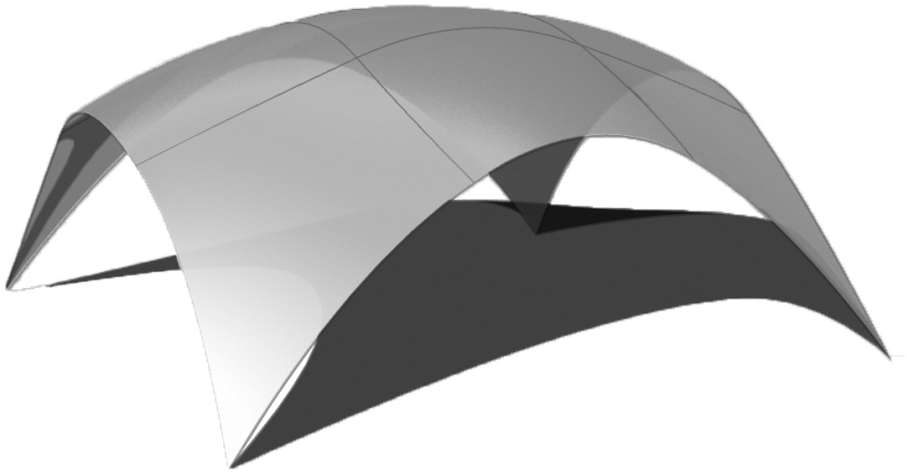
CHALMERS
UNIVERSITY OF TECHNOLOGY



Isogeometric analysis in form finding in architecture

An implementation of a dynamic relaxation solver for Rhinoceros©

Master's thesis in Structural Engineering and Building Technology

ANDREA ALEXANDERSSON

Department of Applied Mechanics
Division of Material and Computational Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015
Master's thesis 2015:44

MASTER'S THESIS IN STRUCTURAL ENGINEERING AND BUILDING TECHNOLOGY

Isogeometric analysis in form finding in architecture

An implementation of a dynamic relaxation solver for Rhinoceros©

ANDREA ALEXANDERSSON

Department of Applied Mechanics
Division of Material and Computational Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015

Isogeometric analysis in form finding in architecture
An implementation of a dynamic relaxation solver for Rhinoceros©
ANDREA ALEXANDERSSON

Master's Thesis 2015:44
ISSN 1652-8557
Department of Applied Mechanics
Division of Material and Computational Mechanics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Cover:
Form found surface using the plug-in program developed in the thesis

Printed by Reproservice
Göteborg, Sweden 2015-08-24

Isogeometric analysis in form finding in architecture
An implementation of a dynamic relaxation solver for Rhinoceros©
Master's thesis in Structural Engineering and Building Technology
ANDREA ALEXANDERSSON
Department of Applied Mechanics
Division of Material and Computational Mechanics
Chalmers University of Technology

Abstract

Today, the use of computers for the design of new buildings is almost crucial. As the built environment is getting more complex, the integration between design and analysis is getting more important and the time setting up models for the geometry as well as the analysis is increasing. To make the design process as efficient as possible there is a need for tools for real-time analysis in the conceptual design phase.

Most CAD geometries consist of NURBS (Non-Uniform Rational B-Splines) which are functions in a mathematical model for representing and generating geometries. By using isogeometric finite elements, i.e. where the basis functions consist of NURBS, the geometry of the CAD model can also be used in the FE-analysis which means that the same model can be used for the geometry as well as for the analysis. This will save a lot of time for the analysts and for the design process in general as it will facilitate the design iteration. Using the correct geometry instead of a simplification will also generate more reliable results.

Dynamic relaxation is an explicit numerical method in which a static problem is solved as a fictious dynamic problem and can be used for form finding. By using this form finding method on elements that only work in tension and compression, i.e. membranes, an idea of an optimal form can be obtained.

The purpose of this thesis has been to develop a plug-in program for the 3D modeling software Rhinoceros©. The plug-in uses dynamic relaxation with isogeometric analysis for the conceptual design of membranes. The program is written in the coding language C# using VisualStudio©. The aim is to explore new methods of FE-modeling and to evaluate the possibilities and difficulties of using isogeometric elements in a design process.

Key words:

Isogeometric analysis, finite element method, form finding, dynamic relaxation, shell structures, object oriented programming, architectural engineering

Isogeometrisk analys för formsökning inom arkitektur
Implementering av en lösare för dynamisk relaxation i Rhinoceros©
Examensarbete inom Konstruktionsteknik och byggnadsteknologi
ANDREA ALEXANDERSSON
Institutionen för tillämpad mekanik
Avdelningen för Material och beräkningsmekanik
Chalmers tekniska högskola

Sammanfattning

Att använda datorer när man designar nya byggnader är idag näst intill nödvändigt. När den byggda miljön blir mer komplex, blir integrationen mellan design och analys allt viktigare och tiden det tar för att modellera geometrin samt modellen för analyser ökar. För att göra designprocessen så effektiv som möjligt finns därför ett behov av att utveckla verktyg för realtidsanalys under den konceptuella designfasen.

De flesta CAD-geometrier består av NURBS (Non-Uniform Rational B-Splines) som är funktioner i en matematisk modell för att representera och skapa geometrier. Genom att använda isogeometriska finita element, där basfunktionerna består av NURBS, kan geometrin i CAD-modellen också användas i FE-analyserna, vilket innebär att samma modell kan användas för geometrin samt för strukturberäkningarna. Detta gör att man kan spara mycket tid på modelleringen och i designprocessen i allmänhet eftersom det underlättar itereringen som processen innebär. Genom att använda den korrekta geometrin istället för en förenkling kommer också beräkningarna att generera mer tillförlitliga resultat.

Dynamisk relaxation är en explicit numerisk metod i vilken man löser ett statiskt problem som ett fiktivt dynamiskt problem och kan användas för formsökning. Genom att använda membran, d.v.s. element som bara kan ta tryck- och dragspänningar, kan man genom denna formsökningsmetod få en uppfattning om den optimala formen för systemet.

Syftet med arbetet har varit att utveckla ett plug-in-program för 3D-modelleringsprogrammet Rhinoceros©. Plug-in-programmet använder dynamisk relaxation med isogeometrisk analys för konceptuell design av membran. Programmet är skrivet i kodspråket C # med VisualStudio©. Syftet är att undersöka nya metoder för FE-modellering och att utvärdera möjligheter och svårigheter med att använda isogeometriska element i en designprocess.

Nyckelord:
Isogeometrisk analys, finita elementmetoden, formsökning, dynamisk relaxation, skalstrukturer, objektorienterad programmering, Arkitektur och teknik

Contents

Preface

This master's thesis of 30 credits has been carried out during the spring of 2015. The work has taken place at the Department of Applied mechanics at Chalmers University and has been a collaboration with the Department of Structural mechanics at the University of Lund.

Senior Lecturer Dr. Mats Ander at Chalmers University has been the examiner of the thesis and has also supervised the work. The supervisor from Lund University has been PhD student, MSc Architectural Engineering, Vedad Alic.

Göteborg 2015-08-24

ANDREA ALEXANDERSSON

Acknowledgement

First and foremost I want to thank my examiner Dr. Mats Ander for his constant feedback and support throughout the thesis and especially for his constant encouragement. I would also like to thank my supervisor Vedad Alic for his help and for the very much appreciated support with the development of the software.

At last I want to thank Prof. Karl-Gunnar Olsson and Prof. Kent Persson for the rewarding discussions during our meetings in Lund.

1 Introduction

Today, the use of computers for the design of new buildings is almost crucial. As the built environment is getting more complex, the need for tools for real-time analysis in the conceptual design phase is increasing. This integration between the design and analysis of the structures will therefore demand a stronger connection between architects and engineers.

1.1 Background

Most CAD (Computer Aided Design) geometries consist of NURBS (Non-Uniform Rational B-Splines). As the built environment gets more and more complex, the CAD models are getting larger and the amount of time setting up the model for the geometry as well as the model for the analysis is increasing. When two different models (geometry and structural analysis) are needed in the design process a lot of information can be lost. Many factors contributes to this loss. The lack of communication between architects and engineers and the overall lack of integration between the two professions has great significance to this. Another important factor is the simplifications that are made in the FE-model.

In isogeometric analysis (IGA) the basis functions used in the FE-calculations consists of NURBS, hence, by using isogeometric finite elements the geometry of the CAD model can also be used in the FE-analysis. This will save a lot of time for the analysts and for the design process in general as it will facilitate the design iteration. Using the correct geometry instead of a simplification will also generate more reliable results.

There are a number of tools for conceptual design already today, for example the SMART© tool for the Rhino© plug in program Grasshopper©. However, most of these programs use classic Lagrangian finite element analysis and therefore translates the NURBS geometry into a simplified model for the structural computations. By implementing isogeometric analysis in these tools the process in the conceptual design phase can be made even more efficient. In most FE-programs, for example ANSYS© the geometries are also simplified to carry out the calculations. With further development of IGA to the FE-programs the work and results of the subsequent design phases can be improved as well.

Shells derives directly from their flow of forces. A well-formed shell transfers the external loads predominantly by membrane forces, i.e. tension and/or compression in the plane of the shell structure. Through this minimization of bending stresses the thickness of a concrete shell loaded with its self-weight can be reduced to around 80mm for reinforced or pre-stressed concrete and 12mm if fiber reinforcement is used (Adriaenssens et al. 2014). Not only does this lightness has an aesthetically appealing effect, the material savings does also have positive effects on the cost and environmental aspects.

Shell structures comprises of both continuous and discrete surfaces, i.e. membranes or grid shells. They can both be generated in three ways;

- By freeform where the shell is generated without consideration to the structural performance (Figure 1).
- Through form finding which includes both natural hanging shapes and digital form finding (Figure 2 and Figure 3).
- Mathematically, where the shape is described by analytical functions (Figure 2).

Figure 1: Shell and Shadows by Zaha Hadid Architects, see [1].

Figure 2: The roof over the British Museum Great Court designed by Foster+Partners. The form is mathematically generated by Chris Williams, but the triangularization of the grid has been form found, see [2].

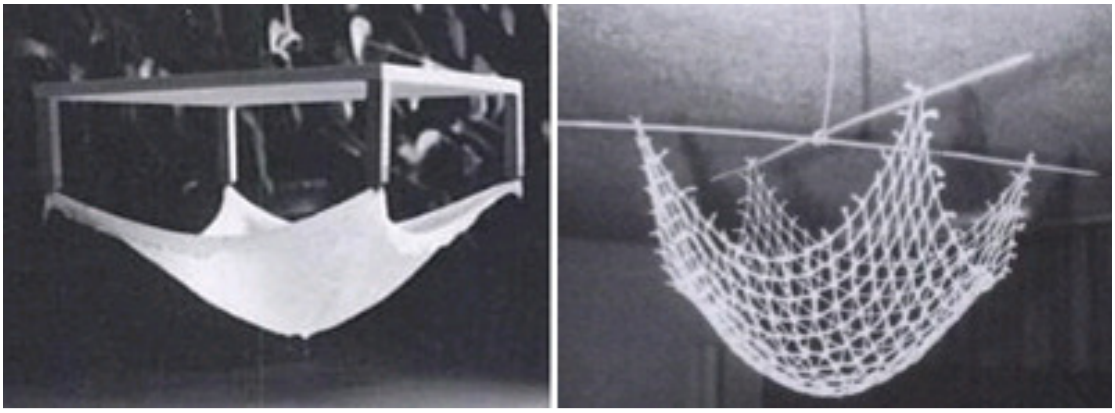


Figure 3: Form finding experiments by Heinz Isler, see [3].

Form finding is the process of finding an optimal form based on criteria regarding for example minimizing material use or deflections. The design loading for form finding the geometry is often chosen to be the structure's self-weight.

Dynamic relaxation is a numerical method, using finite elements, that iteratively searches to find the solution where all forces are in equilibrium. By using this method an idea of an optimal form can be used in the conceptual design of a structure. As a physical example consider the principle of the hanging chain. By applying a gravitational field to the hanging chain, it will eventually find its most optimal form, i.e. the chain will be in equilibrium. One problem that arises when using these types of methods is how to keep free edges straight in the search for equilibrium. This restriction in design limits the design cases for which it can be used. To further develop this method and to be able to use it for a wider set of situations, a way to keep a desired geometry and satisfy the boundary conditions for the specific case is necessary.

1.2 Purpose

The purpose of this thesis has been to develop a form finding program for membranes that uses dynamic relaxation in combination with isogeometric analysis. The program is built as a plug-in for the 3D modeling software Rhinoceros©. The program is written in the coding language C# using VisualStudio©. The aim is to explore new methods of FE-modeling and to evaluate the possibilities and difficulties of using isogeometric elements in a design process.

A literature study followed by the design of a plug-in using form finding will give a deeper understanding of the method. How does the method differ from the traditional form finding method using isoparametric elements? What advantages are there in using isogeometric elements? Is there any differences in mesh sizes and computational effort?

A case study of the roof of the British Museum will be carried out to apply the method to a real problem. What is the context of the roof and the building? How was the form finding process in this case? What difficulties were dealt with then? And could it have been made more efficient by using isogeometric analysis?

1.3 Method

The work during the project has been roughly divided into three main parts with the writing of the report as a continuous part throughout the entire project.

Initially a literature study and a "crash course" was carried out. This first part of the project was necessary to understand the different steps of the methods that were processed in the remaining parts. The literature study consisted of reading old theses, books and articles that treat the subjects. The crash course concerned dynamic relaxation, NURBS and isogeometric analysis with literature and coding examples.

As an application of the first part of the project a plug-in program to the software Rhinoceros© was developed. The program uses dynamic relaxation with isogeometric analysis of membrane structures. The program is built in Visual Studio© using the coding language C#. This choice of language is because of the existing project wizard RhinoCommon©. RhinoCommon© is a .NET plug-in SDK (Software Development Kit) containing templates for creating plug-ins using Visual Studio© for Rhinoceros©.

As a starting point for the program, existing functions for building the geometry and calculating the basis functions in MATLAB© written by Vedad Alic was tested. These MATLAB©-codes was then translated to C#-codes and the program was supplemented with a code for the dynamic relaxation routine. In the beginning these calculations were made purely in Visual Studio© with no connection to Rhinoceros© to make sure the classes and methods in the program were working as intended. Comparisons with results from MATLAB© calculations was made to ensure the accuracy of the results.

Parallel to this "calculation engine" the coding and design for the interface of the plug-in was developed using RhinoCommon©. When both codes worked without any problems they were connected and modified so that the geometry and other input for the analysis was read directly from Rhinoceros©.

After having explored the attributes of structures analyzed with form finding and developed understanding for isogeometric analysis a case study of the roof of the British Museum was carried out. This was done to understand the process from concept to realization and how isogeometric analysis and form finding could have been integrated to facilitate the technique. A problem encountered in form finding methods, such as dynamic relaxation, is how the boundary conditions meet a desired design or the restraint of the context of a structure. Different ways to overcome this limitation in the design process was also explored.

1.4 Limitations

The aim of the thesis has been to develop the conceptual tool for membrane structures. For that purpose the tool is supposed to be used for the process from an initial idea to the concretization of a model for further design. Thus the purpose is not to replace existing software for final design calculations to ensure the load carrying capacity.

The form finding method used in this thesis is dynamic relaxation. There are several other methods, such as the force density method, that could have been used (Adriaenssens et al. 2014). However dynamic relaxation was chosen because of the relatively straightforward implementation of the method.

The structure for analysis that is treated is membranes, although isogeometric analysis can be used for other structural elements with some modifications of the calculations.

In the calculations only evenly distributed loading is considered and the setting of the boundary conditions is limited to the control points.

1.5 Outline of the report

The report is divided into the same parts as the proceeding of the work in this thesis. After the introduction of the project, the theory behind the work is explained. The theory covers NURBS and B-Splines, from which the NURBS are derived, as well as the method of analysis used. These methods are the classic finite element method and the isogeometric finite element method, both constructed on the isoparametric concept. Finally the theory of dynamic relaxation is explained.

Then follows the main section of the thesis which shows the implementation of the theory in the plug-in program in Rhinoceros©.

The case study of the British Museum then follows with an explanation of the design and context of the building. A section of the form finding process for this particular structure is described and connected to the form finding process treated in this thesis.

To sum up the report there is a discussion on what has been brought forward during this work and what conclusions can be drawn. The discussion also reconnects to the argument on how to develop the integration between architects and engineers. Suggestions on how to proceed this work for further development is also added for advice on other forthcoming studies.

2 Theory

This chapter covers the theory of the subjects in the thesis. First are the theory of NURBS and the mathematics of the geometry of the basis used in isogeometric analysis presented. The chapter then moves on to the method of analysis where the isogeometric analysis is compared with classic FE-analysis. The reader is then assumed to already have some basic knowledge of the finite element method. Finally, there is a section covering the theory of dynamic relaxation, which is the form finding method used in the thesis, including a general algorithm showing the important steps of the procedure.

2.1 NURBS

In order to properly explain the structure and behaviour of NURBS it is natural to first provide an introduction to B-Splines. For a more thorough review of B-splines and NURBS the reader is referred to Cottrell, J.A. et al. (2009) from which the notations used below follows.

2.1.1 B-Splines

A B-Spline is a curve that consists of several polynomial segments and is defined by the polynomial degree of each segment, the knot vector and the control points. The polynomial order and the knot vector define the basis functions of the B-spline segments. By adding the control points, the curve is defined as a linear combination of the basis functions.

Knot vectors

The knot vector, $\Xi = [\xi_1, \xi_2, \xi_3, \ldots, \xi_{n+p+1}]$, is a non-decreasing sequence of coordinates in the parameter space, where ξ_i is the ith knot, n is the number of basis functions used to construct the curve and p is the polynomial order of the curve. The knots divide the curve into segments, or the parameter space into elements. The knot vector consists of knot spans that span over the interval $\xi_i \leq \xi \leq \xi_{i+1}$. A knot span can have zero length, the two consecutive knots are then of the same value and the knot is said to be a knot of a certain multiplicity. If all knots are equally spaced in the vector, i.e. the knot spans are of equal lengths, it is a uniform knot vector, and otherwise it is non-uniform. If the first and last knot of a knot vector has the multiplicity $p + 1$, the knot vector is open. A major difference between knots and nodes in classic finite element analysis is that the knots are not necessarily interpolatory. If the knot vector is open the first and last knot will be interpolatory but not, in general, the interior ones. This is why the open knot vector is mostly used in CAD geometries.

As an example, consider the knot vector

$$\Xi = [0,0,0,1,2,3,4,4,5,5,5] \tag{2.1}$$

The first and last knot is of multiplicity three which means that the knot vector is open and non-uniform since the multiplicity implies that the knot spans are of varying lengths. Note that the knot ξ_4 is of multiplicity two. This will have consequences on the continuity over the element which in this case will be C^0. This will be obvious when calculating the basis functions. One can also note that the number of basis functions that will be used to construct the curve is

$$n = k - p - 1 = 8 . \tag{2.2}$$

Where k is the total number of knots in the vector and the polynomial order is 2, since the multiplicity of the first and last knot of the open vector is $p + 1$.

Basis functions

The B-spline basis functions are defined by using equations (2.3) and (2.4) recursively on a knot vector. This procedure follows the Cox de Boor recursion formula.
For $p = 0$ the function is a step function:

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \le \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} . \tag{2.3}$$

For $p > 0$ the function is a linear combination of two previously calculated functions:

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) . \tag{2.4}$$

The basis functions for the knot vector (2.1) are calculated in Appendix AI. The schematic picture in Figure 4 shows how the basis functions for each degree is generated from the previously calculated functions.

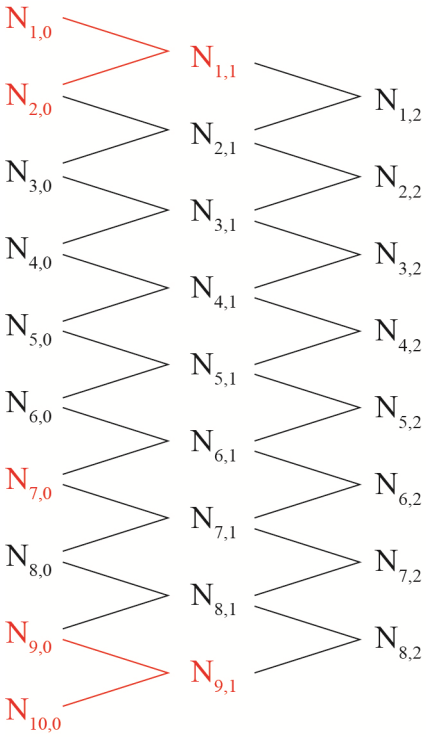


Figure 4: Schematic picture over how the basis functions are generated for p=0, p=1 and p=2. The basis functions in red are zero because of the zero lengths of the knot span.

