speed vs. number of iterations to achieve equilibrium. Some techniques seem to be more stable than others while some are faster. The relaxation algorithm could automatically evaluate the circumstance to pick the most suitable.
- **Add more element types.** The main focus so far has been to simulate the bending of one dimensional elements in the context of gridshells. However, the area of bending active structures could also include two dimensional elements. Simulating bending of steel sheets or strips would require such a formulation.
- **Speed improvements.** When a set of nodes or constraints are added in the structure, the `NodeRepository` class will first search for coordinates in the same position, making sure no duplications occur. This process is performed by measuring the euclidean distance between the nodes which is characterized by an O(N2) algorithm. When the number of nodes become large, the process becomes very time consuming. Suggested is to use another type of data structures such as k-d tree. It is a neat space-partitioning data structure invented 1975 by J.L Bently [5] which organizes the nodes in k-dimensional sub-domains. Another take on speed improvement is the introduction of parallel computing and multithreaded algorithms, which has tremendous potentials in the application of Dynamic Relaxation.
- **Implementation of DSM Solver.** As suggested in [4], the benefits of a Direct stiffness approach (matrix) can be combined with dynamic relaxation to perform faster analysis. In an initial step, the a `DirectStiffnessSolver` shall be created by implementing the interface `ISolver`. The DS/DR hybrid solver can then be created, which aggregates both solvers.
- **Fire.** Questions regarding the fire safety aspects of timber gridshells were risen during the thesis work. For conventional timber frame structures, load bearing elements are usually either covered by gypsum boards or they are sufficiently thick to maintain capacity during the coalification process for a required time span. Timber gridshells are often doubly curved thin shells which may be hard to protect using conventional methods. It would be interesting find a way around this issue by exploring alternative fire protection methods.
- **Automatic spring back analysis.** When artificial shaping forces are present in the form-finding such as pulling a grid to a target surface as described in section 1.1.1.3, the equilibrium configuration will be slightly off. This is due to the fact that the artificial forces do not exist in reality. An extension of EMU.dll could be to automatically perform the cutting of the grid at a certain level, apply new supports and run a second stage form-finding.
- **Cladding.** The pattern produced by the bent network of elements exhibit aesthetical qualities which can be exposed as part of the architectural expression. Since the grid itself only provide a load bearing structure, additional

cladding must be attached in order to function as a climate screen. An interesting topic of further research could be to find a rational cladding technique which doesn't hide the geometrical pattern of the grid.

- **Cross section transformation**. A common method for gaining smaller radii when the laths are bent without using large cross sections is the use of double layering as described in section 3.2.3.1. When simulating the erection process, the layers must slide freely relative each other. When the equilibrium is found and the ULS/SLS analysis is about to be performed, the layers are locked. To execute this instant change of cross section numerically, the bending moments need to be updated accordingly. The node positions remain the same, but not the internal stresses. The logics and equations to implement this are described in [9].
- **Formation stress in ULS/SLS analysis** - To get a more realistic analysis result in the ULS / SLS design checks, the stresses induced by the formation should be included. This could be done in two ways; either by applying external loads after the form-finding in EMU.gh, or export the model with pres-tress into a commercial FEM package.

6
Conclusion

The question this thesis addresses is the feasibility of elastically bent gridshells in a Scandinavian context. Looking worldwide at realized gridshells and existing design methods, a gap has been identified between the required analysis process and existing available design tools. This has led to the pursuit of formulating a design and analysis process. Included in this has been to develop a codebase which can simulate the highly non-linear bending process with automatic supervision of the material capacity. The mechanics used is based on a nodal six degree of freedom formulation of the Dynamic Relaxation method (6DoF DR). All equations and logics are compiled into a C# .NET class library which functions as an application programming interface (API). It is given the name EMU.dll and has been implemented as a plugin to the parametric 3D modelling software Grasshopper3d® (GH) for Rhinoceros®. In order to achieve good code design which easily can be maintained and extended, the concepts of object oriented design patterns are used.

