National Technical University of Athens

School of Civil Engineering

Department of Structural Engineering

Institute of Structural Analysis & Antiseismic Research

Isogeometric Stiffness Matrix Computation in GPU Programming Environment

Supervisors:

Manolis Papadrakakis, Professor NTUA

Panagiotis Karakitsios, PhD Candidate

Athens, November 2016

Christos Gkritzalis

Ευχαριστίες

Θα ήθελα να εκφράσω την αμέριστη ευγνωμοσύνη μου σε όλους όσους συνέβαλαν στην εκπόνηση της διπλωματικής μου εργασίας. Κατ' αρχάς, θα ήθελα να ευχαριστήσω τον επιβλέποντά μου, Καθηγητή του Εργαστηρίου Στατικής & Αντισεισμικών Ερευνών της Σχολής Πολιτικών Μηχανικών του Εθνικού Μετσόβιου Πολυτεχνείου, κ. **Μανόλη Παπαδρακάκη** για το γεγονός ότι με εισήγαγε στο πεδίο της υπολογιστικής μηχανικής και για την ευκαιρία που μου προσέφερε να καταπιαστώ με τη ρηξικέλευθη Ισογεωμετρική Μέθοδο, καθώς επίσης και για την πολύτιμη καθοδήγησή του.

Θα ήθελα επίσης να ευχαριστήσω τον ερευνητή Υποψήφιο Διδάκτορα **Παναγιώτη Καρακίτσιο** για την εξαιρετική μας συνεργασία τα δύο τελευταία έτη, την καθοδήγηση και τις συμβουλές του. Η ερευνητική αξία της συγκεκριμένης εργασίας δε θα ήταν η ίδια χωρίς τη συνεισφορά του. Θερμές ευχαριστίες και στον ερευνητή **Γεώργιο Καραΐσκο**, με τον οποίο έχουμε γράψει αμέτρητες γραμμές κώδικα, απαραίτητες για την εξαγωγή των ερευνητικών αποτελεσμάτων της εργασίας αυτής.

Θα ήθελα επίσης να ευχαριστήσω το **Εθνικό Μετσόβιο Πολυτεχνείο** και τη **Σχολή Πολιτικών Μηχανικών**, όχι μόνο για το ευρύ φάσμα γνώσεων και εξειδίκευσης που μου παρείχε, αλλά και για το γεγονός ότι συνέβαλε στο να αναπτύξω τον κατάλληλο τρόπο σκέψης που απαιτεί το επάγγελμα και η επιστήμη του μηχανικού.

Αθήνα, Νοέμβριος 2016

Χρήστος Γκρίτζαλης

Περίληψη

Η υπολογιστική μηχανική ασχολείται με τη χρήση των υπολογιστικών μεθόδων και των υπολογιστών για τη μελέτη των φυσικών και μηχανικών συστημάτων που διέπονται από τις αρχές της μηχανικής. Στην πράξη συχνά προκύπτουν προβλήματα, τα οποία απαιτούν αριθμητική προσομοίωση με έναν τρόπο που είναι ακριβής και, παράλληλα, υπολογιστικά αποδοτικός. Η ακρίβεια μπορεί να επιτευχθεί χρησιμοποιώντας μία από τις σύγχρονες μεθόδους προσομοίωσης, αλλά τι συμβαίνει με την υπολογιστική αποδοτικότητα; Παρά το γεγονός ότι τις τελευταίες δεκαετίες παρατηρήθηκε μεγάλη βελτίωση στις δυνατότητες των υπολογιστών, ακόμα η επίλυση προβλημάτων μηχανικού μεγάλης κλίμακας παραμένει πρόκληση. Είναι κατανοητό ότι η ακρίβεια και το υπολογιστικό κόστος είναι έννοιες αλληλεξαρτώμενες. Ο σκοπός της συγκεκριμένης διπλωματικής εργασίας είναι η επιτάχυνση της Γραμμικής Στατικής Ισογεωμετρικής Μεθόδου (μία καινοτόμο μεθοδολογία πλήρους ενοποίησης των CAD – CAE, η οποία εισήχθη από τους J. Austin Cottrell, Thomas J.R. Hughes και Yuri Bazilevs) για την επίλυση προβλημάτων μεγάλης κλίμακας.

Για το σκοπό αυτό, κώδικας Ισογεωμετρικής Μεθόδου, βασισμένης στα NURBS, για εκτέλεση σε CPU (κεντρική μονάδα επεξεργασίας) και GPU (κάρτα γραφικών) υλοποιήθηκε. Προς αυτήν την κατεύθυνση, παράλληλοι αλγόριθμοι εκτελέστηκαν στη GPU για τον υπολογισμό του μητρώου στιβαρότητας προβλημάτων Ισογεωμετρικής Ανάλυσης. Η αποδοτικότητα διαφόρων μορφών αποθήκευσης του μητρώου στιβαρότητας διερευνάται. Πολυάριθμες κλασικές εφαρμογές σχεδιάστηκαν και αναλύθηκαν με το Geomiso, το πρώτο παγκοσμίως λογισμικό CAD/ CAE (υπολογιστικής γεωμετρίας/ υπολογιστικής μηχανικής).

Ως κύριες γλώσσες προγραμματισμού χρησιμοποιήθηκαν οι C++ και C#. Για την εφαρμογή του παράλληλου προγραμματισμού σε περιβάλλοντα γραφικής επεξεργασίας χρησιμοποιήθηκε η Quant Alea.

Η εργασία αυτή οργανώνεται ως εξής: Το **Κεφάλαιο 1** αποτελεί μία εισαγωγή στο πεδίο της Ισογεωμετρικής Μεθόδου. Στο **Κεφάλαιο 2** εξετάζονται οι B-SPLines και οι NURBS γεωμετρίες. Το **Κεφάλαιο 3** είναι αφιερωμένο στις CPUs και στις GPUs κάνοντας μία σύντομη εισαγωγή στον παράλληλο προγραμματισμό. Το **Κεφάλαιο 4** ασχολείται με το χειρισμό των πινάκων. Το **Κεφάλαιο 5** πραγματεύεται τις τεχνικές πύκνωσης δικτύου στην ισογεωμετρική ανάλυση και στο **Κεφάλαιο 6** πραγματοποιείται η σειριακή και παράλληλη μόρφωση του μητρώου στιβαρότητας. Η εφαρμογή των εξωτερικών φορτίων και των συνοριακών συνθήκων εξετάζεται στο **Κεφάλαιο 7**. Τέλος, στο **Κεφάλαιο 8**, κλασικές 2D και 3D εφαρμογές παρουσιάζονται. Το **Κεφάλαιο 9** περιέχει τα συμπεράσματα της ερευνητικής εργασίας, ακολουθεί το παράρτημα στο οποίο συνοψίζονται όλες οι εφαρμογές που εξετάζονται.

Abstract

Computational mechanics is concerned with the use of computational methods and computers to study physical and engineering systems governed by the principles of mechanics. In the engineering, practice frequently arise problems that demand a numerical simulation in a way that is both accurate and computationally efficient. The accuracy can be achieved by using one of the many up-to-date simulation methods, but what happens with the computational efficiency? Despite the fact that the last decades a large improvement of the computers capabilities have been observed, still the challenge of solving large-scale problems remains, since more and more difficult problems are tried to be solved by the engineers. It is understandable that accuracy and computational cost go together.

Isogeometric Analysis code, based on NURBS, for execution on CPU (central processing unit) and on GPU (graphic processing unit) was created. In this direction massively parallel algorithms have been implemented on GPU (graphic processing unit) for the computation of the Isogeometric stiffness matrix. Adding to this, the efficiency of different storage formats have been examined in this work. A numerus classical applications have been designed and analyzed in Geomiso, the first worldwide CAD/ CAE (computer aided design/ computer aided engineering) software.

C++ and C# was used as the main programming languages. For the implementation of GPU parallel code, the parallel programming environment and programming model that has been used is the Quant Alea.

This work is organized as follows: **Chapter 1** is a general introduction to the topic of isogeometric analysis. **Chapter 2** examines B-SPLines and NURBS. **Chapter 3** is dedicated to computers' central processing units (CPUs) and graphics processing units making a brief review to parallel computing. **Chapter 4** deals with the handling of matrices. In **Chapter 5**, the refinement techniques applied on isogeometric analysis are outlined and in **Chapter 6** the sequential and the parallel formulation of the stiffness matrix is investigated. The application of external loads and boundary conditions and the resulting displacement, stress and strain fields are investigated in **Chapter 7**. Finally, in **Chapter 8**, classical 2D and 3D application are presented. **Chapter 9** presents the conclusions, followed by the appendix containing the examples tested in this work.

Table of Contents

Table of Figures

1 The Isogeometric Method

1.1 Finite Element Analysis

1.1.1 Historical overview

Isogeometric method as a historic computational mechanics achievement

Isogeometric analysis (IGA) is the powerful generalization of the classical finite element analysis (FEA). It is a recently developed computational approach offering the possibility of fully integrating finite element analysis into conventional NURBS-based computer-aided design (CAD) tools. To date, it is necessary to convert data between CAD and FEA software packages to analyse new designs during development, a demanding task as the computational geometric approach for each is different. This new technique employs complex NURBS geometry (most CAD packages are based on it) in the FEA application directly. This lets models to be designed, examined and adjusted in one effort, using a common data set. The pioneers of this technique are Tom Hughes and his group at the University of Texas at Austin.

The most popular software based on IGA is Geomiso (geomiso.com), the first worldwide fully integrated CAD/ CAE software. This pioneering software has managed to fully integrate computer-aided design (CAD) and computer-aided engineering (CAE), merging the geometric design with the mesh generation into a single procedure, eliminating the geometric errors, significantly increasing the accuracy of the engineering analysis and drastically reducing the required computational time and cost, thus creating high added value for both the engineers and the engineering applications.

Figure 1.1. Thomas J. R. Hughes.
(users.ices.utexas.edu)

Isogeometric analysis is set to bridge the existing gap between CAD and CAE. Thomas J.R. Hughes, J.A. Cottrell & Y. Bazilevs published a book in 2009, which introduces this new method. “Isogeometric Analysis: Towards Integration of CAD and FEA” is the first book to be issued on this new method. It is proved to be a trustworthy guide to researchers worldwide who specialize in IGA. It offers an extensive study of IGA and numerous applications. IGA can lead to innovative research in the field computational mechanics (structural linear/ nonlinear static/ dynamic analysis, biomechanics, etc).

The need for isogeometric analysis

The need for a new method had to be met; greater challenges in finite element analysis were arising every day, demanding faster and more precise results. Even nowadays, although structures worldwide are designed using FEM instead of the traditional by-hand ways, design errors cannot be avoided, especially in the case of demanding engineering project with complex geometry.

Since 2000, structural collapse cost humanity over 1500 lives; many of them could have been spared with a more accurate tool for analysis and design being available to engineers. Unfortunately, present analysis technologies require a lot of man-hours for manual generation of approximated, FEM-suitable geometries. Consequently, they derive the engineer from his main task and force him to devote less time in result evaluation. The risk is even greater, considering the mistakes the engineer can make during this. Furthermore, the approximation of the geometry sometimes is clearly not enough for the desired convergence.

All these led to the isogeometric method. In order to better understand this new method, the engineers have to acknowledge the evolution of computer-aided engineering throughout its history and understand the principles this revolutionary method is based on.

This introduction provides the historical overview of the computer-aided engineering technologies.

Before finite element method

In the past, engineers had to cover the demands for efficient and accurate structural analysis. Construction always needed the cost-efficiency provided by design. In the early years, the only weapon the engineer had was his mind. In order to solve a statically indeterminate structure, equilibrium, constitutive and compatibility laws had to be applied. Several methods were proposed for this purpose.

The **force method**, based on the Betti-Maxwell theorem, was combined with virtual works in order to be able to take into account forces and moments. The main idea is the preservation of equilibrium and the calculation of forces, in order to ensure compatibility.

The **displacement method**, on the other hand, ensures that compatibility is maintained and equilibrium is achieved via calculation of displacements.

The **moment distribution method**, also known as the **Cross method**, relies upon computational iteration cycles to an initial moment distribution, until the desired approximation is achieved. It only yields results for bending effects and ignores axial and shear tension. Its efficiency made it very popular among the engineers in the 1930s.

These methods had their constraints; a vast amount of computational time is required and many errors occur in the manual computational process. More complex problems could only be approximated and sometimes engineers had to solve differential equations by hand in order to obtain the solution.

Finite element method

Structural analysis has been a major part of the engineering field of practice. The knowledge of a structure's reaction to certain loads enhances its safety and makes it cost-effective. It has a wide field of application, including buildings, bridges, airplanes, space shuttles, ships, satellites, nuclear stations and much more. At first, engineers used methods obtained from the solution of differential equations in order to evaluate the displacement, strain and stress field. Structural mechanics theorems were developed and used in order to solve this computational problem. These served their purpose well for relatively simple, everyday linear problems. However, new technologies emerged and the constant demand for faster, more accurate solutions to complicated problems had to be met.

Early computer models had made their appearance and engineering scientists were eager to use them in their problems. The finite element method can be placed in the late stages of World War II. Structural engineers working for the Royal Aeronautical Society of London had to design an innovative type of combat jet aircraft whose speed required swept-back wings. Unfortunately, none of the existing theories could fit to solve such a complex problem. The failure of the German ME262 was palpable proof of that matter.

This demanding task was assigned to one of the brightest minds of the Royal Aeronautical Society of London, John Argyris. Argyris was born in Greece, Volos in 1913 and entered the School of Civil Engineering of National Technical University of Athens. John Argyris was the mentor of Professor Manolis Papadrakakis.

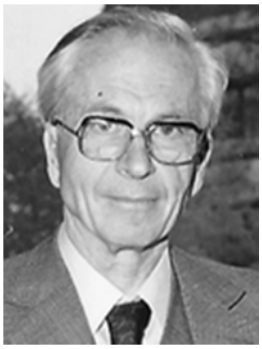


Figure 1.2. John Argyris.

Figure 1.3. Manolis Papadrakakis.

John Argyris graduated from Technical University of Munich in 1936 and begun to work in industrial applications of complex structures. He remained in Germany in the beginning of World War II and was accused of giving research info to the Allies, arrested and sent to a concentration camp when the Axis invaded Greece. He was rescued by a German Admiral, Kanaris, who was of Greek ancestry as well. After breaking out from the concentration camp, Argyris escaped to Switzerland by swimming through the Rhine river, in the middle of a raid holding his passport with his teeth. He finished his Doctoral Degree in Aeronautics from ETH, Zurich in 1942. Afterwards, he moved to England and was engaged with the Royal Aeronautical Society of London, working as a technical officer. As a researcher, he was really skeptical with Cartesian coordinate systems and the way they were used in engineering. He believed that triangular and tetrahedral elements were far more suitable for engineering applications. John Argyris was not devoted to everyday problems, but would rather get busy with difficult, apparently unsolvable problems. His superiors quickly recognized this trait and he in return welcomed the challenges he was given, including the swept-back wings aircraft problem.

It seems he was the right man for the job. In August 1943, after 3 days and nights of devotion to the problem, he had a breakthrough. He used triangular elements to simulate the swept-back wings and solved the model in the electro-mechanical computing device the Society had recently acquired. The device was able to solve an equation with up to 64 unknowns. Analysis results were very close to the experimental results, with a deviation of approximately 8%. This was the birth of the finite element Method. His papers were at once labeled “secret”. This innovative method included a different measurement of strain and stress field, diverging from the classical Cartesian field and was proving to be useful and easily generalized.

In the following years, the «matrix force and displacement method», mostly known as finite element method (or, as Ray Clough wrote in 1960, «The Argyris method») was developed by many researchers, including Turner, Clough, Zienkiewicz and Cheung. Argyris resumed his academic career with drastic contributions to the research and development of FEM as well as many other aspects of the engineering field until he was 88 years old. He invented, among others, the triangular element TRIC and is also well known for the contribution to the solution of the heat protection problem for the NASA space shuttle during the entrance in the atmosphere. He passed away in April 2004 [15].

Due to the swift evolution of computational speed and memory capacity, FEM became very popular within the engineering industry. Millions of dollars were invested in its development. New FEM technologies emerged, such as the isoparametric elements. These allowed for a more general approach and a better adaptation to complex geometries.

NTUA Professor Manolis Papardakakis has devoted his career in the evolution and outspread of FEM technologies. Nowadays, in the middle of the financial crisis in Greece, the Institute of Structural Analysis & Antiseismic Research of National Technical University of Athens has a remarkable research portfolio to show, always being up to date with the latest technological trends. Bright young minds are given a chance to shine, continuing and improving the cycle of expanding the boundaries of human knowledge.

In the dawn of 2000, structural engineering has changed drastically. Personal computers and a variety of FEM software are now available to engineers. All the hard work done so many years ago by hand is now avoided. Greater speeds and bigger rates of convergence are achieved every other month. Problems, once thought to remain unsolved, now seem common and relatively easy. Finite elements are used in a wide range of computational analysis, such as structural and dynamic analysis, fluid mechanics, biomechanics, earthquake engineering and many more. The modern engineer does not need to solve complex mathematical equations by hand, but has to pursue a global and thorough understanding of this field as well as knowledge of the innovative computational methods.

The engineering software market consists of many products devoted to the analysis of FEM models. There are generalized and more theoretical software that can solve almost any type of structure. NASTRAN, a widely used FEM platform, was originally developed by NASA in the 1960s in order to cover the Agency’s special needs. Simulia Abaqus, originally released in 1978, was developed using an open-source language, Python was initially intended for non-linear problems. It is particularly popular due to its wide range of modeling capabilities, both for linear and non-linear problems. ADINA (Automatic Dynamic Incremental Non-Linear Analysis), developed in 1974, is used in a wide range of non-linear problems. It has applications in static and dynamic analysis, heat transfer, compressible and incompressible flows and electromagnetic phenomena. FEMAP is used as an input creation and output processing tool for the engineers. It cooperates with the solver routines from other platforms (e.g. NASTRAN) and focuses on the easy and accurate communication between software and user.

Specialized software is also available in the engineering market. The specific characteristics and complexity of today’s structures require a more personal and delicate approach. ATENA, standing for Advanced Tool for Engineering Non-Linear Analysis, is specialized in everything to do with reinforced concrete structures. SOFiSTIK, first used in 1987, is directed towards bridge linear and non-linear analysis. Furthermore, there are platforms dedicated to the special needs of the Greek market. Designing structures in the country with the biggest seismic activity in Europe is not an easy task.
Geomiso is the first worldwide «two-in-one» fully integrated CAD/ CAE software and the first commercial one based on IGA

1.1.2 Basic idea

The basic idea of FEM is the approximation of the solution field via piecewise polynomial functions, the so-called **shape functions**. The multiplication of the shape functions' value **[R(x,y,z)]** by the nodes' displacements **{D}** leads to the displacement vector **{d(x,y,z)}**.

$$\underset{(6\times1)}{\{d(x,y,z)\}} = \underset{(6\times3n)}{[R(x,y,z)]} \cdot \underset{(3n\times1)}{\{D\}} \quad (3D)$$

where n is the number of nodes.

The problem is directly downsized from infinite unknowns to a finite number of degrees of freedom.

The strain **{ε(x,y,z)}** and stress **{σ(x,y,z)}** vectors are given by the following equation.

$$\underset{(6\times1)}{\{\varepsilon(x,y,z)\}} = \begin{bmatrix} \varepsilon_X(x,y,z) \\ \varepsilon_Y(x,y,z) \\ \varepsilon_Z(x,y,z) \\ \gamma_{XY}(x,y,z) \\ \gamma_{YZ}(x,y,z) \\ \gamma_{ZX}(x,y,z) \end{bmatrix} \quad (3D)$$

$$\underset{(6\times1)}{\{\sigma(x,y,z)\}} = \begin{bmatrix} \sigma_X(x,y,z) \\ \sigma_Y(x,y,z) \\ \sigma_Z(x,y,z) \\ \sigma_{XY}(x,y,z) \\ \sigma_{YZ}(x,y,z) \\ \sigma_{ZX}(x,y,z) \end{bmatrix} \quad (3D)$$

Multiplying the deformation matrix **[B]** by the nodes' displacement vector **{D}**, the strain vector **{ε(x,y,z)}** is given by the following equation.

$$\underset{(6\times1)}{\{\varepsilon(x,y,z)\}} = \underset{(6\times3n)}{[B(x,y,z)]} \cdot \underset{(3n\times1)}{\{D\}} \quad (3D)$$

The stress vector **{σ(x,y,z)}** is calculated from the corresponding strain vector **{ε(x,y,z)}**, using the Hooke's constitutive law.

$$\underset{(6\times1)}{\{\sigma(x,y,z)\}} = \underset{(6x6)}{[E]} \cdot \underset{(6\times1)}{\{\varepsilon(x,y,z)\}} \quad (3D)$$

The stiffness matrix **[k]** of each element (internal/ external virtual work equilibrium) is equal to:

$$\underset{(3n\times3n)}{[k]} = \int_V \underset{(3n\times6)}{[B]^T} \cdot \underset{(6\times6)}{[E]} \cdot \underset{(6\times3n)}{[B]}\, dV$$

Distributed loads **{f}** can be transformed into equivalent nodal loads **{F_R}**.

$$\underset{(3n\times1)}{\{F_R\}} = \int_V \underset{(3n\times3)}{[R]^T} \cdot \underset{(3\times1)}{\{f\}}\, dV$$

The contribution of each element is added to the total stiffness matrix **[K]** and the total load vector **{F_R}**.

The nodes' displacement vector **{D}** is calculated from the following equation.

$$\underset{(3n\times1)}{\{F_R\}} = \underset{(3n\times3n)}{[K]} \cdot \underset{(3n\times1)}{\{D\}} \Rightarrow \underset{(3n\times1)}{\{D\}} = \underset{(3n\times3n)}{[K]^{-1}} \cdot \underset{(3n\times1)}{\{F_R\}}$$

where n is the nodes' number.

1.1.3 Drawbacks

Despite the capabilities of FEM, numerous problems have to be solved. Finite elements approximate the geometry, thus introducing additional geometric errors, which decreases the accuracy of the analysis results. After the meshing has been completed, the initial geometry plays no more role in the analysis procedure. Due to this mesh, quite many problems arise.

If the initial mesh doesn't give an accurate solution, engineers have to use refinement algorithms and create from the very beginning a new finer one. The new mesh cannot be directly created from the coarse one. Efficiency is certainly at a low, as procedures already completed have to be repeated in order for the new mesh to be created. Additional computational time is required and the geometrical differences.

The lack of integration between design and mesh generation is crucial. Computational geometry provides simply the input data for the mesh generation. Exact geometrical representation is not reflected in the new mesh, nor is the smoothness of the initial model, which leads to a slightly different model that analysis solves. Even for a small change in the geometrical model, reintegration and new mesh generation is required. This process is not efficient for the modern engineer; instead of devoting his time in creativity and design, he has to undergo the time consuming task of regenerating a slightly different mesh over and over again.

Computational geometry has evolved; new and optimized structures are being used more often nowadays. Finite Element geometries fail to keep up with that pace and as a result, computer-aided engineering is separated even more from computer-aided design. Finite elements cannot cooperate with the modern technologies of T-SPLines and subdivision surfaces. These problems had always been present throughout the history of finite elements. Complicated computational methods and algorithms have been developed in order to overcome them. The problem is that the nature of FEM does not allow for significant steps toward CAD-FEM integration. Improvements to the basic structure of finite elements are difficult and quite inefficient.

In order to create a NURBS model, designers have to define the degree and the knot vector for each parametric axis and the control points (Cartesian coordinates, weights). **Figure 1.4** depicts the NURBS model of a car.

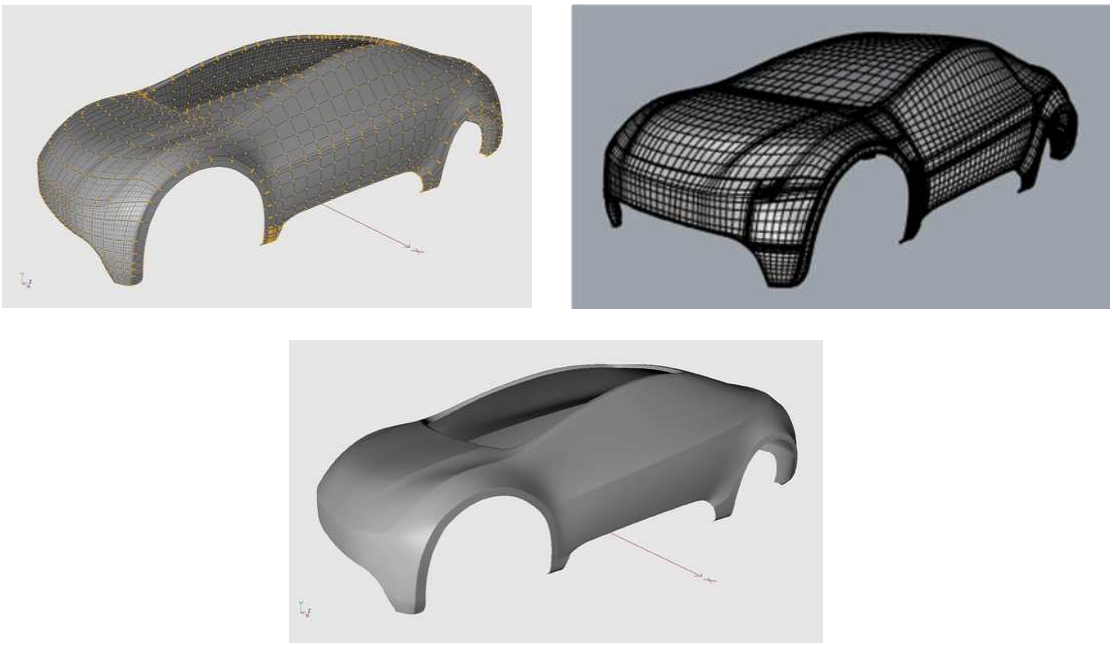
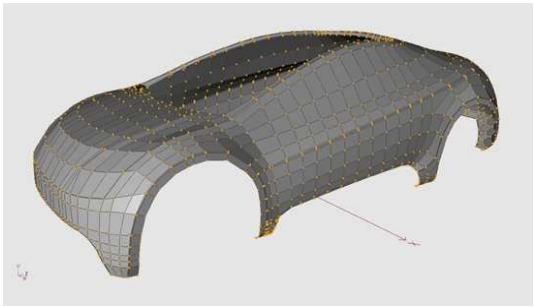
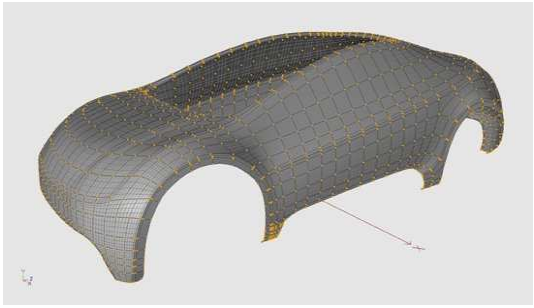


Figure 1.4. NURBS model of a car.
(grabcad.com)

Finite element method is unable to utilize the existing exact geometry mesh, thus creates a new approximate one, the finite element mesh. On the contrary, isogeometric analysis takes advantage of the exact geometry mesh, while refinement is conducted only for accuracy purposes while the geometry remains intact.