As an example, by using the knot vector from (2.1), tracing the fifth basis function of the second degree, $N_{5,2}(\xi)$, it is obvious that three functions from the zeroth degree is needed.

$$N_{5,0} = \begin{cases} 1 & \text{if } 2 \le \xi < 3 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{6,0} = \begin{cases} 1 & \text{if } 3 \le \xi < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{7,0} = 0$$

These three step functions gives two functions of the first degree using equation (2.4)

$$N_{5,1} = \begin{cases} \xi - 2 & \text{if } 2 \le \xi < 3 \\ 4 - \xi & \text{if } 3 \le \xi < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{6,1} = \begin{cases} \xi - 3 & \text{if } 3 \le \xi < 4 \\ 0 & \text{otherwise} \end{cases}$$

Finally by using equation (2.4) again on these two first degree functions, $N_{5,2}(\xi)$ is obtained.

$$N_{5,2} = \begin{cases} \dfrac{(\xi-2)^2}{2} & \text{if } 2 \le \xi < 3 \\ (\xi-3)\cdot(4-\xi)+(4-\xi)^2 & \text{if } 3 \le \xi < 4 \\ 0 & \text{otherwise} \end{cases}$$

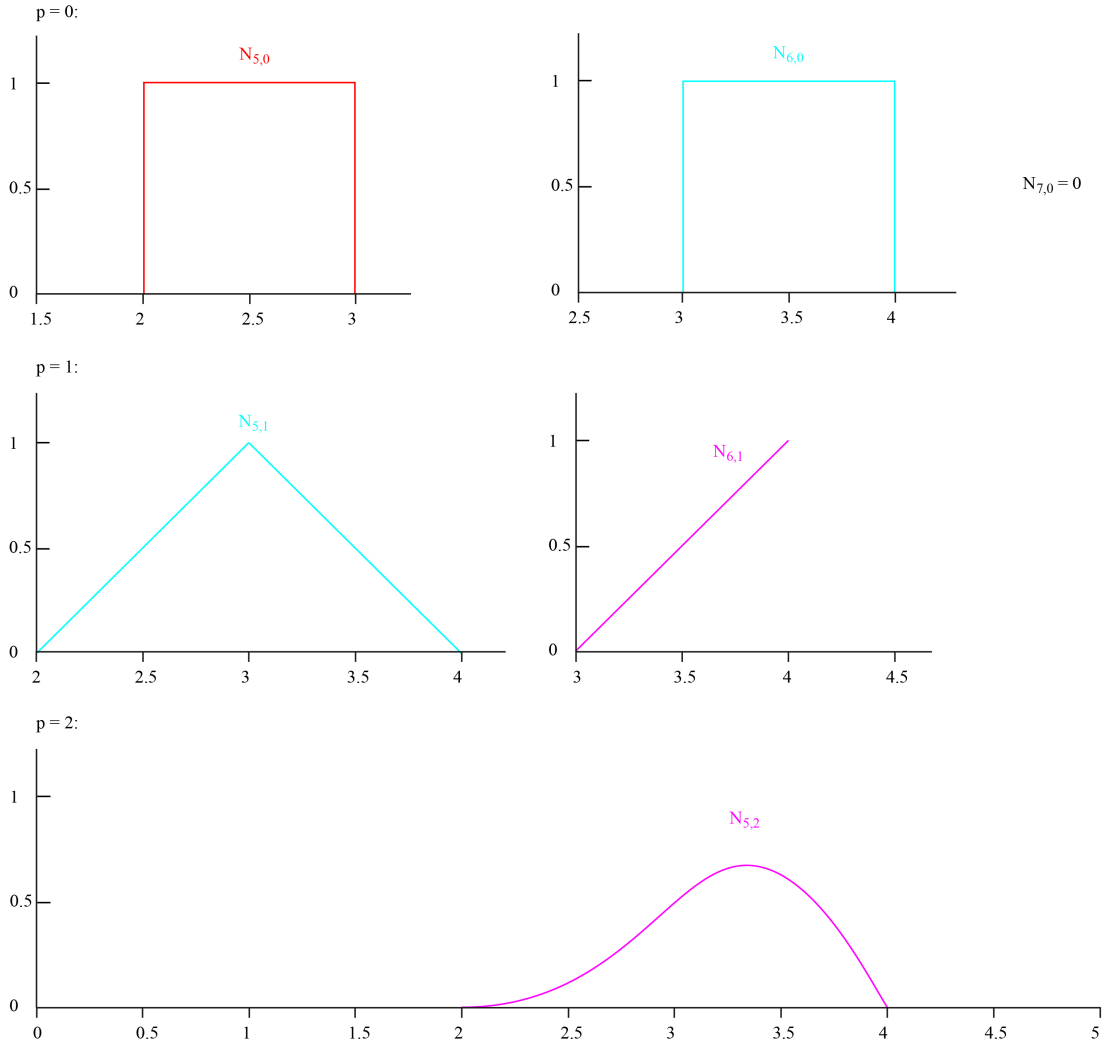


Figure 5: Generating the fifth basis function from the previous functions of lower degree.

The resulting plots for all basis functions calculated in Appendix AI are shown in Figure 6, Figure 7 and Figure 8. The step functions of the zeroth degree basis functions $p = 0$:

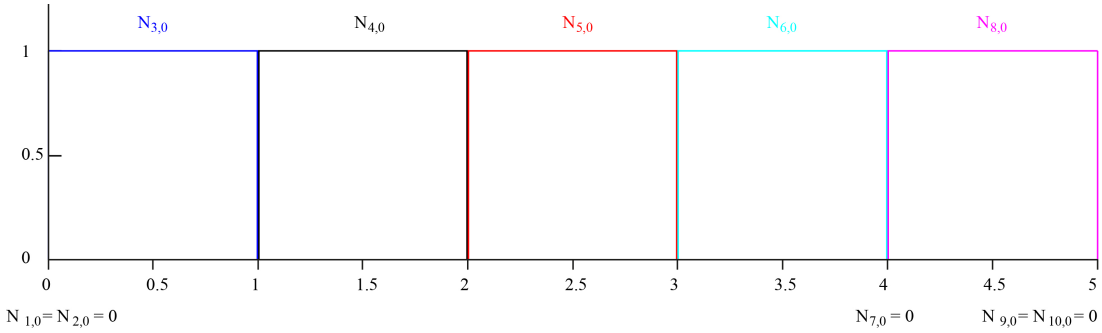


Figure 6: Form functions for p=0

For $p = 1$ the step functions are interpolated to linear functions:

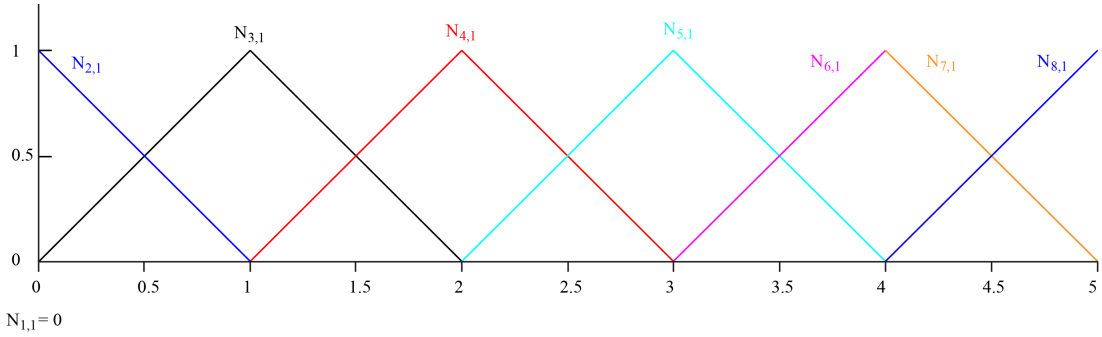


Figure 7: Form functions for p=1

The second degree basis functions, $p = 2$, are interpolated from the linear functions:

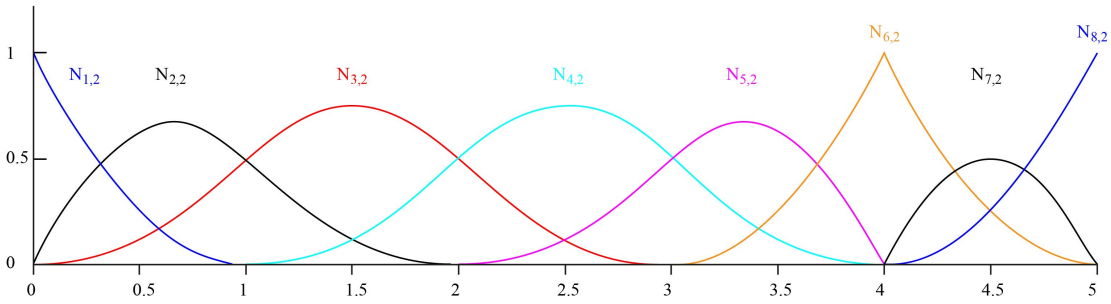


Figure 8: Form functions for p=2

Properties of the basis

There are several important properties of these functions, some which are common for the Lagrangian elements. The property of partition of unity, i.e. the sum of all basis in each point is equal to one is one of them.

$$\sum_{i=1}^{n} N_{i,p}(\xi) = 1 \tag{2.5}$$

A distinctive and important property of isogeometric analysis is that a p^{th} order function always has p-1 continuous derivatives over the knots, provided that the interior knots are of multiplicity one. This means that it's possible to obtain a higher continuity over element boundaries, which leads to a more accurate approximation of the sought physical field, i.e. the displacements in this case. A B-spline function of p^{th} order also has support over p+1 knot spans, which means that a higher order function has support over a larger portion of the domain compared to classic Lagrangian functions. However the bandwidth of the matrices used in the numerical method will not change. Regardless of the method of analysis, the number of functions that any function shares support with will be $2p$+1.

Derivatives of the basis functions

The derivative of the basis function $N_{i,\,p}$ is given by

$$\frac{d}{d\xi} N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)\,. \tag{2.6}$$

In a generalized formula for higher derivatives this is written as

$$\frac{d^k}{d^k\xi} N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}\left(\frac{d^{k-1}}{d^{k-1}\xi} N_{i,p-1}(\xi)\right) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}\left(\frac{d^{k-1}}{d^{k-1}\xi} N_{i+1,p-1}(\xi)\right). \tag{2.7}$$

Control points

The control points are the coefficients of the basis functions that are used to create the B-spline curves. By taking the linear combination of the basis functions the curve is given by

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{B}_i\,. \tag{2.8}$$

By using different control points for the same basis functions, two different curves are created. However, as the same basis is used, the curves have the same properties regarding the degree and continuity over the element boundaries. In Figure 9 and Figure 10 the basis from Figure 8 is used with two different sets of control points, resulting in two different curves.

The piecewise linear combination of the control points gives the control polygon. The control polygon is tangent to the curve in the interior knots which means that the curve has continuity $C^{p-1} = C^1$ at these element boundaries. At the kink at the repeated knot, the continuity is $C^{p-2} = C^0$. Note the repeated knot at control point (10, 7) in Figure 9 and (10, 9) in Figure 10. Because of the C^0 continuity followed by the repeated knot, the curve is interpolatory in the control point, i.e. the knot coincides with the control point.

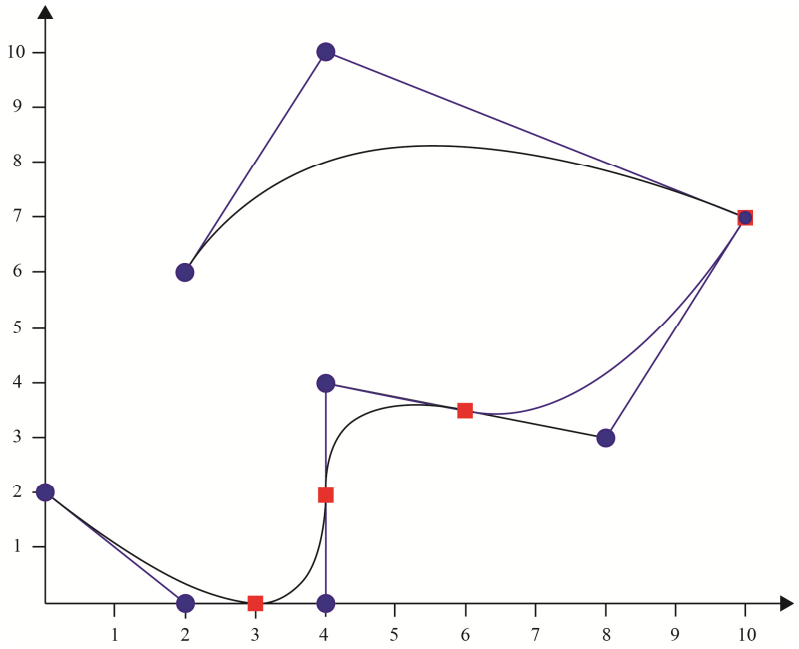


Figure 9: B-Spline curve where the blue dots mark the control points, or nodes, connected by the control polygon. The red squares mark the knots, i.e. the element boundaries.

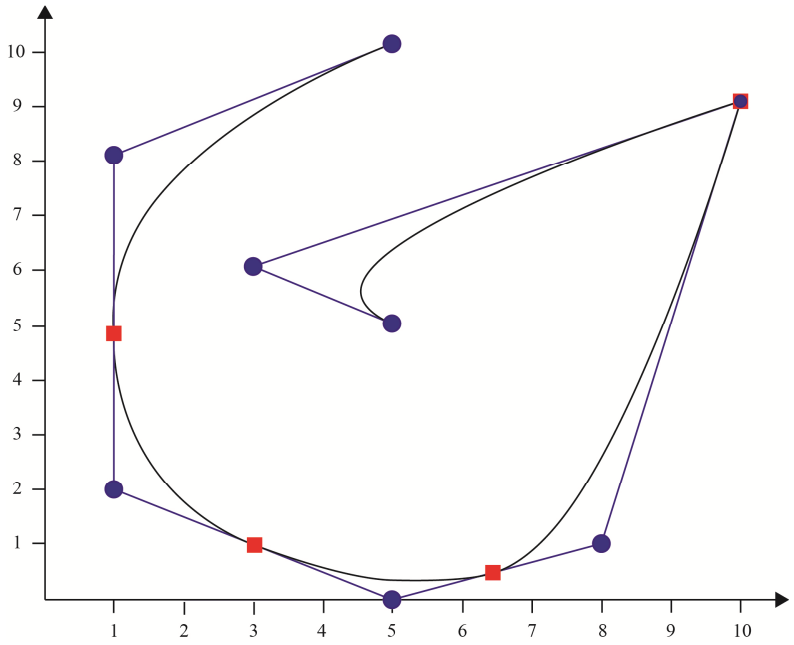


Figure 10: B-Spline curve created from the same knot vector as in Figure 9, but with other control points.

B-Spline surfaces

The discussion so far has only covered the B-spline curve but moving on to surfaces is more or less the same thing only in two directions. The B-spline surface is defined by two knot vectors $\Xi = [\xi_1, \xi_2, \xi_3, \ldots, \xi_{n+p+1}]$ and $H = [\eta_1, \eta_2, \eta_3, \ldots, \eta_{m+q+1}]$, the polynomial orders p and q and the control net $\{\mathbf{B}_{i,j}\}$ where i = 1,2,…,n and j = 1,2,…,m. The tensor product of two univariate basis functions gives the bivariate surface defined by

$$\mathbf{S}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta)\mathbf{B}_{i,j}\,. \tag{2.9}$$

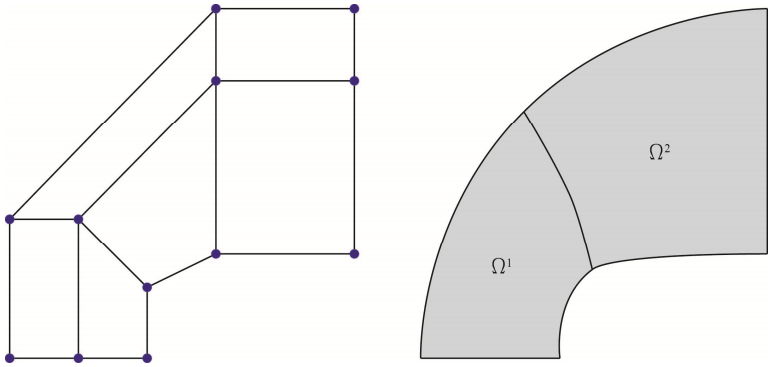


Figure 11: Biquadratic surface from the knot vectors Ξ= [0 0 0 0.5 1 1 1] and H= [0 0 0 1 1 1]. To the left is the net of control points and to the right is the mesh consisting of two elements.

Refinement

There are several ways to enrich a B-spline without changing the initial geometry. Two of these tools are fundamental in isogeometric analysis; knot insertion and degree elevation. The changes are made in to the knot vector in the parameter space hence why it doesn't change the geometry in the physical space.

Knot insertion is either when a new knot is inserted and by doing so, creating a new subdivision of the curve. This is typically done to obtain a finer mesh of the geometry. Knot insertion is also when an existing knot is inserted hence decreasing the continuity over the element boundary. By inserting a new knot, one more control point, one more element and one more basis function than before the refinement is added. Knot insertion has similarities with the h-refinement in classical finite element analysis where elements are split into new elements. The difference is however that the h-refinement always will have C^0 continuity over the element boundaries.

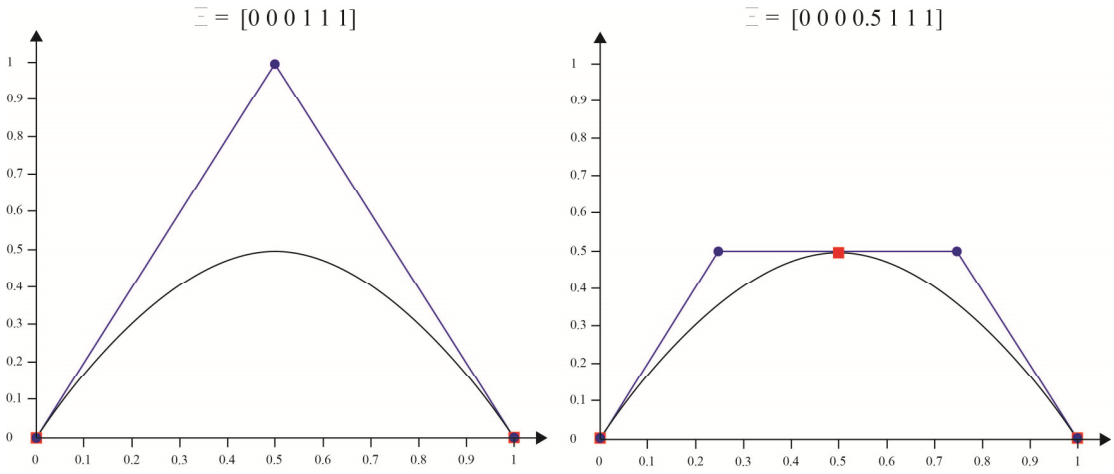


Figure 12: Knot insertion. The original curve to the left has three control points (blue dot) and two knots (red square), i.e. the curve consists of one element. The refined curve to the right has one more control point and one more knot, hence two elements.

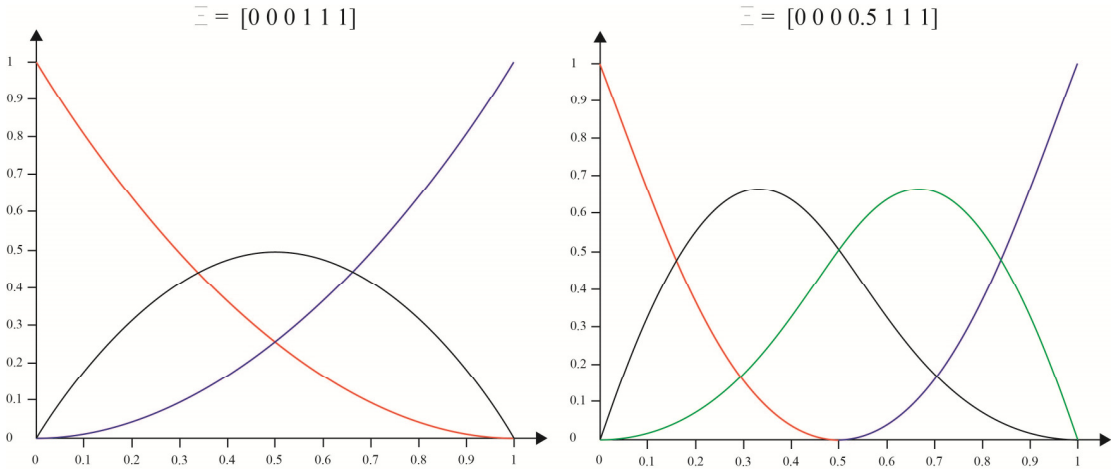


Figure 13: Knot insertion. The basis on the left with three functions is the original. The refined basis on the right has one more basis function.

With degree elevation the polynomial order can be raised without changing the continuities of the knots. This is achieved by increasing the multiplicity of all existing knots, i.e. no new knots are inserted. Since all the knots are raised, the continuity of the interior knots is preserved. Degree elevation is similar to p-refinement in classic finite element analysis. In p-refinement, the order is increased but the basis is always C^0 before the refinement. The degree elevation can be performed no matter what the initial degree is, which makes the isogeometric analysis more flexible in dealing with higher order techniques.

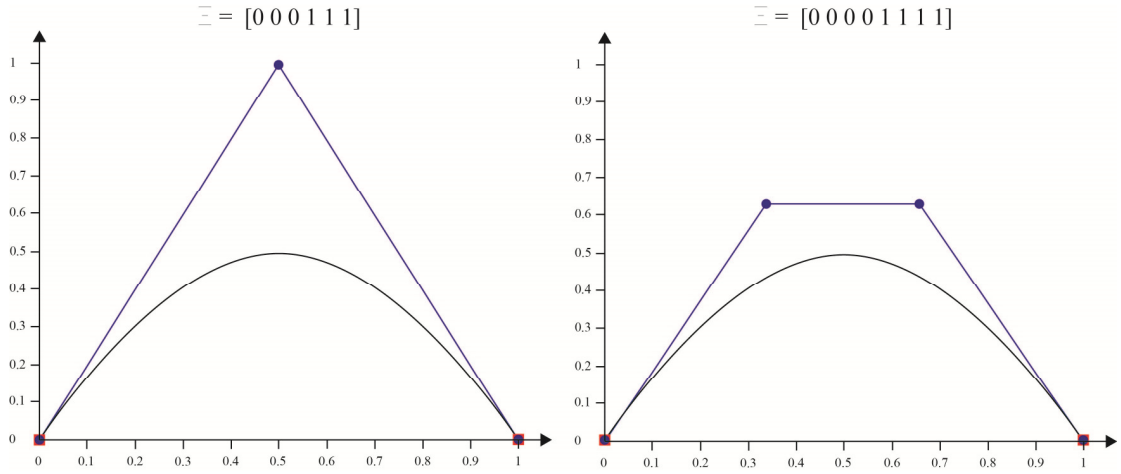


Figure 14: Order elevation. The curve on the left is the original curve, as in Figure 12. The refined curve on the right has one more control point but the curve still consists of only one element.