The structural output of the developed numerical framework is compared against analytical and physical models. Through four test cases the code is benchmarked and proven to be performing accurately. As a real-time structural analysis plug-in in a parametric environment, the user can easily interact with the model during run-time. Suggestions of how the developed tool fits into a bigger system of the gridshell design and analysis process are presented. The implementation of software design patterns makes EMU.dll straightforward to extend and modify to suit project specific needs.

The developed tool can be used in the structural design and analysis process of elastic gridshells. The combination of the underlying 6DoF DR engine and the implementation in GH makes EMU a novel tool with capabilities to interactively control structural models. Although the tool was written in the purpose of elastic gridshells, EMU can be used to perform various non-linear frame analysis including buckling, progressive collapse and form-finding.

Bibliography

[1] B.C.P. Heng, R.I. Mackie (2008) *Using design patterns in object-oriented finite element programming.* Computers and Structures 87(2009) 952-961.

[2] Adriaenssens SML (2000) *Stressed spline structures.* PhD thesis. University of Bath (2000).

[3] Kuijvenhoven M, Hoogenboom P. *Particle-spring method for form finding grid shell structures consisting of flexible members.* Journals of the International Association for Shell and Spatial Structures 53(1) (2012).

[4] Olsson J. *Form finding and size optimization: Implementation of beam elements and size optimization in real time form finding using dynamic relaxation.* Master thesis. Chalmers University of Technology (2012)

[5] Bently J. L. (1975). *Multidimensional binary search trees used for associative searching.* Communications of the ACM 18 (9)

[6] Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert (2004). *Head First Design Patterns.* (paperback) 1. O'REILLY

[7] Happold, E Liddell, W.I. (1975). *Timber lattice roof for the Mannheim Bundesgartenschau.* The Structural Engineer, No. 3, Volume 53, March 1975

[8] Kelly O, Harris R, Dickson M, Rowe J. (2001). *The construction of the downland gridshell.* The Structural Engineer, No. 17, Volume 79, 2001

[9] B. D'Amico, A. Kermani, H. Zhang, P. Shepherd, C.J.K. Williams (2015). *Optimization of cross-section of actively bent grid shells with strength and geometric compatibility constraints.* Computers Structures, Volume 154, 1 July 2015, Pages 163–176

[10] Naicu D.I. (2012). *Geometry and Performance of Timber Gridshells.* Master thesis. University of Bath (2012)

[11] B. D'Amico, A. Kermani, H. Zhang, (2014). *Form-finding and structural analysis of actively bent timber grid shells.* Engineering Structures, Volume 81, 2014, Pages 195-207.

[12] Quinn G.C, Gengnagel C. (2015). *Simulation Methods for the Erection of Strained Grid Shells Via Pneumatic Falsework.* Design Modelling Symposium 2015, Volume: Modelling Behaviour (2015). Pages 257-268

[13] F. Tayeb, J.-F. Caron, O. Baverel, L. Du Peloux (2013). *Stability and robustness of a 300 m^2 composite gridshell structure.* Construction and Building Materials, Volume 49 (2013). Pages 926-938

[14] Toussaint, M.H. (2007). *A Design Tool for Timber Gridshells.* Master thesis. T.U. Delft (2007)