(a) FEA



(b) IGA

Figure 1.5. FEA vs. IGA.

1.2 Computer-Aided Design

1.2.1 Historical overview

Isogeometric analysis as a historic computer-aided engineering achievement

The evolution of computing systems made the design on a platform very attractive. Drafts could be edited easily and data could be stored and transferred at much higher speeds. Computer-aided design has many applications in today's world and a huge industrial support. Computer-generated imagery (CGI) is used in movies even more often; 3D and 2D cartoons are drafted and animated through computer software. Engineers draft complex designs such as cars, space shuttles, long span bridges and so on. All the drafts can be edited and the escalation to optimized drafting is easier than ever. Designers' time is now being devoted to creative thinking and taking ideas to the next level, rather than useless drafting by hand for hours.

However, there is still a lot to be improved. Computational geometry is involved in numerous engineering applications and should not be considered independently. Reinventing and improving computational geometry structures is the first step in completing this task. In order to understand the modern world of computer-aided design, one has to study the history, the evolution and the need that led to the creation of this scientific field.

Computer-aided design (CAD) emerged in the 1950s from the automotive, shipyard and aircraft industries. In those times, designers were able to produce accurate drafts by hand, but when ship cross sections had to be drafted in real-life size, pencils could not help anymore and things became a bit more complex. The main problem was the definition of a real-size curve, which smoothly interpolated several predetermined points in order to create the shell of the ship. This task was usually carried out in the loft of a building, due to the large amount of space needed. The loftsman, as he was called, used easy-to-bend pieces of steel or wood, the spline, in order to interpolate the points. In order to maintain the spline's shape, he usually put weights on them on certain points.

Figure 1.6. Ducks.
(boatdesign.net)

The development of NURBS arose from the need for effectively representing freeform surfaces. Two engineers in France stood at the forefront of this approach, Pierre Bezier from Renault and Paul de Casteljau from Citroen. Bezier's work was the first to reach publication, and soon after the CAD industry started using and enhancing Bezier curves. However, certain disadvantages of Bezier curves led to the search for more convenient forms of representation. Researchers introduced B-SPLines (similar to Bezier curves) with the curve defined by a set of points, the so-called **control points**, but their number was independent of the polynomial order of the curve. B-SPLines were a generalization of Bezier curves, more convenient to edit; changing a point no longer changed the whole curve, but only part of it.

Another problem is that even B-SPLines cannot produce an exact representation of conic sections. This is where NURBS came along. Ken Versprille was the first to work with NURBS on his dissertation in 1975. Later, when he acquired a top position in Computervision, the company began to support NURBS. Boeing, in its ambitious project to create a single curve representation that included Bezier curves and conic sections, became the first to industrialize NURBS.

After that, NURBS began to spread across the global CAD industry. They possessed a lot of interesting attributes. The parameterization of the whole curve was downsized to a few control points, numerically stable mathematical procedures and easy modification. Cartoon characters, videogame graphics, ships, cars, airplanes are designed using NURBS. This led to major investments from research and industrial faculties. Graphic designers became accustomed to them and students were taught about the theory and implementation of NURBS in real-life problems. This cycle led to their increasing popularity in the CAD industry.

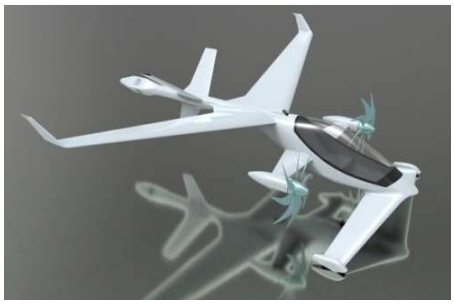
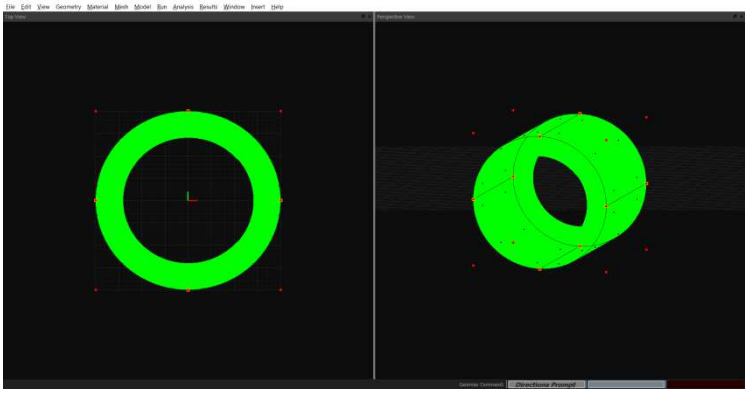
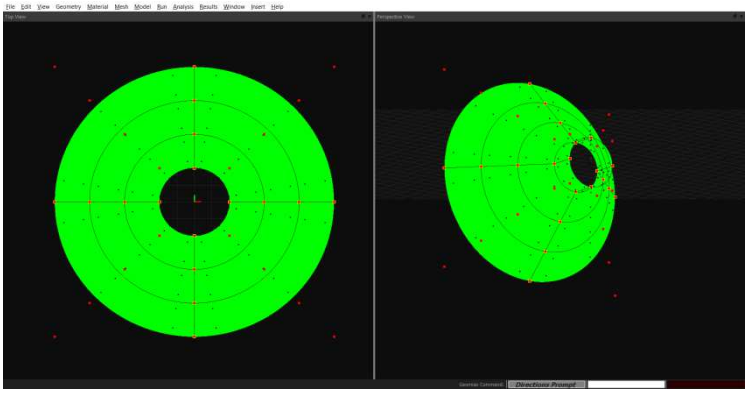


Figure 1.7. NURBS model of an airplane.
(creativecrash.com)

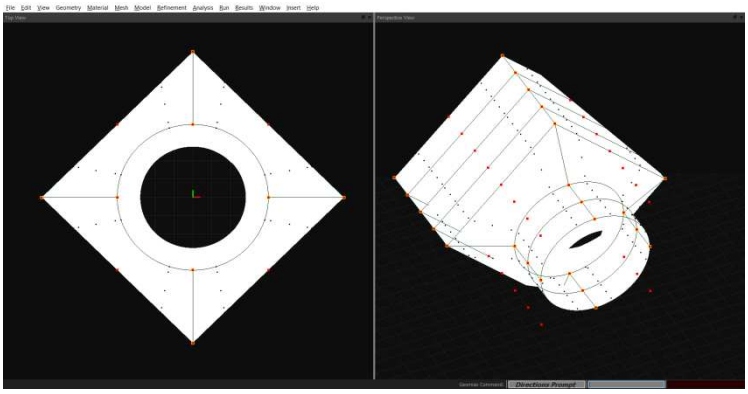
Nowadays, despite the fact that new more efficient CAD technologies have arisen (T-SPLines, polycube SPLines, subdivision surfaces), NURBS still remain the most popular one and ubiquitous in CAD systems. CAD industry has already invested billions of dollars in them. They are easier in use and not sophisticated. Their major strengths are that they are convenient for free-form surface modeling, can exactly represent all conic sections (circles, cylinders, spheres, ellipsoids, etc.), and that one can find numerous efficient and numerically stable algorithms to generate NURBS objects. NURBS also possess useful mathematical properties, such as refinement through knot insertion, C^{p-1}-continuity for order p, and the variation diminishing and convex hull properties. One of their main disadvantages is that they do not ensure watertight patch connection and don't allow local refinement. Despite their drawbacks versus other CAD technologies, their superiority in engineering design is indisputable.



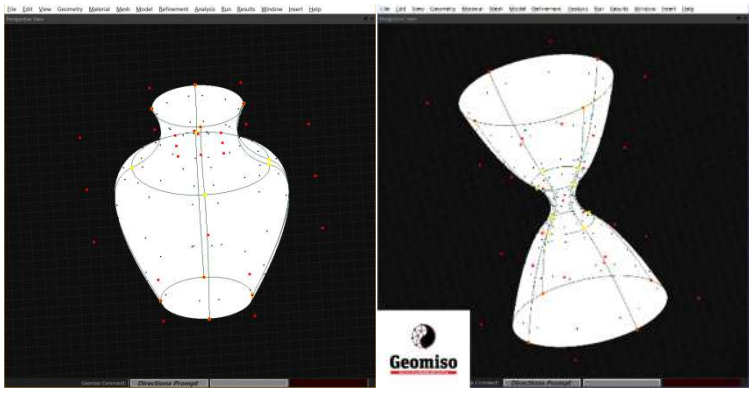
(a) Cylinder



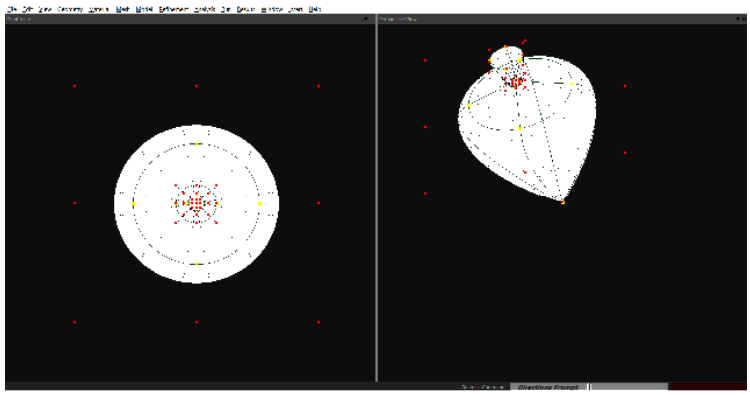
(b) Collar



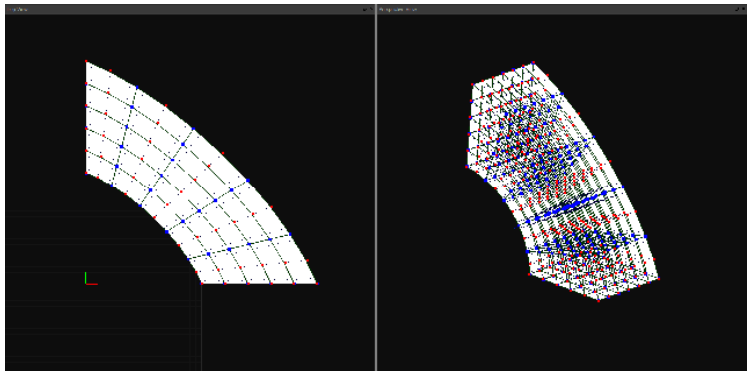
(c) Junction



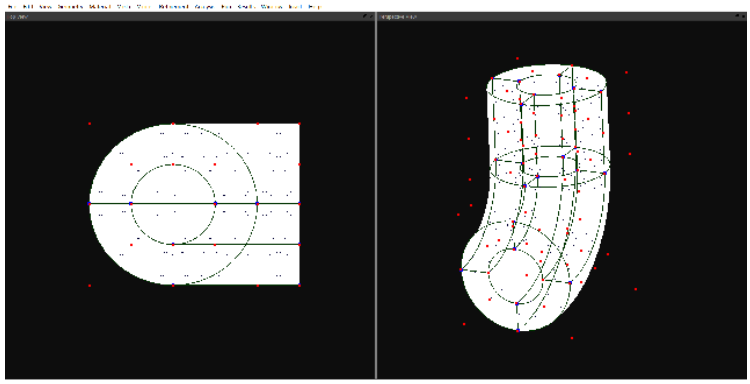
(d) Amphora (e) Hourglass



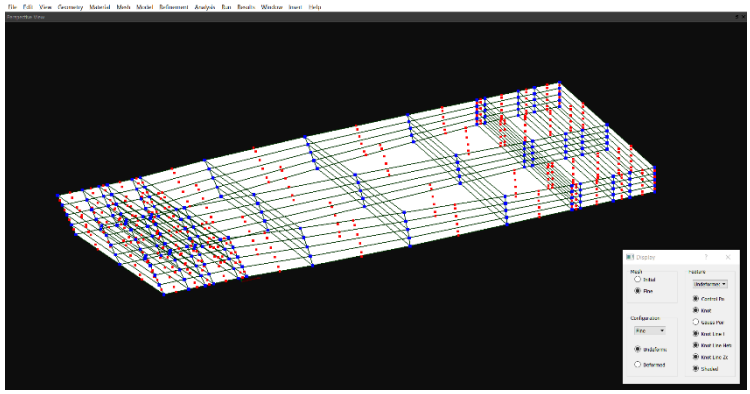
(f) Spinning top



(g) Annulus quarter



(h) Pipe



(i) Twisted pipe

Figure 1.8. NURBS models.
(geomiso.com)

1.3 Isogeometric Analysis

Although both the global CAD and the global CAE industries are growing at impressive compound annual growth rate, engineers are not able to benefit from the latest CAD advanced techniques. The main obstacle is the fact that engineers have a limited knowledge of computational geometry and designers know nothing about computational mechanics.

To date, designers create the product design using CAD software and then deliver it to an engineer, who transfers the data of the CAD design to another programme, a CAE one, so as to carry out the engineering analysis and study its behaviour. Using finite element software packages, engineers are not able to efficiently utilize the exact existing mesh of the CAD model and they have to create from the very beginning a new one, the so-called finite element mesh, which is not precise and only approximates the geometric domain. Consequently, geometric errors are introduced, while the engineering analysis isn't accurate enough to study complex CAD models.

Isogeometric method is the powerful generalization of the classical finite element method. The basic idea is to utilize the very same functions used for the geometry design in order to approximate the solution field. As a result, IGA is able to utilize the existing exact finite element mesh and eliminate the time-consuming procedure of mesh generation. This key attribute enables IGA to be clearly superior to the finite element analysis. Additionally, this new technique offers better capabilities, as it is able to encompass all the latest and most advanced CAD tools (NURBS, T-SPLines, polycube SPLines and subdivision surfaces). Therefore, engineering applications end up being better and more optimized.

Refinement in isogeometric analysis is quite more efficient than refinement in finite element analysis and lead to more richness in the overall refinement space. A major difference is that, in IGA it is conducted only for increasing the accuracy of the analysis results, while leaving the geometry and its parameterization intact. There are numerous ways in which the basis may be enriched. In particular, engineer is now able to control not only the element size and the order, but also the continuity of the basis. It's worth mentioning that in IGA there is an extra refinement type, the k-refinement.

The first mechanism by which one can enrich the basis is **knot insertion**. Knots may be inserted without changing a curve geometrically or parametrically. The second one is **order elevation**, which involves raising the polynomial order of the basis. As the basis has $p - m_i$ continuous derivatives across element boundaries, it is obvious that when the order is increased, the multiplicity need also to be increased if we are to preserve the discontinuities in the various derivatives already existing in the original curve. During order elevation, the multiplicity of each knot value is increased by one, but no new knot values are added. As with knot insertion, neither the geometry nor the parameterization are changed.

Implementing the new **k-refinement**, the engineer has the ability to both add knots and increase the order of the basis.

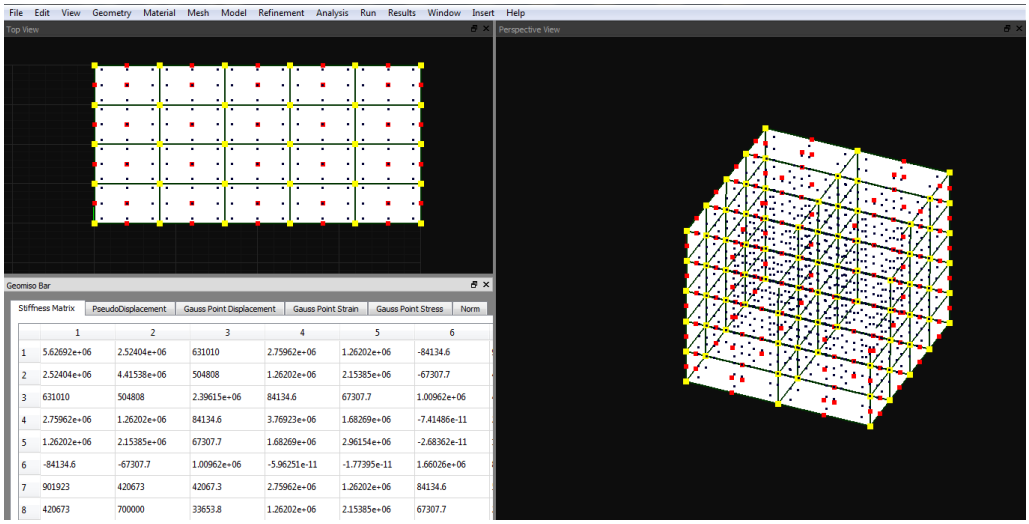
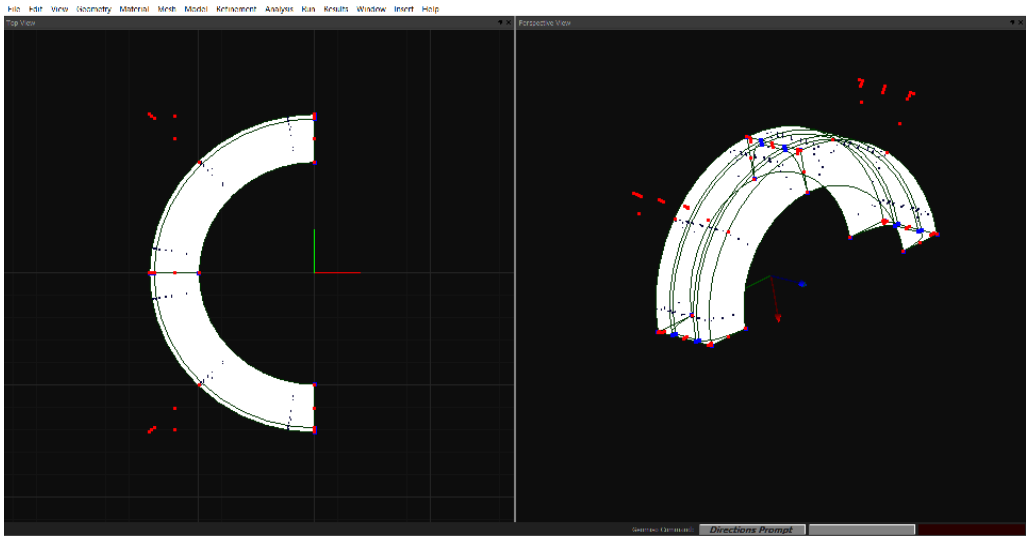
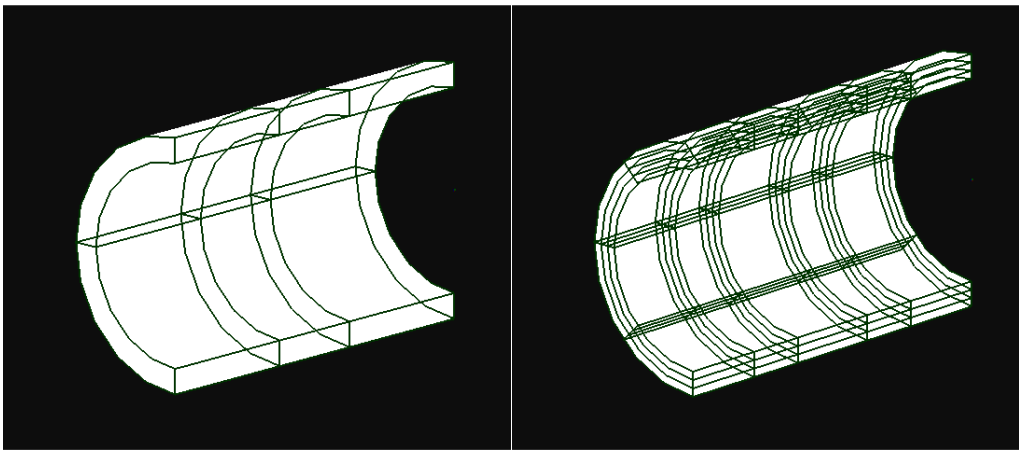


Figure 1.9. Isogeometric analysis with Geomiso.
(geomiso.com)



(a) NURBS model



(b) Refinement

Figure 1.10. The half part of a tyre. Refinement with Geomiso.
(geomiso.com)

2 Isogeometric Analysis

2.1 Introduction

Until to date, engineers had not realized that by designing a model, they simultaneously created its corresponding finite element mesh. This information, although not necessary for designers, is a revolutionary remark for the engineers. Before the isogeometric method, engineers had to create a new approximate finite element mesh instead of taking advantage of the existing exact one. As a result, geometric errors are introduced, which make the process both less accurate and more time-consuming.

The main idea is the elimination of the mesh generation. The role and properties of the node mesh are divided into two separate meshes, obtained directly from the geometrical representation.

- The **control net**, which defines the geometry and the finite number of degrees of freedom that form the problem equation.

- The **knot mesh**, which defines the appropriate discretization for the numerical integration.

For the scope of this thesis, I have worked exclusively with **Non-Uniform Rational B-SPLines** (NURBS), as they are the most popular computational geometry technology. Despite the fact that quite more advanced SPLines have emerged, CAD industry still invests in NURBS. Since 1970, billions of dollars have been invested, establishing them as a common tool for computer-aided design. Both designers and engineers still use NURBS despite their disadvantages, such as difficulties as far as the patch connection and the local refinement is concerned. The reason for this is that NURBS are not only much more simple in their formulation and use, but also can exactly represent all conic sections (circles, cylinders, spheres, ellipsoids, etc.)

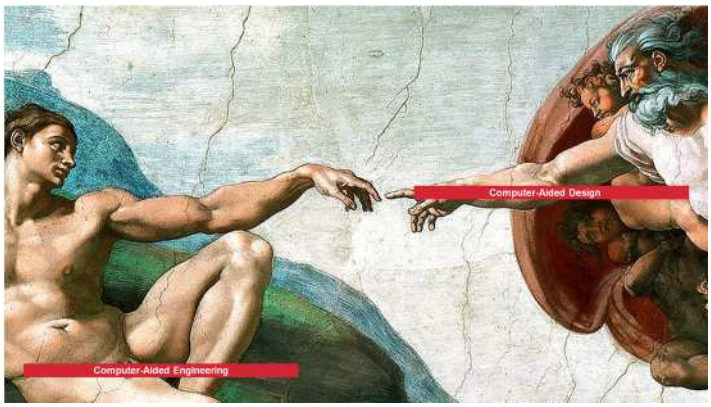
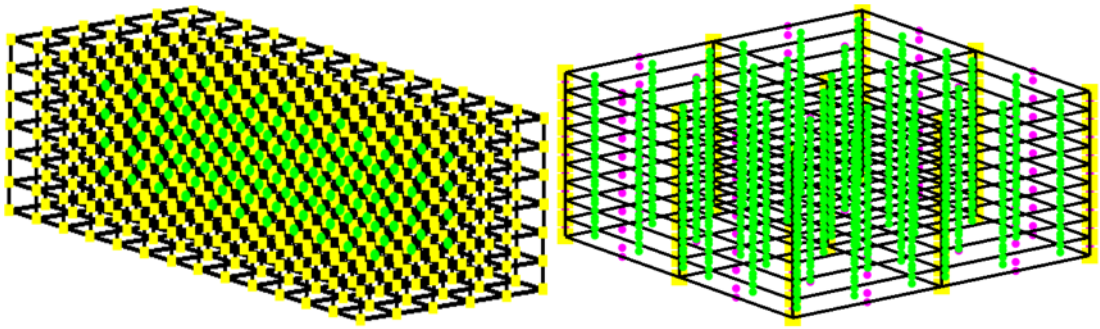


Figure 2.1. Full integration of CAD and CAE.

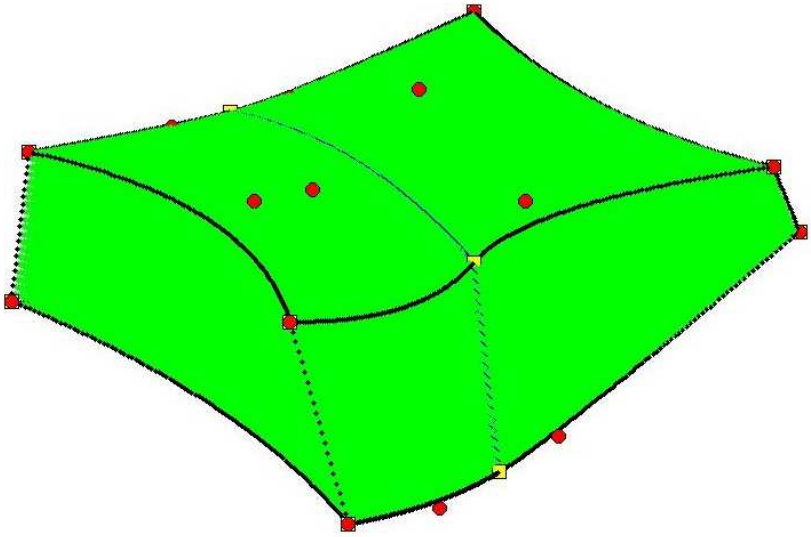
2.2 Index, Parameter & Physical Space

In most occasions, the exact solution of an engineering problem is neither possible nor necessary. The actual objective is to find an accurate solution that satisfies a selected convergence criterion. The ultimate challenge for an engineer is to balance between accuracy and time.

Design and analysis of complex geometries is a powerful asset for modern engineers, who are capable of facing surprisingly more complicated problems. The geometry design is conducted in the Cartesian coordinate system, the so-called **physical space**. It is useful to represent the NURBS model in the **parameter space**, where 1D, 2D, 3D models are represented as lines, rectangles and rectangular parallelepipeds. Parameter space is already known from the isoparametric concept of finite element method. In the case of the isogeometric method, the parameter space shows major differences. Additionally, there is an additional space, the **index space**, which plays an important role for T-SPLines, but it is only auxiliary for NURBS.