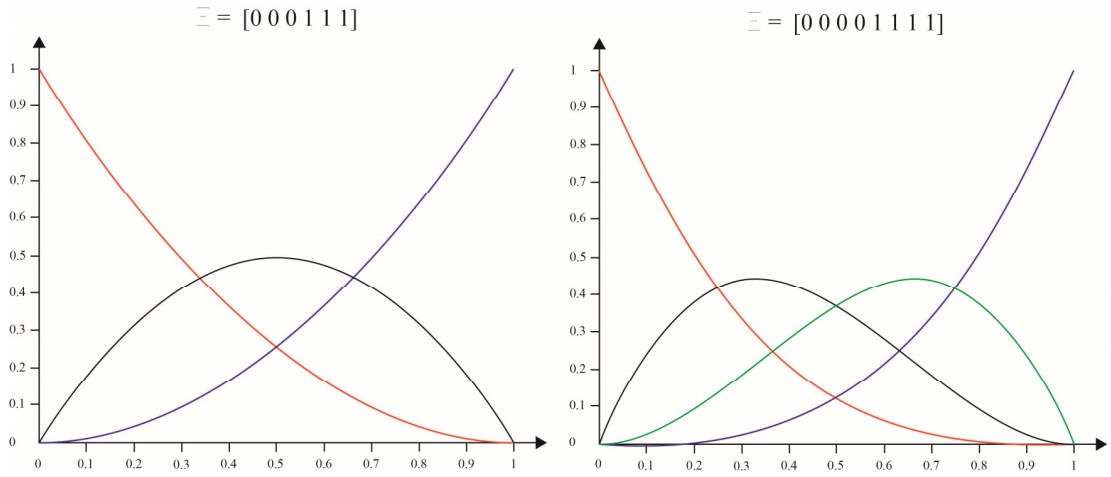


Figure 15: Order elevation. The refined basis on the right consists of one more basis function, as in Figure 13.

If both higher order and higher continuity is desirable, the so-called k-refinement can be used. The k-refinement is a combination of knot insertion and order elevation and is unique for the isogeometric analysis. The order in which the actions are performed is crucial. If a new knot is inserted into a vector of degree p, the continuity over the new knot will be p-1. When performing order elevation of the vector to degree q the new knot will still have p-1 continuity. If the procedure is instead reversed, so that the vector is elevated from degree p to q and then the new knot is inserted, the knot will have continuity q-1.

2.1.2 Non-Uniform Rational B-Splines

NURBS stands for Non-Uniform Rational B-Splines and are weighted B-Splines with weighted control points. In CAD geometries one usually wants a knot vector with uneven knot spans, for example so that the curve is interpolatory in the start and end points, hence the knot vector is non-uniform. The rational property comes from the fact that, as will be obvious further down in the text, the shape functions are piecewise rational functions, i.e. functions consisting of one polynomial divided by another polynomial.

The major difference when moving on to NURBS is the weighting of the control points. By assigning different weights to the control points, the points will have a different amount of influence on the curve. This is what makes one of the defining features of isogeometric analysis since it makes it possible to construct any polynomial or rational (such as a circle for example). Since the weights are in proportion to each other one could say that the B-Splines also indirectly has weights of equal value.

The *i*th control point of the NURBS curve, $\mathbf{B}_i$, is obtained from the corresponding B-Spline control point, $\mathbf{B}_i^w$ and weight w_i:

$$\left(\mathbf{B}_i\right)_j = \frac{\left(\mathbf{B}_i^w\right)_j}{w_i} \qquad j = 1,\ldots,d \tag{2.10}$$

$$w_i = \left(\mathbf{B}_i^w\right)_{d+1}, \tag{2.11}$$

where d is the dimension of the entity.

In order to apply the transformation to an entire curve, the weighting function is defined as:

$$W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) w_i \,, \tag{2.12}$$

where $N_{i,p}$ is the B-Spline basis function. With the weighting function the NURBS basis function is defined as:

$$R_i^p(\xi) = \frac{N_{i,p}(\xi) w_i}{W(\xi)} = \frac{N_{i,p}(\xi) w_i}{\sum_{i=1}^{n} N_{i,p}(\xi) w_i} \,. \tag{2.13}$$

Combining the control points from (2.10) and the basis functions (2.13) the NURBS curve is given by:

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} R_i^p(\xi) \mathbf{B}_i \,. \tag{2.14}$$

Expanding into two dimensions, the basis for the NURBS surface is obtained in a similar way:

$$R_{i,j}^{p,q}(\xi,\eta)=\frac{N_{i,p}(\xi)M_{j,q}(\eta)w_{i,j}}{\sum_{i=1}^{n}\sum_{j=1}^{m}N_{i,p}(\xi)M_{j,q}(\eta)w_{i,j}}. \tag{2.15}$$

The derivatives of the basis functions are obtained through the quotient rule

$$\frac{d}{d\xi}R_i^p(\xi)=w_i\frac{W(\xi)N_{i,p}^{'}(\xi)-W^{'}(\xi)N_{i,p}(\xi)}{(W(\xi))^2}, \tag{2.16}$$

where

$$W^{'}(\xi)=\sum_{i=1}^{n}N_{i,p}^{'}(\xi)w_i\ . \tag{2.17}$$

For higher order derivatives, the expression gets more complicated

$$\frac{d^k}{d\xi^k}R_i^p(\xi)=\frac{A_i^{(k)}(\xi)-\sum_{j=1}^{k}\binom{k}{j}W^{(j)}(\xi)\frac{d^{(k-j)}}{d\xi^{(k-j)}}R_i^p(\xi)}{W(\xi)}, \tag{2.18}$$

where

$$A_i^{(k)}(\xi)=w_i\frac{d^k}{d\xi^k}N_{i,p}(\xi) \tag{2.19}$$

and

$$\binom{k}{j}=\frac{k!}{j!(k-j)!}. \tag{2.20}$$

2.1.3 Multiple patches

One mayor advantage of using isogeometric analysis is the ability to use multiple patches. This can be necessary if different parts of the model for example has different material properties in different parts of the model or a complex geometry.

Each patch represents a subdomain of elements in the same parameter space. The transition between patches, i.e. the knots where two patches meet, will be C^0 since the edges has to be interpolatory to meet.

2.2 Methods of analysis

The finite element method is a numerical method that is used to solve partial differential equations. The method was developed from two independent directions; structural mechanics and mathematics. It was first introduced in the 1930s on solving problems of mathematical physics and was then further developed. After the Second World War and during the 60s the method grew, partly because of the introduction of computers and became well established for structural analysis of elasticity problems and structural mechanics (Thomée, 1999). After further development the method was generalized to function in several other areas in engineering, such as dynamics of fluids and heat transfer.

The basis of the analysis is to divide a continuous domain into discrete subdomains connected by nodes. Each node has a function that is nonzero and continuous over the elements connected by the node. This function is known as the form function or basis function of the node and describes how a specific node influences the solution. By taking the linear combination of all element functions, the approximate solution for the whole problem is obtained.

By providing a short introduction to both classic isoparametric and isogeometric analysis the two methods will be further evaluated and compared. For more information about isoparametric analysis the reader is referred to Ottosen, P. (1992). A thorough description of isogeometric analysis can be found in Cottrell, J. A. et al. (2009).

2.2.1 Classic Finite Element Analysis

The finite element method used is based on the Galerkin method for solving boundary value problems, see Ottosen, P. (1992) for explanation and references to other methods. The method starts with stating the strong form of the problem, i.e. the differential equation of the problem at hand. As an example, consider the boundary value problem

$$\begin{aligned} \boldsymbol{\nabla}^T \boldsymbol{D} \boldsymbol{\nabla} \boldsymbol{u} + \boldsymbol{f} &= 0 \ \text{ on } \Omega \\ \boldsymbol{u} &= \boldsymbol{g} \ \text{ on } \Gamma_g \\ \boldsymbol{\nabla} \boldsymbol{u} \cdot \boldsymbol{n} &= \boldsymbol{h} \ \text{ on } \Gamma_h \end{aligned} \tag{2.21}$$

where $\Gamma_g \cup \Gamma_h = \Gamma = \partial\Omega$ denotes the boundary and **n** is the normal of $\partial\Omega$. The unknown solution, **u,** is sought. For an elastic problem $\boldsymbol{f}$ denotes the known body force and $\boldsymbol{h}$ denotes a known traction force on the boundary.

By multiplying the differential equation with a weighting function, **v**, and integrate by parts using the Green Gauss theorem the weak form of the problem is achieved

$$\int_\Omega (\boldsymbol{\nabla} \boldsymbol{v})^T \boldsymbol{D} \boldsymbol{\nabla} \boldsymbol{u} d\Omega = \int_\Omega \boldsymbol{v}^T \boldsymbol{f} d\Omega + \int_\Omega \boldsymbol{v}^T \boldsymbol{h} d\Gamma \tag{2.22}$$

$$\boldsymbol{u} = \boldsymbol{g} \ \ on \ \Gamma_g.$$

To obtain a solvable system of linear equations an approximation of the solution field is required. According to the Galerkin method, the weighting function should be chosen to be equal to the approximation function of the solution. Hence,

$$\begin{aligned} \mathbf{u} = \mathbf{N} \cdot \mathbf{a} \quad &\Rightarrow \quad \mathbf{v} = \mathbf{N} \cdot \mathbf{c} \\ \nabla \mathbf{u} = \nabla \mathbf{N} \cdot \mathbf{a} = \mathbf{B} \cdot \mathbf{a} \quad &\Rightarrow \quad \nabla \mathbf{v} = \nabla \mathbf{N} \cdot \mathbf{c} = \mathbf{B} \cdot \mathbf{c} \end{aligned} \tag{2.23}$$

where **N** is the vector containing the form functions, or basis functions, of the elements and **a** the vector of unknown scalar displacements of the nodes. Inserting these relationships into the weak form in (2.22) the matrix equations are obtained:

$$\int_\Omega \mathbf{B}^T \boldsymbol{D} \mathbf{B} \mathbf{a} \, d\Omega = \int_\Omega \boldsymbol{N}^T \boldsymbol{f} d\Omega + \int_\Omega \boldsymbol{N}^T \boldsymbol{h} d\Gamma \quad \Leftrightarrow \quad \boldsymbol{K}\boldsymbol{a} = \boldsymbol{f} \tag{2.24}$$

$$\boldsymbol{K} = \int_\Omega \mathbf{B}^T \boldsymbol{D} \mathbf{B} \, d\Omega, \quad \boldsymbol{f} = \int_\Omega \boldsymbol{N}^T \boldsymbol{f} d\Omega + \int_\Omega \boldsymbol{N}^T \boldsymbol{h} d\Gamma$$

where **D** is the constitutive matrix.

Basis functions

Classic FEA using Lagrangian elements follows the isoparametric concept. The concept of isoparametric analysis is that the same basis that is used in the numerical method for the structural analysis of the model is also used for describing the geometry of the model.

The choice of approximation has to fulfil the convergence criteria which comprises of the requirements of completeness and compatibility. The completeness requirement is that the approximation must be able to describe at least an arbitrary linear function in each element. The compatibility requirement is that the approximation of the solution must be continuous over the element boundaries. The simplest shape functions for a one-dimensional element are linear functions, see equation (2.25) and Figure 16. It is obvious from Figure 16 that the shape functions are continuous over the element (the dotted lines represent an adjacent element) however the gradient of the approximated solution will be discontinuous, i.e. the boundary has C^0-continuity.

$$\begin{aligned} N_i^e &= -\frac{1}{L}\left(x - x_j\right) \\ N_j^e &= \frac{1}{L}\left(x - x_i\right) \end{aligned} \tag{2.25}$$

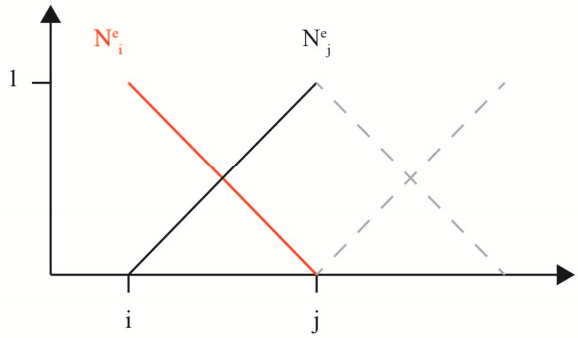


Figure 16: Variation of linear element shape functions.

As the polynomial degree is raised to quadratic elements, equation (2.26) and Figure 17, it is clear that the C^0-continuity is still present over the element boundary. Since the solution is interpolated between the values of the nodal points, the solution will vary continuously over boundaries but the continuity of the gradient of the solution cannot be assured.

$$
\begin{aligned}
N_i^e &= \frac{2}{L^2}(x - x_j)\cdot(x - x_k) \\
N_j^e &= -\frac{4}{L^2}(x - x_i)\cdot(x - x_k) \\
N_k^e &= \frac{2}{L^2}(x - x_i)\cdot(x - x_j)
\end{aligned}
\tag{2.26}
$$

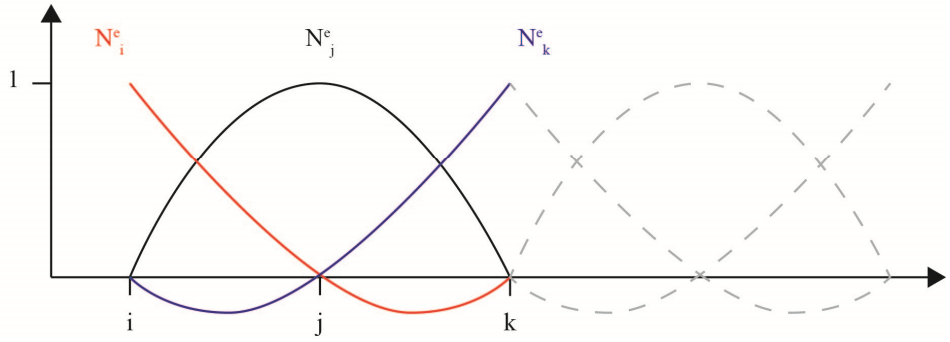


Figure 17: Variation of quadratic element shape functions.

The Lagrange element is a simple rectangular two dimensional element with four nodal points. The bilinear shape functions of the element is

$$
\begin{aligned}
N_i^e &= \frac{1}{4ab}(x - x_j)\cdot(y - y_m) \\
N_j^e &= -\frac{1}{4ab}(x - x_i)\cdot(y - y_k) \\
N_k^e &= \frac{1}{4ab}(x - x_m)\cdot(y - y_j) \\
N_m^e &= -\frac{1}{4ab}(x - x_k)\cdot(y - y_i)\cdot
\end{aligned}
\tag{2.27}
$$

Mapping

When modelling structures with arbitrary geometries the elements has to be mapped from a square region, called the parent domain, in order to fulfil the requirement of compatibility, see Figure 18. The integrations are carried out in the parent element using the shape functions for the Lagrange element in equations (2.27) which are also the shape functions used in the mapping to the global domain, equation (2.28), which is the isoparametric concept. The parent domain or parameter space is local to each of the elements in the global domain, see Figure 18.

$$x = x(\xi,\eta) = \mathbf{N}^e(\xi,\eta)\cdot\mathbf{x}^e \qquad y = y(\xi,\eta) = \mathbf{N}^e(\xi,\eta)\cdot\mathbf{y}^e. \tag{2.28}$$

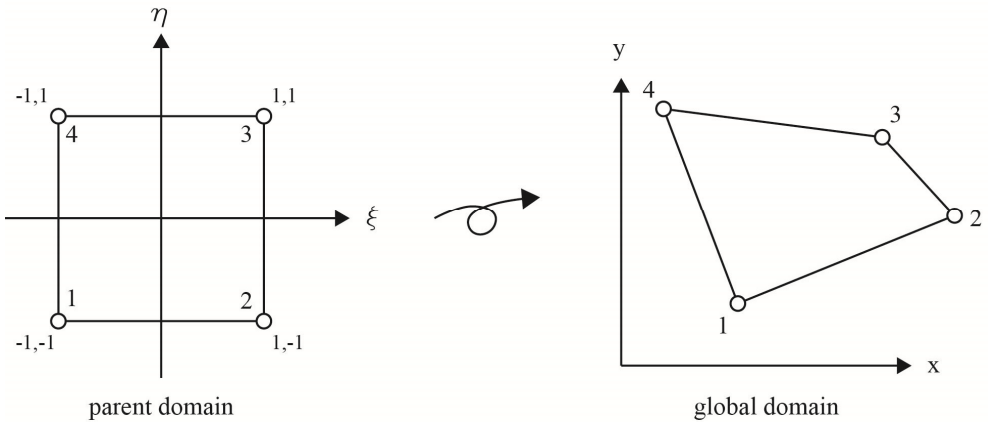


Figure 18: Mapping of an element from the parent domain to the global domain.

Connectivity arrays

Connectivity arrays are used to organize the structure of a model. These arrays contain information that keep track of which elements are connected to a certain node and which degrees of freedom (DOF) belong to which node. In classic FEA there are two such arrays. The DOF-array handles the numbering of the degrees of freedom. Each row in the DOF represents one node and its degrees of freedom. The EDOF-matrix handles the elements of a structure and what DOFs are connected to each element. Hence, by combining the two arrays one can track which nodes are connected to which elements and which DOFs belongs to which node (Austrell et al., 2004)

Code architecture

The flow chart of a program using classical FEA code is shown in Figure 19. The program starts with some pre-processing steps where the data defining the boundary value problem is read. From the input data the connectivity arrays can be set up and the memory for the global arrays is located and set to zero. After these initializations the algorithm for assembling the system starts. This algorithm consists of two loops. The first one loops through all elements where a local stiffness and force matrices are initiated. For each element a second loop goes through the quadrature points where the basis functions and its derivatives are calculated. The contribution from the quadrature point to the element stiffness matrix and force vector are assembled in

each iteration before the second loop is completed. Using the connectivity arrays, the element contribution can then be assembled into the global arrays. When all element stiffness matrices and contributions to the load vector are assembled the system can be solved.

To obtain a code for a single-patch isogeometric analysis the steps in yellow in Figure 19 need to be modified. As the basis is changed the input data will be different as well as the connectivity arrays. The evaluation of the basis functions must be updated as the basis consists of NURBS which are calculated according to Section 2.1.2. As can be seen in the flowchart, the isogeometric analysis fits well into the existing FE-codes and only a few changes are necessary.

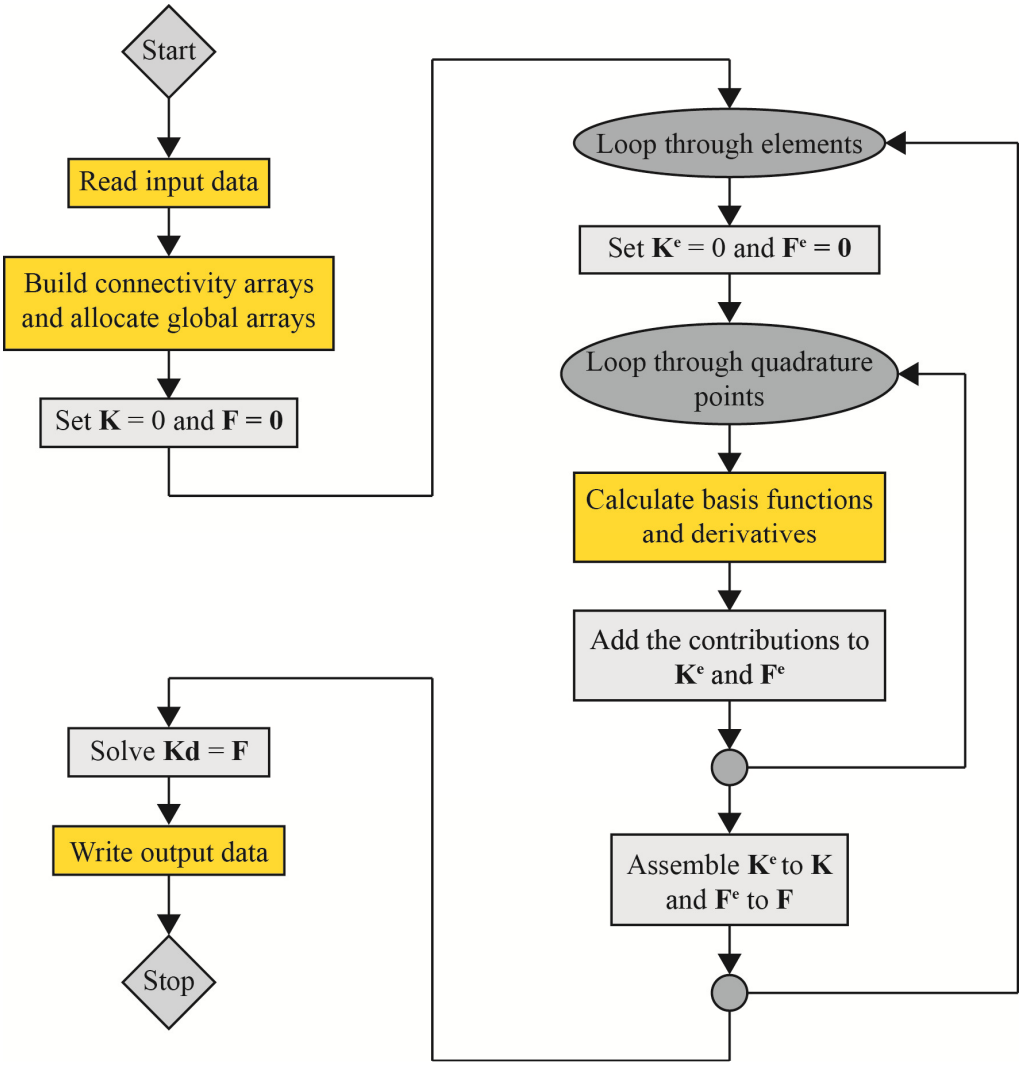


Figure 19: Flow chart of a classical FEA program.

2.2.2 Isogeometric Finite Element Analysis

The concept of isogeometric analysis was conceived by Tom Hughes, a professor at the University of Texas in Austin. The idea was brought to him while studying how the models for finite element analysis are generated from CAD geometries. In 2003 the graduate students Austin Cottrell and Yuri Bazilevs started to study the subject in their PhD work under the supervision of Hughes. During their work they studied the technology behind NURBS and develop NURBS based finite element codes. After them other students have developed the method further under the supervision of Hughes (Cottrell et al., 2009).