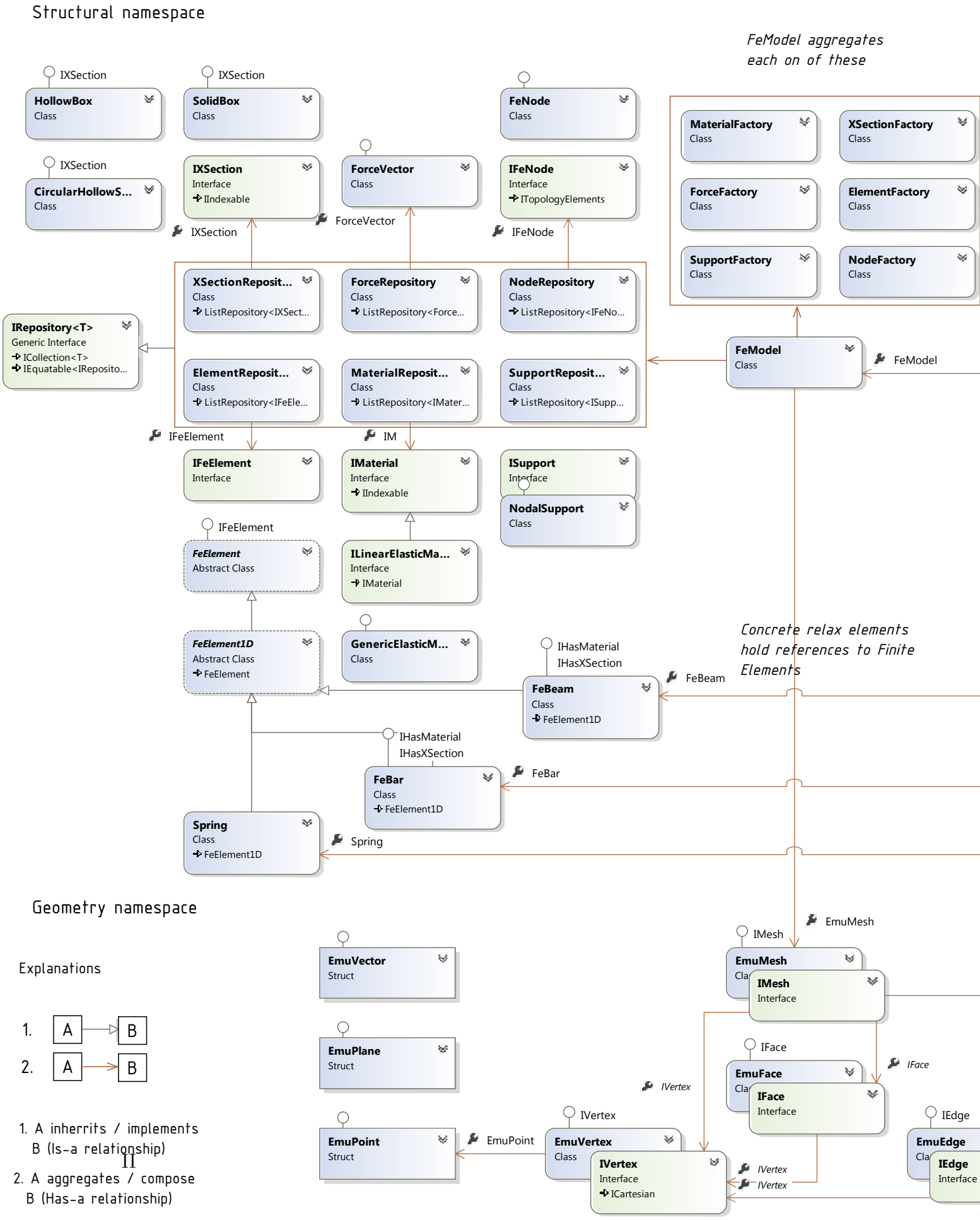
[15] Van Verth J. M. (2010). *Mathematical Background.* Game Physics Pearls (2010) Pages 3-27