(a) Index space **(b)** Parameter space



(c) Physical space

Figure 2.2. NURBS solid.
(geomiso.com)

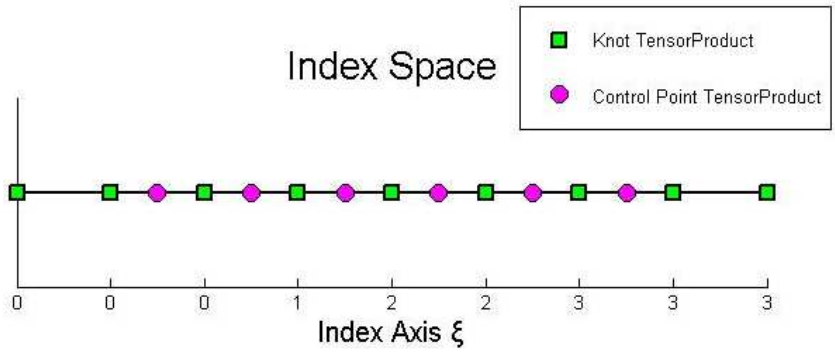
2.2.1 Index space

In **index space**, control points are defined as the center of their support. The support of each control point is its domain of influence consisting of p+1 knot value spans for 1D, $(p+1)^2$ rectangles for 2D and $(p+1)^3$ cuboids for 3D. Index space indicates the:

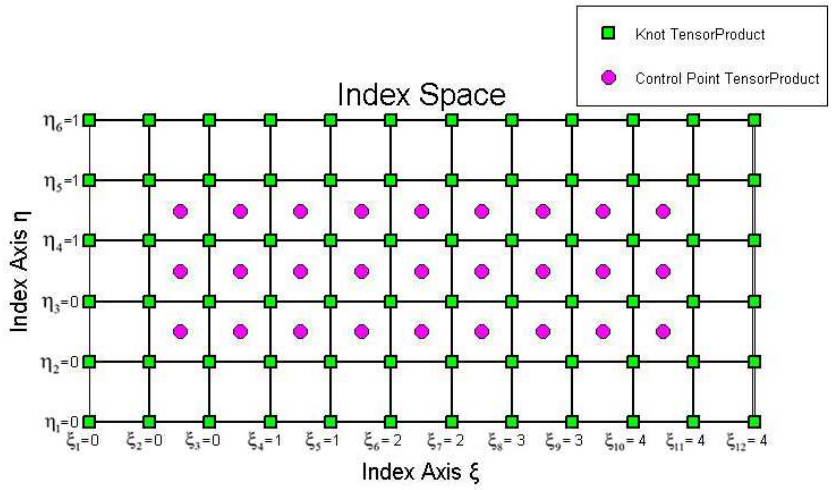
- interconnection between the control points
- support of a basis function
- contribution of a knot value to the basis
- overlapping between the finite elements

1D, 2D, 3D models are represented as lines, rectangles and rectangular parallelepipeds respectively, while knots are equally spaced regardless their value.

Expansion to 2D or 3D leads to the creation of rectangles or cuboids respectively. Due to full tensor product nature, everything mentioned about 1D extends and applies to both 2D and 3D. Thus, index space provides data necessary for the comprehension of the NURBS model.



(a) Curve.

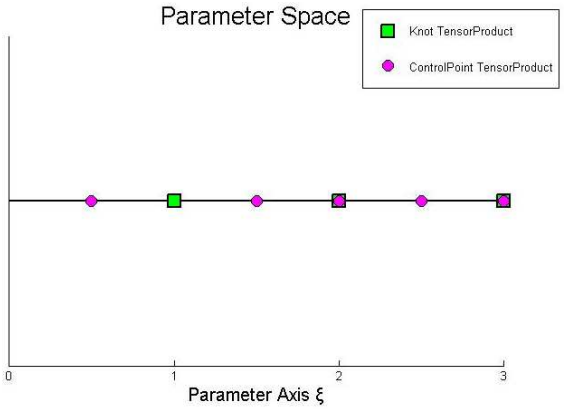


(b) Surface.

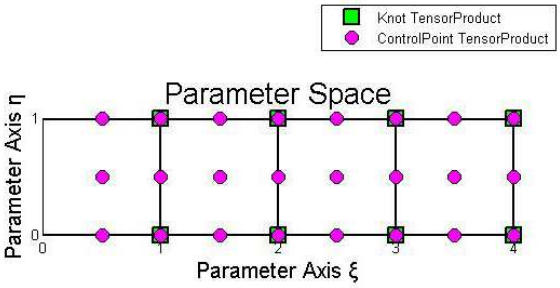
Figure 2.3. NURBS model in index space.

2.2.2 Parameter space

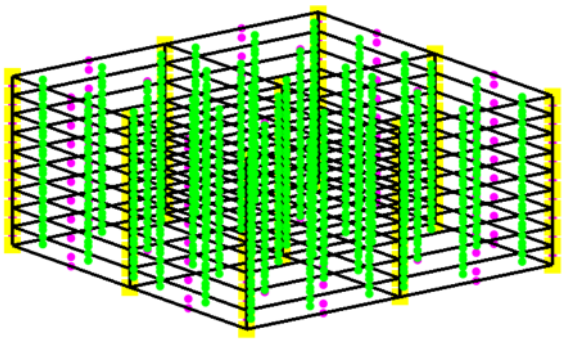
Parameter space is the representation of the model with respect to knots. SPLine entities are represented as lines, rectangles and rectangular parallelepipeds, which are transformed into real geometries in physical space through the Jacobian transformation. This mapping is already known from FEM. In the 1D case, each knot corresponds to the starting point of a specific knot span and to the ending point of the next one. Support is the domain, in which the basis function has non-zero value. Basis functions with intersected supports overlap in parameter space and control a shared domain of the model in physical space.



(a) Curve.



(b) Surface.



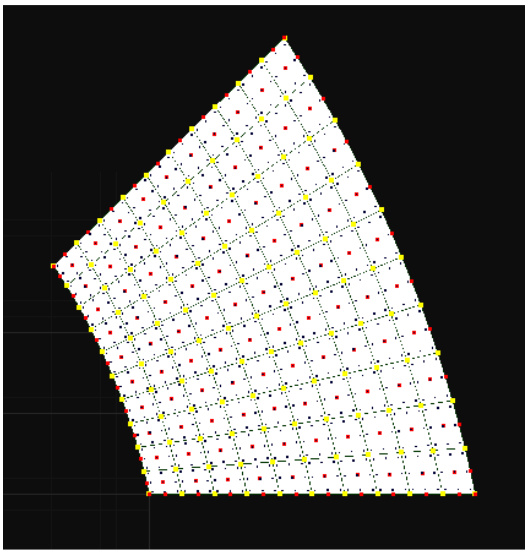
(c) Solid.

Figure 2.4. NURBS model in parameter space.

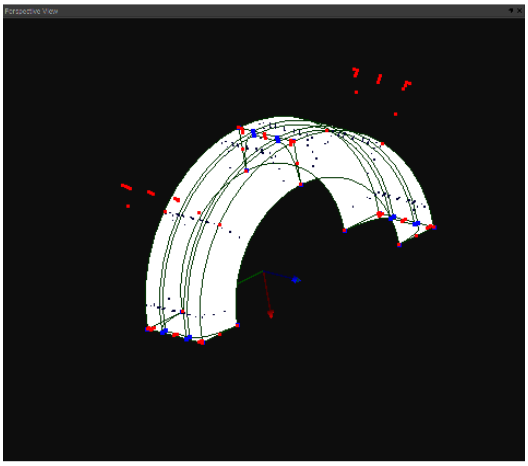
2.2.3 Physical space

Physical space, the well-known Cartesian space, is the one where the real geometry of the examined structure is represented. The lines, rectangles and rectangular parallelepipeds of the parameter space are transformed into real geometries in physical space. Both the control point variables (physical coordinates, weights) and the basis play a significant role in this mapping. It is worth mentioning that for a given set of control points, only a single set of basis functions leaves the geometry intact.

Control points are often located outside the model. This is why they are not material points and don't belong to the model in contrast to FEM's nodes, which always belong to the mesh. Due to this, isogeometric analysis is able to utilize the existing finite element mesh of the NURBS model, something impossible in the case of FEA.



(a) Surface.



(b) Solid.

Figure 2.5. NURBS model in physical space.

2.3 B-SPLines

2.3.1 Basis function

Given a sequence of non-decreasing numbers

$$\Xi = \left\{ \xi_1 \quad \xi_2 \quad \dots \quad \xi_{n+p} \quad \xi_{n+p+1} \right\}$$

we can evaluate the value of basis functions at $\xi \in \left[\xi_1, \xi_{n+p+1} \right]$ using the following Cox-de Boor recursive formula.

First, for degree p=0 (piecewise constant, box BSPLine)

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

The box function does not include the right edge ξ_{i+1} in order to ensure partition of unity, as the following basis function begins at that edge. The last basis function, however, includes both left and right edge, in order to be defined for the whole knot span.

$$N_{n+p,0}(\xi) = \begin{cases} 1, & \text{if } \xi_{n+p} \le \xi \le \xi_{n+p+1} \\ 0, & \text{otherwise} \end{cases}$$

For degree p=1,2,…:

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

with the assumption of $\frac{0}{0} \doteq 0$.

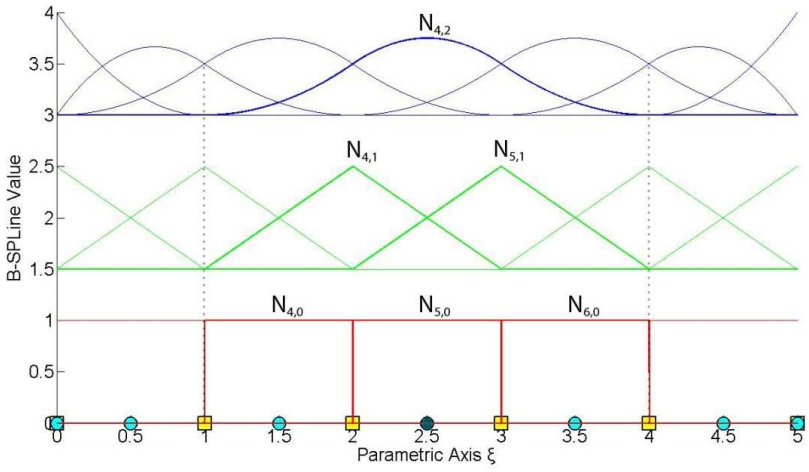


Figure 2.6. Quadratic basis. Recursive formula.

2.3.2 Knot vector

Knots define the boundaries of the basis. A knot vector Ξ is usually defined as a set of non-decreasing parametric coordinates ξ_i, with $\xi_i \leq \xi_{i+1}$. A value can be repeated multiple times, affecting the continuity of the basis. Let p the degree of the basis. If the first and the last knots are repeated p+1 times, the knot value vector is **open**. In the case of an open knot value vector, the continuity is C^{-1} at the edges. An open curve is interpolatory at these knots. If knot values are equally spaced, the knot value vector is **uniform**. A knot value vector may contain integers or decimals. What really matters is not the numerical value, but the relative relation between them.

2.3.3 Control point

Control points are defined as the centers of their support in the index space. Recall that for the i^{th} basis function of order p, the support is $[\xi_i, \xi_{i+p+1})$. The support contains p+1 knot value spans, therefore p+2 knot values (including the right boundary value ξ_{i+p+1}).

2.3.4 Full tensor product

B-SPLine basis functions have a full-tensor product nature. 2D shape functions are defined as the full tensor product of the corresponding 1D basis functions $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$. 2D/ 3D shape functions are defined as the full tensor product of the corresponding 1D basis functions $N_{i,p}(\xi)$, $M_{j,q}(\eta)$ and $L_{k,r}(\zeta)$.

$$R_{i,j}^{p,q}(\xi,\eta) = N_{i,p}(\xi) \cdot M_{j,q}(\eta)$$

$$R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) = N_{i,p}(\xi) \cdot M_{j,q}(\eta) \cdot L_{k,r}(\zeta)$$

2.3.5 Properties

1. **Local support**: $N_{i,p}(\xi) = 0 \ \forall \xi \notin \left[\xi_i, \xi_{i+p+1}\right)$
2. In any given knot span, **at most p+1** functions of order p are **non-zero**.
3. **Non-negativity**: $N_{i,p}(\xi) \geq 0, \ \forall \ \xi, i, p$
4. **Partition of unity**: $\sum_{i=1}^{n} N_{i,p}(\xi) = 1, \forall \ \xi, p$
5. **C^{p-m}**-continuity across knots with multiplicity m.
6. $N_{i,p}(\xi)$ has exactly one maximum value, except for p=0.
7. A non-periodic knot value vector (n functions, order p) has **n+p+1** knot values.
8. Every B-SPLine basis function **shares support with other 2p** B-SPLines.

2.3.5.1 Local support

The basis functions are non-zero in certain knot spans in parameter space.

$$N_{i,p}(\xi) = 0 \quad \forall \xi \notin \left[\xi_i, \xi_{i+p+1}\right)$$

Local support comes from the recursive nature of basis functions. For the creation of a B-SPLine basis function of degree p, two consecutive functions of order p-1 are used. For the creation of those functions, three consecutive ones of order p-2 are needed. Inductively, p+1 consecutive box basis functions are required.

Each box function has a support of one knot span. As a result, the support of the final basis function is defined by the union of the supports of the box functions, hence p+1 consecutive knot spans.

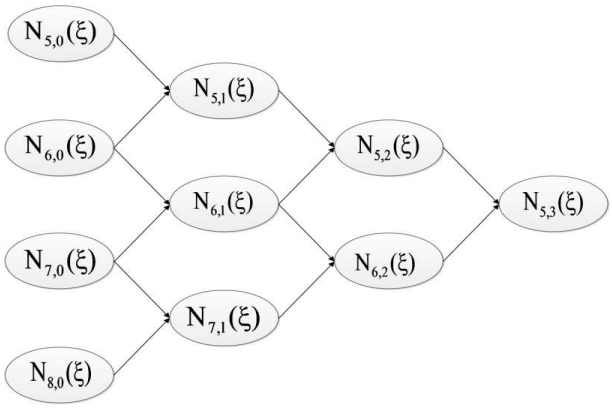


Figure 2.7. Lower-order basis functions required for $N_{5,3}(\xi)$.

In **Figure 2.8**, the recursive character of B-SPLines is depicted. The box functions, drawn in red, are required in order to build the linear basis functions, drawn in green. Linear functions are combined in order to define the quadratic (p=2) ones. It's worth mentioning that there are linear and box functions being zero across the entire domain. Despite that, these functons still contribute to the higher order B-SPLines.

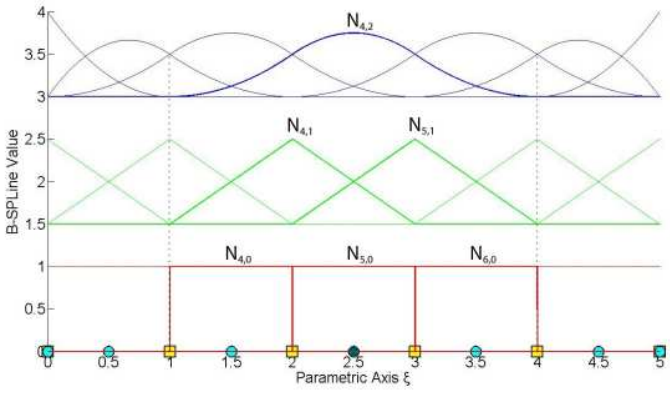


Figure 2.8. Quadratic basis. Recursive formula.

For example, $N_{4,2}(\xi)$ in **Figure 2.8** is defined by $N_{4,1}(\xi)$ and $N_{5,1}(\xi)$, which, in turn, are evaluated from $N_{4,0}(\xi)$, $N_{5,0}(\xi)$ and $N_{5,0}(\xi)$, $N_{6,0}(\xi)$ respectively. The corresponding box functions are non-zero in the knot spans [1,2), [2,3) and [2,3), [3,4), thus creating the support [1,4) of $N_{4,2}(\xi)$.

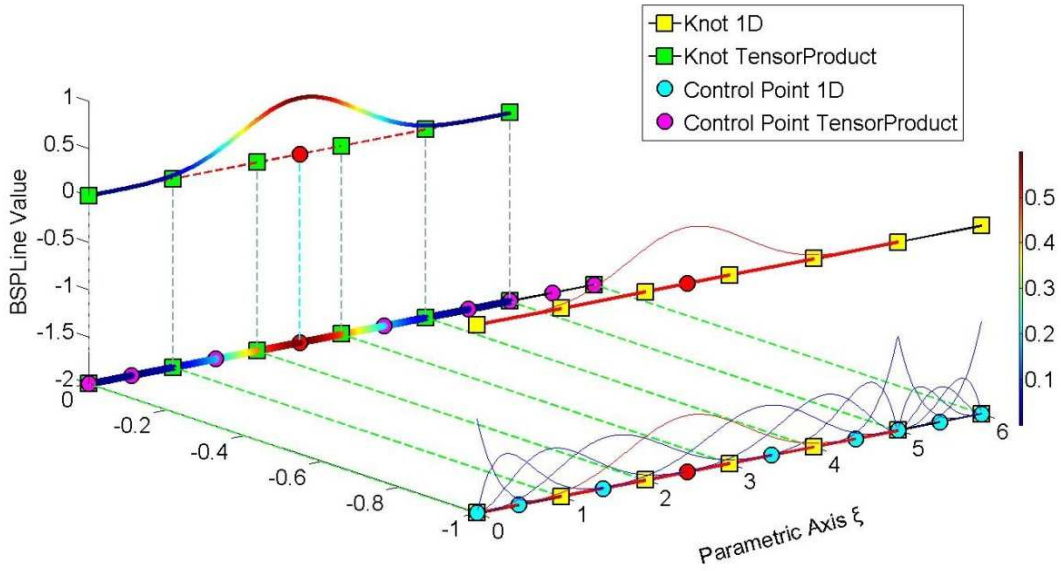


Figure 2.9. Quadric basis. Shape function $N_{5,4}(\xi)$.
Ξ={0,0,0,0,0,1,2,3,4,5,5,5,5,6,6,6,6,6}

In **Figure 2.9**, the degree is equal to p=4, so each quadric basis function has a support of p+1=4+1=5 knot value spans.

The function $N_{5,4}(\xi)$, drawn in red, is non-zero only across the knot value spans [0,1), [1,2), [2,3), [3,4) and [4,5).

The p=2=6 knot values that define this function are {0,1,2,3,4,5}, presented in green.

There are no trivial spans affected this basis function.

Bear in mind that knot values can be repeated, thus creating trivial spans.

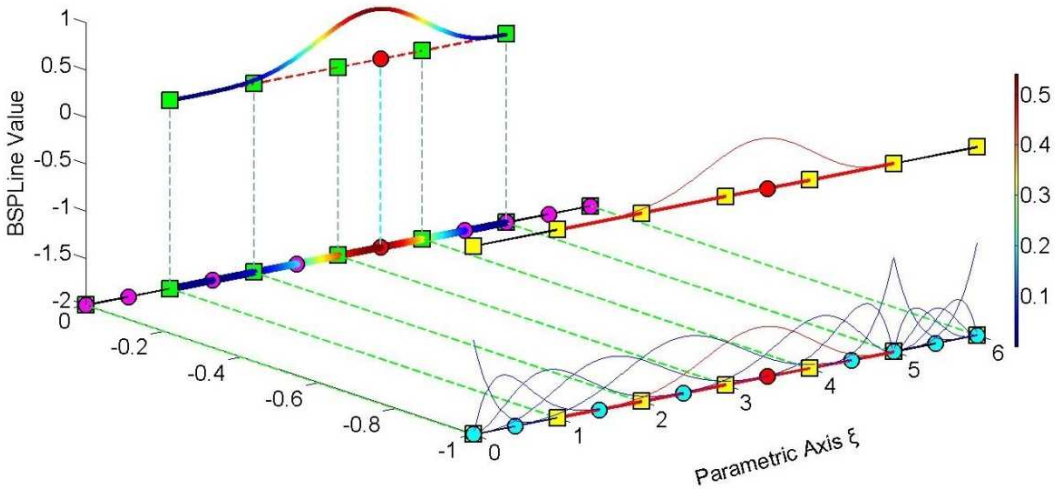


Figure 2.10. Quadric basis. Shape function $N_{6,4}(\xi)$.
Ξ={0,0,0,0,0,1,2,3,4,5,5,5,5,6,6,6,6,6}.

Figure 2.10 depicts the shape function $N_{6,4}(\xi)$, which has non-zero value across p+1=5 knot value spans, means [1,2), [2,3), [3,4), [4,5) and [5,5). The corresponding knot values, shown in green, are {1,2,3,4,5,5}. The support contains five knot value spans, which correspond to four knot spans, as there is one trivial.

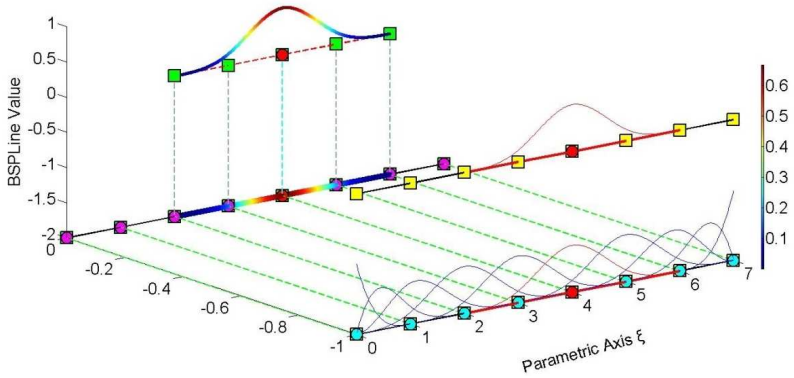


Figure 2.11. Cubic basis. Basis function $N_{6,3}(\xi)$.
Ξ={0,0,0,0,1,2,3,4,5,6,7,7,7,7}

$N_{6,3}(\xi)$ has non-zero value across the knot value spans [2,3), [3,4), [4,5) and [5,6). There are no trivial spans. Its support contains the knot values {2,3,4,5,6}. Full tensor product nature allows the expansion of 1D properties to 2D and 3D. $N_{3,2}(\xi)$ (**Figure 2.12**) has support of p+1=3 knot value spans, means [0,1), [1,2) and [2,3). It contains the knot values {0,1,2,3}. $M_{4,2}(\eta)$ has support of 3 knot value spans, means [1,2), [2,3) and [3,4). It corresponds to the knot values {1,2,3,4}.

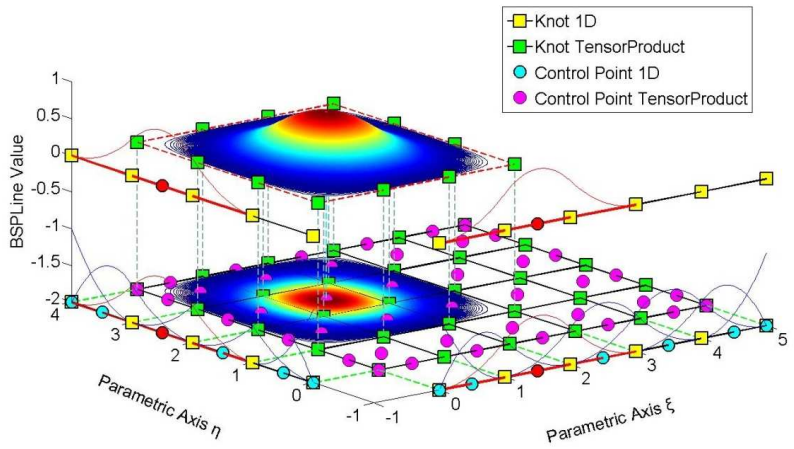


Figure 2.12. Shape function $R_{3,4}^{2,2}(\xi,\eta)$ as a tensor product of $N_{3,2}(\xi)$ and $N_{4,2}(\eta)$.
Ξ={0,0,0,1,2,3,4,5,5,5}, H={0,0,0,1,2,3,4,4,4}

The 2D shape function $R_{3,4}^{2,2}(\xi,\eta)$ is the full tensor product of $N_{3,2}(\xi)$ and $M_{4,2}(\eta)$. Its support is a full tensor product of the 1D supports of the corresponding basis functions and contains $(p+1)^2$=9 knot value rectangles.

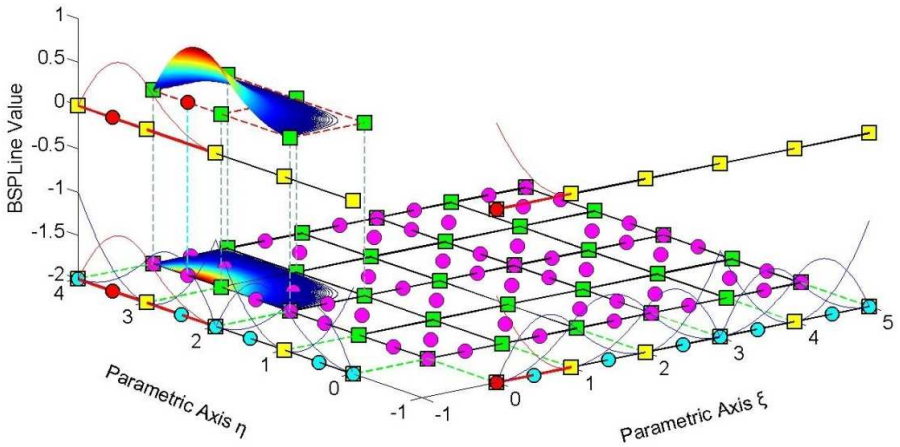


Figure 2.13. Shape function $R_{1,6}^{2,2}(\xi,\eta)$ as a tensor product of $N_{1,2}(\xi)$ and $N_{6,2}(\eta)$. Ξ={0,0,0,1,2,3,3,4,5,5,5}, H={0,0,0,1,2,2,3,4,4,4}

Both 1D basis functions have reduced continuity in **Figure 2.11**. $N_{1,2}(\xi)$ is affected by the knot value spans [0,0), [0,0) and [0,1), thus its support contains a single knot span, the [0,1). The corresponding knot values are {0,0,0,1}. $M_{6,2}(\eta)$ are affected by the knot values {2,3,4,4} and its support contains the spans [2,3), [3,4), [4,4). There are one trivial and two non-trivial spans. The support of the shape function $R_{1,6}^{2,2}(\xi,\eta)$ is defined as full tensor product of the supports of the corresponding basis functions, thus contains 1*2=2 knot rectangles.