The isogeometric analysis also inherits the isoparametric concept, as the same basis is used for describing both the geometry and the solution space. However, there is a fundamental difference in the implementation of the concept. Instead of approximating the geometry, which in fact is already known, after the unknown solution space, the course of action is reversed. By implementing isogeometric analysis the chosen basis will exactly represent the geometry and is then used for approximation of the unknown fields.

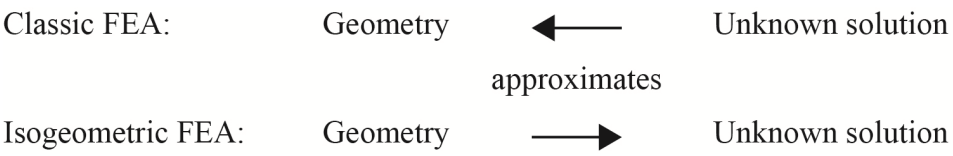


Figure 20: The fundamental difference between classic isoparametric and isogeometric finite element analysis.

Basically the only thing that differ isogeometric analysis from classic FEA is the basis that is being used. However, this difference has large consequences.

Mapping

In isogeometric analysis the parameter space is local to patches, rather than of single elements as in classical FEA. The mapping is therefore carried out in two steps; an affine mapping from the parent element to the parametric domain and a geometrical mapping from the parametric domain to the global domain.

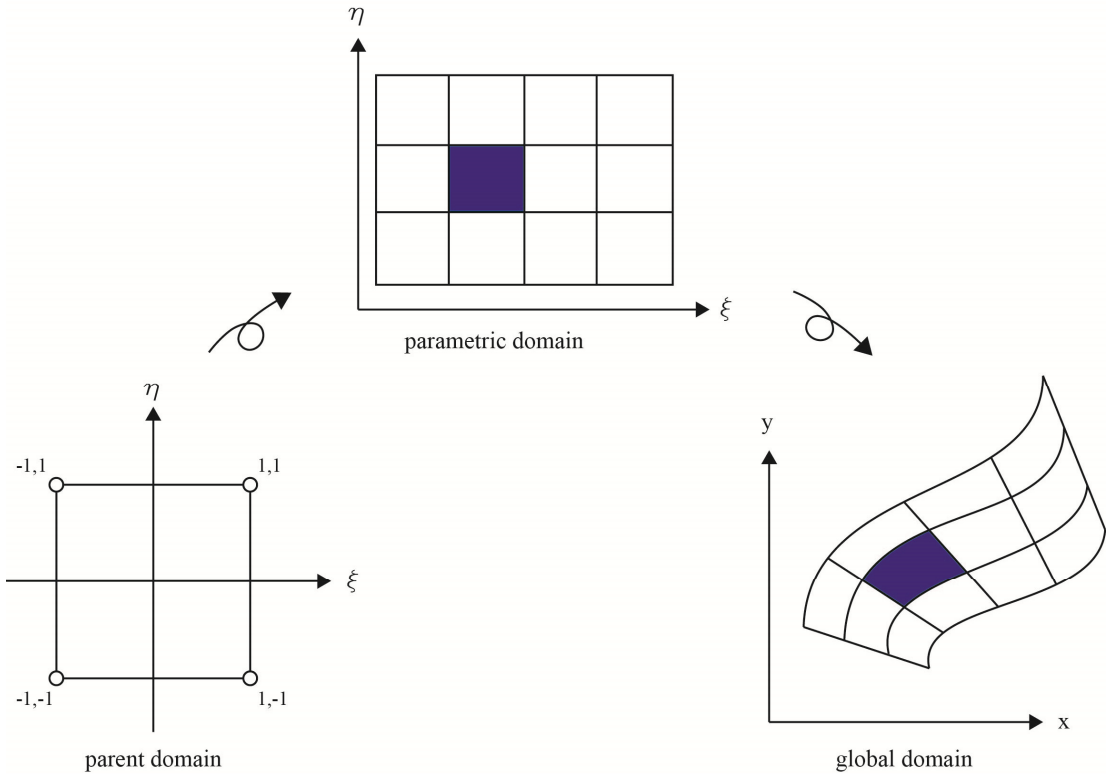


Figure 21: Mapping in two steps; from parent element to parametric domain and from parametric domain to the physical element.

Connectivity arrays

With the difference in the parametric domain from the classic FEA comes a need of using two additional connectivity arrays. The NURBS coordinates are introduced as the indices of the knots in the knot vectors, defining the index space, see Figure 22.

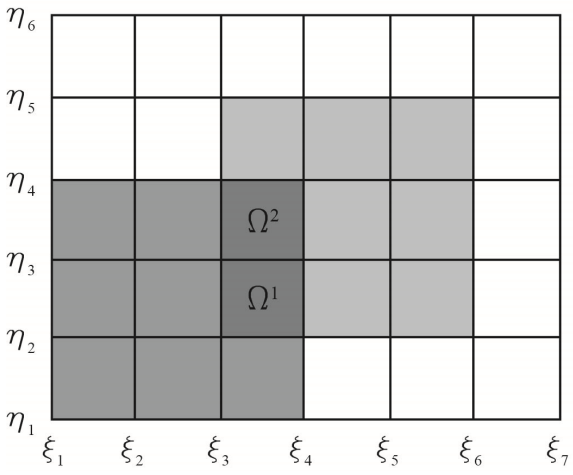


Figure 22: The index space of the mesh in Figure 11 showing the two elements.

The INC array connects the NURBS coordinates to their global shape function number, where each row represent one function. The IEN array tells in which elements the shape functions have support (Cottrell et al., 2009).

Code architecture

The flowchart of the code for multipatch isogeometric analysis is shown in Figure 23. The major difference from the classic FEA is the extra loop through the patches. The change of using multiple patches also results in the partitioning of the input data. The global input that is common for all patches is, as before, read at the starting point. Local input, such as the knot vectors and control points are read in the loop for the different patches. The differences in the routine from the single patch code are highlighted in yellow.

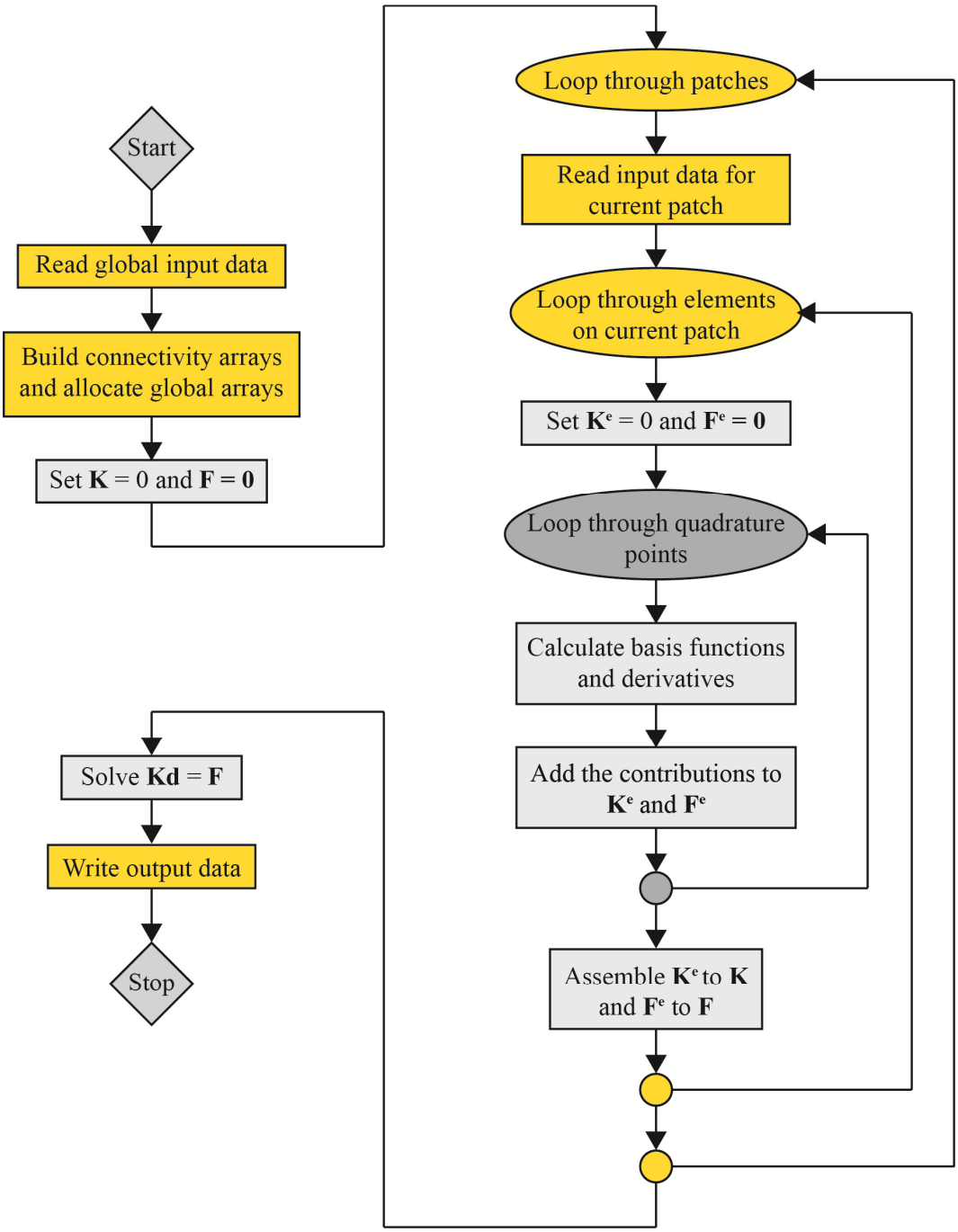


Figure 23: Flow chart of an isogeometric analysis program using multiple patches

An important difference from the classical FEA is that in isogeometric analysis the exact geometry is always used, independent of the level of discretization. The accuracy of the computations is of course the major advantage of this geometric exactness but it also affects the whole analysis process. By using the exact geometry from the beginning, there is no need for an external description of the geometry which will simplify the refinement of the model.

2.3 Dynamic Relaxation

Dynamic relaxation is an explicit numerical method in which a static problem is solved as a fictious dynamic problem and can be used for form finding. The method traces the motion of the nodes step by step for small time increments, Δt, until the structure reaches a state of static equilibrium. It was invented in 1965 by Alistair Day and was originally developed for tidal flow computations (Barnes, 1988 and Adriaenssens et al., 2014). The notations follow the notation according to Barnes, M. R. (1988).

The method is based on Newton's second law, force equals mass times acceleration. The residual force R_{ix} in node *i* in the direction *x* is

$$R_{ix} = M_i \cdot \dot{v}_{ix}, \tag{2.29}$$

where

M_i is the lumped mass in the node *i*,

$\dot{v}_{ix}$ is the acceleration of the node *i* in the *x*-direction.

From equation (2.29) the recurrence equation for calculating the velocity for each time step is obtained

$$v_{ix}^{t+\Delta t} = v_{ix}^{t} + \Delta t \cdot \frac{R_{ix}}{M_i}, \tag{2.30}$$

where

$v_{ix}^{t+\Delta t}$ is the updated velocity at time $t+\Delta t$,

v_{ix}^{t} is the velocity at the previous iteration,

Δt is the time step.

From the updated velocities the new displacements are computed by

$$u_{ix}^{t+\Delta t} = u_{ix}^{t} + \Delta t \cdot v_{ix}^{t+\Delta t}, \tag{2.31}$$

where

$u_{ix}^{t+\Delta t}$ is the updated displacement at time $t+\Delta t$,

u_{ix}^{t} is the displacement at the previous iteration.

To make the structure come to rest in equilibrium, damping is introduced to the solution procedure. There are several kinds of damping, however for this project kinetic damping has been chosen because of its satisfactory convergence properties (Barnes, 1988). When using kinetic damping, the kinetic energy of the nodes is traced. If a peak in the energy is found the system is brought to rest by setting the velocity to zero, see Figure 25. The iteration process is then continued from the current geometry.

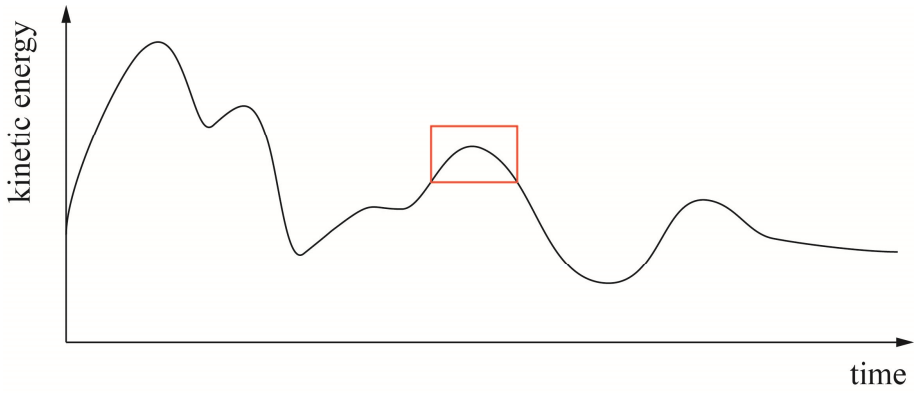


Figure 24: The kinetic energy and its peaks during the dynamic relaxation iterations without damping. A magnified picture of the highlighted piece of the curve is shown in Figure 26.

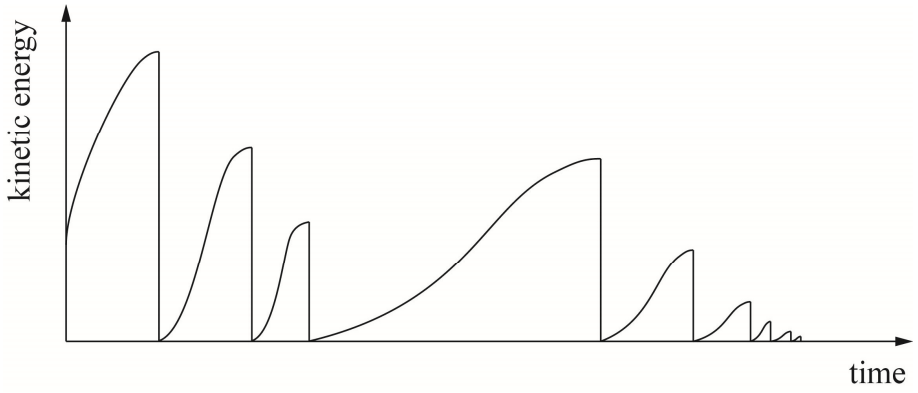


Figure 25: The kinetic energy during the dynamic relaxation routine with kinetic damping. At each peak the kinetic energy is set to zero and continued from that point.

The peak is found when the current kinetic energy is lower than the previous iteration, i.e.

$$K_e^{t+\Delta t} < K_e^t, \tag{2.32}$$

where

$K_e^{t+\Delta t}$ is the current kinetic energy, i.e. the energy at time $t+\Delta t$,

K_e^t is the previous kinetic energy, i.e. the energy at time t.

To find the displacement at the true peak, which occurs sometime between t and $t+\Delta t$, the kinetic energies calculated at t, $t+\Delta t/2$ and $t+\Delta t$ are interpolated, see Figure 26. The time elapsed since the peak is obtained by

$$\delta t^* = \Delta t \cdot \frac{E}{E - D} = \Delta t \cdot q \,, \tag{2.33}$$

where

$$E = C - B \quad \text{and} \quad D = B - A \,, \tag{2.34}$$

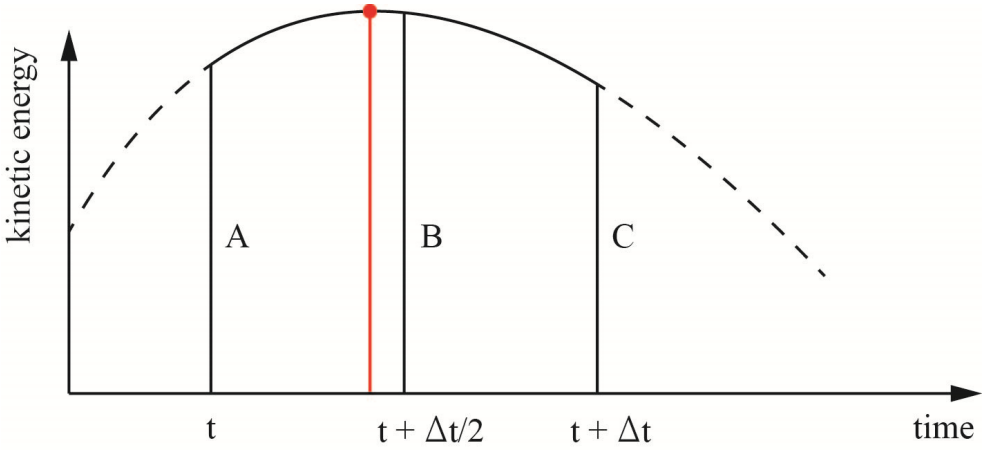


Figure 26: Magnification of the highlighted area in Figure 24. A, B, and C is the kinetic energy at the specific times. The red dot marks the location of the true peak.

With the time for the true peak the displacement is calculated by

$$u_{ix}^{t^*} = u_{ix}^{t+\Delta t} - \Delta t \cdot (1+q) \cdot v_{ix}^{t+\Delta t} + \frac{\Delta t^2}{2} \cdot q \cdot \frac{R_{ix}}{M_i} \,, \tag{2.35}$$

As an alternative the true peak can be assumed to occur at $t+\Delta t/2$ which would mean that the q in equation (2.33) is equal to 1/2.

A general algorithm for a dynamic relaxation program using kinetic damping is presented in Figure 27.

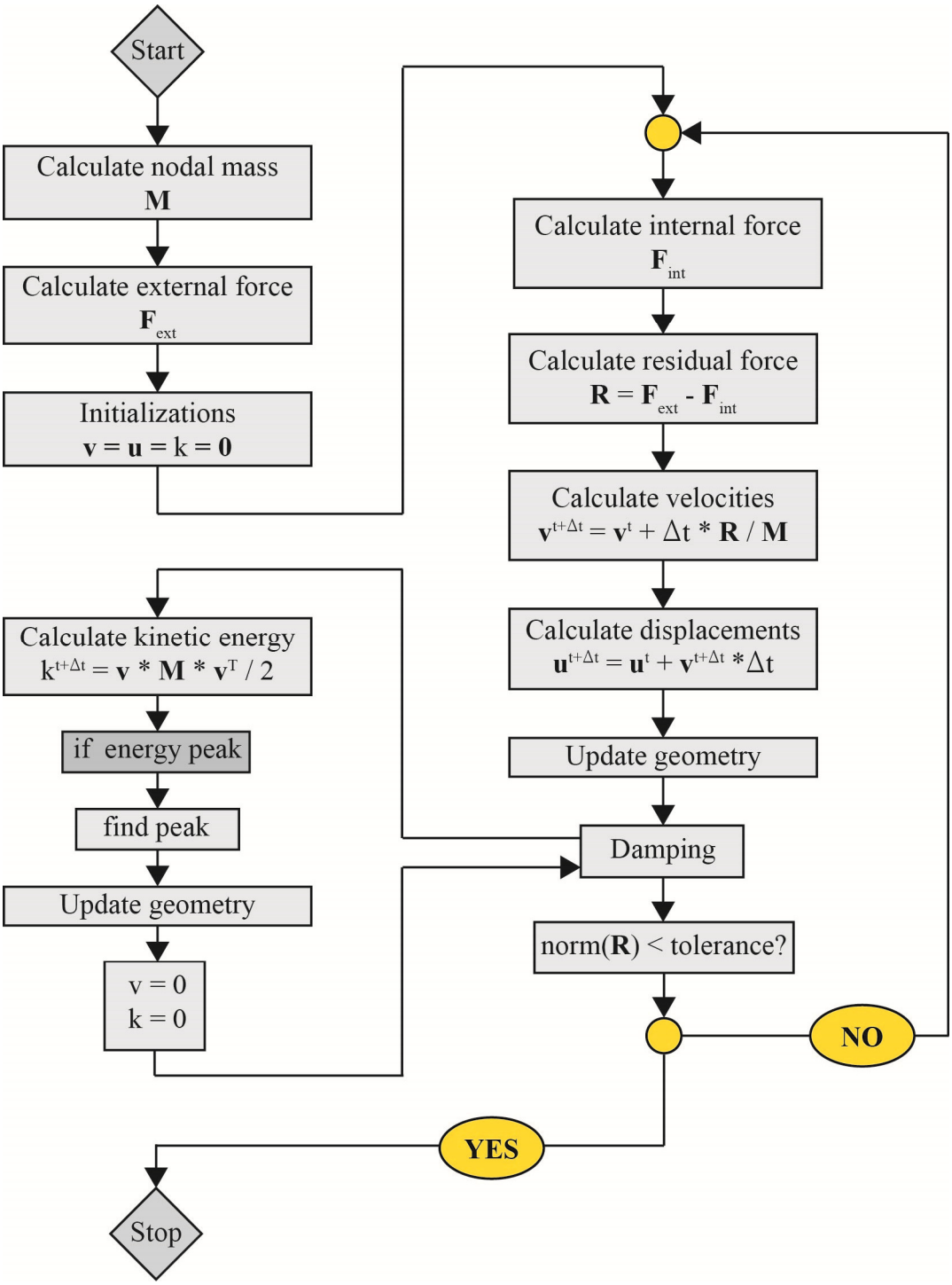


Figure 27: General algorithm for the dynamic relaxation routine with kinetic damping.

3 Implementation

The plug-in program created during the work of the thesis has been carried out using the project wizard RhinoCommon©. RhinoCommon© is a .NET plug-in SDK containing templates for creating plug-ins using Visual Studio© for Rhinoceros©. Programming in Visual Studio© offers a variety of coding languages, however for this thesis the object oriented programming language C# has been chosen. To perform the numerical calculations the Math.NET Numerics library has also been used.