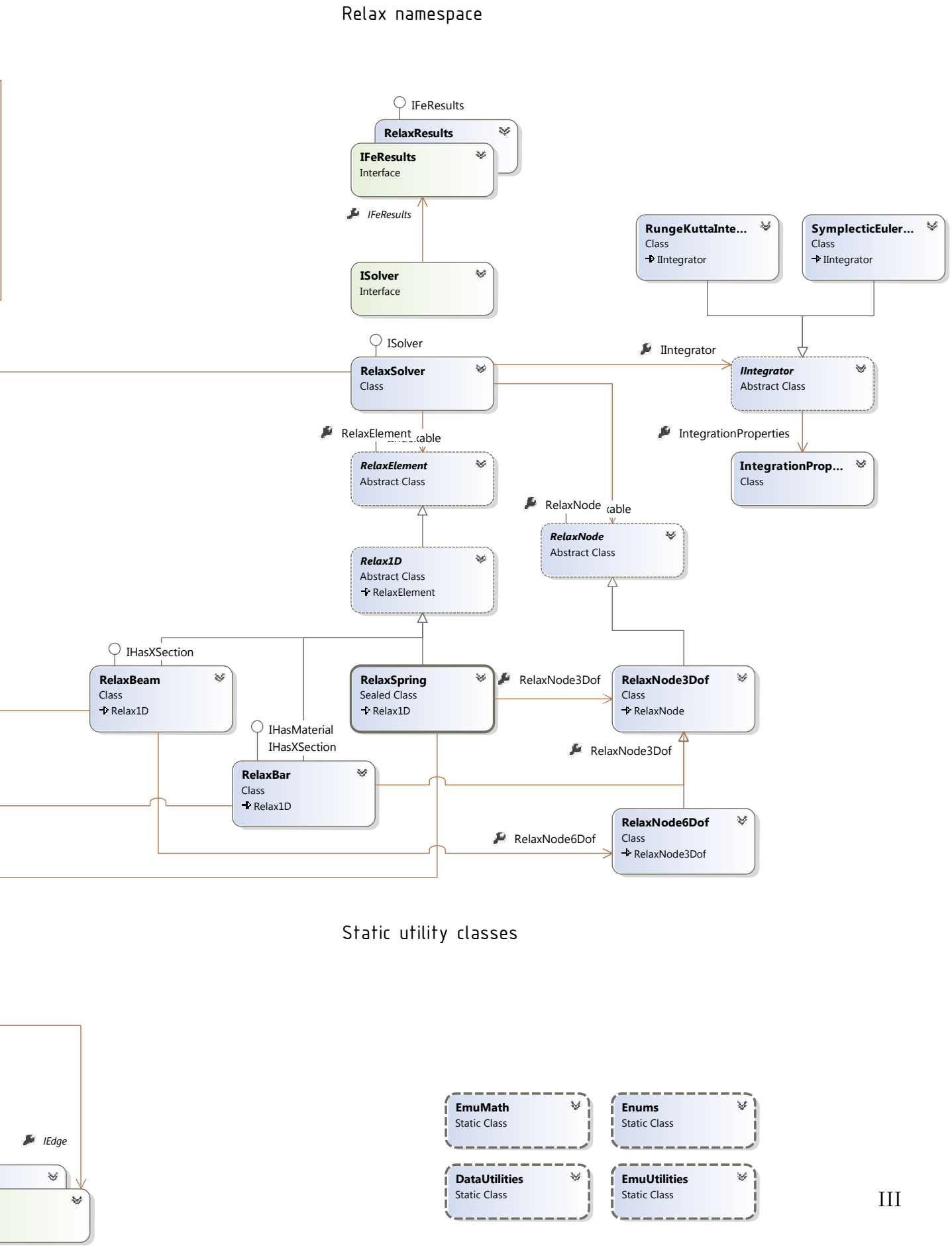
[16] Department of Civil and Environmental Engineering - Princeton University. *MANNHEIM MULTIHALLE* [Online] Available at: http://shells.princeton.edu/Mann1.html [Accessed 6 November 2015].
[17] Otto F. *IL10 Gitterschalen*. Institut für leichte Flächentragwerke (IL), 1974.
[18] Lundh H. *Grundläggande hållfasthetslära*. Department of Solid Mechanics - Royal Institute of Technology, 2000.
[19] Craig R. R., Kurdila A. J. *Fundamentals of Structural Dynamics, 2nd Edition*. JOHN WILEY SONS, 2006.
[20] Melgar E. R. *Integrating Physics into the Design Process*. Master of Science Thesis, University College London (2010)
[21] The American Wood Council *Beam formulas with shear and moment diagrams*. American Forest Paper association (2007)
[22] Dahlblom O., Olsson K.G. *Strukturmekanik, modellering och analys av ramar och fackverk*. Studentlitteratur AB, Lund, 2010