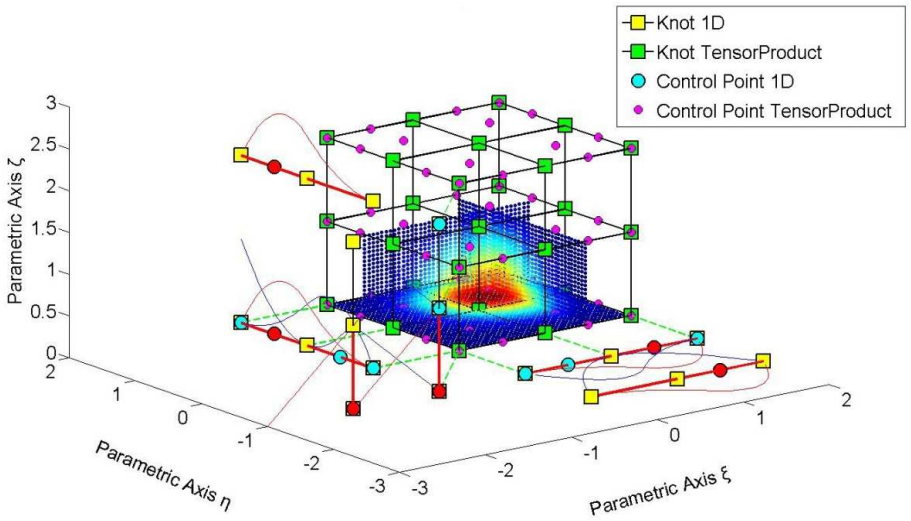


Figure 2.14. Shape function $R_{3,3,1}^{2,2,1}(\xi,\eta,\zeta)$ as tensor product of $N_{3,2}(\xi)$, $M_{3,2}(\eta)$, $L_{1,1}(\zeta)$. Ξ={0,0,0,1,2,2,2}, H={0,0,0,1,2,2,2}, Z={0,0,0,1,2,2}

Figure 2.14 depicts the three parametric axes ξ, η and ζ. Basis functions for ξ are presented in the ξ-η plane, for η in the η-ζ plane and for ζ in the ζ-ξ plane. $N_{3,2}(\xi)$ and $M_{3,2}(\eta)$ have a support of 3 knot value spans ([0,1), [1,2) and [2,2)). These supports correspond to the knot values {0,1,2,2}. $L_{1,1}(\zeta)$ has support of a single span.

Their full tensor product is calculated as a function of two 1D basis functions (control point of the third direction). The contour is projected on a plane parallel to the two directions and intersecting with this specific point. This process is repeated for all the combinations, thus creating contours in ξ-η, η-ζ and ζ-ξ planes.

For example, control point (3,3,1) has coordinates (ξ,η,ζ)=(1.5,1.5,0).

$R^{2,2,1}_{3,3,1}(\xi,\eta,0)$ is calculated and projected in ζ=0 plane. The support of the projection shows the support of this function per ξ, η, namely 2*2=4 knot rectangles.

$R^{2,2,1}_{3,3,1}(\xi,1.5,\zeta)$ is represented in the plane η=1.5. Its support per ξ, ζ contains 2*1=2 knot rectangles.

$R^{2,2,1}_{3,3,1}(1.5,\eta,\zeta)$ is projected in ξ=1.5 plane. Its support per η, ζ contains 2*1=2 knot rectangles.

The support of the 3D shape function across the entire domain is the tensor product of the corresponding 1D ones. In this case, it is about 2*2*1=4 knot cuboids. The support comes from the tensor product of the projections in **Figure 2.14**.

2.3.5.2 Non-zero functions per knot span

A box function, that has non-zero values across one knot span, contributes to the two consecutive B-SPLine basis functions of order p=1. These two functions lead to the creation of three consecutive basis functions of order p=2. Inductively, a box function contributes to the creation of p+1 basis functions of order p.

According to Cox de Boor recursive formula, only one box function has non-zero values across a certain knot span. As a result, only the corresponding p+1 basis functions of order p can be non-zero in that specific knot span.

Therefore, at a non-trivial knot value span:

$$[\xi_i, \xi_{i+1})$$

only the basis functions

$$N_{i-p,p}(\xi),\ N_{i-p+1,p}(\xi),\ ...,\ N_{i,p}(\xi)$$

are non-zero.

This is used efficiently for the formulation of the total stiffness matrix, in order to reduce the computational cost.

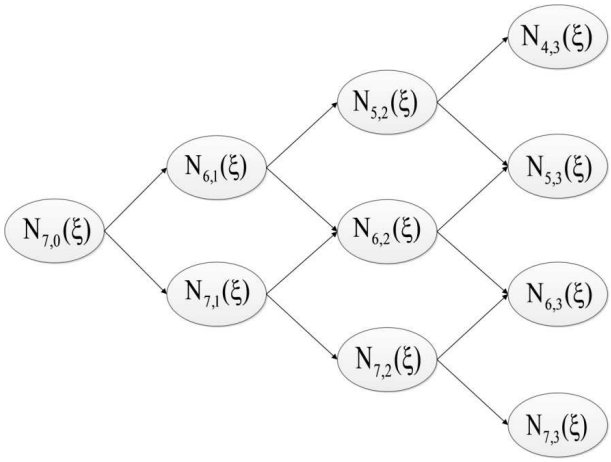


Figure 2.15. From a box function to higher-order ones.

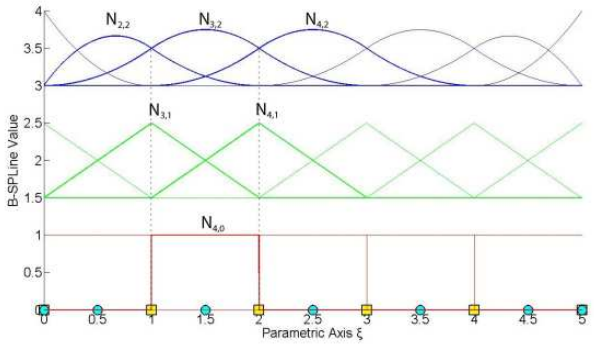


Figure 2.16. Contribution of a box function to non-zero higher-order ones.
Across knot span [1,2).

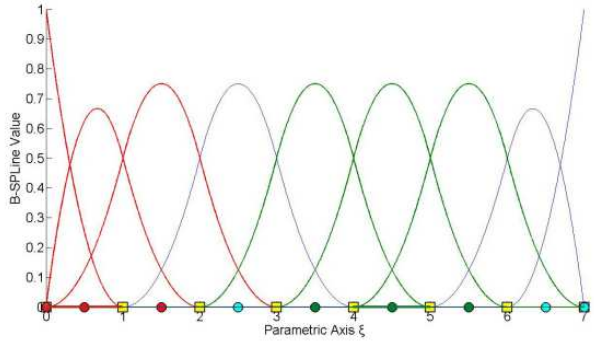


Figure 2.17. Quadratic basis.
Ξ={0,0,0,1,2,3,4,5,6,7,7,7}
Basis functions, that are non-zero across knot span [0,1) & [4,5), are shown in red & green respectively.

In **Figure 2.15**, two knot spans are investigated. Across each one, p+1=3 basis functions has non-zero value. For span [0,1)=[ξ_3,ξ_4), basis functions $N_{1,p}(\xi)$, $N_{2,p}(\xi)$, $N_{3,p}(\xi)$ are non-zero, while for [4,5)=[ξ_7,ξ_8), $N_{5,p}(\xi)$, $N_{6,p}(\xi)$, $N_{7,p}(\xi)$ are non-zero. As a result, for 2D problems, across a given knot rectangle, the tensor product of the non-zero basis functions defines (p+1)x(q+1) shape functions. For 3D, across a specific knot cuboid, the tensor product of the non-zero basis functions defines (p+1)x(q+1)x(r+1) non-zero shape functions.

2.3.5.3 Non-negativity

$$N_{i,p}(\xi) \geq 0, \ \forall \xi, i, p$$

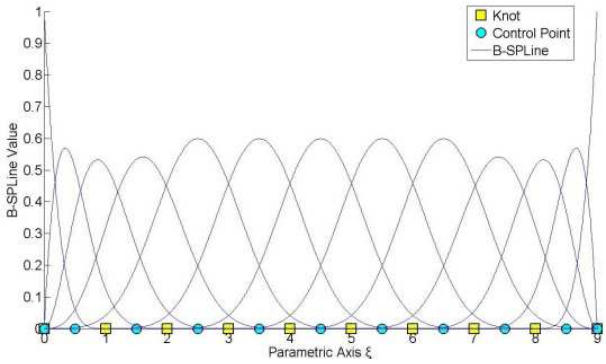


Figure 2.18. Quadric basis.
Ξ={0,0,0,0,0,1,2,3,4,5,6,7,8,9,9,9,9,9}

2.3.5.4 Partition of unity

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

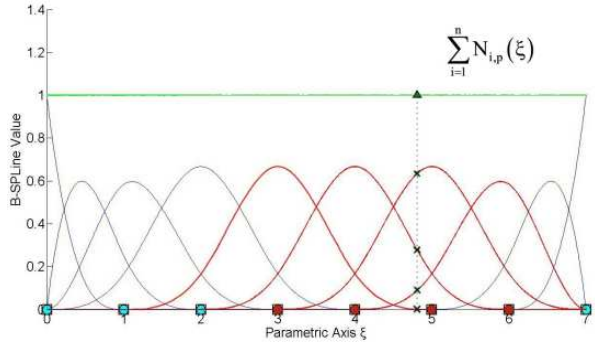


Figure 2.19. Cubic basis. Partition of unity.
Ξ={0,0,0,0,1,2,3,4,5,6,7,7,7,7}

The green horizontal line represents the sum of the basis functions' value, which is equal to 1 for every ξ. Partition of unity also applies for multivariate shape functions as well.

2D

$$\sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q} = \sum_{i=1}^{n}\sum_{j=1}^{m} \left\{ N_{i,p}(\xi) \cdot M_{j,q}(\eta) \right\} = 1$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q} = \sum_{i=1}^{n}\sum_{j=1}^{m} \left\{ N_{i,p}(\xi) \cdot M_{j,q}(\eta) \right\} = \left(\sum_{i=1}^{n} N_{i,p}(\xi) \right) \cdot \left(\sum_{j=1}^{m} M_{j,q}(\eta) \right) = 1$$

3D

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} R_{i,j,k}^{p,q,r} = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} \left\{ N_{i,p}(\xi) \cdot M_{j,q}(\eta) \cdot L_{k,r}(\zeta) \right\} = 1$$

2.3.5.5 C^{p-m} continuity

At a knot with multiplicity m, $N_{i,p}(\xi)$ has C^{p-m} continuity, means that there are p-m continuous derivatives. Continuity less than C^0 is not possible for internal knots, which means they can be repeated up to p times. As continuity decreases, basis functions have increased overlapping.

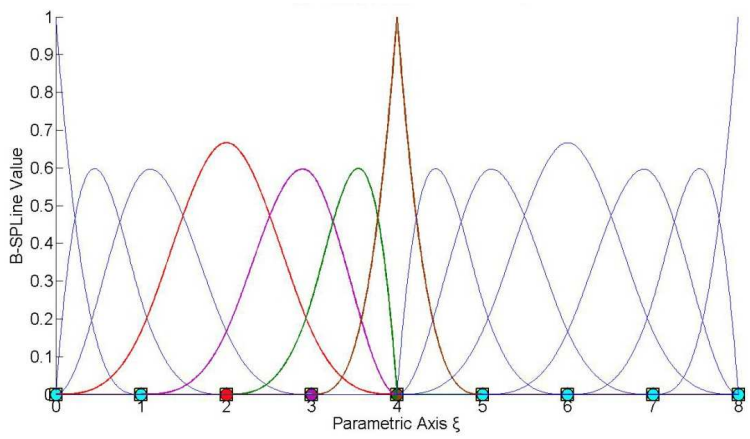


Figure 2.20. Cubic basis.
Ξ={0,0,0,0,1,2,3,4,4,4,5,6,7,8,8,8,8}

Continuity of a basis depends on both the polynomial order and the knot value multiplicity.

The polynomial order of the basis, depicted in **Figure 2.20**, is equal to p=3. $N_{4,3}(\xi)$, drawn in red, has a knot value support of {0,1,2,3,4}. The knot ξ=4 has multiplicity m=3, so the basis $N_{4,3}(\xi)$ is $C^{p-m}=C^{3-3}=C^0$ continuous at it. The purple basis function $N_{5,3}(\xi)$ has a support of {1,2,3,4,4}. Knot value support for $N_{6,3}(\xi)$ and $N_{7,3}(\xi)$ is {2,3,4,4,4,} and {3,4,4,4,5} respectively.

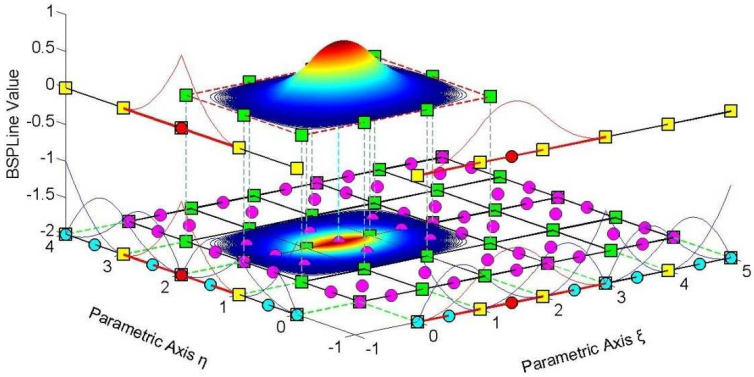
In 2D and 3D problems, continuity is inherited directly from the continuity of the corresponding univariate basis functions.

In **Figure 2.21.a**, $R^{2,2}_{3,4}(\xi,\eta)$ is the full tensor product of $N_{3,2}(\xi)$ and $M_{4,2}(\eta)$. $N_{3,2}(\xi)$ has $C^{p-m}=C^{2-2}=C^0$ continuity at ξ=3 and $M_{4,2}(\eta)$ has $C^{p-m}=C^{2-2}=C^0$ continuity at η=2. Therefore, $R^{2,2}_{3,4}(\xi,\eta)$ has C^0 continuity at (ξ,η)=(3,2).

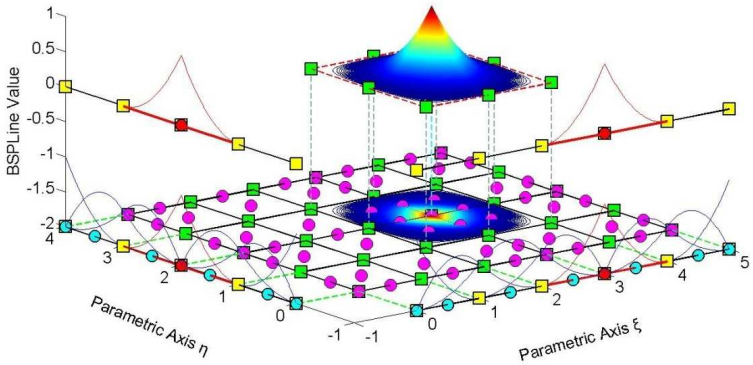
In **Figure 2.21.b**, $R^{2,2}_{5,4}(\xi,\eta)$ is the tensor product of $N_{5,2}(\xi)$ and $M_{4,2}(\eta)$. Both basis functions have C^0-continuity across ξ=3, η=2 respectively. As a result, $R^{2,2}_{5,4}(\xi,\eta)$ has C^0-continuity at (ξ,η)=(3,2).

Non-negativity and partition of unity properties require that:

$$R^{2,2}_{i,j}(3,2)=0\,,\ \forall(i,j)\neq(5,4)$$



(a) $R_{i,j}^{p,q} = R_{3,4}^{2,2}(\xi,\eta)$, the tensor product of $N_{3,2}(\xi)$ & $M_{4,2}(\eta)$



(b) $R_{i,j}^{p,q} = R_{5,4}^{2,2}(\xi,\eta)$, the tensor product of $N_{5,2}(\xi)$ and $M_{4,2}(\eta)$

Figure 2.21. Biviriate shape functions. Quadratic basis.
$\Xi = \{0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 3\ \ 3\ \ 4\ \ 5\ \ 5\ \ 5\}$, $H = \{0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 2\ \ 3\ \ 4\ \ 4\ \ 4\}$

Figure 2.22 depicts the trivariate shape function $R_{4,3,3}^{2,2,2}(\xi,\eta,\zeta)$, the full tensor product of three C^0-continuous univariate basis functions across (2,1,1). These are the $N_{4,2}(\xi)$, $M_{3,2}(\eta)$ and $L_{3,2}(\xi)$.

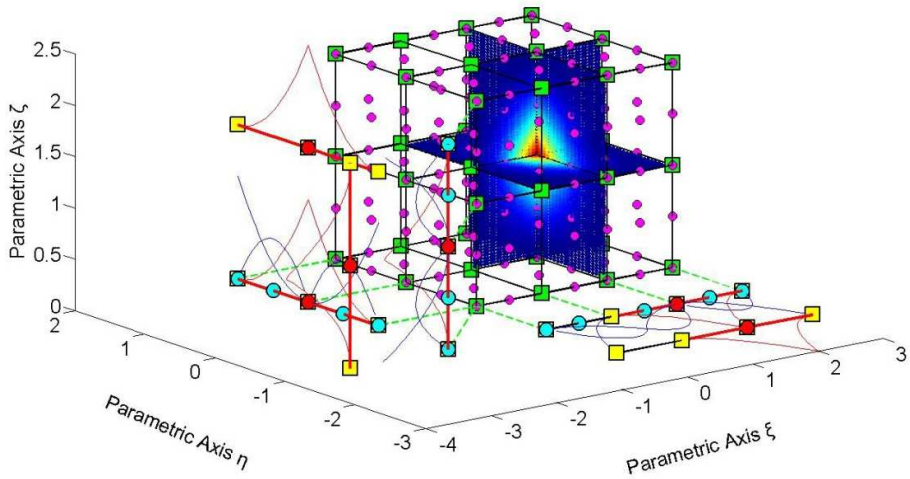


Figure 2.22. Trivariate shape functions. Quadratic basis.
$R_{4,3,3}^{2,2,2}(\xi,\eta,\zeta)$, Ξ={0,0,0,1,2,2,3,3,3}, H={0,0,0,1,1,2,2,2}, Z={0,0,0,1,1,2,2,2}

2.3.5.6 Linear independence

Given a finite number of distinct vectors $\{u_1, u_2, \dots u_n\}$ and scalars $\{a_1, a_2, \dots a_n\}$, the subset S of a vector space V is **linearly independent**, if the equation

$$a_1 \cdot u_1 + a_2 \cdot u_2 + ... + a_n \cdot u_n = 0$$

has unique solution $a_1=a_2=\dots=a_n=0$.

Thus, the subset is linearly independent, if a linear combination of the vectors is the zero vector, only for $a_1=a_2=\dots=a_n=0$.

The linear independence in the isogeometric method applies for the basis $\{N_{1,p}(\xi), N_{2,p}(\xi), \dots N_{n,p}(\xi)\}$.

$$a_1 \cdot N_{1,p}(\xi) + a_2 \cdot N_{2,p}(\xi) + ... + a_n \cdot N_{n,p}(\xi) = 0 \Rightarrow a_1 = a_2 = ... = a_n = 0$$

No B-SPLine basis function can be expressed as a linear combination of the other B-SPLine basis functions.

These linearly independent vectors form a basis for the vector space V. Interesting attributes of such vectors include:

- The basis of the vector space V can be formed by different sets of linearly independent vectors. Any set can be used, provided that the vectors are linearly independent and all the properties above apply for every vector. Thus, in isogeometric analysis, we can choose different sets of basis functions in order to represent the vector space.

- The number of vectors of any basis chosen is equal to the dimension of V, often represented as dim(V). In the isogeometric method, dim(V) is the control points' number.

- Let's assume a set of vectors $\{u_1, u_2, \dots, u_n\}$, which form the basis of a vector space with dim(V)=n and the numbers$\{a_1, a_2, \dots a_n\}$, called the coordinates of u. In the isogeometric method, the basis of the vector space is the set of B-SPLine basis functions $\{N_{1,p}(\xi), N_{2,p}(\xi), \dots N_{n,p}(\xi)\}$ and the numbers are the coordinates$\{X_1, X_2, \dots, X_n\}$ of the control points of the curve. A random vector u of the vector space V can be represented by the equation:

  $$u = a_1u_1 + a_2u_2 + ... + a_nu_n$$

  which, for isogeometric analysis, applies as:

  $$C(\xi) = X_1 \cdot N_{1,p}(\xi) + X_2 \cdot N_{2,p}(\xi) + ... + X_n \cdot N_{n,p}(\xi)$$

Any vector in V can be described as a linear combination of the basis. The numbers $\{a_1 \quad a_2 \quad ... \quad a_n\}$ are the coordinates of u and they are unique for this specific u and this specific basis.

Vector space V in $\mathbb{R}$ is a set of vectors with the following properties.

1. Commutativity of addition:

$$u + v = v + u \ , \ \forall u, v \in V$$

   In IGA, this property applies as

$$C_1(\xi) + C_2(\xi) = C_2(\xi) + C_1(\xi)$$

2. Associativity of addition:

$$u + (v + w) = (u + v) + w \ , \ \forall u, v, w \in V$$

   Respectively, in IGA,

$$C_1(\xi) + (C_2(\xi) + C_3(\xi)) = (C_1(\xi) + C_2(\xi)) + C_3(\xi)$$

3. Identity element of addition: There is an element $0 \in V$, the zero vector, so that

$$u + 0 = u \ , \ \forall u \in V$$

   In IGA, the equation applies as

$$C(\xi) + 0 = C(\xi)$$

4. Inverse elements of addition: There is an element $-u \in V$, such that

$$u + (-u) = 0 \ , \ \forall u \in V$$

   The -u is called the additive inverse of u. The equivalent in IGA is:

$$C(\xi) + (-C(\xi)) = 0$$

5. Identity element of scalar multiplication: For the real number 1, it applies:

$$1 \cdot u = u \ , \ \forall u \in V$$

   Respectively, in IGA:

$$1 \cdot C(\xi) = C(\xi)$$

6. Distributivity of scalar multiplication with respect to vector addition: $\forall a \in \mathbb{R}$ and $\forall u, v \in V$ applies the relation

$$a \cdot (u + v) = a \cdot u + a \cdot v$$

In IGA:

$$a \cdot (C_1(\xi) + C_2(\xi)) = a \cdot C_1(\xi) + a \cdot C_2(\xi)$$

7. Distributivity of scalar multiplication with respect to field addition: $\forall a, b \in \mathbb{R}$ and $\forall u \in V$ applies the relation

$$(a + b) \cdot u = a \cdot u + b \cdot u$$

In IGA:

$$(a + b) \cdot C(\xi) = a \cdot C(\xi) + b \cdot C(\xi)$$

8. Compatibility of scalar multiplication with field multiplication: $\forall a, b \in \mathbb{R}$ and $\forall u \in V$ applies the equation

$$a \cdot (b \cdot u) = (a \cdot b) \cdot u$$

In IGA:

$$a \cdot (b \cdot C(\xi)) = (a \cdot b) \cdot C(\xi)$$

B-SPLine basis functions are indeed the basis for a vector space with dim(V)=n, where n is the number of control points.

Control points are the coordinates that transform the basis functions at any point in the given physical space. Conclusively,

$$\{N_{1,p}(\xi) \quad N_{2,p}(\xi) \quad ... \quad N_{n,p}(\xi)\}$$

is the basis of the vector space, while

$$\{X_1 \quad X_2 \quad ... \quad X_n\}$$

are the coordinates for a vector C(ξ). The familiar linear combination applies:

$$C(\xi) = X_1 \cdot N_{1,p}(\xi) + X_2 \cdot N_{2,p}(\xi) + ... + X_n \cdot N_{n,p}(\xi)$$

Due to linear independence and vector space properties, for a specific set of basis functions, only one set of control points can define the appropriate geometry.

2.3.5.7 Fixed knot value number

A non-periodic knot value vector, that produces n functions of order p, has $n+p+1$ knot values.

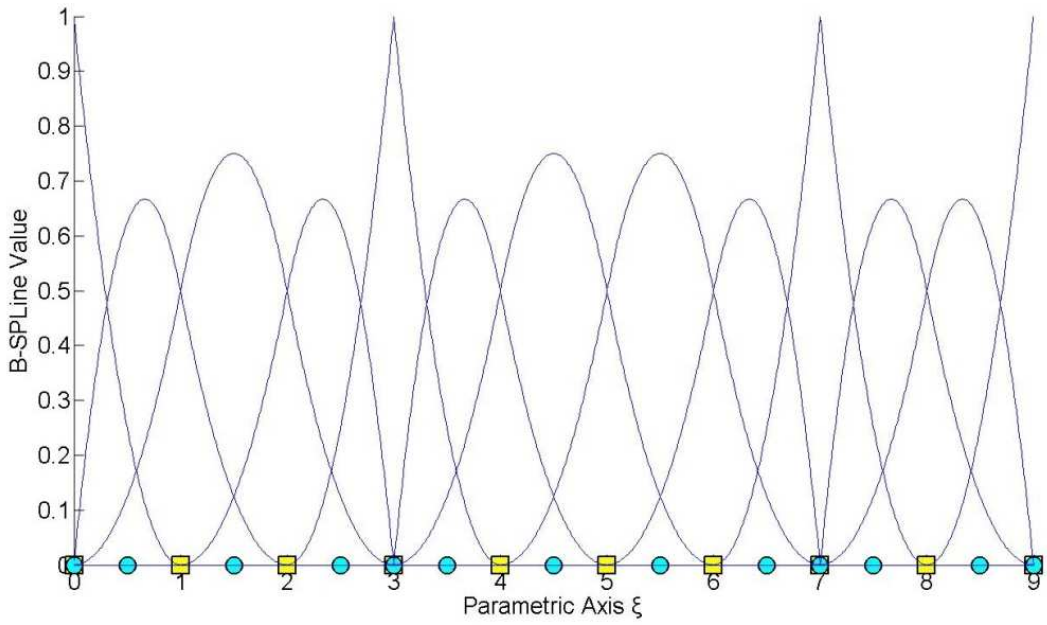


Figure 2.23. Compact support.
$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 7 \quad 8 \quad 9 \quad 9 \quad 9\}$

In **Figure 2.23**, the degree is p=2 (quadratic basis) and the control point number is equal to n=13. The knot value number is $k = n+p+1 = 13+2+1 = 16$ and the knot span number is equal to 9.

In order to have n basis functions of degree p, n+p basis functions of order p=0 are used, hence n+p+1 knot values are needed.

Each control point has a knot value support of p+2 knot values. For n control points, $n \cdot (p+2)$ knot values are needed.