By using the .NET SDK provided by RhinoCommon© a large library of commands and classes for the geometric objects is offered. As mentioned before the geometric objects in Rhinoceros© are built up as NURBS which is convenient as the knot vectors and control points are then given by Rhinoceros© directly. The geometric data can then be used in the classes for calculations and analysis created during this work. The output, i.e. the form found geometry can then be exported back to Rhinoceros©.

To create the computational classes in C# existing codes in MATLAB was translated and completed with some new routines.

3.1 Structure of the software

The program consists of two interacting *projects* called IGA and SampleCsWpfPanel. The structure of the two projects are shown in Table 1 and Table 2.

IGA (Table 1) contains three *classes* with *methods* for calculations; FEMFunctions, IGAFunctions and IGAMembraneFunctions, where the latter class has been the main focus for this project in this thesis apart from some of the methods in IGAFunctions regarding the calculation of membranes. This project was developed before the plug in was created to make sure that the functions worked properly and is therefore reusable in other projects. The accuracy of the results was compared to results obtained by a similar program written in MATLAB.

SampleCsWpfPanel (Table 2) was the second main focus of the thesis which consists of two central parts; SampleCsWpfPanelUserControl and SampleCsWpfViewModel. The UserControl consists of both the GUI (Graphic User Interface), which is basically the design of the interface to Rhino, and the interaction logic to the GUI which handles the *events* performed by the user at runtime.

The GUI is developed using WPF (Windows Presentation Foundation) which is a subset of .NET Framework types in VisualStudio©. The interface contains different *controls*, for example buttons, that has events handled by the interaction logic. The controls for display have *bindings* to objects in the ViewModel class.

The interaction logic of UserControl is called at start up where the program is initialized and the ViewModel class is instantiated. The *event handlers* creates objects of the classes (FormFindingPatch for example) and calls the methods in ViewModel at runtime.

ViewModel contains the surface and point classes and methods that handles the *lists* of them. When creating an *instance* of the FormFindingPatch or BoundaryPoint, the *constructor* demands an argument of a reference to a Rhino© object. The parameter is controlled in the UserControl before creating the instance to avoid errors.

When the button Calculate (see Figure 31) is pressed at runtime by the user the *void* threadTest is called from the event handler of the button. The Rhino© object reference connected to the FormFindingPatch is first converted to a NURBS geometry and then sent as input to the method getIGAGeometry where the control points and knot vectors are obtained through properties in the Rhino© object. The calculations can then be carried out and the geometry is written back to Rhino© surfaces. The method threadTest, containing the dynamic relaxation routine together with some of the other classes can be found in Appendix AII.

Table 1: Classes and methods of the program IGA.

IGA	
FEMFunctions	
GetGaussPoints Assem Solve	Examples of methods in the class
IGAFunctions	
- SurfacePatch	
- IGASurfacePatch : SurfacePatch	
- IGAElement	
- IGAIntegrationPoint	
- IGAFunctions	
DerBasisFun BuildINCIEN2D BuildDOF BuildEDOF ShapeFunction2DShell BuildBaseVectors BuildDMatrix BuildBCoefficients	Methods used in the calculation methods in IGAMembraneSurfacePatch
IGAMembraneFunctions	
- IGAMembraneSurfacePatch : SurfacePatch	Inherits from SurfacePatch
elements	Property: Array of IGAMembraneElements
INC IEN (ENOD) ID (DOF) LM (EDOF)	Properties: Connectivity arrays calculated in IGAFunctions
extF intF	Internal and external force vectors
ComputeBasisFunctions ComputeSurfaceParameters ComputeExternalForce ComputeInternalForce	Void methods performing calculations for all integrationPoints in all elements in the patch using methods in IGAFunctions
- IGAMembraneElement	
gpXi wXi gpEta wEta	Property: Arrays of gauss points and weights in ξ and η direction from FEMFunctions
integrationPoints	Property: Array of IGAMembraneIntegrationPoints
- IGAMembraneIntegrationPoint	
Basis functions, derivatives and jacobians Surface basis vectors Constitutive matrix and stiffness Coefficents for B-matrix Gauss points and weights Local internal force vector	Properties

Table 2: Classes and methods of the program SampleCsWpfPanel

SampleCsWpfPanel	
SampleCsWpfPanelUserControl	
- SampleCsWpfPanelUserControl	GUI (Graphical User Interface) Bindings to SampleCsWpfViewModel
- SampleCsWpfPanelUserControl	Interaction logic for GUI
ShowSelectedSurfaceCommand ShowSelectedPointCommand ShowSelectedPatchesCommand RemoveSurfaceCommand DeletePointCommand AddSurfaceCommand AddPointCommand	Commands connected to GUI
surfaceLoadCmb_SelectionChanged patchLst_SelectionChanged pickedPatchLst_SelectionChanged calculateBtn_Click ComparePoints	Event-handlers connected to GUI Starts calculation, calls threadTest()
threadTest getIGAGeometry	Contains the dynamic relaxation routine Called from threadTest Input: NURBS-surface (Rhino object reference) Output: knot vectors, degrees, control points
SampleCsWpfViewModel	
- SampleCsWpfViewModel	Class
surfaceList boundaryPointList patchList	Lists of FormFindingPatch and BoundaryPoint instances. Binding to GUI
selectedSurface selectedPoint selectedPatch	Objects of FormFindingPatch or BoundaryPoint. Binding to GUI
AddSurface AddPoint SelectPatch UnselectPatch RemoveSurface DeletePoint	Void methods with FormFindingPatch or BoundaryPoint as argument
- FormFindingPatch (Rhino object reference)	Class (Constructor argument)
E (Elastic modulus) nu (Poisson's ratio) t (thickness) xLoad yLoad zLoad	Double properties
- BoundaryPoint (Rhino object reference)	Class (Constructor argument)
xLocked yLocked zLocked	Boolean properties

3.2 Solution algorithm

The following algorithm is implemented in the class SampleCsWpfPanelUserControl for the form finding. The algorithm is meant to give an overview of the program together with the tables in the preceding section. The code for the algorithm can be found in Appendix AII.

1. Global input is read from SampleCsWpfViewModel
2. Lists are allocated for the patches
3. For each FormFindingPatch:
   a) Convert Rhino object to NURBS surface
   b) Create object of IGAMembraneSurfacePatch
   c) Add patch to list
4. Create array of global control points
5. Create global DOF array
6. Allocate vectors for global internal and external forces
7. For each IGAMembraneSurfacePatch:
   a) Build DOF and EDOF matrices from global numbering
   b) Get surface properties and load from FormFindingPatch-properties
   c) Calculate basis functions using IGAMembraneSurfacePatch - for each integration point in each element:
      i. Using IGAFunctions, calculate:
         - Basis functions
         - Derivatives of the basis functions with respect to parametric coordinates
         - The Jacobian for the mapping from parent element to parameter space
      ii. Store the results in the integration point
   d) Calculate surface parameters using IGAMembraneSurfacePatch – for each integration point in each element:
      i. Build base vectors using IGAFunctions
      ii. Calculate the Jacobian for the mapping from parametric domain to physical element
      iii. Construct D-matrix using IGAFunctions
      iv. Build coefficients for the B-matrix using IGAFunctions
      v. Store the results in the integration point
   e) Calculate external force using IGAMembraneSurfacePatch – for each integration point in each element:
      i. Calculate internal force
      ii. Assemble into global force
8. Get boundary conditions by comparing the control points to the BoundaryPointList
9. Create mass matrix
10. Set initial forces, energies and displacements to zero
11. Set an initial residual norm

12. While the solution has not converged:
    a) Set the internal forces to zero
    b) Compute the internal force using IGAMembraneSurfacePatch – for each element:
        i. Extract the element displacements
            - For each integration point, determine:
                - Derivatives of displacements
                - Strains
                - B-matrix
                - Internal force for current Gauss point
                - Add contribution of internal force from current Gauss point to element force vector
        ii. Assemble the element forces to the global internal force vector
    c) Compute the out of balance force vector
    d) Set the constrained degrees of freedom in the out of balance vector to zero
    e) Calculate the residual norm of the out of balance force vector
    f) Update the velocities
    g) Update displacements
    h) For all IGAMembraneSurfacePatches:
        i. Write the solution to Rhino geometry
        ii. Update the surface to Rhino
    i) Kinetic damping:
        i. Set the previous kinetic energy
        ii. Compute the current kinetic energy
        iii. If a peak in the kinetic energy is found:
            - Find the time location of the peak
            - Compute the displacement of the peak
            - Set the velocities to zero
    j) If the norm of the out of balance force vector is smaller than a certain tolerance, the analysis is done, if not the while loop continues.

3.3 User's manual

1. Initializations
    a) Open the user interface.
    b) Draw the surface or surfaces for form finding in Rhinoceros©. Rebuild the surfaces to raise the degree of the surfaces to at least 2.
    c) Draw points to represent supports. The points have to coincide with the control points of the surfaces generated by Rhino©.

2. Geometry (Figure 28)
    a) Add the surfaces to the list by selecting them and press the button Add surface.
    b) Erase or show the surface in Rhinoceros© by highlighting the surface in the list and press the buttons on the right hand side of the user interface.
    c) The material parameters are typed in for the highlighted surface.

3. Boundary conditions (Figure 29)
    a) Add the boundary conditions by selecting the points and press the button Add point.
    b) Erase or show the point in Rhinoceros© by highlighting the point in the list and press the buttons on the right hand side of the user interface.
    c) When a point in the list is selected, the user can decide in which directions the boundary is restrained.

4. Load (Figure 30)
    a) Select a surface in the list box
    b) For the selected surface, type in the load in each direction.

5. Dynamic relaxation (Figure 31 and Figure 32)
    a) The list on the right represent the list of patches added to the analysis
    b) To add a surface, select the surface in the list on the left
    c) To erase a surface from the analysis, select the patch in the list with the added patches
    d) To show the selected patches for the analysis in Rhinoceros©, press the button Show patches
    e) To start the analysis, press the Start button.
    f) The form found surface is added as a new surface to Rhinoceros©, Figure 32.

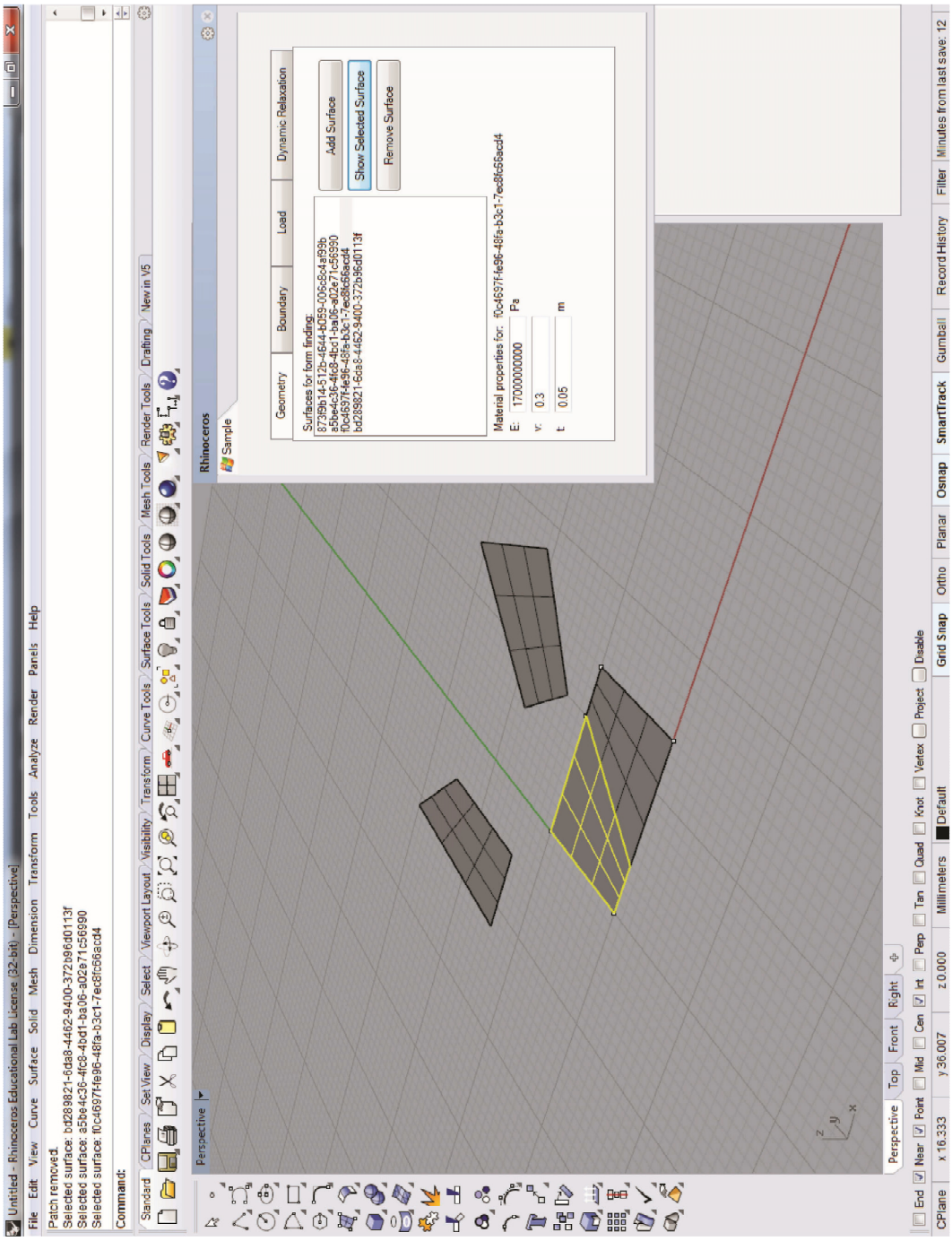


Figure 28: The geometry tab is opened when the GUI is started. One patch is highlighted. The material parameters can be typed in for each patch.

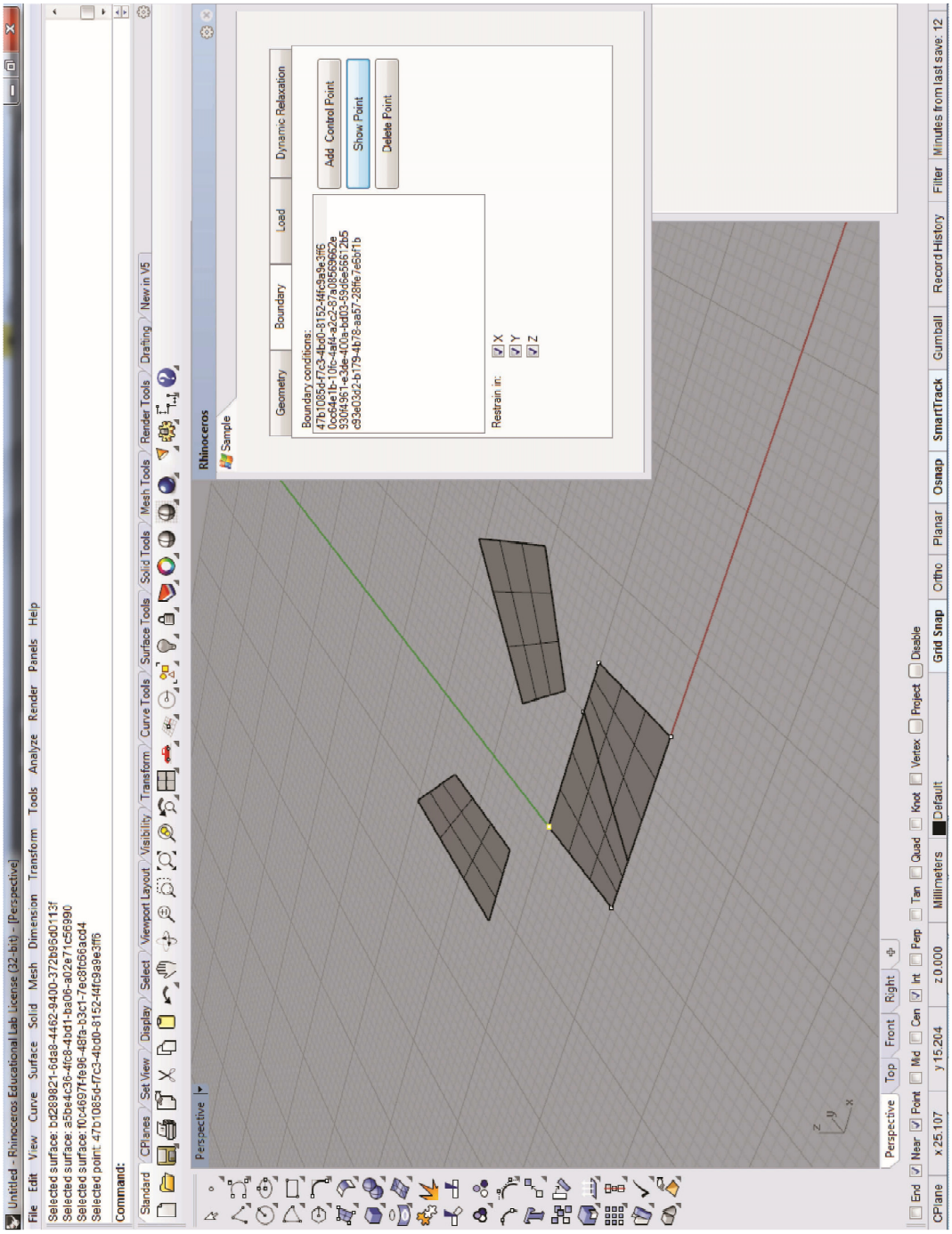


Figure 29: The tab for choosing boundary conditions in the GUI. Once a point is selected, the directions of the restraint can be chosen.

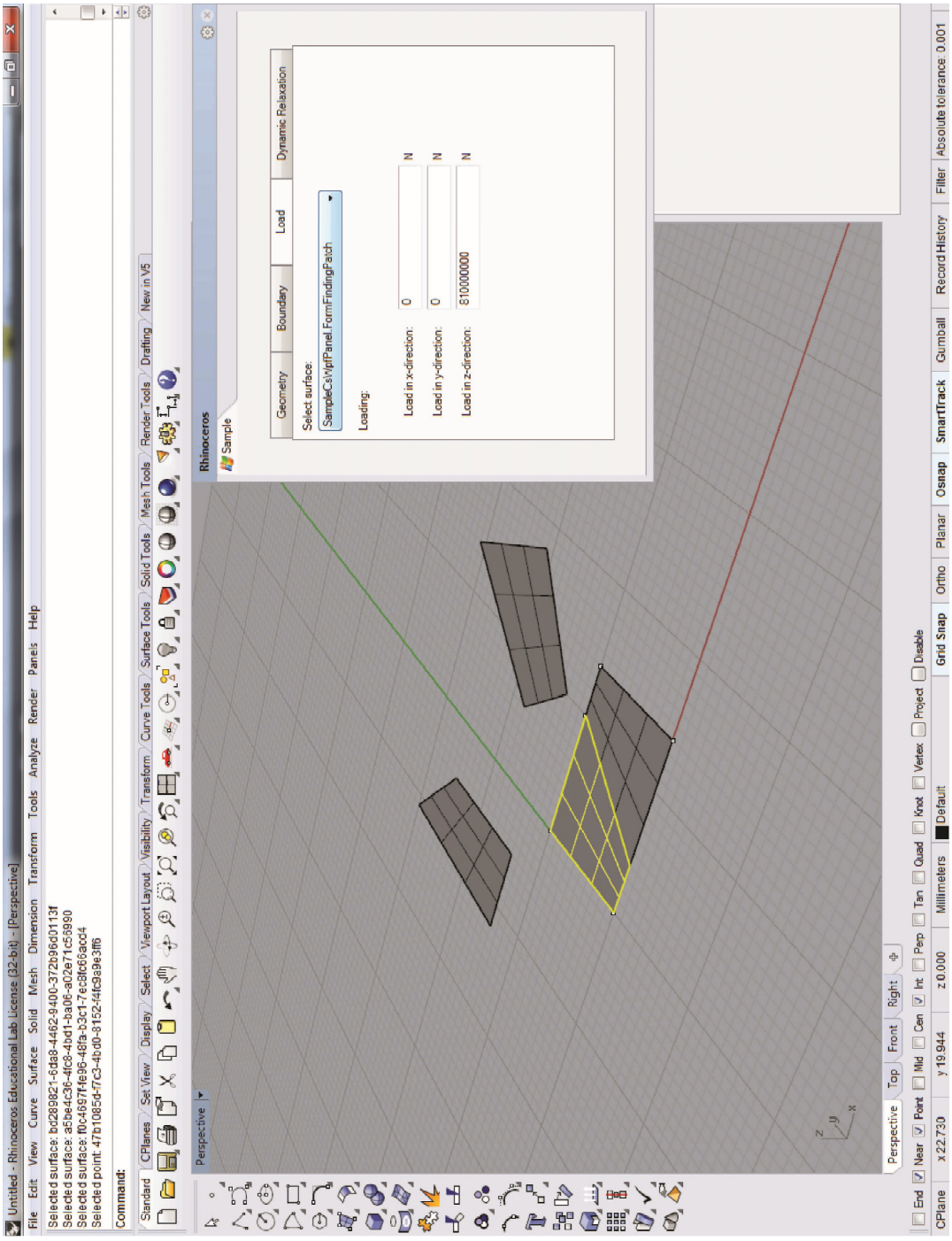


Figure 30: The loading tab in the GUI. When a patch is selected, the load can be typed in for each direction.