A

Appendix 1: Class diagram

This appendix includes a simplified UML (Unified Modeling Language) class diagram over EMU.dll. For the sake of readability, not all classes and relationships are illustrated. It aims to expose the overall design concept and the general system rather than a complete description of all components present in the application. The diagram has been presented as partitions in section 4.1.1,4.1.2 and 4.1.3 where an associated explanation for each selection can be found.


Structural namespace
FeModel aggregates each on of these
IXSection
HollowBox
Class
IXSection
SolidBox
Class
FeNode
Class
IXSection
CircularHollowS...
Class
IXSection
Interface
IIndexable
ForceVector
Class
IFeNode
Interface
ITopologyElements
MaterialFactory
Class
XSectionFactory
Class
ForceFactory
Class
ElementFactory
Class
SupportFactory
Class
NodeFactory
Class
IXSection
ForceVector
IFeNode
XSectionReposit...
Class
ListRepository<IXSect...
ForceRepository
Class
ListRepository<Force...
NodeRepository
Class
ListRepository<IFeNo...
IRepository<T>
Generic Interface
ICollection<T>
IEquatable<IReposito...
ElementReposit...
Class
ListRepository<IFeEle...
MaterialReposit...
Class
ListRepository<IMater...
SupportReposit...
Class
ListRepository<ISupp...
FeModel
Class
FeModel
IFeElement
IM
IFeElement
Interface
IMaterial
Interface
IIndexable
ISupport
Interface
NodalSupport
Class
IFeElement
FeElement
Abstract Class
ILinearElasticMa...
Interface
IMaterial
FeElement1D
Abstract Class
FeElement
GenericElasticM...
Class
IHasMaterial
IHasXSection
FeBeam
Class
FeElement1D
FeBeam
Concrete relax elements hold references to Finite Elements
IHasMaterial
IHasXSection
FeBar
Class
FeElement1D
FeBar
Spring
Class
FeElement1D
Spring
Geometry namespace
EmuMesh
IMesh
EmuVector
Struct
EmuMesh
IMesh
Interface
EmuPlane
Struct
IFace
EmuFace
IFace
Interface
IFace
IVertex
IVertex
EmuPoint
Struct
EmuPoint
EmuVertex
Class
IVertex
Interface
ICartesian
IVertex
IVertex
IEdge
EmuEdge
IEdge
Interface
Explanations
1. A
B
2. A
B
1. A inherrits / implements B (Is-a relationship)
2. A aggregates / compose B (Has-a relationship)


Relax namespace
IFeResults
RelaxResults
IFeResults
Interface
IFeResults
ISolver
Interface
RungeKuttaInte...
Class
IIntegrator
SymplecticEuler...
Class
IIntegrator
ISolver
RelaxSolver
Class
IIntegrator
IIntegrator
Abstract Class
RelaxElement
IntegrationProperties
RelaxElement
Abstract Class
IntegrationProp...
Class
RelaxNode
RelaxNode
Abstract Class
Relax1D
Abstract Class
RelaxElement
IHasXSection
RelaxBeam
Class
Relax1D
RelaxSpring
Sealed Class
Relax1D
RelaxNode3Dof
RelaxNode3Dof
Class
RelaxNode
IHasMaterial
IHasXSection
RelaxNode3Dof
RelaxBar
Class
Relax1D
RelaxNode6Dof
Class
RelaxNode3Dof
RelaxNode6Dof
Static utility classes
IEdge
EmuMath
Static Class
Enums
Static Class
DataUtilities
Static Class
EmuUtilities
Static Class