There are (p+1) knot values repeated in (n-1) control point interconnections. The total number of knot values is equal to:

$$n \cdot (p+2) - (n-1) \cdot (p+1) =$$
$$n \cdot p + 2n - n \cdot p - n + p + 1 =$$
$$n + p + 1$$

Therefore, $n+p+1$ knot values are required.

2.3.5.8 Shared support

Each B-SPLine shares support with at most 2p other B-SPLines. Specifically, each basis function shares support with at most p basis functions on each side. This results in higher overlapping, compared to the polynomial shape functions of the finite element method. Basis functions' overlapping leads to interconnectivity between control points as well.

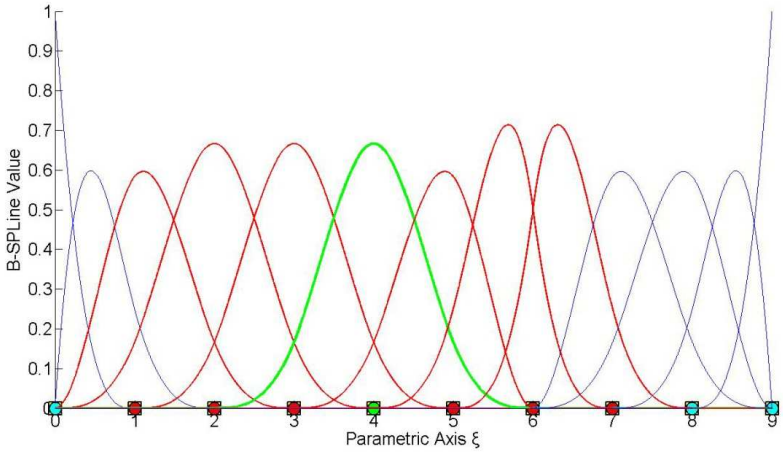


Figure 2.24. Shared support for $N_{6,3}(\xi)$.
$\Xi=\{0,0,0,0,1,2,3,4,5,6,6,7,8,9,9,9,9\}$

$N_{6,3}(\xi)$ interacts with p=3 basis functions on each side, $N_{3,2}(\xi)$, $N_{4,2}(\xi)$ and $N_{5,2}(\xi)$ on the left, $N_{7,2}(\xi)$, $N_{8,2}(\xi)$, $N_{9,2}(\xi)$on the right. As a result, the corresponding stiffness matrix coefficients are non-zero.

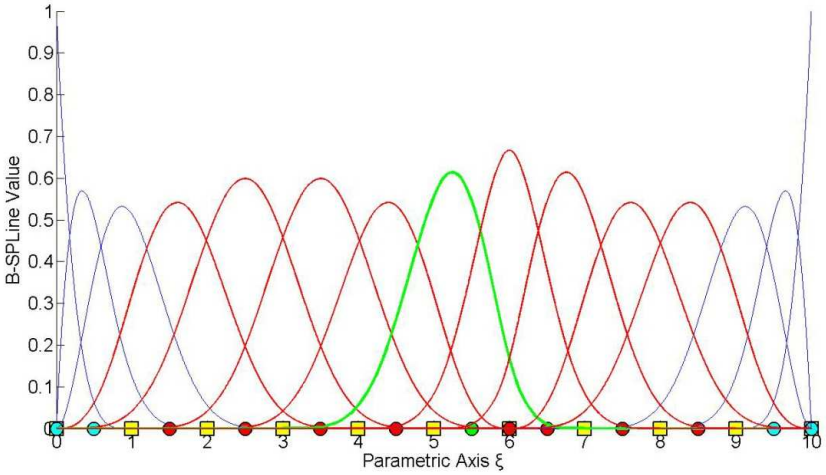


Figure 2.25. Shared support for $N_{8,4}(\xi)$.
$\Xi=\{0,0,0,0,0,1,2,3,4,5,6,6,7,8,9,10,10,10,10,10,\}$

$N_{8,4}(\xi)$ shares support with 2p=8 basis functions, 4 on each side. Greater-order basis functions tend to create more dense stiffness martices and therefore demand more computational power.

2.3.6 Derivatives

Deformation and stiffness matrices depend on the shape functions' derivatives. As a result, and the stress and strain field is based on them. The derivatives of B-SPLines, as defined from the recursive formula, are represented as a linear combination of previous polynomial order basis functions:

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

This leads to a generalized equation for the k^{th} derivative:

$$\frac{d^k}{d\xi^k} N_{i,p}(\xi) = \frac{p!}{(p-k)!} \cdot \sum_{j=0}^{k} \left\{ a_{k,j} \cdot N_{i+j,p-k}(\xi) \right\}$$

$$a_{0,0} = 1$$

$$a_{k,0} = \frac{a_{k-1,0}}{\xi_{i+p-k+1} - \xi_i}$$

$$a_{k,j} = \frac{a_{k-1,j} - a_{k-1,j-1}}{\xi_{i+p+j-k+1} - \xi_{i+j}}, \quad j=1, \ldots, k\text{-}1$$

$$a_{k,k} = \frac{-a_{k-1,k-1}}{\xi_{i+p+1} - \xi_{i+k}}$$

The partial derivatives of 2D shape functions are defined, as follows:

$$\frac{\partial}{\partial \xi} R_{i,j}^{p,q}(\xi,\eta) = \left(\frac{d}{d\xi} N_{i,p}(\xi) \right) \cdot M_{j,q}(\eta)$$

$$\frac{\partial}{\partial \eta} R_{i,j}^{p,q}(\xi,\eta) = N_{i,p}(\xi) \cdot \left(\frac{d}{d\eta} M_{j,q}(\eta) \right)$$

The partial derivatives of 3D shape functions are defined, as follows:

$$\frac{\partial}{\partial \xi} R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) = \left(\frac{d}{d\xi} N_{i,p}(\xi) \right) \cdot M_{j,q}(\eta) \cdot L_{k,r}(\zeta)$$

$$\frac{\partial}{\partial \eta} R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) = N_{i,p}(\xi) \cdot \left(\frac{d}{d\eta} M_{j,q}(\eta) \right) \cdot L_{k,r}(\zeta)$$

$$\frac{\partial}{\partial \zeta} R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) = N_{i,p}(\xi) \cdot M_{j,q}(\eta) \cdot \left(\frac{d}{d\zeta} L_{k,r}(\zeta) \right)$$

2.3.7 Curves, surfaces & solids

Given a knot value vector Ξ and a polynomial order p, the B-SPLine basis functions and the corresponding shape functions are able to be calculated.

In order to define a B-SPLine curve/ surface/ solid, the control point variables $X_i=\{X_i, Y_i, Z_i\}$ are also needed.

Curve

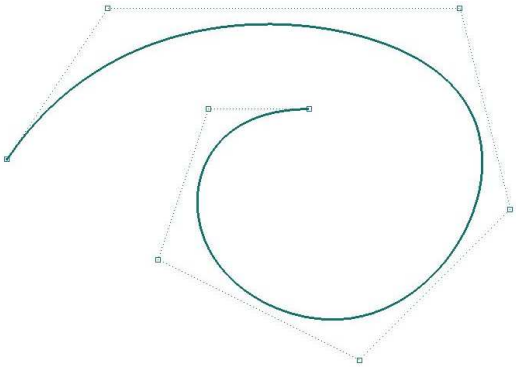
$$C(\xi) = \sum_{i=1}^{n} \left\{ N_{i,p}(\xi) \cdot X_i \right\}$$

Surface

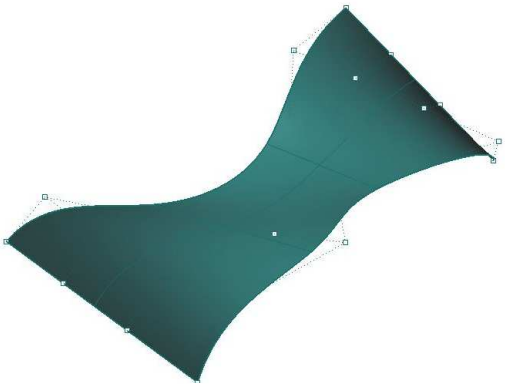
$$S(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} \left\{ N_{i,p}(\xi) \cdot M_{j,q}(\eta) \cdot X_{i,j} \right\} = \sum_{i=1}^{n}\sum_{j=1}^{m} \left\{ R_{i,j}^{p,q} \cdot X_{i,j} \right\}$$

Solid

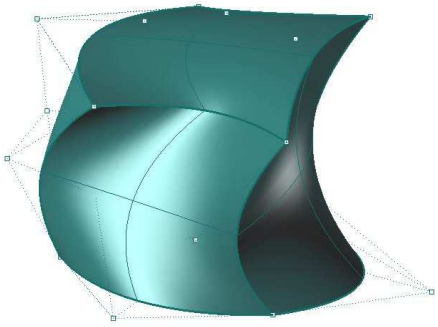
$$S(\xi,\eta,\zeta) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} \left\{ N_{i,p}(\xi) \cdot M_{j,q}(\eta) \cdot L_{k,q}(\zeta) \cdot X_{i,j,k} \right\} = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} \left\{ R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) \cdot X_{i,j,k} \right\}$$



(a) Curve



(b) Surface



(c) Solid

Figure 2.26. NURBS models.

2.3.8 Curve properties

B-SPLine curve entities have the following properties (Piegl, Tiller):

1. B-SPLine curves are a generalization of Bezier curves.
2. C(ξ) is a piecewise polynomial curve.
3. Each basis function corresponds to a specific control point.
4. The first and last control point, as well as the control points that corresponds to C^0-continuous basis functions, are interpolatory to the curve.
5. B-SPLine curves possess strong convex hull property.
6. Moving a control point X_i only changes part of the curve.
7. The control polygon represents a piecewise linear approximation to the curve.
8. It is possible to use multiple control points with the same coordinates.
9. Any transformation applied to the curve can be applied directly to the control points. This property is known as “affine invariance” or “affine covariance”.
10. In every knot span, at most p+1 control points contribute to the definition of the curve, corresponding to the p+1 non-zero basis functions.
11. Since C(ξ) is the linear combination of $N_{i,p}(\xi)$, curve’s continuity and differentiability comes from the basis functions.
12. No line has more intersections with the curve, than with the control polygon.

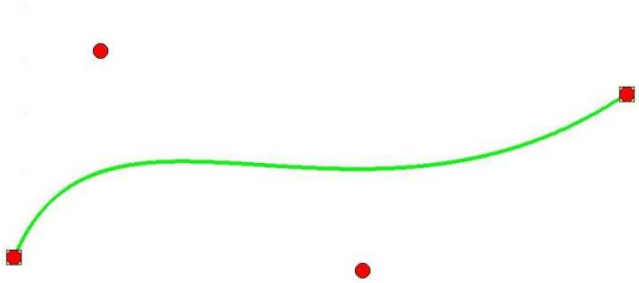
2.3.8.1 Generalization of Bezier curves

B-SPLine curves are the generalization of Bezier curves.

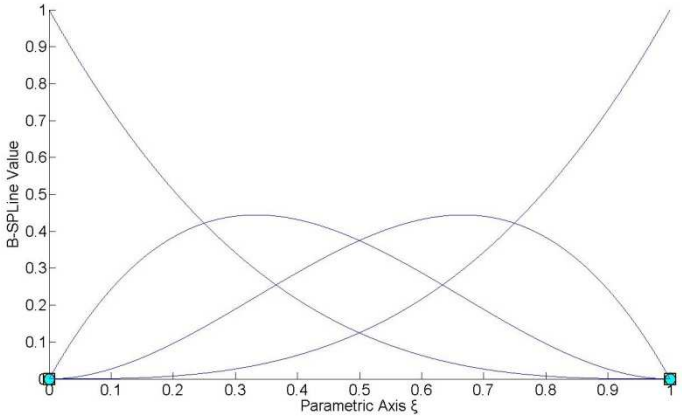
Given an open knot vector and n=p+1 control points, a Bezier curve is produced.

Bezier curves were utilized by CAD methods before B-SPLines. **Figure 2.27** depicts a Bezier curve with p=3, an open knot value vector {0,0,0,0,1,1,1,1} and n+p+1=4 control points.

Bezier curves are B-SPLine curves defined in only one knot span. As a result, every basis function is non-zero across the entire parameter space and control points affect the shape of the entire curve. Bezier surfaces are a special case of one-rectangle B-SPLine surfaces and Bezier solids a special case of one-cuboid B-SPLine solids as well.



(a) Curve.



(b) Basis.

Figure 2.27. B-SPLine curve and its basis.

2.3.8.2 Piecewise polynomial curve

C(ξ) is formed from piecewise polynomials $N_{i,p}(\xi)$, thus it is a piecewise polynomial curve.

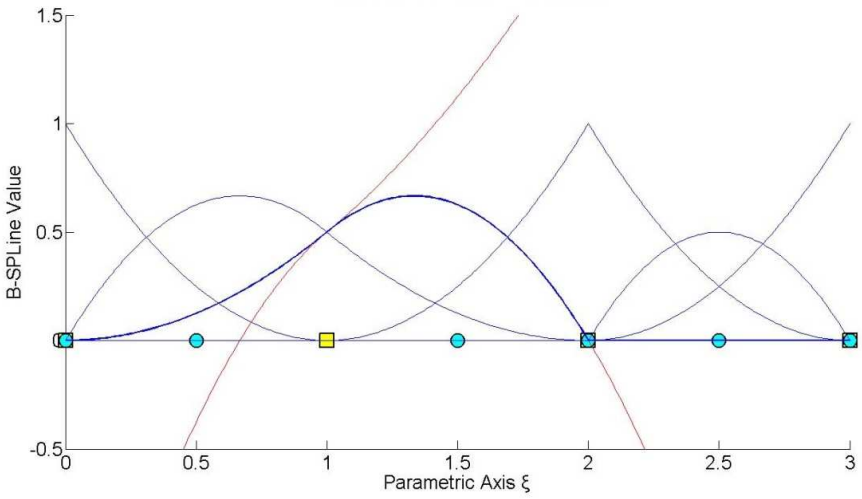


Figure 2.28. Piecewise polynomial curve.

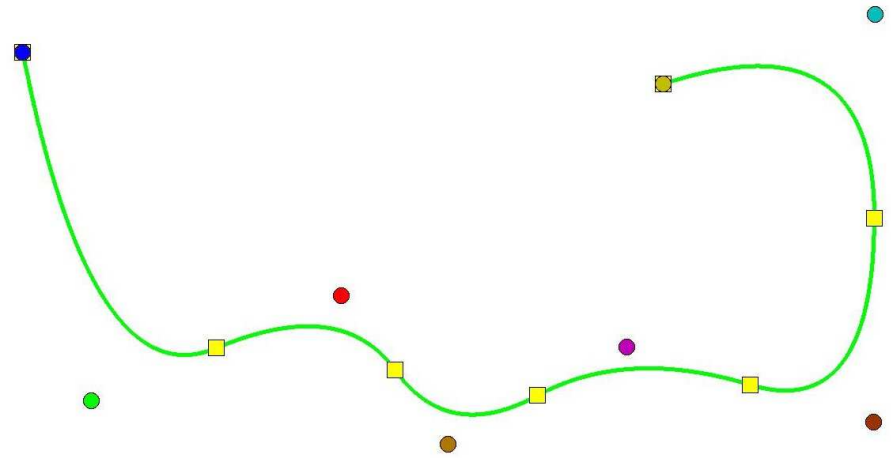
A B-SPLine curve is obtained by the following equaiton:

$$C(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) \cdot X_i$$

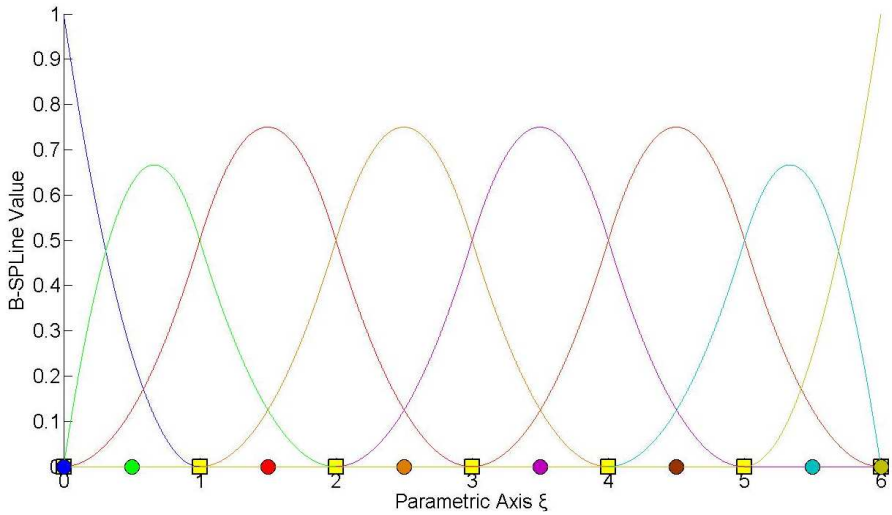
which is a linear combination of piecewise polynomial basis functions. This is also valid for multi-directional entities as well. Shape functions (full tensor product) are piecewise polynomials with respect to the corresponding directions, thus B-SPLine surfaces and solids are also piecewise polynomials. The term B-SPLine, after all, stands for Basis - Smooth Polynomial Line.

2.3.8.3 Control point – basis function correspondence

Each basis function corresponds to a specific control point.



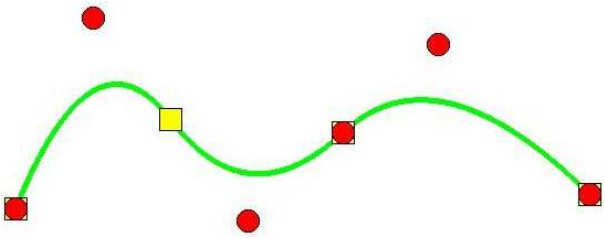
(a) Curve.



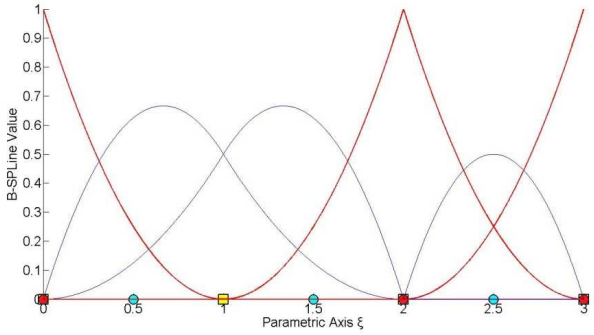
(b) Basis.

Figure 2.29. B-SPLine curve and its basis.

2.3.8.4 Interpolation



(a) Curve.



(b) Basis.

Figure 2.30. Control point interpolation.

$$C(\xi) = \sum_{i=1}^{n} \left\{ N_{i,2}(\xi) \cdot X_i \right\}$$

$$C(0) = \sum_{i=1}^{n} N_{i,2}(0) \cdot X_i$$

$$N_{1,2}(0) = 1,\ N_{i,2}(0) = 0,\ i = 2,...,6$$

so,

$$C(0) = N_{1,2}(0) \cdot X_1 = X_1$$

$$C(3) = \sum_{i=1}^{n} N_{i,2}(3) \cdot X_i$$

$$N_{6,2}(3) = 1$$

$$N_{i,2}(3) = 0,\ i = 1,...,5$$

so,

$$C(3) = N_{6,2}(3) \cdot X_6 = X_6$$

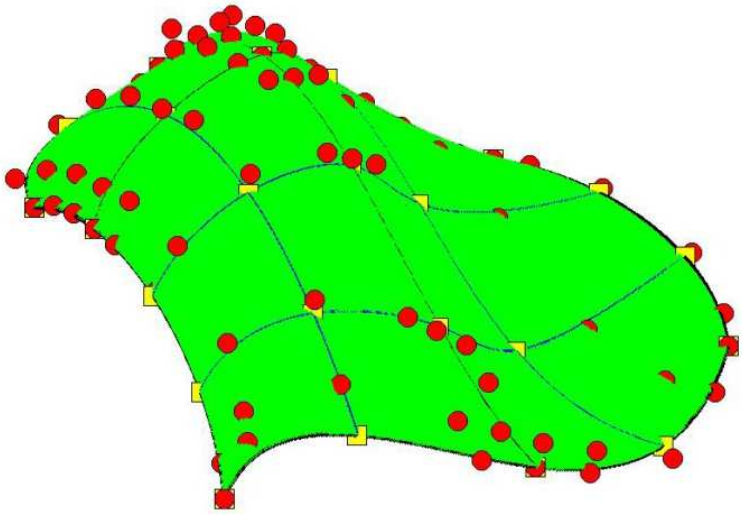
$$C(2) = \sum_{i=1}^{n} \left\{ N_{i,2}(2) \cdot X_i \right\}$$

$N_{4,2}(2) = 1$, as this is the only non-zero basis function across ξ=2

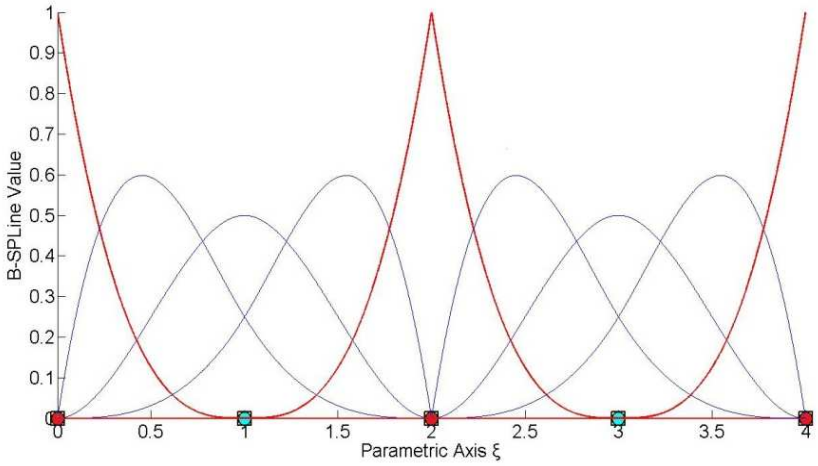
$$N_{i,2}(2) = 0,\ i \neq 4$$

so,

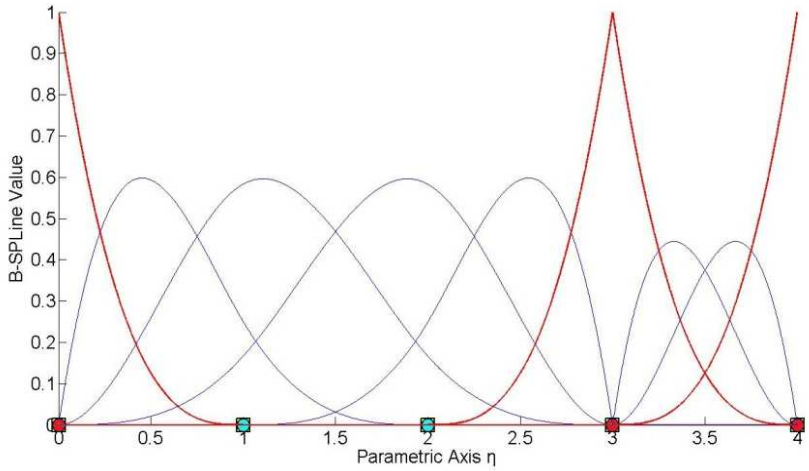
$$C(2) = N_{4,2}(2) \cdot X_4 = X_4$$



(a) Surface.

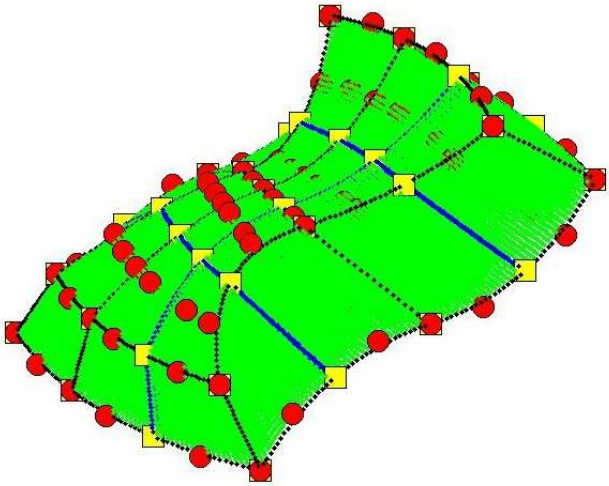


(b) Basis (ξ).

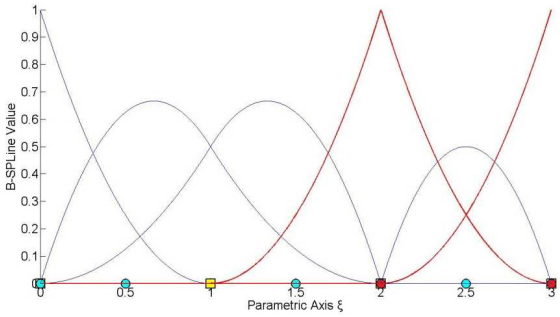


(c) Basis (η).

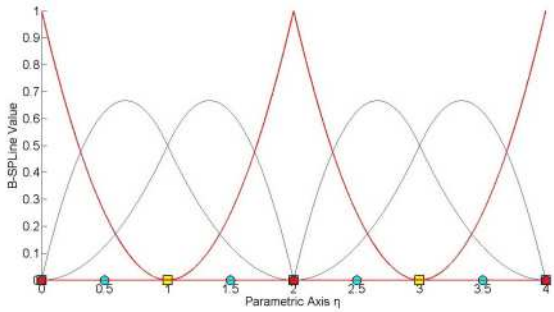
Figure 2.31. B-SPLine surface.



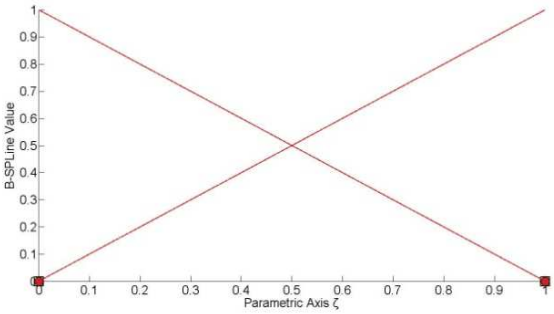
(a) Solid.



(b) Basis (ξ)



(c) Basis (η)



(d) Basis (ζ)

Figure 2.32. B-SPLine solid.

2.3.8.5 Convex hull

B-SPLine curves possess strong convex hull property. The convex hull of the curve is defined as the sum of the convex hulls of p+1 consecutive control points. The curve is always contained in the convex hull.