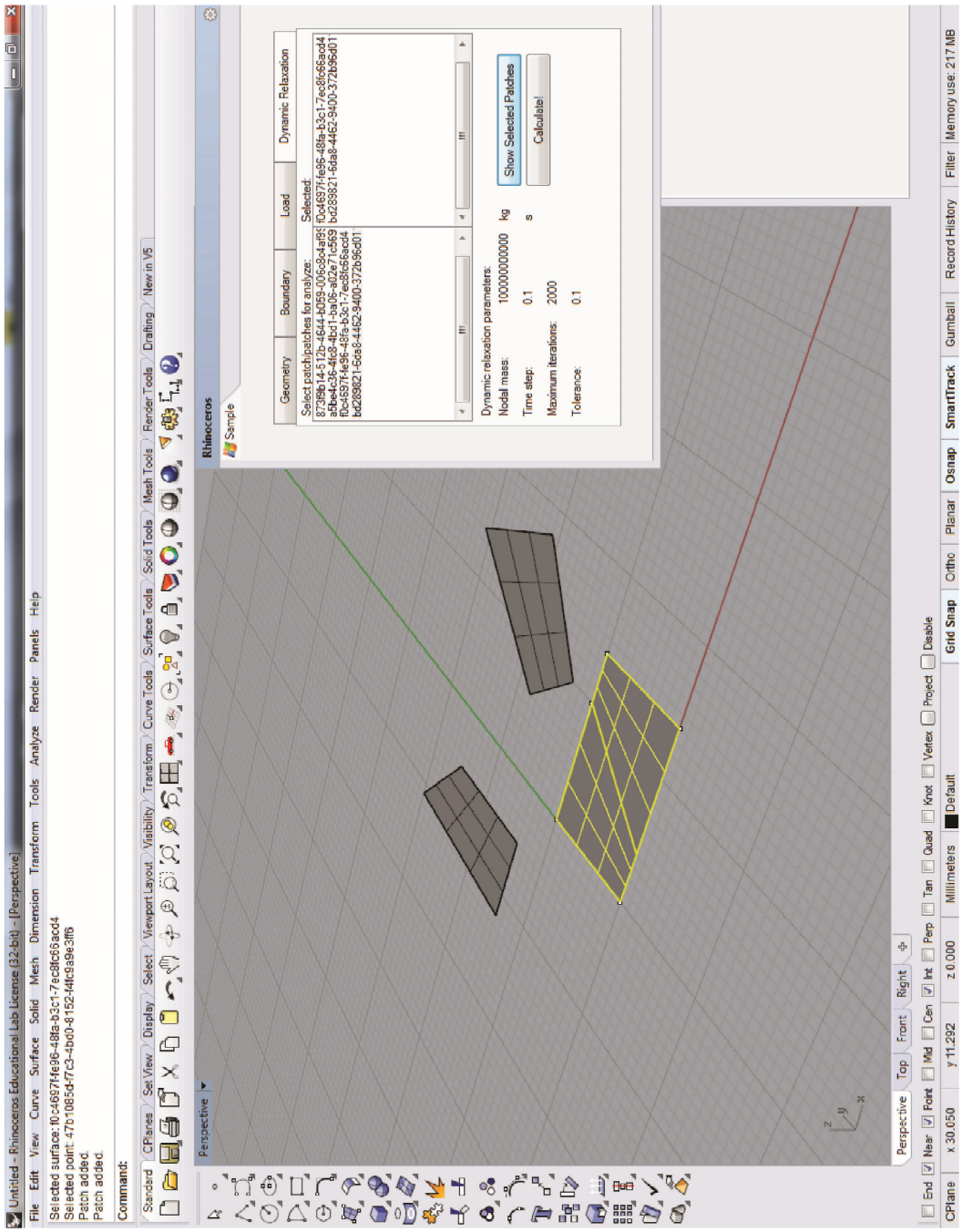


Figure 31: Dynamic relaxation tab of the GUI. The patches for form finding are selected.

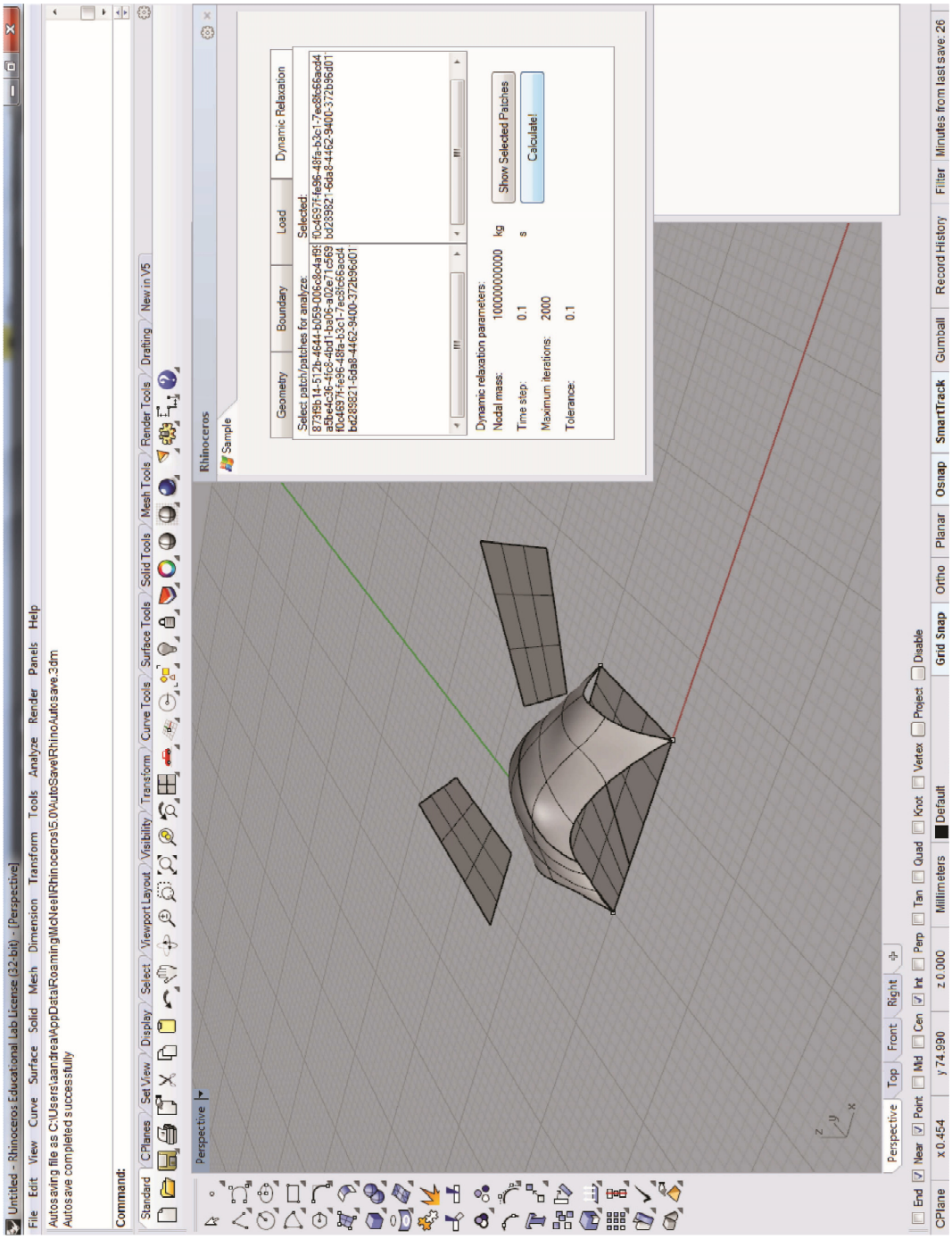


Figure 32: Dynamic relaxation tab after relaxation of the two patches.

3.4 Boundary conditions

One issue that arises when working with form finding techniques such as dynamic relaxation is dealing with the horizontal forces that occur along the edges of the model. The structures obtained through a form finding process often end up with the similar attribute of arches along the unrestrained sides to account for the horizontal forces. This behavior causes a huge limitation to the range of design situations where form finding can be used. By finding a way of keeping a desired geometry while the boundary conditions are still satisfied this design method could be used more generally.

Consider, for example, a square patch with fixed supports in the corners and the edges locked in the z-direction. In the form finding process the sides of the patch will move more and more towards the middle, forming a vault to obtain equilibrium for the horizontal forces along the edges, see Figure 33.

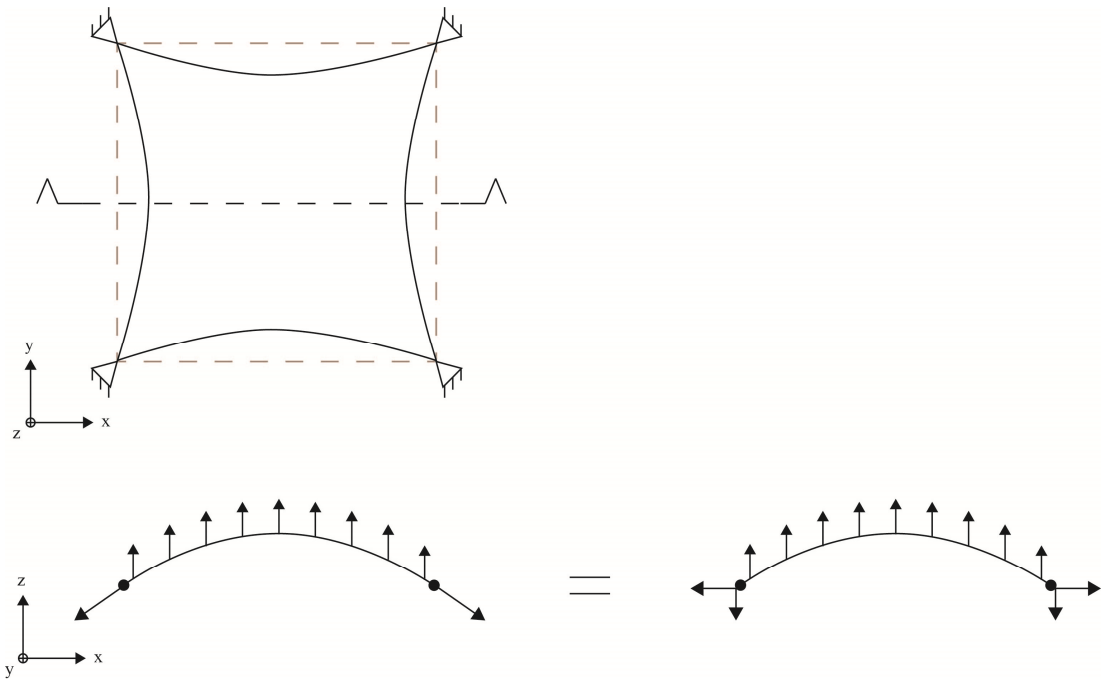


Figure 33: Sketch of a form found square patch with four locked corners in plan (above) and a section through the middle (below).

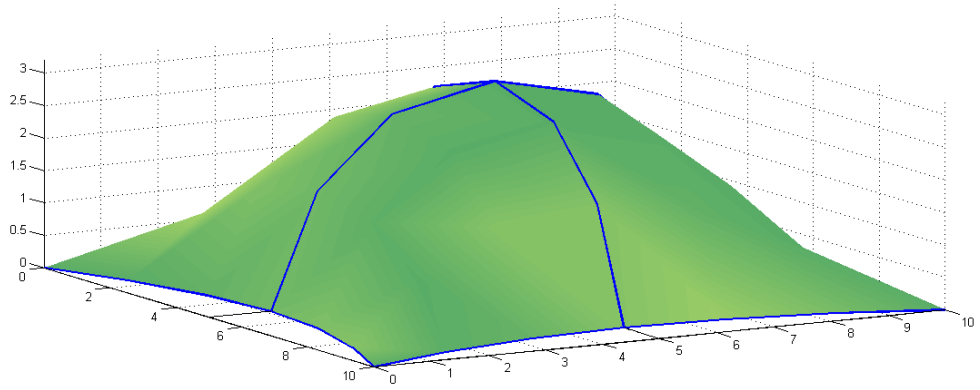


Figure 34: Form found square patch with four locked corners and the sides locked in the y-direction.

The question of how to keep the edges in line with the initial patch, the dotted square in Figure 33, was raised.

An idea to counteract for these forces that occur was to add patches along the sides. The hope was that in the relaxation of the model these patches would pull the edges of the square patch towards the desired straight line forming “wings” along the sides, see Figure 35. The horizontal components of the reaction forces should ideally cancel each other out. Unfortunately this effect was not obtained.

The kink that arises along the intersection of two patches has C^0 continuity. This could have consequences on the transferring of the forces over this kink and could be a reason for this undesired result.

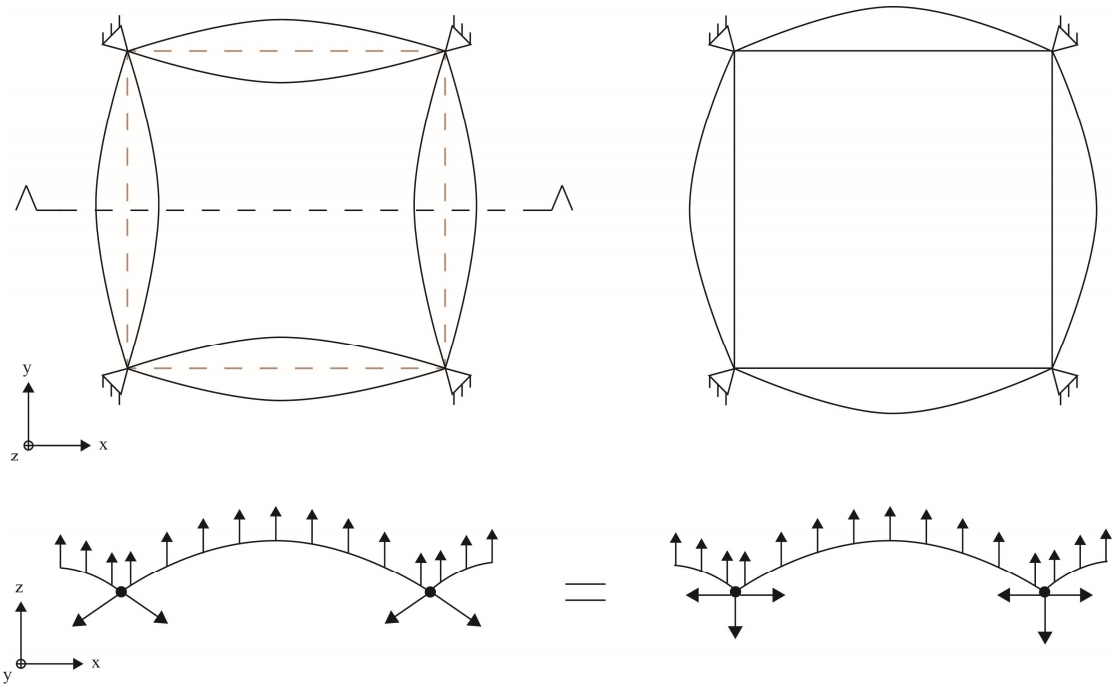


Figure 35: Attempt to keep straight edges by adding four edge patches to pull the edges out creating "wings" on the sides of the square.

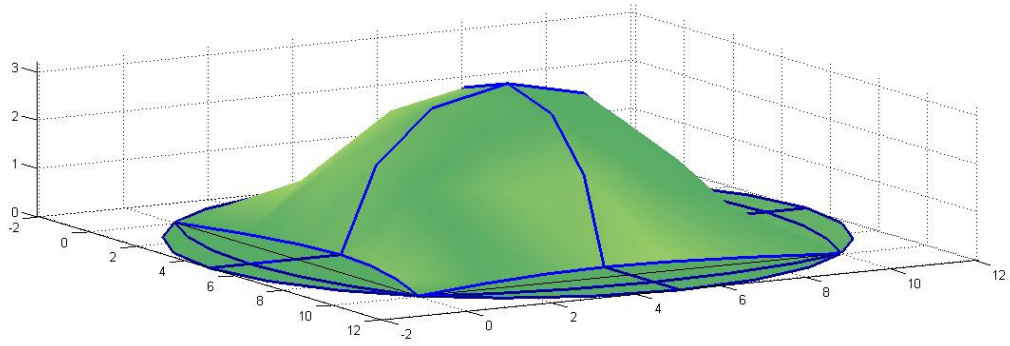


Figure 36: Attempt with four unloaded side patches.

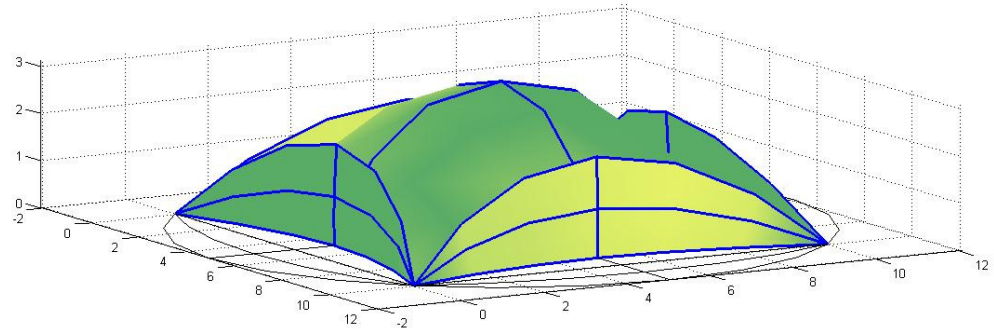


Figure 37: Attempt with four loaded side patches.

To overcome this kink another attempt was made using only one patch with an initial curvature outwards, see Figure 38. During the relaxation these curved edges would move towards the desired straight lines. Performing the dynamic relaxation on a patch using four elements, this effect was obtained, see Figure 39. However when the mesh was refined the form found edges was relaxed into another curvature, see Figure 40.

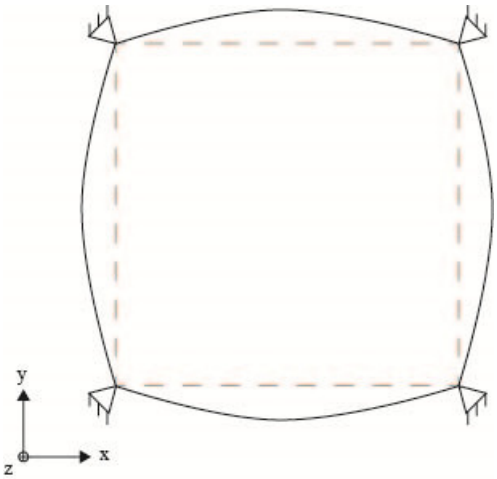


Figure 38: Attempt using one patch with initial curvature of the sides.

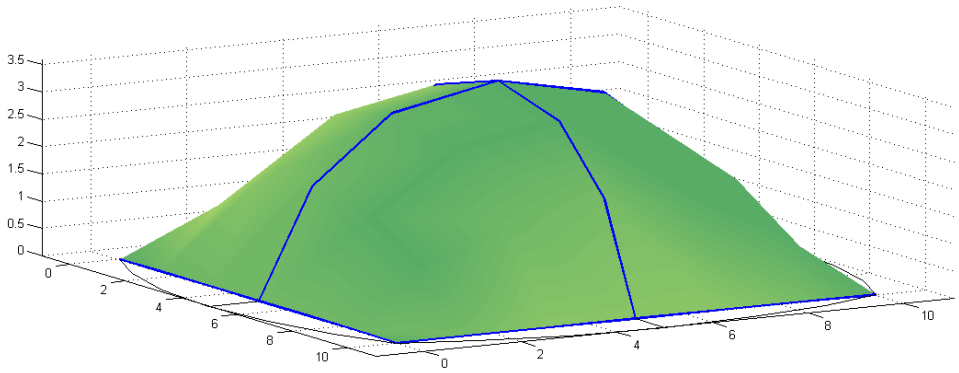


Figure 39: Form found patch consisting of four elements with initial curvature.

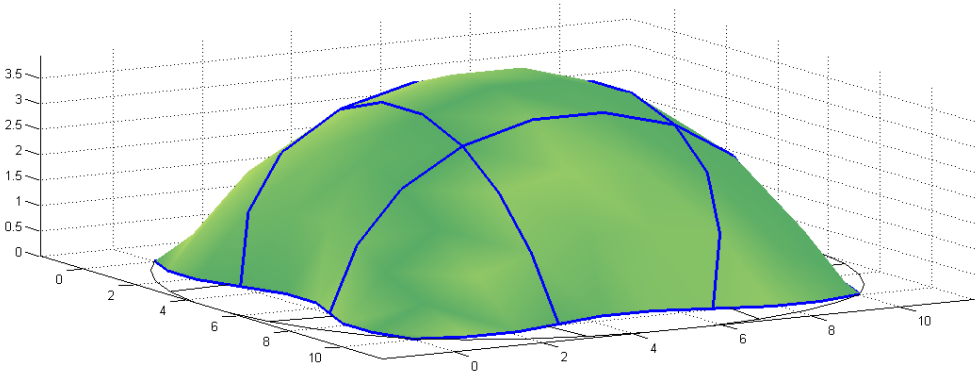


Figure 40: Form found patch consisting of nine elements with initial curvature.

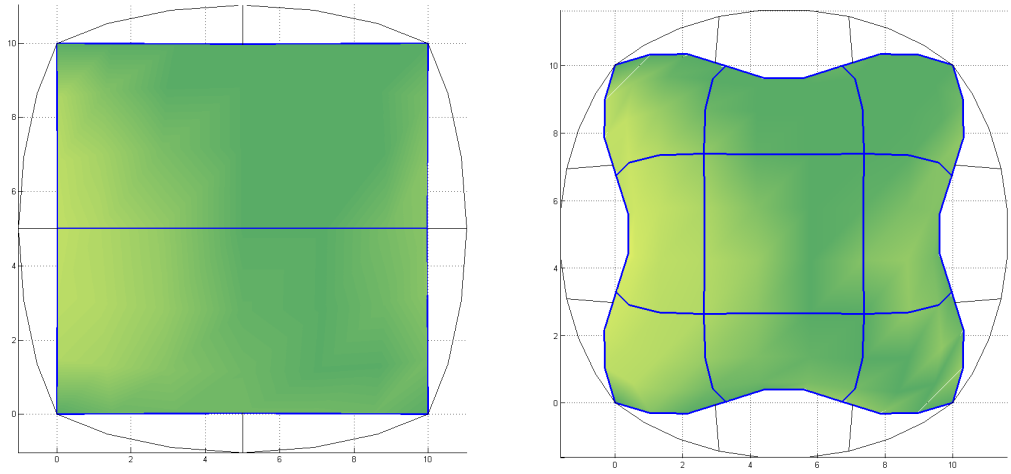


Figure 41: Top view of the form found patches in Figure 39 and Figure 40.

4 British Museum Case Study

The roof over the British Museum Great Court consisting of a triangular grid shell was completed in 2000. The architects of the project were Foster and Partners and the engineers were Buro Happold.