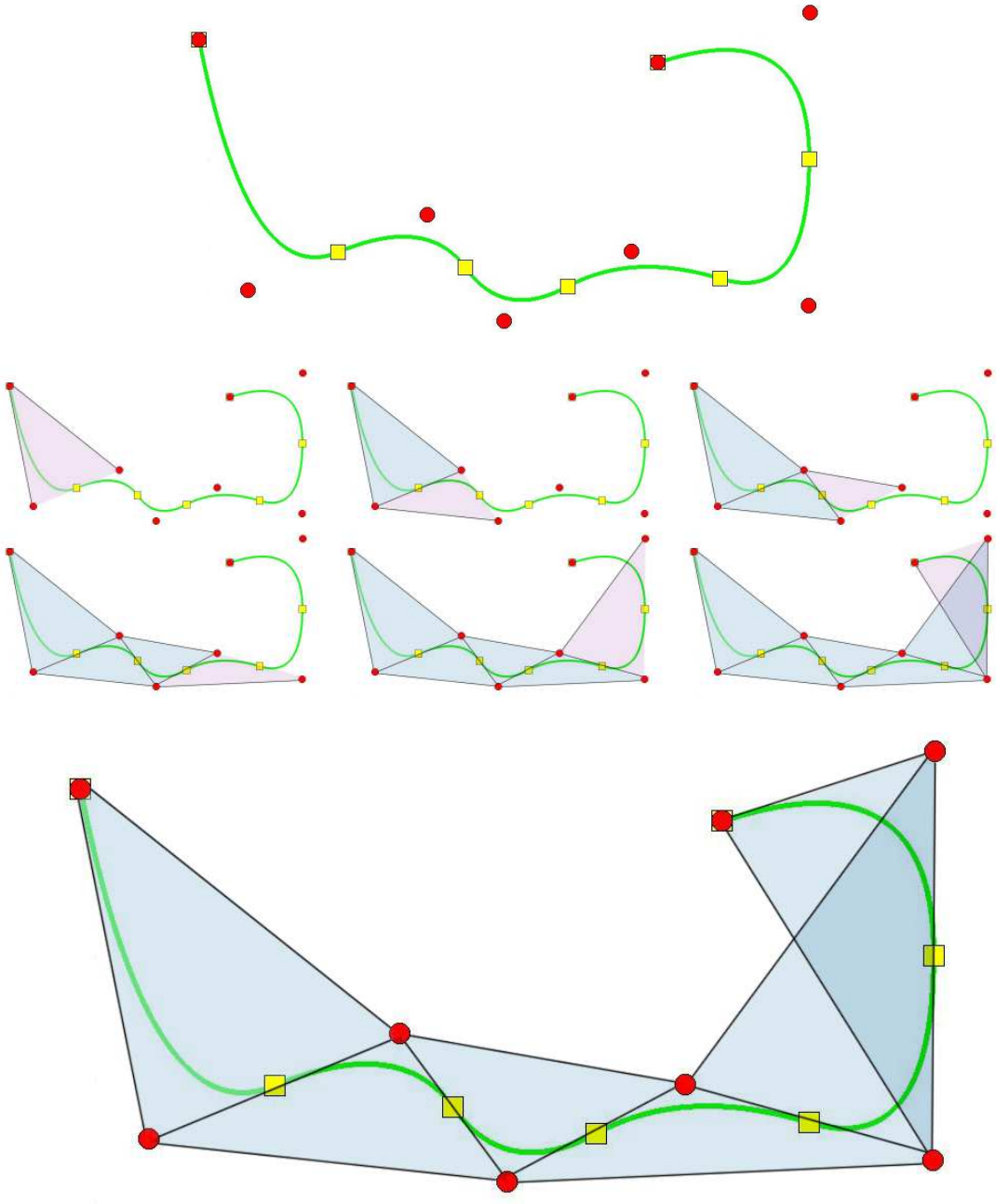


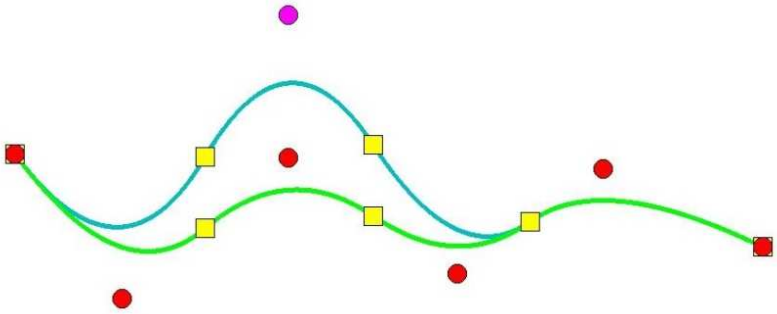
Figure 2.33. Convex hull.

The curve, depicted in **Figure 2.33**, is quadratic. The convex hull is formed by connecting each control point with other p ones. The union of the convex hulls contains the curve. The convex hull is a way to assume the general form of a B-SPLine curve.

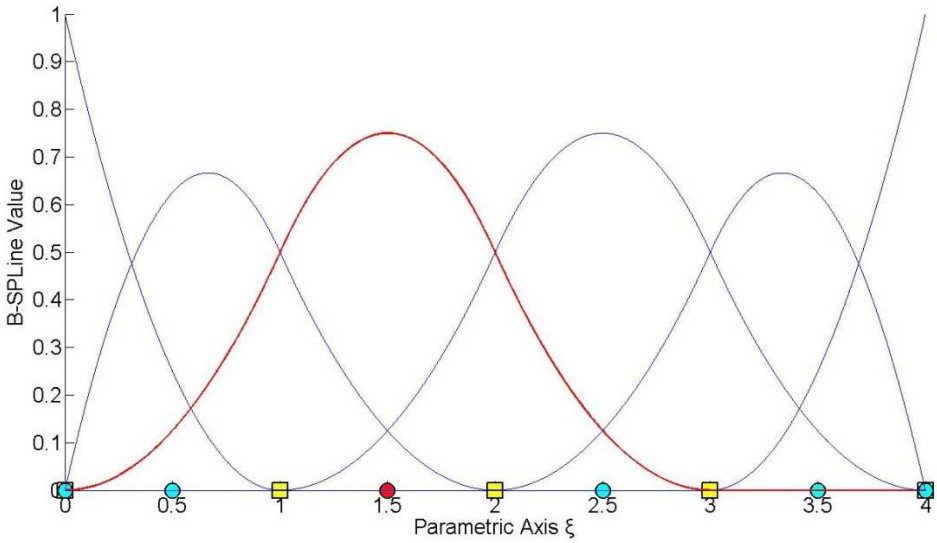
2.3.8.6 Local support

Moving a control point X_i only changes part of the curve, more specifically the part corresponding to the $[\xi_i, \xi_{i+p+1})$ knot value spans. This is a result of the local support of the corresponding B-SPLine function.

Moving along the curve, basis function $N_{i,p}(\xi)$ is switched "on" at the knot value ξ_i and then again switched "off" at the knot value ξ_{i+p+1}, where the function $N_{i+p+1,p}(\xi)$ is switched "on".



(a) B-SPLine curve.



(b) Quadratic basis.

Figure 2.34. Control point. Support.

A control point is associated with a basis function. Support of the control point is defined by the support of the corresponding basis function. Therefore, control points affect only part of the curve. In **Figure 2.34**, a curve with p=2 and knot value vector {0,0,0,1,2,3,4,4,4} is depicted. The third control point is moved and this affects partially the entity.

In this example, $N_{3,2}(\xi)$ spans from $\xi = 0$ to $\xi = 3$. The knots act as boundaries of the support. For $\xi = 3$, $N_{3,2}(\xi)$ is switched "off" and $N_{6,2}(\xi)$ is switched "on".

Local support of control points is also expanded by tensor product properties. The local support of a multi-directional control point is the tensor product of the respective supports.

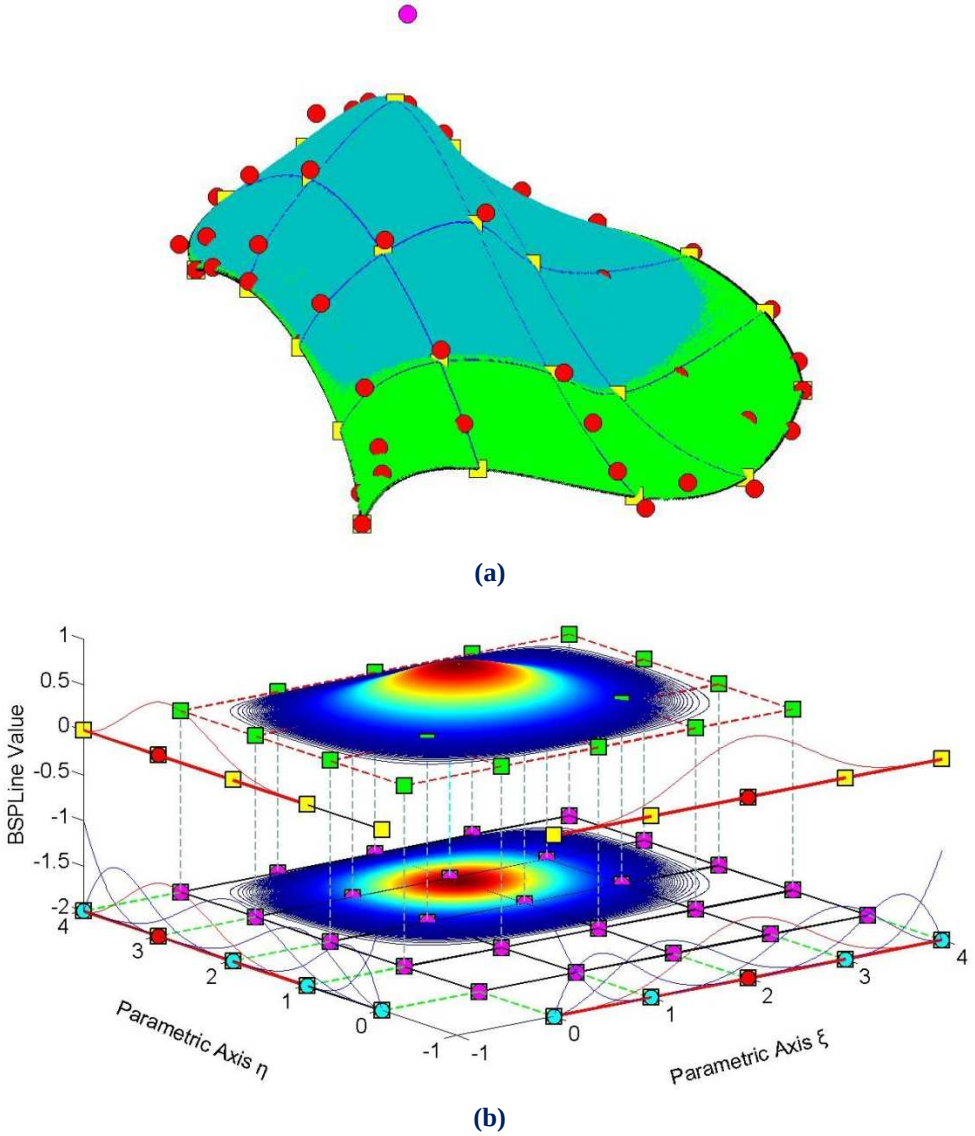
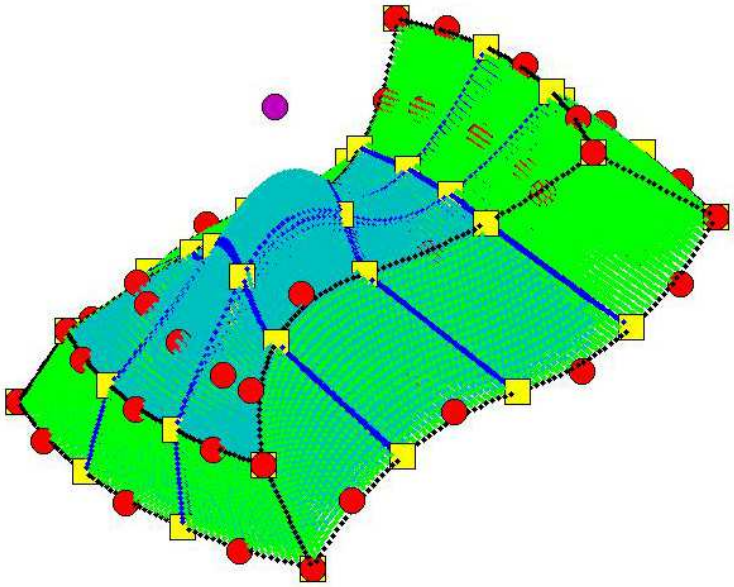


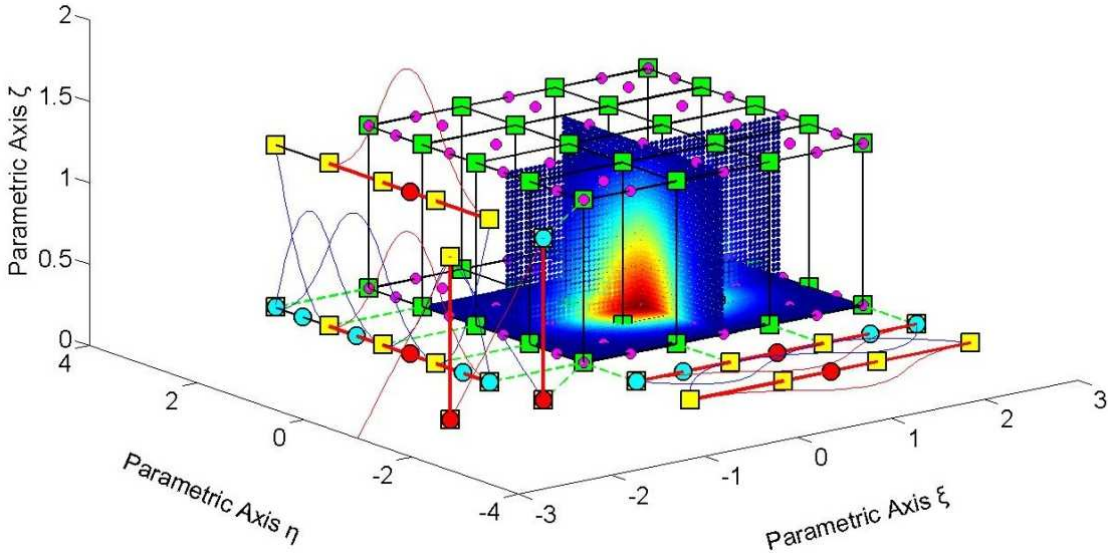
Figure 2.35. The support of a control point. 2D case.
Surface **(a)** in Physical and **(b)** in Parameter Space.

Figure 2.35 shows the local support in the parametric axes ξ, η of $N_{4,2}(\xi)$ and $M_{3,2}(\eta)$ B-SPLine curves respectively. In parametric axis ξ, the local support expands throughout the axis, whereas in parametric axis η, the basis function is switched "on" at the second knot span. In **Figure 2.35.a**, the tensor product of the respective supports for ξ, η is represented in cyan. It spans across $4x3 = 12$ knot rectangles in total.

The same properties apply for B-SPLine solids as well.



(a)



(b)

Figure 2.36. The local support of a control point. 3D case.
Solid **(a)** in Physical and **(b)** in Parameter Space.

Figure 2.36.b presents the local support of $N_{3,2}(\xi)$, $M_{4,2}(\eta)$ and $L_{1,1}(\zeta)$. In parametric axes ξ, ζ, the local support expands at all knot spans, whereas in the parametric axis η the local support spans between knots 1 and 4. In **Figure 2.36.a**, the local support is displayed in cyan and it does not reach all the knot spans in parametric axis η. A total of 3x3x1=9 knot cuboids present the support of the control point.

2.3.8.7 Control polygon approximation

The control polygon is a piecewise linear approximation to the curve. Due to convex hull properties, refinement by knot insertion or order elevation brings the control polygon closer to the curve.

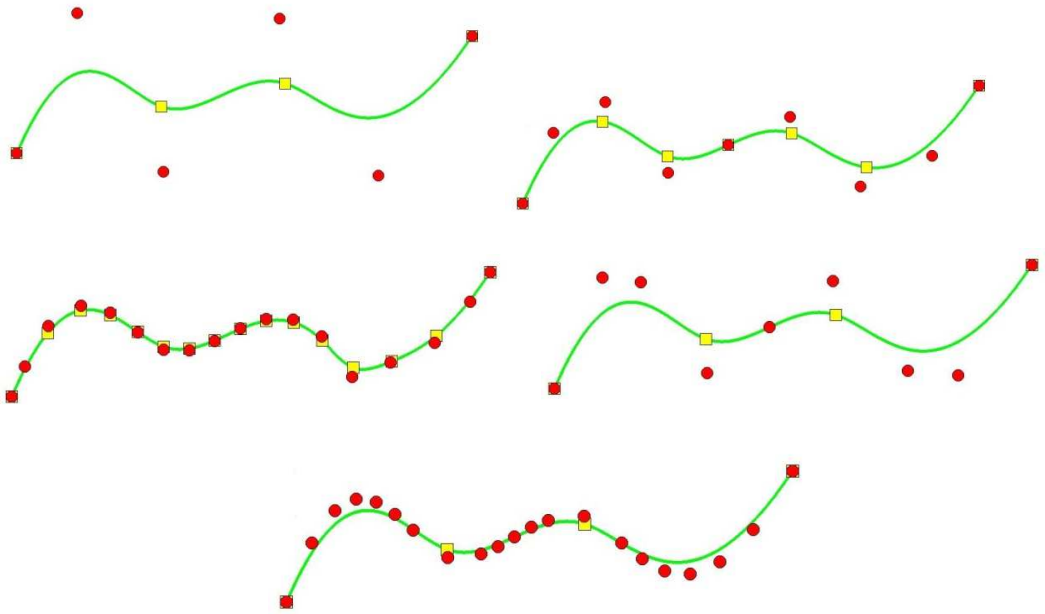


Figure 2.37. Control Polygon approximation through Refinement.

In **Figure 2.37** a curve of degree p=3 is designed. The control polygon is a linear approximation to the curve. When consecutive h- or p- refinements are applied, the control polygon is brought even closer to the curve. Refined control polygons provide a general idea of the form of the curve. This property also applies for multiple directions.

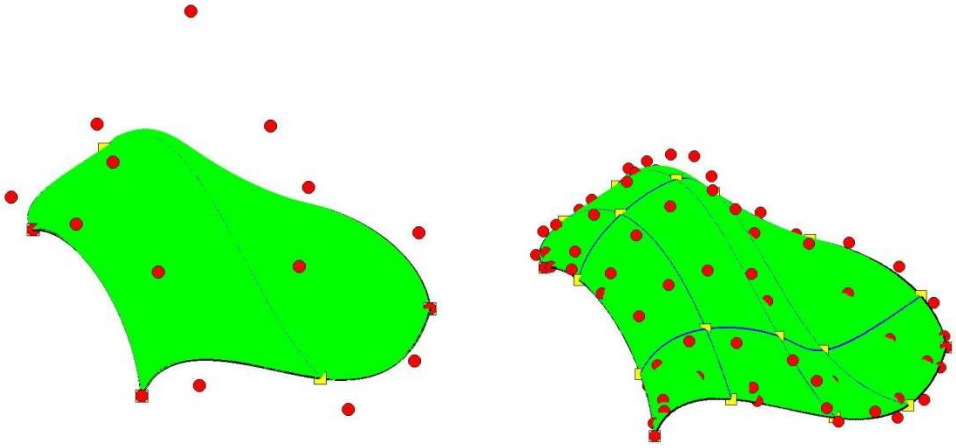
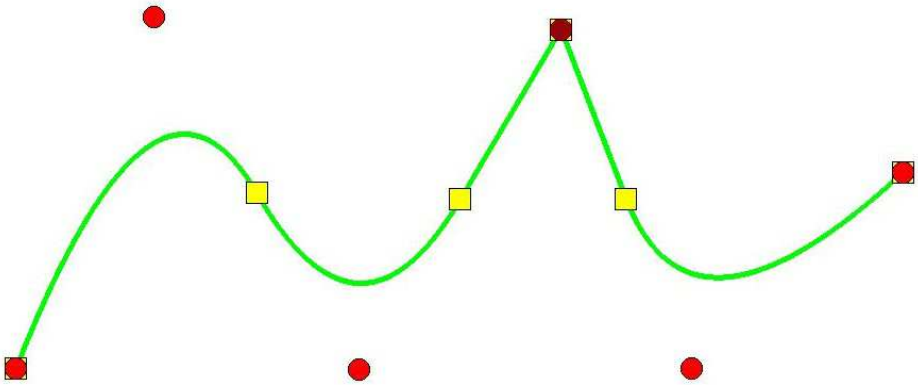


Figure 2.38. Control net. Knot insertion.

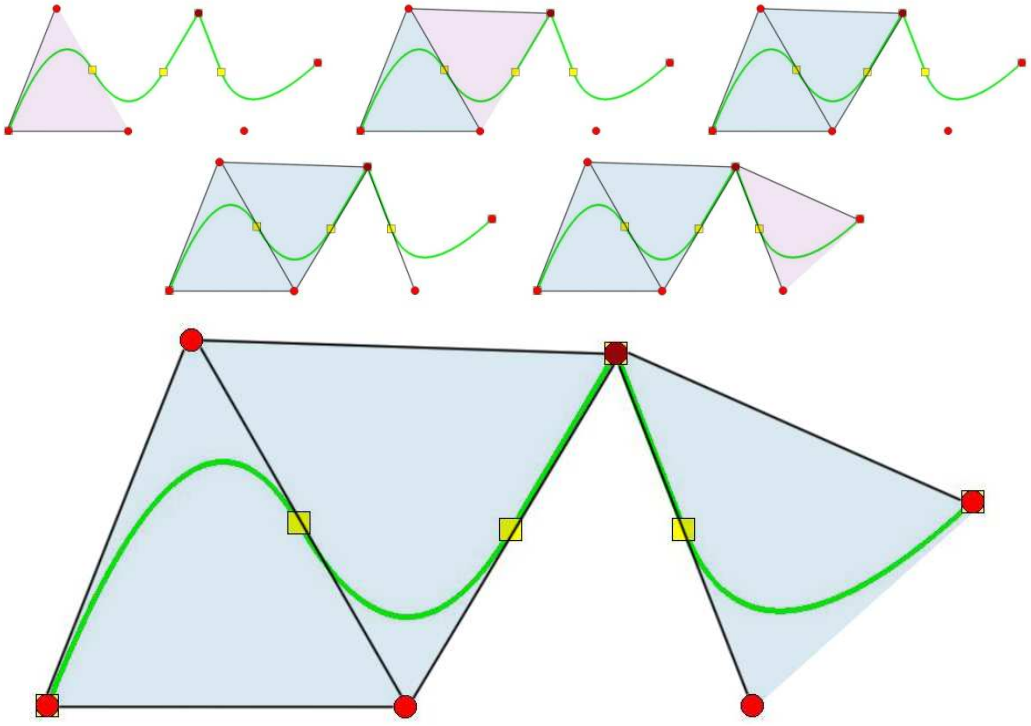
For example, in **Figure 2.38**, refinements bring the control net closer to the surface.

2.3.8.8 Multiple control points

It is possible to use multiple control points with the same coordinates.



(a) Curve. Physical space.



(b) Convex hull.

Figure 2.39. Convex hull.
Curve with two coincident control points (drawn in red).

In **Figure 2.39.a**, a quadratic curve with a double control point is designed. The curve is interpolatory at these points and a sharp edge is formed. This is explained in **Figure 2.39.b**, where the convex hull of the curve is designed. The curve is always contained in the convex hull, therefore a sharp edge has to be formed exactly at the double point coordinates. Inductively, this applies when p coincident control points are used in a curve of polynomial degree p.

2.4 Non-Uniform Rational B-SPLines

B-SPLine geometries may have many promising attributes, but they also have several weaknesses in geometrical representation. Some basic geometrical forms cannot be presented as B-SPLine entities, such as circles or conic sections in general. In order to solve this problem, the CAD industry is based on Non-Uniform Rational B-SPLines (NURBS). The basic idea is simple.

2.4.1 Basic idea

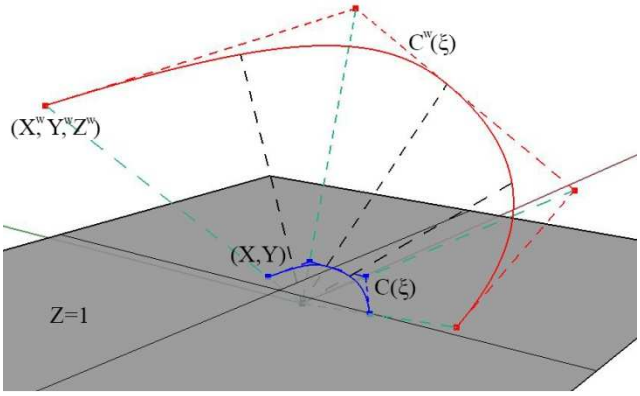


Figure 2.40. Projective transformation. NURBS curve.

As shown in **Figure 2.40**, the projective B-SPLine curve $C^w(\xi)$ is created from the projective 3D control points $X^w = \{X^w \quad Y^w \quad Z^w\}$.

Projection of the curve and control points on the plane $z = 1$ produces the NURBS curve $C(\xi)$ and the 2D control points:

$$X = \{X \quad Y\}$$

where,

$$\{X_i \quad Y_i\} = \left\{\frac{X_i^w}{Z_i^w} \quad \frac{Y_i^w}{Z_i^w}\right\}$$

The weights of the NURBS curve are defined as:

$$w = \{Z^w\}$$

Generally, n-dimensional Rational B-SPLines are projections of $(n+1)$-dimensional non-rational B-SPLines.

2.4.2 Shape functions

In order to evaluate NURBS shape functions, the weighting function is defined:

$$W(\xi) = \sum_{i=1}^{n} \left\{ N_{i,p}(\xi) \cdot w_i \right\}$$

In most engineering applications, weights have positive values. Unless otherwise stated, they will be considered positive for the scope of this thesis. $W(\xi)$ is in fact the Z-coordinate of the projective B-SPLine curve. Projective transformation is applied by dividing the other two coordinates of the B-SPLine curve with the Z-coordinate. NURBS shape functions are calculated by:

$$R_i^p(\xi) = \frac{N_{i,p}(\xi) \cdot w_i}{W(\xi)} = \frac{N_{i,p}(\xi) \cdot w_i}{\sum_{i'=1}^{n} \left\{ N_{i',p}(\xi) \cdot w_{i'} \right\}}$$

$R_i^p(\xi)$ are piecewise rational functions. The expression "the order of NURBS" refers to the order of the projective B-SPLine curve.
NURBS shape functions in multiple directions can be obtained as tensor products of one-directional basis functions:
2D shape functions:

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_{i,p}(\xi) \cdot M_{j,q}(\eta) \cdot w_{ij}}{\sum_{i'=1}^{n} \sum_{j'=1}^{m} \left\{ N_{i',p}(\xi) \cdot M_{j',q}(\eta) \cdot w_{i'j'} \right\}}$$

$$W(\xi,\eta) = \sum_{i'=1}^{n} \sum_{j'=1}^{m} \left\{ N_{i',p}(\xi) \cdot M_{j',q}(\eta) \cdot w_{i'j'} \right\}$$

Similarly, we expand the equations in order to obtain tensor product 3D shape functions:

$$R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) = \frac{N_{i,p}(\xi) \cdot M_{j,q}(\eta) \cdot L_{k,r}(\zeta) \cdot w_{ijk}}{\sum_{i'=1}^{n} \sum_{j'=1}^{m} \sum_{k'=1}^{l} \left\{ N_{i',p}(\xi) \cdot M_{j',q}(\eta) \cdot L_{k',r}(\zeta) \cdot w_{i'j'k'} \right\}}$$

The weighting function is now defined as:

$$W(\xi,\eta,\zeta) = \sum_{i'=1}^{n} \sum_{j'=1}^{m} \sum_{k'=1}^{l} \left\{ N_{i',p}(\xi) \cdot M_{j',q}(\eta) \cdot L_{k',r}(\zeta) \cdot w_{i'j'k'} \right\}$$

Observe that for $w_{ijk} = 1, \ \forall\, i,j,k$, it applies that NURBS shape functions downgrade to B-SPLine basis functions. NURBS entities are a generalization of B-SPLine entities.

2.4.3 Derivatives

Simple application of the quotient rule yields the derivatives of NURBS shape functions for one and multiple directions.

$$\frac{d}{d\xi}R_i^p(\xi) = w_i \cdot \frac{\left(\frac{d}{d\xi}N_{i,p}(\xi)\right)\cdot W(\xi) - \left(\frac{d}{d\xi}W(\xi)\right)\cdot N_{i,p}(\xi)}{\left(W(\xi)\right)^2}$$

where

$$\frac{d}{d\xi}W(\xi) = \sum_{i=1}^{n}\left(\frac{d}{d\xi}N_{i,p}(\xi)\right)\cdot w_i$$

For bidirectional shape functions:

$$\frac{\partial}{\partial\xi}R_{i,j}^{p,q}(\xi,\eta) = w_{ij} \cdot \frac{\left(\frac{d}{d\xi}N_{i,p}(\xi)\right)\cdot M_{j,q}(\eta)\cdot W(\xi,\eta) - \left(\frac{\partial}{\partial\xi}W(\xi,\eta)\right)\cdot N_{i,p}(\xi)\cdot M_{j,q}(\eta)}{\left(W(\xi,\eta)\right)^2}$$

$$\frac{\partial}{\partial\eta}R_{i,j}^{p,q}(\xi,\eta) = w_{ij} \cdot \frac{N_{i,p}(\xi)\cdot\left(\frac{d}{d\eta}M_{j,q}(\eta)\right)\cdot W(\xi,\eta) - \left(\frac{\partial}{\partial\eta}W(\xi,\eta)\right)\cdot N_{i,p}(\xi)\cdot M_{j,q}(\eta)}{\left(W(\xi,\eta)\right)^2}$$

Derivatives of 3D shape functions per direction are evaluated as shown

$$\frac{\partial}{\partial\xi}R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) =$$

$$= w_{ijk} \cdot \frac{\left(\frac{d}{d\xi}N_{i,p}(\xi)\right)\cdot M_{j,q}(\eta)\cdot L_{k,r}(\zeta)\cdot W(\xi,\eta,\zeta) - \left(\frac{\partial}{\partial\xi}W(\xi,\eta,\zeta)\right)\cdot N_{i,p}(\xi)\cdot M_{j,q}(\eta)\cdot L_{k,r}(\zeta)}{\left(W(\xi,\eta,\zeta)\right)^2}$$

$$\frac{\partial}{\partial\eta}R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) =$$

$$= w_{ijk} \cdot \frac{N_{i,p}(\xi)\cdot\left(\frac{d}{d\eta}M_{j,q}(\eta)\right)\cdot L_{k,r}(\zeta)\cdot W(\xi,\eta,\zeta) - \left(\frac{\partial}{\partial\eta}W(\xi,\eta,\zeta)\right)\cdot N_{i,p}(\xi)\cdot M_{j,q}(\eta)\cdot L_{k,r}(\zeta)}{\left(W(\xi,\eta,\zeta)\right)^2}$$

$$\frac{\partial}{\partial\zeta}R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) =$$

$$= w_{ijk} \cdot \frac{N_{i,p}(\xi)\cdot M_{j,q}(\eta)\cdot\left(\frac{d}{d\zeta}L_{k,r}(\zeta)\right)\cdot W(\xi,\eta,\zeta) - \left(\frac{\partial}{\partial\zeta}W(\xi,\eta,\zeta)\right)\cdot N_{i,p}(\xi)\cdot M_{j,q}(\eta)\cdot L_{k,r}(\zeta)}{\left(W(\xi,\eta,\zeta)\right)^2}$$

2.4.4 NURBS entities

NURBS entities are created as a linear combination of NURBS shape functions, exactly the same way as B-SPLine entities. The following is the equation for the creation of NURBS curves:

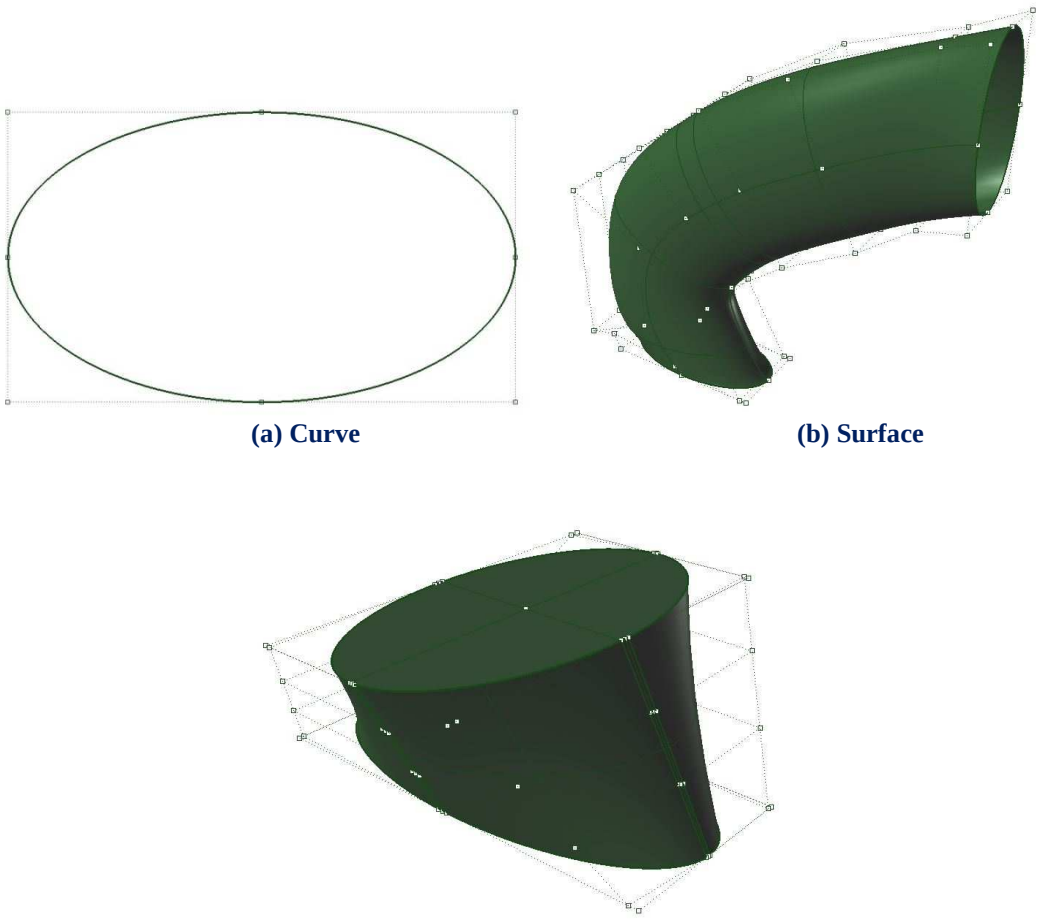
$$C(\xi) = \sum_{i=1}^{n} \left\{ R_i^p(\xi) \cdot X_i \right\}$$

Surfaces:

$$S(\xi,\eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left\{ R_{i,j}^{p,q}(\xi,\eta) \cdot X_{i,j} \right\}$$

Solids:

$$S(\xi,\eta,\zeta) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{l} \left\{ R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) \cdot X_{i,j,k} \right\}$$

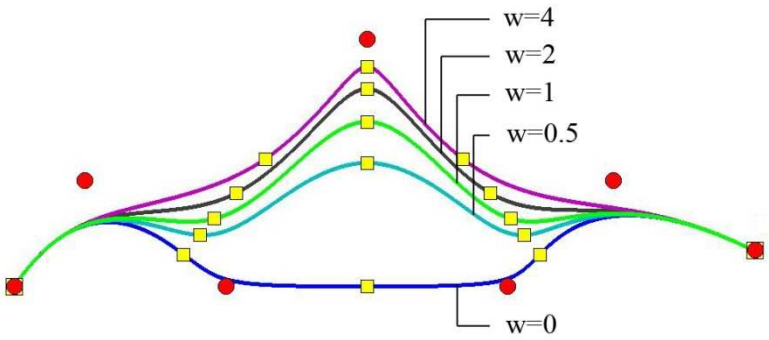


(a) Curve **(b) Surface**

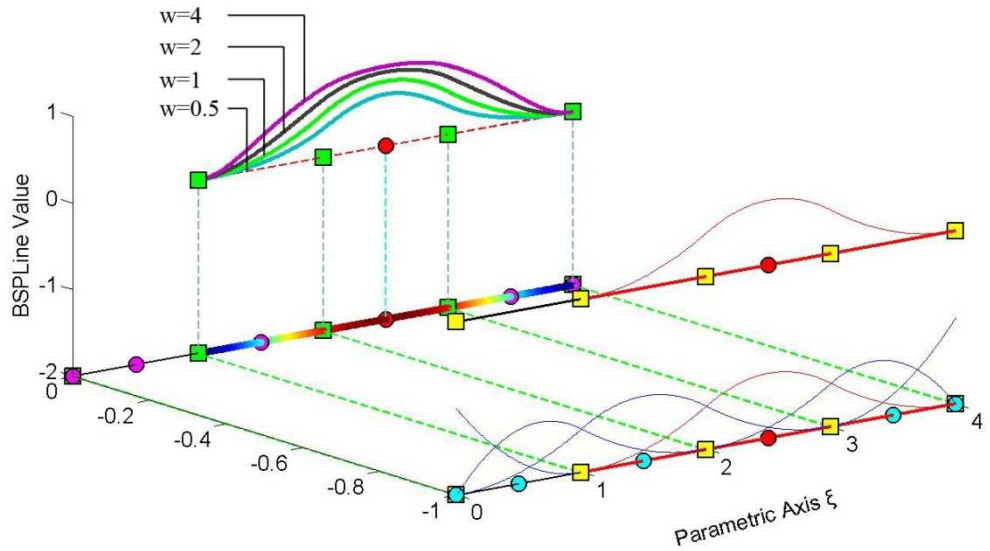
(c) Solid

Figure 2.41. NURBS elliptical entities.

2.4.5 NURBS examples



(a) Curve. Weight.



(b) Shape function. Weight.

Figure 2.42. NURBS curves, shape functions and weights.

In **Figure 2.42**, five NURBS curves with the same set of control point coordinates are represented. The corresponding weights are $w_i = 1$, for $i \neq 3$. The third control point has a different weight for each curve. Observe that, as the weight value increases, the shape function and, as a result, the corresponding control point tends to dominate the $p+1$ knot spans of the support. Thus, the knots are gravitated closer to the corresponding control point.

In order to accurately represent an arc of $\theta < 180$ degrees, three control points are required. Weights for the first and last points are $w_1 = w_3 = 1$, whereas the middle one has a weight of $w_2 = \cos\left(\frac{\theta}{2}\right)$.

In **Figure 2.43**, the same circle is represented by NURBS shape functions of different order. This is a closed curve, so the first and last control points coincide. Weights are shown for each control point. A NURBS circle is usually represented by four consecutive patches, bound together by a common knot value vector.

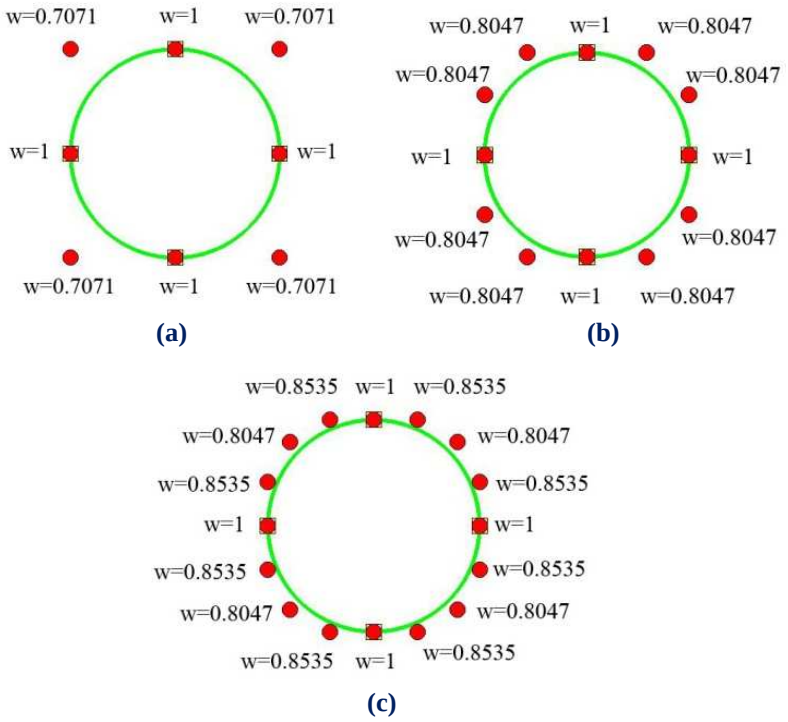


Figure 2.43. NURBS circle, different degree & weights.
(a) Quadratic basis functions. Knot value vector

$$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4\}$$

(b) Cubic basis functions. Knot value vector

$$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4 \quad 4\}$$

(c) Quadric basis functions. Knot value vector

$$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4\}$$

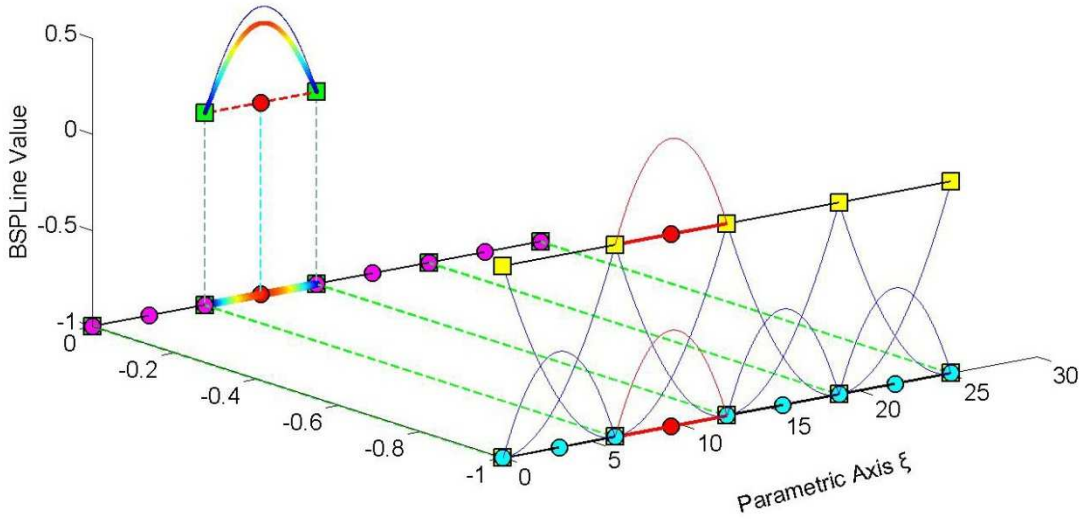


Figure 2.44. Basis needed for a circle.

Contour distribution for shape function $R_5^2(\xi)$, with corresponding weight $\frac{\sqrt{2}}{2} = 0.7071$.

Comparison with basis function $N_{5,2}(\xi)$ shown in blue.

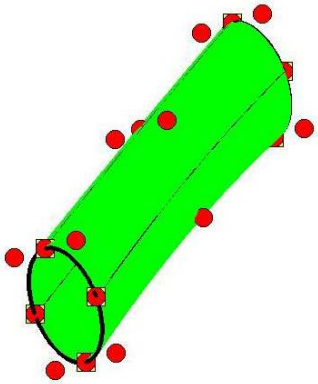


Figure 2.45. NURBS surface generated from consecutive circle cross-sections.

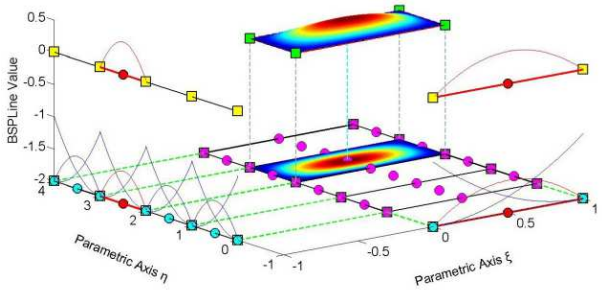
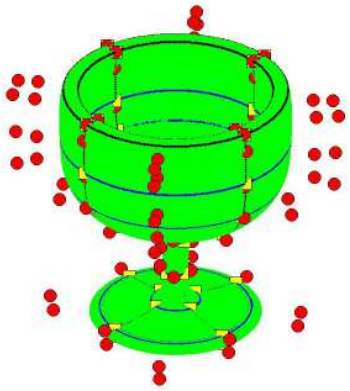
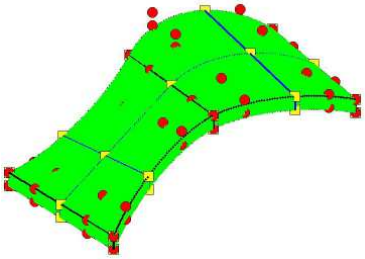


Figure 2.46. 2D Shape function.



(a) Wine glass.



(b) Abstract NURBS solid.

Figure 2.47. NURBS solids.

.

2.5 Patch

NURBS entities are created by transforming a simple parametric shape (line, rectangle, cube) to a model in physical Space (curve, surface, solid). They are used for the exact and efficient representation of complex geometrical structures. Sometimes, the mapping of a single parametric shape is not the optimum solution. A designer might need two or three parametric cubes in order to efficiently represent solids with major changes in geometrical attributes. As displayed in **Figure 2.48**, each of these cubes, mapped to a portion of the solid in physical space, is a NURBS patch. As expected, each patch has continuity C^{p-m} on interior knots and C^{-1} on the edge.

(a) **(b)**

Figure 2.48. Falkirk Wheel "abutment".
(a) Geometrical representation. Five separate patches are used.
(b) Each patch portrays a cube in parameter space mapped as a complex shape in physical space.

Interconnection between patches can be roughly achieved by choosing coincident control points on the edges. Still, patch connection rarely is leak-proof. This is one of the major disadvantages of NURBS, downsized and eliminated in the next version of SPLines (T-SPLines etc.).

Sometimes, patches exist for other reasons. For example, a major change in material properties requires a patch boundary. Interpolation through a certain control point calls for patch boundary to be established there. Even application of C^0 continuity, for analysis purposes, is enabled by introduction of a patch. If the same polynomial order is used, the mapping can be unified. In these special occasions, the separate parameter spaces of the patches can be united in one parameter space, using one set of basis functions and one knot value vector. The distinction between patches can be applied by enforcing C^0 continuity across the boundary.

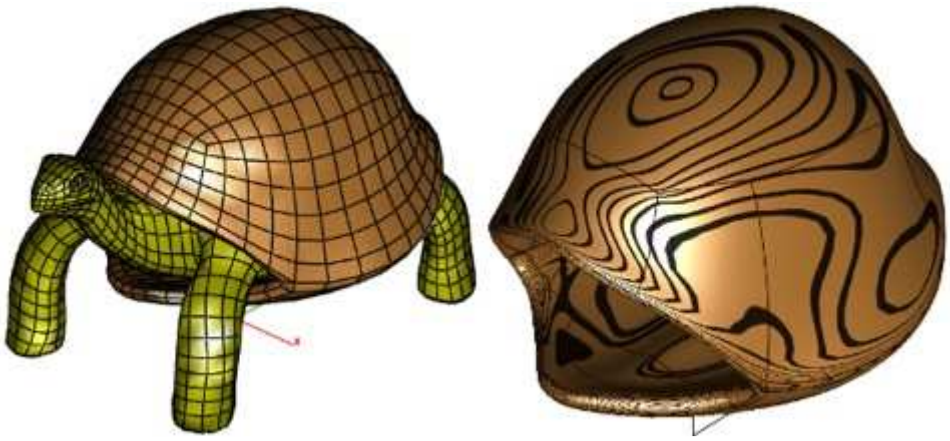


Figure 2.49. The NURBS model of a turtle.
NURBS patches enforced in order to distinct shell from turtle.
Each material requires separate patch stiffness matrix evaluation,
before Global stiffness matrix creation.
(masterviacad.com)

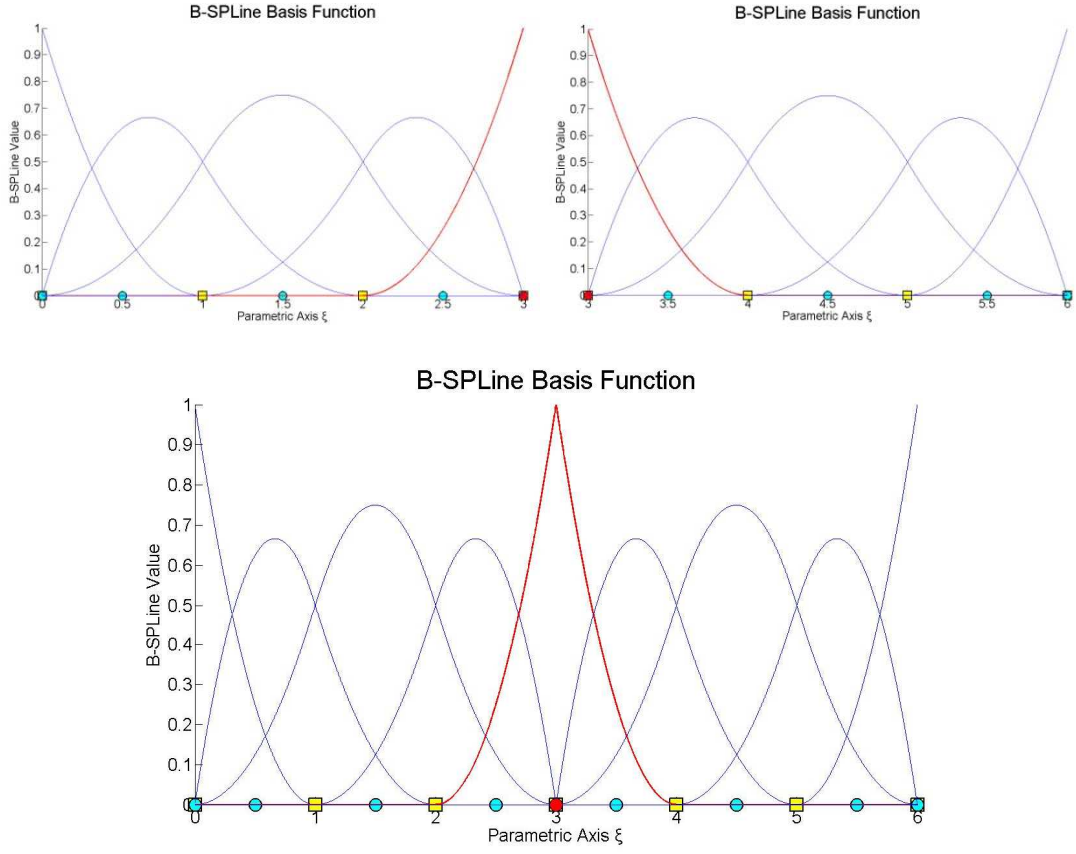


Figure 2.50. Separate knot vectors united into one.
The control points at the boundary are merged. C^0 Continuity is applied.

In the geometrical models presented in this thesis, knot boundaries are drawn in blue and patch boundaries in black. This separates C^{-1} and C^0 continuity from C^1 and greater continuity.

2.6 Relations between basic entities

Many times it is useful to know the interaction that exists between basic entities of isogeometric analysis. In IGA the three basic entities are the control points, the elements and the Gauss points. The last two entities are related to each other since the quadratic rule determines the number of Gauss points per element. Relations between control points and elements are established for the assembly of the stiffness matrix as for other parts of analysis. For the program implementation it is more efficient and not complicated to employ elements to Gauss points during the determination of relations.

It is worth to note that the number of elements that correspond to a single control point is variable and as a result the storage of relations in a single two dimensional array would be inefficient since there will be empty entries. For this reason, it is preferable to use jagged arrays. Reminded that a jagged array is an array whose elements are arrays and its elements can be of different dimensions and sizes.

2.6.1 N-E correlations

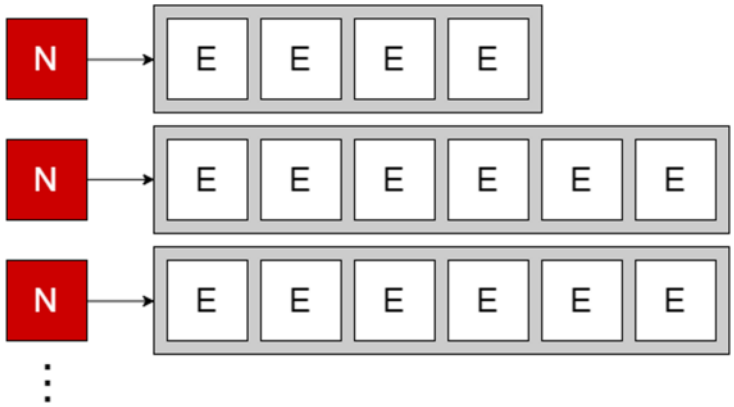


Figure 2.51. N-E correlations.
Each node N is associated to the element entities E that influence it.