Figure 42: The roof over the Great Court from above, see [5]. A view from inside is seen on Figure 2.

4.1 Context

The British Museum was founded in the mid-18th century to house the collection of the physician Sir Hans Sloan which he in his will bequeathed to King George II. The collection included chiefly natural history specimens from his travels but also books, coins and medals.

Today's quadrangle building, designed by Sir Robert Smirke, was completed in 1852 and is in the Greek revival style. The Great Court was originally meant to be a garden but, due to the need of more space for storing books, the Reading Room and some book stacks were built just a few years later. After that the court became a lost space.

In 1997, as the library department of the Museum was relocated, the architectural competition for the redesign of the courtyard was launched. The main aims of the competition were to reveal hidden spaces, revise old spaces and to create new spaces. The winning entry by Foster and Partners together with Buro Happold was completed in 2000 (History of the British Museum).

The buildings surrounding the courtyard are old and designed for the prevailing conditions at that time and can therefore not restrain any horizontal forces from the new roof. The roof is therefore designed to take only vertical forces along the edges using sliding supports and all horizontal forces must be transferred either to the corners of the square or to the edge of the circular Reading Room in the courtyard where they can be handled. The circular Reading Room is reinforced with columns to transfer the extra weight to the ground and the horizontal forces cancel each other through a compression ring around the room.

The Great Court measures 73 meters from east to west and 97 meters from north to south, see Figure 43. The circular Reading Room is offset 3 meters to the north from the middle and has a diameter of 44 meters (Williams, 2001).

4.2 Form Finding Process

Form finding techniques such as dynamic relaxation demands that the supports are fixed in order to be able to meet the surrounding buildings in straight lines around the edges. Also, the shell needed to be able to take both compression and tension. The techniques of form finding origins from the physical hanging models hence the resulting model works in either one or another.

Since a form finding technique couldn't be used the shape of the roof was determined geometrically by Chris Williams. The structural grid was then found through dynamic relaxation on the given surface where the nodes of the grid lie on the surface.

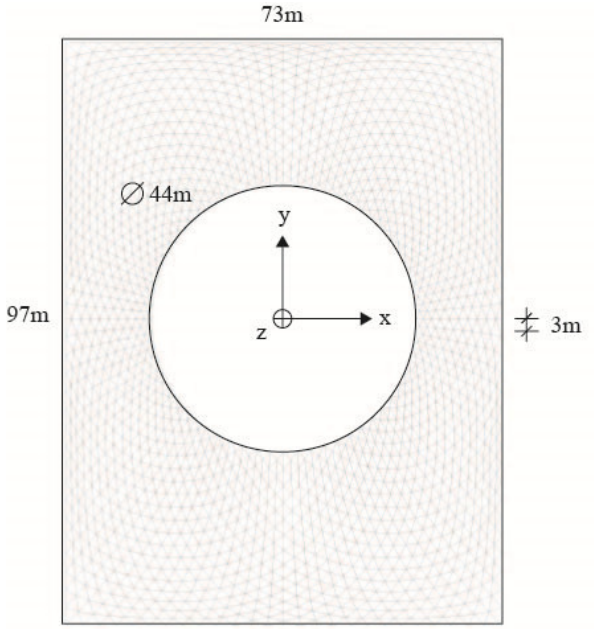


Figure 43: The boundary of the roof.

The height of the roof, z, is the sum of three different functions of x and y which are the directions in east and north. The centre of the reading room is the origin, see Figure 43. The first function z_1 describes the change in the height between the boundaries from the Reading Room and the surrounding buildings, i.e. the rectangular court, see Figure AIII: 1 in Appendix AIII. The second and third functions describe the curvature of the roof and are both zero along the edges, see Figure AIII: 2 and Figure AIII: 3 in Appendix AIII. The function describing the roof shape is presented in Appendix AIII. Figure 44 shows the final surface generated by the sum of the three functions.

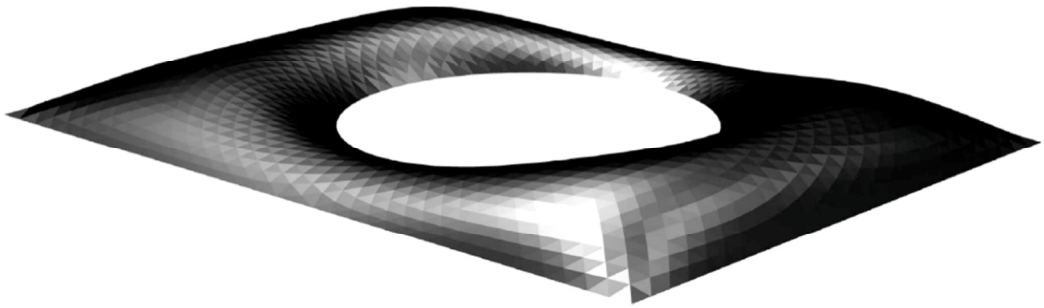


Figure 44: The final surface comprised of the three functions. [4]

Once the shape of the roof was finished the structural grid could be determined. The node points along the boundaries were equally spaced around the Reading room and the rectangle and joint together. These radial lines were then divided into segments of equal length and by joining the dots a grid was obtained, see Figure 45. The discontinuities in the grid were eliminated using dynamic relaxation where a fictitious force was applied to the nodes. Allowing the nodes to slide on the given surface a relaxed grid was obtained; see Figure 46 (Williams, 2001).

4.3 Relation to form finding

By describing a shell mathematically the possibility to try out different options is decreasing due to the time effort for setting up the analytical expressions. This means that the final form might not be the most optimal. The form finding of the grid layout will on the other hand generate the most optimal grid for the chosen form.

Form finding technique such as dynamic relaxation are relatively easy and can generate suggestions of a final structure quickly. This makes the process of generating a form much more efficient and several different shapes can be evaluated during the conceptual phase. However, the limitations in the technique discussed in Section 3.4 makes it complicated to satisfy the demands of the surrounding structures. To keep the edges straight the boundary along the whole rectangle needs to be fixed which in this case is not possible due to the surrounding old structures.

The dome of the Reading Room is an old landmark and the sightlines to it could therefore not be disturbed. Hence, the shell had a restriction of the height which further complicates the use of form finding techniques. The more shallow vault the more the roof thrusts outwards generating larger horizontal forces to take care of.

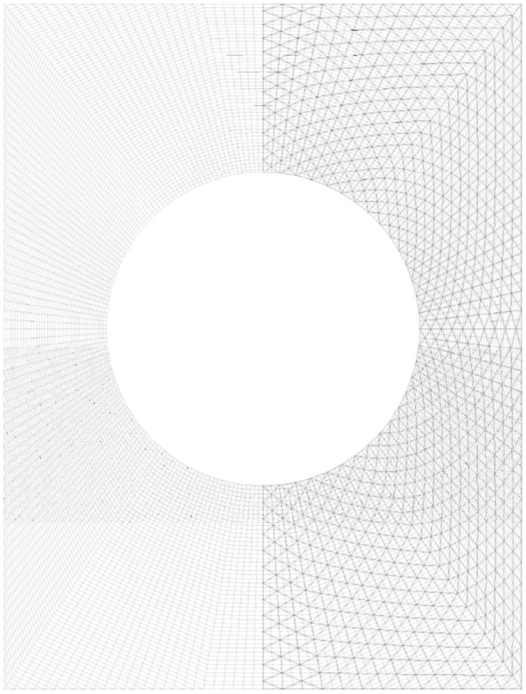


Figure 45: Starting grid [4].

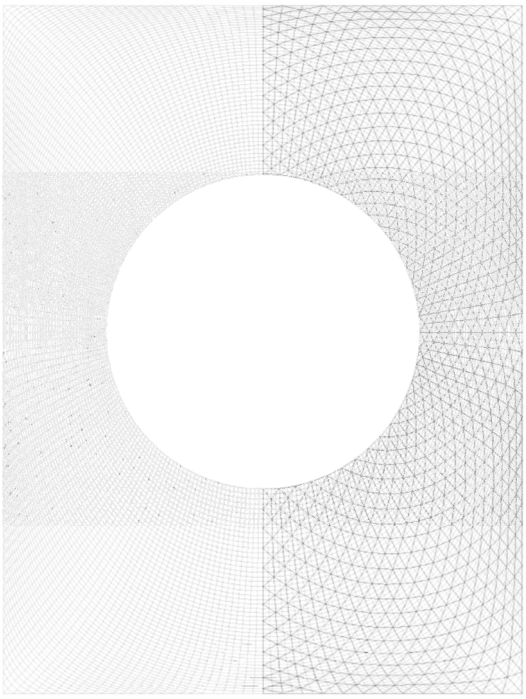


Figure 46: Relaxed grid [4].

Figure 47: Elevation of the final grid looking west [4].

5 Discussion

5.1 The conceptual design tool

During this thesis a conceptual design tool for membranes has been developed. The tool can be used to get a quick estimation of the shape of a structure. Other existing form finding tools are constructed as components to the plug in program Grasshopper© which, for a beginner to Rhino©, can be complicated to use. The fact that this program is a plug in to Rhino© directly makes it more simple to use, which means that the tool can be used by a broader community and by new users of Rhino©. There are still several improvements that needs to be done. To make the program more user-friendly some developments of the user interface needs to be done especially regarding how the boundary points are chosen and what types of boundary conditions that can be applied.

To speed up the convergence of the method the implementation of the mass matrix needs to be adjusted. The one that is currently used is a unity matrix multiplied with a scalar typed in by the user. By implementing a routine that adjusts the nodal masses to be proportional to the stiffness components the convergence would be improved (Barnes, 1988).

5.2 Isogeometric analysis

There are several advantages of using isogeometric finite elements for analysis. First and foremost is the fact that it uses the correct geometry which will generate more accurate approximation of the solution. When Lagrangian finite elements are used, the demand of the mesh size increases which has a negative effect on the computational effort of a program. Also, the refinement of a FEA mesh requires that the exact geometry is provided, i.e. that the CAD model communicates with the model for the analysis, which it's not the case.

Shell structures are particularly sensitive to imperfections with regard to buckling, which means that the approximation of the shell has to be very exact. This is also a reason to implement isogeometric analysis where the exact geometry is used.

During the work of the thesis isogeometric analysis has been compared to the classic finite element analysis regarding the theory of the two methods. A further comparison between the two could have included the design of a shell using both methods. This would give a deeper understanding of the differences in results and computational effort of the implementation of the two methods.

5.3 The roof of the British Museum Great Court

The case study of the roof of the British Museum was carried out to apply the method to a real problem. The specific context of the roof where the surrounding buildings are unable to handle the horizontal forces narrows the possible choices of form finding methods down. The surface of the roof is mathematically determined and the grid is subsequently determined using dynamic relaxation on the surface. Finding a form mathematically is a time consuming process and by finding a way to deal with the arching free edges during a form finding process, dynamic relaxation would have been a possible choice for the surface instead. This problem is discussed in Section 3.4 and the issue needs further investigation. Perhaps by making the side patches in Figure 35 stiffer than the mid-patch the edges could be kept straight. Also a generic algorithm to find the starting curvature of the edges in Figure 38 that would generate straight edges in the form found patch is a possible way of dealing with the problem. By using isogeometric analysis for the form finding of the roof a more accurate model would offer less geometrical imperfections generated during the analysis.

5.4 Connecting architects and engineers

The use of computers is close to necessary today. There are several programs for design and analysis that work differently and are suited for different things. By the use of a common tool for design and analysis less time is wasted on translating geometry for calculations. Perhaps a wider understanding and more creative work collaboration of the two disciplines can be achieved if a common platform is used.

The design process is an iterative process especially when working with more complex geometries like shells. To find the optimal form, the architectural design and structural analysis will undergo several stages. By compromising the work at the conceptual stage into the same computational toolbox this update is easier and faster.

6 Recommendations for further work

There are several interesting subjects of future work with isogeometric analysis. Other subjects that has been discussed during the work has been how to experiment with the thickness of the membrane to be able to generate a wider range of shapes. Another development could be to allow for a limited amount of bending stresses in the membranes.

The plug in created in this work has a great potential but needs more development, as discussed in Section 5.1. Besides working on the existing parts of the program further implementations dealing with other types of elements, for example beams, could be done.

The discussion regarding the boundary conditions in Section 3.4 is an issue that would be very interesting to continue working with. Studies on a generic algorithm to find an initial shape of the boundary edges that would generate a straight edge was started. However, due to the lack of time, no results are yet available.

7 References

Adriaenssens, S., Block, P., Veenendaal, D., Williams, C. (2014) *Shell Structures for Architecture: form finding and optimization.* Abingdon, Oxon, United Kingdom: Routledge.

Austrell, P-E., Dahlblom, O., Lindeman, J., Olsson, A., Olsson, K-G., Persson, K., Petersson, H., Ristinmaa, M., Sandberg, G., Wernberg, P-A. (2004) *CALFEM A Finite Element Toolbox*. Lund University, Sweden.

Barnes, M. R. (1988) Form-finding and analysis of prestressed nets and membranes. *Computers & Structures*, vol. 30, no. 3, pp 685-695.

Cottrell, J. A., Hughes, T. J. R., Bazilevs, Y. (2009) *Isogeometric Analysis: Toward Integration of CAD and FEA.* Chichester, West Sussex, United Kingdom: John Wiley & Sons, Ltd.

Ottosen, N., Petersson, H. (1992) *Introduction to the finite element method.* Harlow, Essex, England: Pearson Education Limited.

Thomée, V. (1999) From finite differences to finite elements; A short history of numerical analysis of partial differential equations. *Journal of computational and applied mechanics*. 128, pp.1-54.

Williams, C. J. K. (2001) The analytic and numerical definition of the geometry of the British Museum Great Court roof. In Burry, M., Datta, S., Dawson, A., Rollo, A. J., eds. *Mathematics & Design 2001*. Geelong, Victoria, Australia: Deakin University, pp. 434-440.

History of the British Museum. *The British Museum*. http://www.britishmuseum.org (2015-05-19).

Pictures

[1] http://www.evolo.us/architecture/shell-and-shadow-for-nordpark-railway-stations-in-innsbruck-austria-zaha-hadid-architects/ (2015-08-03).
[2] http://www.britishmuseum.org (2015-08-03).
[3] http://blog.buildllc.com/2009/04/heinz-isler-a-few-important-things/ (2015-08-03).
[4] Williams, C. J. K. (2001) The analytic and numerical definition of the geometry of the British Museum Great Court Roof. In Burry, M., Datta, S., Dawson, A., Rollo, A. J., eds. *Mathematics & Design 2001*. Geelong, Victoria, Australia: Deakin University, pp. 434-440.
[5] http://www.e-architect.co.uk/london/british-museum-building (2015-08-11).

AI: Calculation of B-Spline basis functions

To calculate the B-spline basis functions for a knot vector equations (2.3) and (2.4) are repeated recursively on the knot vector. The calculations are carried out for the knot vector in (2.1) i.e.

$$\Xi=[0,0,0,1,2,3,4,4,5,5,5]$$

For p=0 the results are the step functions

$$N_{1,0}=N_{2,0}=0$$

$$N_{3,0}=\begin{cases}1 & \text{if } 0\le\xi<1\\ 0 & \text{otherwise}\end{cases}$$

$$N_{4,0}=\begin{cases}1 & \text{if } 1\le\xi<2\\ 0 & \text{otherwise}\end{cases}$$

$$N_{5,0}=\begin{cases}1 & \text{if } 2\le\xi<3\\ 0 & \text{otherwise}\end{cases}$$

$$N_{6,0}=\begin{cases}1 & \text{if } 3\le\xi<4\\ 0 & \text{otherwise}\end{cases}$$

$$N_{7,0}=0$$

$$N_{8,0}=\begin{cases}1 & \text{if } 4\le\xi<5\\ 0 & \text{otherwise}\end{cases}$$

$$N_{9,0}=N_{10,0}=0$$

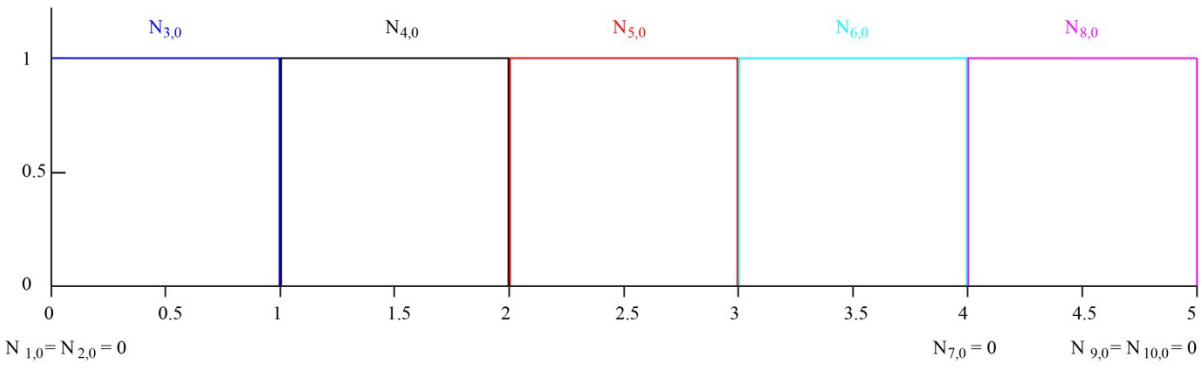


Figure AI: 1 The basis functions for p=0 are step functions.

The next step is for $p = 1$

$$N_{1,1} = \frac{\xi - 0}{0 - 0} \cdot N_{1,0} + \frac{0 - \xi}{0 - 0} \cdot N_{2,0} = 0$$

$$N_{2,1} = \frac{\xi - 0}{0 - 0} \cdot N_{2,0} + \frac{1 - \xi}{1 - 0} \cdot N_{3,0} = (1 - \xi) \cdot N_{3,0} = \begin{cases} 1 - \xi & \text{if } 0 \le \xi < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{3,1} = \frac{\xi - 0}{1 - 0} \cdot N_{3,0} + \frac{2 - \xi}{2 - 1} \cdot N_{4,0} = \xi \cdot N_{3,0} + (2 - \xi) \cdot N_{4,0} = \begin{cases} \xi & \text{if } 0 \le \xi < 1 \\ 2 - \xi & \text{if } 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{4,1} = \frac{\xi - 1}{2 - 1} \cdot N_{4,0} + \frac{3 - \xi}{3 - 2} \cdot N_{5,0} = (\xi - 1) \cdot N_{4,0} + (3 - \xi) \cdot N_{5,0} = \begin{cases} \xi - 1 & \text{if } 1 \le \xi < 2 \\ 3 - \xi & \text{if } 2 \le \xi < 3 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{5,1} = \frac{\xi - 2}{3 - 2} \cdot N_{5,0} + \frac{4 - \xi}{4 - 3} \cdot N_{6,0} = (\xi - 2) \cdot N_{5,0} + (4 - \xi) \cdot N_{6,0} = \begin{cases} \xi - 2 & \text{if } 2 \le \xi < 3 \\ 4 - \xi & \text{if } 3 \le \xi < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{6,1} = \frac{\xi - 3}{4 - 3} \cdot N_{6,0} + \frac{4 - \xi}{4 - 4} \cdot N_{7,0} = (\xi - 3) \cdot N_{6,0} = \begin{cases} \xi - 3 & \text{if } 3 \le \xi < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{7,1} = \frac{\xi - 4}{4 - 4} \cdot N_{7,0} + \frac{5 - \xi}{5 - 4} \cdot N_{8,0} = (5 - \xi) \cdot N_{8,0} = \begin{cases} 5 - \xi & \text{if } 4 \le \xi < 5 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{8,1} = \frac{\xi - 4}{5 - 4} \cdot N_{8,0} + \frac{5 - \xi}{5 - 5} \cdot N_{9,0} = (\xi - 4) \cdot N_{8,0} = \begin{cases} \xi - 4 & \text{if } 4 \le \xi < 5 \\ 0 & \text{otherwise} \end{cases}$$

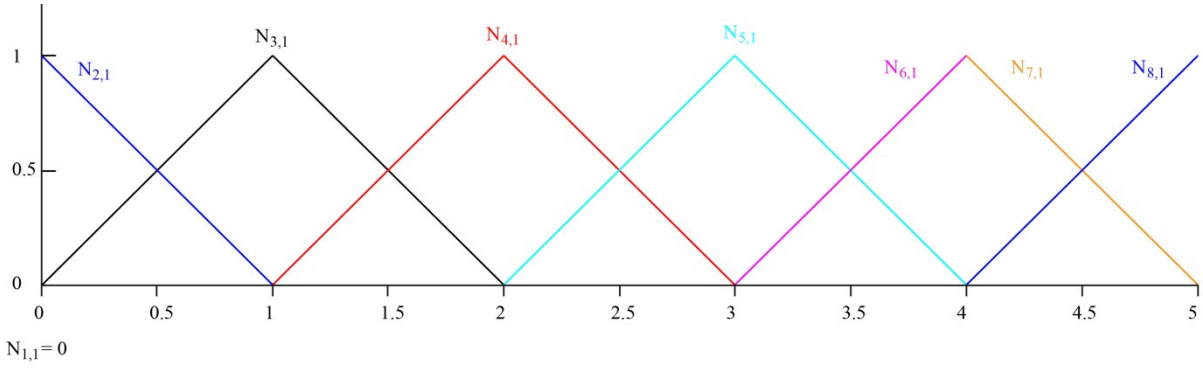


Figure AI: 2 The basis functions for p=1 are linear functions.

For p=2:

$$N_{1,2} = \frac{\xi - 0}{0 - 0} \cdot N_{1,1} + \frac{1 - \xi}{1 - 0} \cdot N_{2,1} = (1 - \xi) \cdot N_{2,1} = \begin{cases} (1-\xi)^2 & \text{if } 0 \le \xi < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{2,2} = \frac{\xi - 0}{1 - 0} \cdot N_{2,1} + \frac{2 - \xi}{2 - 0} \cdot N_{3,1} = \xi \cdot N_{2,1} + \frac{2 - \xi}{2} \cdot N_{3,1} =$$

$$= \begin{cases} \xi(1-\xi) + \dfrac{\xi(2-\xi)}{2} & \text{if } 0 \le \xi < 1 \\ \dfrac{(2-\xi)^2}{2} & \text{if } 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{3,2} = \frac{\xi - 0}{2 - 0} \cdot N_{3,1} + \frac{3 - \xi}{3 - 1} \cdot N_{4,1} = \frac{\xi}{2} \cdot N_{3,1} + \frac{3 - \xi}{2} \cdot N_{4,1} =$$

$$= \begin{cases} \dfrac{\xi^2}{2} & \text{if } 0 \le \xi < 1 \\ \dfrac{\xi(2-\xi)}{2} + \dfrac{(3-\xi)(\xi-1)}{2} & \text{if } 1 \le \xi < 2 \\ \dfrac{(3-\xi)^2}{2} & \text{if } 2 \le \xi < 3 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{4,2} = \frac{\xi - 1}{3 - 1} \cdot N_{4,1} + \frac{4 - \xi}{4 - 2} \cdot N_{5,1} = \frac{\xi - 1}{2} \cdot N_{4,1} + \frac{4 - \xi}{2} \cdot N_{5,1} =$$

$$= \begin{cases} \dfrac{(\xi-1)^2}{2} & \text{if } 1 \le \xi < 2 \\ \dfrac{(\xi-1)\cdot(3-\xi)}{2} + \dfrac{(4-\xi)(\xi-2)}{2} & \text{if } 2 \le \xi < 3 \\ \dfrac{(4-\xi)^2}{2} & \text{if } 3 \le \xi < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{5,2} = \frac{\xi - 2}{4 - 2} \cdot N_{5,1} + \frac{4 - \xi}{4 - 3} \cdot N_{6,1} = \frac{\xi - 2}{2} \cdot N_{5,1} + (4 - \xi) \cdot N_{6,1} =$$

$$= \begin{cases} \dfrac{(\xi-2)^2}{2} & \text{if } 2 \le \xi < 3 \\ (\xi-3)\cdot(4-\xi)+(4-\xi)^2 & \text{if } 3 \le \xi < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{6,2} = \frac{\xi-2}{4-2}\cdot N_{6,1} + \frac{4-\xi}{4-3}\cdot N_{7,1} = \frac{\xi-2}{2}\cdot N_{6,1} + (4-\xi)\cdot N_{7,1} =$$

$$= \begin{cases} \dfrac{(\xi-2)\cdot(\xi-3)}{2} & \text{if } 3 \le \xi < 4 \\ (4-\xi)\cdot(5-\xi) & \text{if } 4 \le \xi < 5 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{7,2} = \frac{\xi-4}{5-4}\cdot N_{7,1} + \frac{5-\xi}{5-4}\cdot N_{8,1} = (\xi-4)\cdot N_{7,1} + (5-\xi)\cdot N_{8,1} =$$

$$= \begin{cases} 2(5-\xi)(\xi-4) & \text{if } 4 \le \xi < 5 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{8,2} = \frac{\xi-4}{5-4}\cdot N_{8,1} = (\xi-4)\cdot N_{8,1} = \begin{cases} (\xi-4)^2 & \text{if } 4 \le \xi < 5 \\ 0 & \text{otherwise} \end{cases}$$

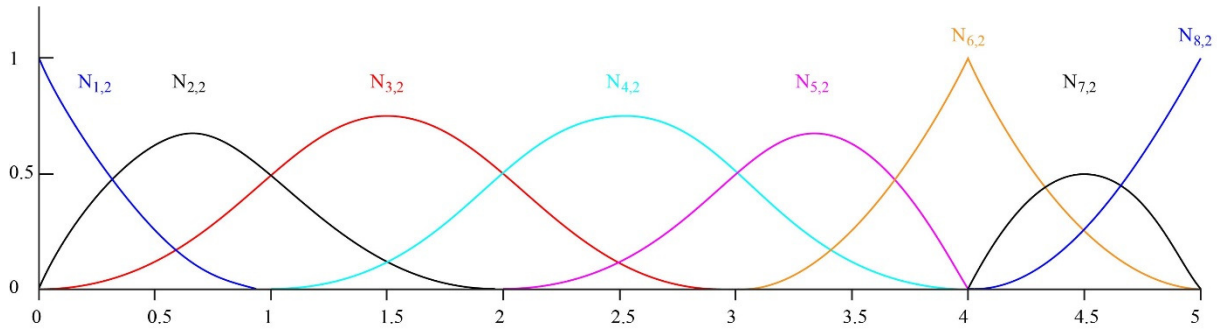


Figure AI: 3 The basis functions for p=2 are quadratic functions. The continuity over the elements are p-1=1 except for the double knot where it's C^0.

AII: Coding example – threadTest()

```
static void threadTest()
{
// DYNAMIC RELAXATION PARAMETERS:
double nodalMass = SampleCsWpfViewModel.Instance.NodalMass;
double h = SampleCsWpfViewModel.Instance.TimeStep;
int maxIter = SampleCsWpfViewModel.Instance.MaxIter;
double tol = SampleCsWpfViewModel.Instance.Tol;
int nIter = 0;

// CREATE IGA SURFACE FROM RHINO SURFACE:
List<IGAMembraneSurfacePatch> IGAPatches = new List<IGAMembraneSurfacePatch>();
List<FormFindingPatch> formfindingPatches = new b List<FormFindingPatch>();
List<NurbsSurface> nurbsPatches = new List<NurbsSurface>();
List<System.Guid> srfGuids = new List<Guid>();
List<int> bc = new List<int>();
int numOfPatches = SampleCsWpfViewModel.Instance.PatchList.Count;

if (numOfPatches < 1) // If no patch is selected for analysis don't crash!!
       { return; }

List<double[]> uniqueB = new List<double[]>();

for (int i = 0; i < numOfPatches; i++)
{
FormFindingPatch currPatch = new
FormFindingPatch(SampleCsWpfViewModel.Instance.PatchList[i].surfacePatch);

formfindingPatches.Add(currPatch);
System.Guid srfGuid = new Guid();
srfGuids.Add(srfGuid);

Rhino.Geometry.Surface srf = currPatch.surfacePatch.Surface();
// Convert the surface to a NURBS surface
Rhino.Geometry.NurbsSurface nurbsSrf;
nurbsSrf = srf.ToNurbsSurface();
nurbsPatches.Add(nurbsSrf);
// IGA geometry data from Rhino.Geometry.NurbsSurface
int degP, degQ;
double[][] B;
double[] knotXi, knotEta;
getIGAGeometry(nurbsSrf, out degP, out degQ, out B, out knotXi, out knotEta);

// Create IGAsurface patch from Rhino Geometry:
IGAMembraneSurfacePatch IGAPatch = new IGAMembraneSurfacePatch(B, knotXi,
knotEta, degP, degQ);
IGAPatches.Add(IGAPatch);

for (int j = 0; j < IGAPatch.B.GetLength(0); j++)
{
double[] currPoint = new double[4] { B[j][0], B[j][1], B[j][2], B[j][3] };
bool alreadyExist = ComparePoints(currPoint, uniqueB);

if (!alreadyExist)
{
uniqueB.Add(currPoint);
}}

// CONNECTIVITY ARRAYS FOR THE PATCH:
int[][] INC = IGAPatch.INC;
int[][] ENOD = IGAPatch.ENOD;
```

```
int[][] DOF = IGAPatch.DOF;
int[][] EDOF = IGAPatch.EDOF;
// Numbers for the patch                          Number of:
int ndof = IGAPatch.ndof;                         // global degrees of freedom
int ldof = IGAPatch.nen * IGAPatch.dim;           // local degrees of freedom
int nel = IGAPatch.nel;                           // elements
int nen = (IGAPatch.p + 1) * (IGAPatch.q + 1);    // local basis functions
}

double[][] B_Global = uniqueB.ToArray();

int ndofGlobal = B_Global.GetLength(0) * 3;
int[][] DOF_Global = IGAFunctions.BuildDOF(3, ndofGlobal / 3);

Vector<double> extF_Global = new DenseVector(ndofGlobal);
Vector<double> intF_Global = new DenseVector(ndofGlobal);

for (int k = 0; k < IGAPatches.Count; k++)
{
IGAMembraneSurfacePatch IGAPatch = IGAPatches[k];
int dim = IGAPatch.dim;
int nel = IGAPatch.nel;                           // elements
int nen = (IGAPatch.p + 1) * (IGAPatch.q + 1);    // local basis functions

for (int m = 0; m < B_Global.GetLength(0); m++)
{
       for (int n = 0; n < IGAPatch.B.GetLength(0); n++)
       {
       if ((B_Global[m][0] == IGAPatch.B[n][0]) && (B_Global[m][1] ==
       IGAPatch.B[n][1]) && (B_Global[m][2] == IGAPatch.B[n][2]))
              {
              IGAPatch.DOF[0][n] = DOF_Global[0][m];
              IGAPatch.DOF[1][n] = DOF_Global[1][m];
              IGAPatch.DOF[2][n] = DOF_Global[2][m];
              }
       }
}

IGAPatch.EDOF = IGAFunctions.BuildEDOF(dim, nel, nen, IGAPatch.DOF,
IGAPatch.ENOD);
int[][] DOF = IGAPatch.DOF;
int[][] EDOF = IGAPatch.EDOF;

// SURFACE PROPERTIES:
double t = formfindingPatches[k].t;
double E = formfindingPatches[k].E;
double nu = formfindingPatches[k].nu;

double[] force = new double[IGAPatch.dim];
force[0] = formfindingPatches[k].xLoad;
force[1] = formfindingPatches[k].yLoad;
force[2] = formfindingPatches[k].zLoad;

// GET BASIS FUNCTIONS, SURFACE PARAMETERS AND EXTERNAL FORCE:
IGAPatch.ComputeBasisFunctions();
IGAPatch.ComputeSurfaceParameters(E, nu, t);

IGAPatch.ComputeExternalForce(t, force, ndofGlobal);
extF_Global = extF_Global + IGAPatch.extF;
}

// BOUNDARY CONDITIONS:
```

```
// Loop through all control points
for (int r = 0; r < B_Global.GetLength(0); r++)
{   // Loop through all boundary points
for (int s = 0; s < SampleCsWpfViewModel.Instance.BoundaryPointList.Count; s++)
{
// The x- and y-valaue of the current boundary point
BoundaryPoint currPoint =
SampleCsWpfViewModel.Instance.BoundaryPointList.ElementAt(s);
double x = currPoint.boundaryPoint.Point().Location.X;
double y = currPoint.boundaryPoint.Point().Location.Y;

if ((Math.Abs(B_Global[r][0] - x) < 1e-4) && ((Math.Abs(B_Global[r][1] - y) <
1e-4)))
{
if (currPoint.lockX == true)
{ bc.Add(DOF_Global[0][r]); }
if (currPoint.lockY == true)
{ bc.Add(DOF_Global[1][r]); }
if (currPoint.lockZ == true)
{ bc.Add(DOF_Global[2][r]); }
}}}

// MASS MATRIX:
DiagonalMatrix massMdiag = new DiagonalMatrix(ndofGlobal, ndofGlobal,
nodalMass);
Vector<double> massMvec = new DenseVector(ndofGlobal);
for (int i = 0; i < ndofGlobal; i++)
{
massMvec[i] = nodalMass;
}

// INITIALIZATIONS:
// Initial displacements and velocities
Vector<double> u = new DenseVector(ndofGlobal);
Vector<double> v = new DenseVector(ndofGlobal);

// Initial forces
Vector<double> intF = new DenseVector(ndofGlobal);
Vector<double> tau = new DenseVector(new double[] { 1, 1, 0 });
tau = tau * 5000000000 * 0;

// Initial kinetic energy
double kinE = massMdiag.LeftMultiply(v).DotProduct(v);
double kinEPrev = 0;
List<double> kinS = new List<double>();

// Initial norm
double residualNorm = extF_Global.L2Norm();

// DYNAMIC RELAXATION ITERATION:
// While the sum of the residual force is larger than the tolerance, do:
while (residualNorm > tol)
{
intF_Global.Clear();
for (int patchNum = 0; patchNum < IGAPatches.Count; patchNum++)
{
IGAMembraneSurfacePatch IGAPatch = IGAPatches[patchNum];
double t = formfindingPatches[patchNum].t;
int nel = IGAPatch.nel;
int ldof = IGAPatch.nen * 3;
// Extract the element displacements from the global displacements
Matrix<double> ed = new DenseMatrix(nel, ldof);
for (int i = 0; i < nel; i++)
{
```

```
for (int j = 0; j < ldof; j++)
{
ed[i, j] = u[IGAPatch.EDOF[i][j]];
}}

// Calculate the internal forces in the membrane
IGAPatch.ComputeInternalForceParallel(ed, tau, t, ndofGlobal);
intF_Global = intF_Global + IGAPatch.intF;
}

// Calculate residual
Vector<double> residual = extF_Global - intF_Global;

// Set the residual to zero in the boundary DOFs
for (int i = 0; i < bc.Count(); i++)
{
residual[bc[i]] = 0;
}

// Residual norm
residualNorm = residual.L2Norm();

// Update velocities
if ((nIter > 0) && !((kinE - kinEPrev) < 0))
{
v = v + residual.PointwiseDivide(massMvec).Multiply(h);
}
else
{
v = residual.PointwiseDivide(massMvec * 2).Multiply(h);
kinE = 0;
}

// Update displacements
u = u + v.Multiply(h);
for (int patchNum = 0; patchNum < IGAPatches.Count; patchNum++)
{
IGAMembraneSurfacePatch IGAPatch = IGAPatches[patchNum];
NurbsSurface nurbsSrf = nurbsPatches[patchNum];

//----------------------------------------
// Write solution to rhino geometry

int nodeNumber = 0;
for (int j = 0; j < IGAPatch.getM(); j++)
{
for (int i = 0; i < IGAPatch.getN(); i++)
{
Point3d pt3 = new Point3d();
pt3.X = IGAPatch.getB(i, j)[0] + u[IGAPatch.DOF[0][nodeNumber]];
pt3.Y = IGAPatch.getB(i, j)[1] + u[IGAPatch.DOF[1][nodeNumber]];
pt3.Z = IGAPatch.getB(i, j)[2] + u[IGAPatch.DOF[2][nodeNumber]];
nurbsSrf.Points.SetControlPoint(i, j, pt3);
nodeNumber = nodeNumber + 1;
}}

// Add new surface to rhino
if (nurbsSrf.IsValid)
{
if (nIter < 1)
{
srfGuids[patchNum] = RhinoDoc.ActiveDoc.Objects.AddSurface(nurbsSrf); // Adds
the surface
```

```
}
RhinoDoc.ActiveDoc.Objects.Replace(srfGuids[patchNum], nurbsSrf);
RhinoDoc.ActiveDoc.Views.Redraw();
}
//----------------------------------------
}
// Kinetic damping
kinEPrev = kinE;
kinE = massMdiag.LeftMultiply(v).DotProduct(v) * 0.5;
kinS.Add(kinE);
// If peak is detected, do:
if ((kinE - kinEPrev) < 0)
{
// Find location of the peak
double eC = kinE; double eB = kinS[nIter - 1]; double eA = kinS[nIter - 2];
double eE = eC - eB; double eD = eB - eA;
double eQ = eE / (eE - eD);

u = u - v.Multiply(h * (1 - eQ)) +
massMdiag.Inverse().LeftMultiply(residual).Multiply(Math.Pow(h, 2) / 2 * eQ);
v.Clear();
kinS[nIter] = 0;
}
nIter++;

if (nIter >= maxIter)
break;
}
RhinoApp.WriteLine("residual norm = {0}", residualNorm);
RhinoApp.WriteLine("total iterations = {0}", nIter);
RhinoApp.Write("Number of degrees of freedom: ");
RhinoApp.WriteLine(ndofGlobal.ToString());
```

AIII: British Museum Great Court roof function

The height of the roof, z, is a function of three fundamental functions
$z = z_1 + z_2 + z_3$

The first one defining the difference in height between the rectangular buildings and the circular Reading Room

$$z_1 = \left(h_{centre} - h_{edge}\right)\eta + h_{edge}$$

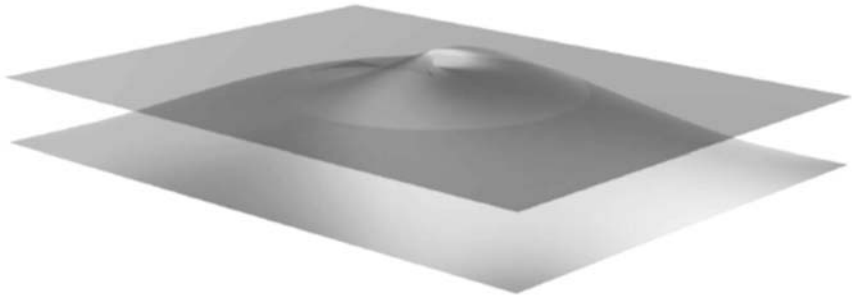


Figure AIII: 1 Surface corresponding to the first function [4].

The second creates a horizontal surface at the edges. Together with the third function it makes sure z=0 around the boundaries

$$\frac{z_2}{\alpha} = \left(1-\lambda\right)\left(\begin{array}{l}\left(35+10\psi\right)\frac{1}{2}\left(1+\cos 2\theta\right)+\frac{24}{2}\left(\frac{1}{2}\left(1-\cos 2\theta\right)+\sin\theta\right)\\ +\left(7.5+12\psi\right)\left(\frac{1}{2}\left(1-\cos 2\theta\right)-\sin\theta\right)-1.6\end{array}\right)$$
$$-\frac{10}{2}\left(1+\cos 2\theta\right)+10\left(\frac{1}{2}\left(\frac{1}{2}\left(1-\cos 2\theta\right)+\sin\theta\right)\right)^2\left(1-3\alpha\right)$$
$$+2.5\left(\frac{1}{2}\left(\frac{1}{2}\left(1-\cos 2\theta\right)-\sin\theta\right)\right)^2\left(\frac{r}{a}-1\right)^2$$

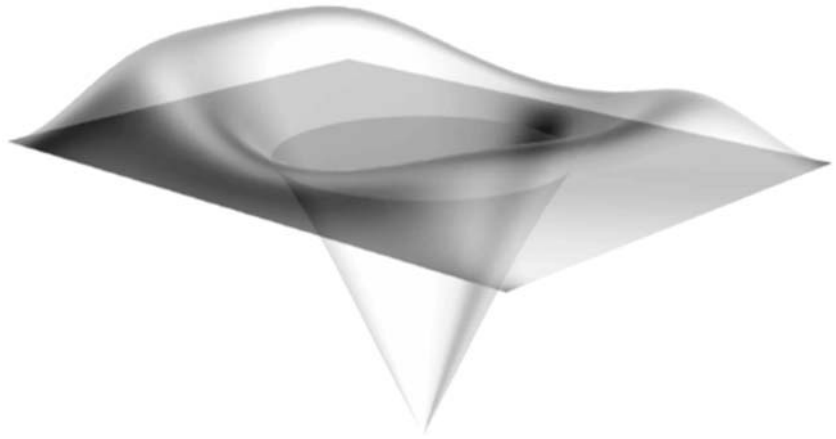


Figure AIII: 2 Surface corresponding to the second function [4].

The third function with infinite curvature at the corners

$$\frac{z_3}{\beta}=\lambda\left(\frac{3.5}{2}(1+\cos 2\theta)+\frac{3}{2}(1-\cos 2\theta)+0.3\sin\theta\right)$$
$$+1.05\left(e^{-\mu\left(1-\frac{x}{b}\right)}+e^{-\mu\left(1+\frac{x}{b}\right)}\right)\left(e^{-\mu\left(1-\frac{y}{c}\right)}+e^{-\mu\left(1+\frac{y}{d}\right)}\right)$$

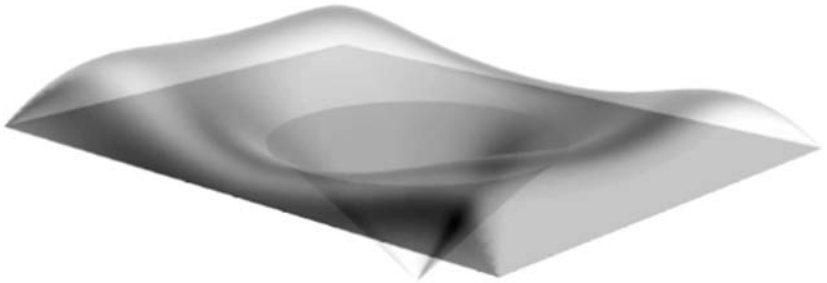


Figure AIII: 3 Surface corresponding to the third function [4].

The polar co-ordinates used in the expressions are

$r=\sqrt{x^2+y^2}$ and $\theta=\cos^{-1}\frac{x}{r}=\sin^{-1}\frac{y}{r}$

And the parameters

$$\eta=\frac{\left(1-\frac{x}{b}\right)\left(1+\frac{x}{b}\right)\left(1-\frac{y}{c}\right)\left(1+\frac{y}{d}\right)}{\left(1-\frac{ax}{rb}\right)\left(1+\frac{ax}{rb}\right)\left(1-\frac{ay}{rc}\right)\left(1+\frac{ay}{rd}\right)},\quad \psi=\left(1-\frac{x}{b}\right)\left(1+\frac{x}{b}\right)\left(1-\frac{y}{c}\right)\left(1+\frac{y}{d}\right),$$

$\alpha=\left(\frac{r}{a}-1\right)\psi$ and

$$\frac{1-\frac{a}{r}}{\beta}=\frac{\sqrt{(b-x)^2+(c-y)^2}}{(b-x)(c-y)}+\frac{\sqrt{(b-x)^2+(d+y)^2}}{(b-x)(d+y)}+\frac{\sqrt{(b+x)^2+(c-y)^2}}{(b+x)(c-y)}+\frac{\sqrt{(b+x)^2+(d+y)^2}}{(b+x)(d+y)}$$

The constants in the expressions are

$a=22.245,\quad b=36.625,\quad c=46.025,\quad d=51.125,\quad \lambda=0.5,\quad \mu=14,$
$h_{centre}=20.955\quad$ and $\quad h_{edge}=19.71.$