An extensive search that tests all nodal-element entity combinations is not practical and therefore the techniques that will be used to determine the N-E correlations will be oriented in the isogeometric simulation method and its ingredients. For each parametric axis (ξ, η, ζ) the mutual relationships between control points and knot spans are indetified. Thus, a jagged array, as shown in the following figure, is created per axis. Every row has a unique ID, which corresponds to the parametric ID of a control point for a specific axis. The number of the columns that a row has, is equal to the number of influenced elements.

Having determine the N-E correlations per axis is easy to exploit the structured grid that isogeometric analysis provides and determine the N-E correlations for the whole geometric domain.

2.6.2 Interactions

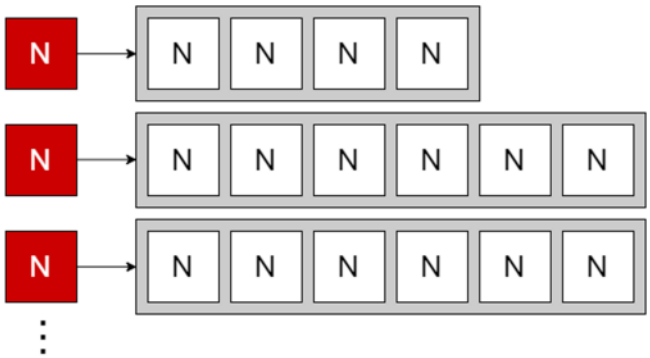


Figure 2.52. Nodal entity interactions.

There are many ways to find the interactions between nodal entities. The most generalized way, which is also applicable on other computational methods like meshless and finite elements, is to devote the N-E and E-N correlations. Each nodal entity has a set of elements influencing it and each element has a list of nodes it influences. In order to identify the interactions of a nodal entity, the nodal entities associated with the related element of the nodal entity are inspected. In this work, a more customized to IGA technique has been used. Taking the advantage of the B-SPLine basis functions support, the interactions between a particular axis can be stored in a single jagged array. Therefore, devoting the full tensor product, which allows the inheritance of the support property from B-SPLines basis functions to NURBS shape functions, the interactions can easily be specified for the whole structure.

2.6.3 Synergies

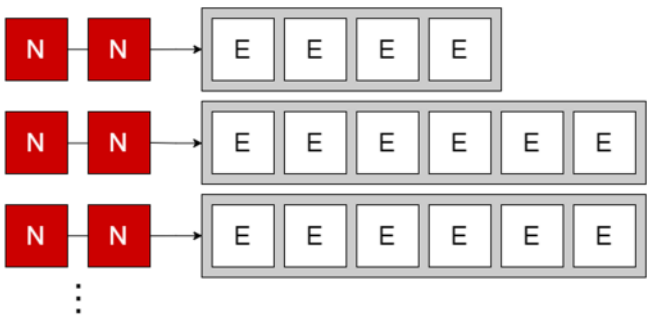


Figure 2.53. Interacting node pairs with shared elements.

Having determined the N-E correlations and the interactions between control points it is not difficult to identify the shared elements of control point pairs. Specifically, for each i-j pair the shared elements arising from the intersection of the elements that both influence i and j. Again, shared elements per pair could be stored, for better utilization of memory resources, in a jagged array where in each row the shared elements of each control point pair will be located.

2.6.4 Domain of influence

The basic entities in IGA are control points and elements. The total stiffness matrix in IGA can be assembled as in FEA with a direct reference in the elements that form a patch. This approach demands the determination of the areas that a control point influences.

In **Figure 2.54**, the areas influencing a control point are shown. Due to the support of B-SPLine basis functions, moving a single control point can affect the geometry of no more than p+1 elements of the curve. When there is no repetition of knot values, and therefore no zero measured knot spans, a single control point affects exactly p+1 control points.

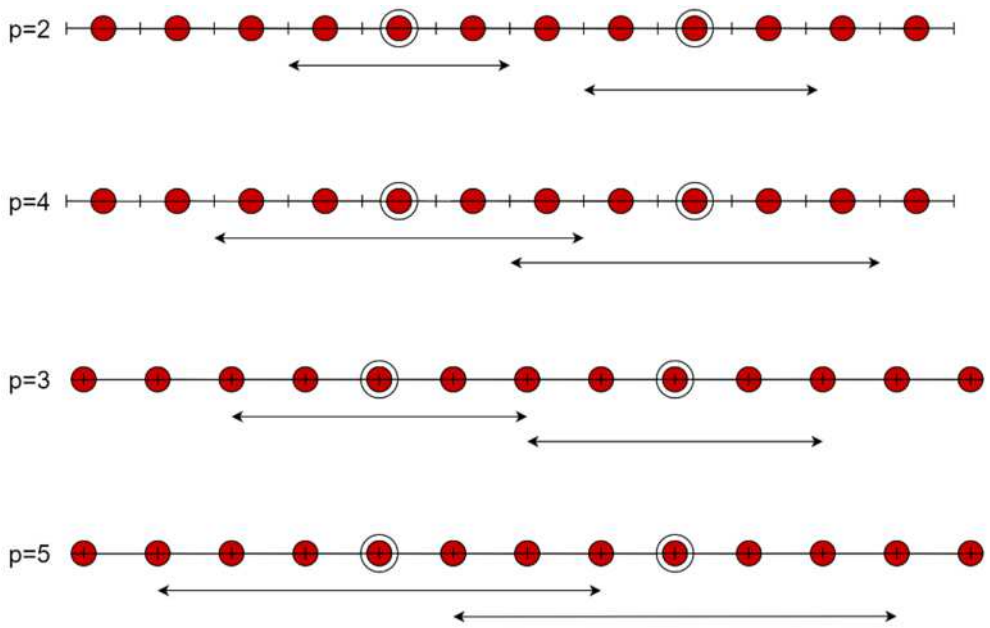


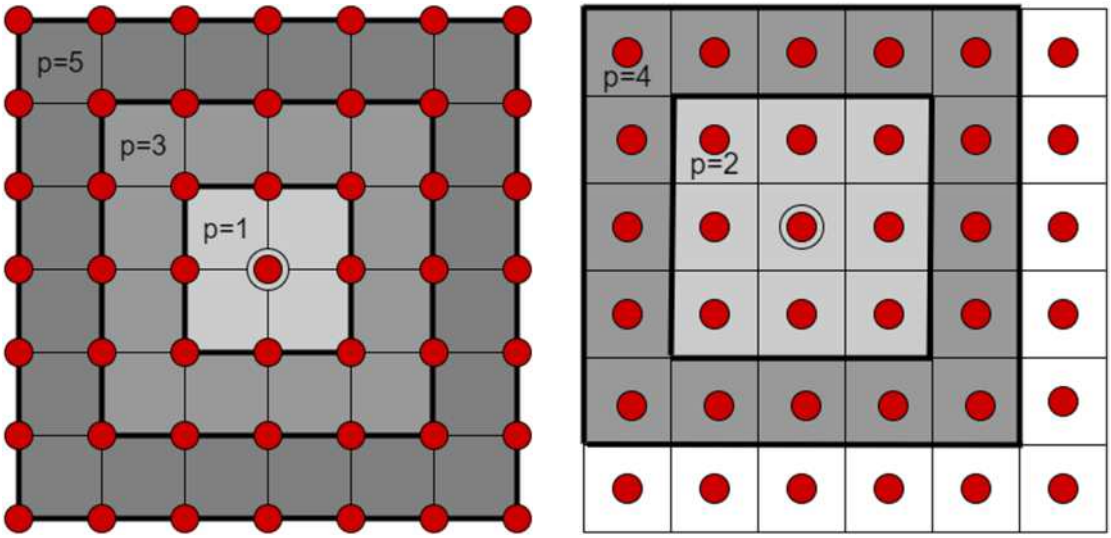
Figure 2.54. Influencing areas of control points. 1D case. Degree.

It is obvious, that for the even degrees p=2 and p=4, control points are located between two sequential knot values. In other words, the parametric coordinates of control points are defined as the average of the coordinates of the knot spans by whom they are surrounded.

For the odd degrees p=3 and p=5, control points and knots are located in the same position.

The previous remarks are direct consequences of the support property of B-SPLine basis functions. Control points are known as the center of support, because the fact they influence the geometric domain in the same way through one direction. Specifically, each basis function shares support with at most p B-Spline basis functions on each side.

In the same way, for a 2D problem, a single control point influences up to $(p+1)\cdot(q+1)$ knot spans.



(a) Odd degree. **(b) Even degree.**

Figure 2.55. Domain of influence. Control point.

In **Figure 2.55**, areas influencing control point situated within cycle for (a) IGA (p odd); (b) IGA (p even). The influenced areas are colored in grey color. Contrary, in Finite Element Method, each node only interacts with the elements in which it belongs as shown in **Figure 2.56**. Thus, for a problem with the same number of degrees of freedom, IGA has more control point interactions and it leads to denser stiffness matrices that FEA. However, IGA does not lead to increased bandwidth of the stiffness matrix in relation to the classical FEA due to the overlapping of B-SPLines basis functions.

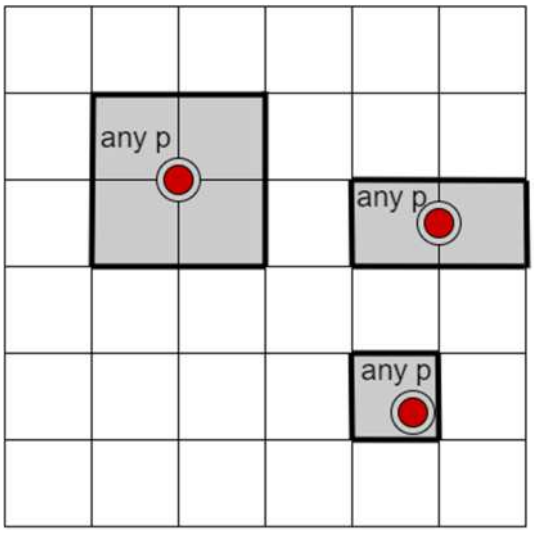
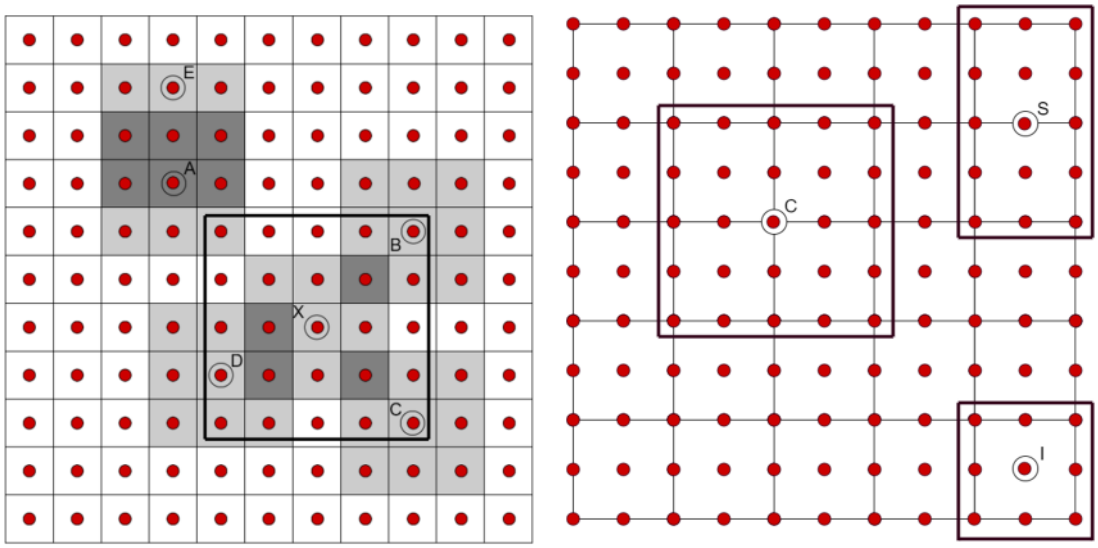


Figure 2.56. Areas influencing node (in red). FEA.
The influenced areas are colored in grey.

2.6.5 Node interaction

It has been referred that each B-Spline shares support with at most 2p B-Splines. Specifically, each basis function shares support with at most p B-Spline basis functions on each side. This fact has a great impact on the formulation of the stiffness matrix and especially for the interaction-wise approach that will be discussed later. In IGA, two control points interact between them if there is at least one shared element. The shape of knot value vector for a particular degree is crucial for the total number of interacting control points. Trivial knot spans reduce the continuity and result in a lower number of interactions.



(a) IGA **(b)** FEA

Figure 2.57. Interacting control points/ nodes for p=2.

In **Figure 2.57.a** the interactions between control points are shown. The degree of the B-SPLine basis function is p=2 in all directions. Due to the fact that there is not any trivial knot span, the shared support is 2p+1=5. Thus, X control point interacts with B, C, D but not with A or E. Similarly, A interacts with E sharing three knot spans. Shared elements appear in deep gray color while the influence domains of each control point is colored in light grey. The thick-lined centered square includes all the control point that are interacting with X control point.

Contrary, in FEA the interaction of the nodes is limited to the adjacent elements. Thus, for the mesh shown in **Figure 2.57.b**, C node interacts only with the four elements from which it is surrounded. Similarly, S node interacts only with the two elements that it belongs to. Attention is needed for the nodes that are not placed in the boundary of the elements but they are enclosed on them. They interact only with one element. Node I is a characteristic example of this case. It is the middle node of a nine nodded element and as a result it interacts only with it.

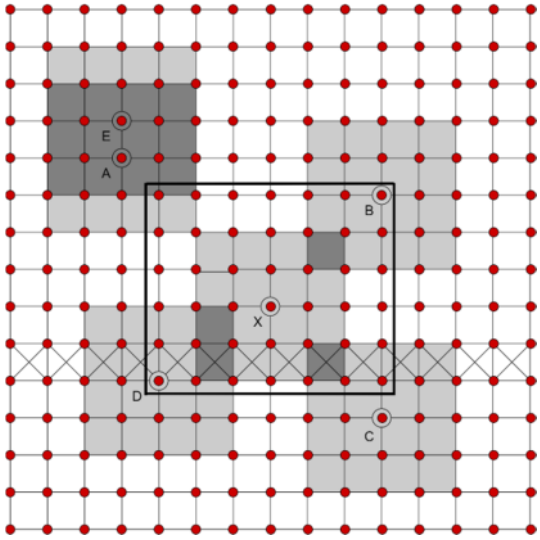


Figure 2.58. IGA Interacting control points for p=3.

It is obvious, that the presence of trivial knot spans reduces the number of interacting control points and results a sparser shape of the stiffness matrix. **Figure 2.58** reflects the impact of zero knot spans in an isogeometric analysis domain. Specifically, the trivial knot spans are on the y-axis and create a row of empty elements reducing the interaction between control points that are located up and down of the row. This state reflects in the relation between X and C. Despite they share one knot span, they are not interacting since this knot span has no Gauss points. However, X and D control points continue to interact sharing the Gauss points of one knot span.

2D Problem Type	Example	DOF	Interacting Control Pairs Number	Time Needed (ms)
	P1-1	2.420.000	10.876.804	1.898
	P1-2	2.000.000	8.988.004	1.767
	P1-3	1.620.000	7.279.204	1.337
	P1-4	1.280.000	5.750.404	1.051
	P2-1	768.800	9.572.836	1.947
	P2-2	500.000	6.220.036	1.291
	P2-3	245.000	3.041.536	599
	P2-4	101.250	1.252.161	237
	P3-1	301.088	7.311.616	1.842
	P3-2	204.800	4.963.984	1.193
	P3-3	151.250	3.659.569	884
	P3-4	101.250	2.442.969	584
	P4-1	151.250	6.027.025	1.826
	P4-2	101.250	4.020.025	1.414
	P4-3	51.200	2.016.400	654
	P4-4	20.000	774.400	206

Table 2.1. Number of interacting control point pairs. 2D examples.

3D Problem Type	Example	DOF	Interacting Control Pairs Number	Time Needed (ms)
	P1-1	648.000	5.639.752	1.420
	P1-2	375.000	3.241.792	603
	P1-3	192.000	1.643.032	364
	P1-4	81.000	681.472	111
	P2-1	117.912	4.410.944	1.345
	P2-2	107.811	4.019.679	1.232
	P2-3	52.728	1.906.624	464
	P2-4	20.577	704.969	142
	P3-1	36.501	3.307.949	1.354
	P3-2	27.783	2.460.375	1.085
	P3-3	20.577	1.771.561	653
	P3-4	14.739	1.225.043	402
	P4-1	14.739	2.352.637	1.133
	P4-2	10.125	1.520.875	1.021
	P4-3	6.591	912.673	433
	P4-4	3.993	493.039	157

Table 2.2. Number of interacting control point pairs. 3D examples.

Table 2.2 contains the number of interacting control point pairs and the time needed for their formulation. These pairs are determined according to the functions' support. After the identification of the interacting nodes per axis and by taking advantage of the structured isogeometric grid, the interacting nodes pairs are formed.

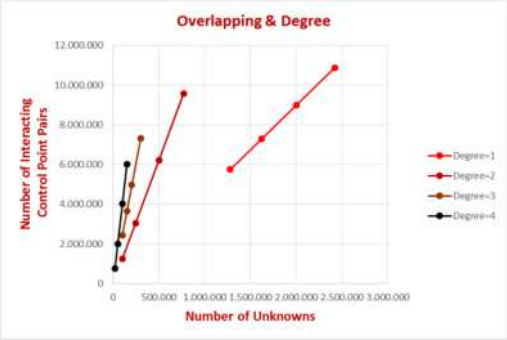


Figure 2.59. Influence of degree on the control point pairs' number. 2D problems.

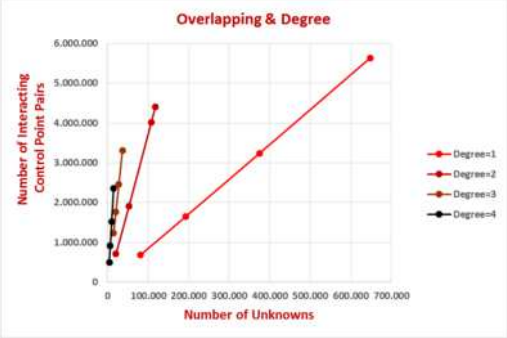


Figure 2.60. Influence of degree on the control point pairs' number. 3D problems.

The degree of the basis affects the overlapping between elements. This is obvious from **Figures 2.59** & **2.60** that depict the increase of the number of interacting control point pairs for different degrees of freedom & different degrees. As the degree increases, the number of interactions also increases. This leads to denser stiffness matrices and reduces the capability of the code to solve problems with high interconnection between elements and concurrently high number of unknowns for the same number of degrees of freedom. Linear basis functions create a geometric domain whose elements appear interconnectivity that is peculiar to the equivalent FEA mesh created by elements with linear functions, since only the adjacent elements interact.

3 CPUs & GPUs

3.1 CPUs

3.1.1 Sequential execution

Traditionally, computers had a single CPU that could run one software thread at a time. It is reminded that thread is the smallest sequence of programmed instructions that can be managed by the operating system. For the execution of a program in a single core CPU, the software is written for serial computation. Each problem should be broken into a discrete series of instructions, where they are implemented sequentially one after another. A computer has a clock, that controls the execution of the CPU. Each time the clock counts one unit, the CPU executes an instruction.

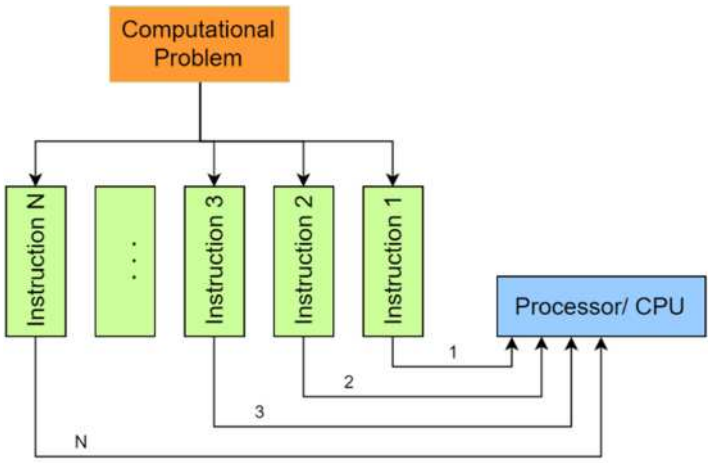


Figure 3.1. Execution progress of a sequential program.

3.1.2 Von Neumann architecture

Modern computers are based on the primitives that were expressed by the famous mathematician John von Neumann. These primitives constitute the Von Neumann architecture. It is about a mono processor system, where the microprocessor receives an input stream, executes the necessary processes, and sends the results in an output stream. Note that there is only one memory, where the input/ output data and instructions are saved. When the input data are more than computer can manage, the operating system enqueues them (multiplexing) and they are executing sequentially one after another. The communication between the components of the system (CPU, memory, input/ output devices) is achieved by groups of cables called buses. This procedure is known input-processing-output (IPO) or single instruction, single data (SISD).

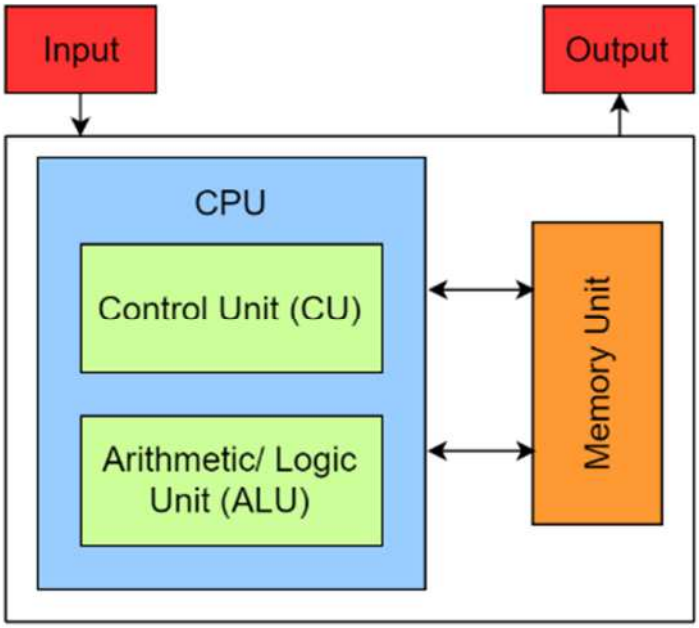


Figure 3.2. Von Neumann architecture.

In recent years, CPU clock rate has increased significantly. However, memory improvement that was achieved mainly concerns their storage capacity and not their transfer rate. Because of this mismatch between CPU and memory, many times is observed to waiting for data to be restored from the memory, while the CPU is keeping unused. Thus, the efficiency of the system configuration depends on the relationship of transfer rates between CPU and memory. The disadvantage of Von Neumann's architecture is known as von Neumann bottleneck.

3.1.3 Facing von Neumann's bottleneck-Genesis of parallel computing

The bypass of Von Neumann architecture disadvantages could be achieved by:

- Creating a cache between the CPU and the memory.
- Increasing the number of installed processors or the number of processing cores.
- Providing the capacity to processor to process instructions in parallel.

Over the years, microprocessor manufactures followed the third option which is noted above. The capacity of processor to run tasks in parallel has improved drastically. This is supported by the fact that the continuous duplication of processing structures such as Arithmetic and Logic Unit (ALU) and Floating point Unit (FPU), and the growing number of processing cores that are included in one single physical processor that have been observed the last few years.

For the full exploitation of these capabilities, that modern processors provide the code of a program, should be written for parallel computation. At first, by the selection of the algorithm, the program must be broken into discrete parts (tasks) that can be solved concurrently, having independence between them. Each part is broken down to a series of instructions that execute on different processors.

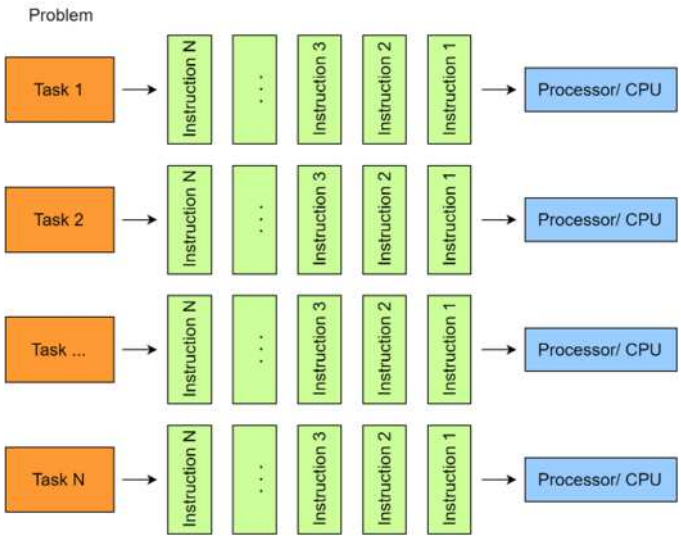


Figure 3.3. Execution progress of a parallel program.

Systems with more than one processors are amenable for parallel computing. In order to fully take advantage of the capabilities that parallel computing provides, the comprehension of hardware specifications is necessary.

There are two basic procedure to distribute tasks in systems with multiple processors:

- **Symmetrical multiprocessing (SMP)**: In this system there are more than one installed processors that are connected to a single, shared memory having access to all Input/ Output devices. This way each processor could be used in order to execute its individual task.
- **Asymmetric multiprocessing (AMP)**: This method was applied before SMP. The difference between SMP is that the input tasks are not distributed equally for each processor, but usually one processor act, as the main processor and scheduling the execution of tasks to the other CPUs.

Symmetrical multiprocessing proves as the best solution for exceeding von Neumann's bottleneck and it ensures the optimized utilization of resources. Each task is scheduled to be executed in any available processor. The operation system accomplishes the efficient distribution of workload to the CPUs.

Hyper-threading technology (HTT or HT)

Until now it has become reference to single and multiple processors with the assumption that one processor can implement one thread each time. Recently, a new technology has been developed called hyper-threading technology, which is used to improve parallelization of computations (doing multiple tasks at once) performed on microprocessors.

Hyper-threading technology was first developed by Intel as a simultaneous multithreading (SMT) implementation which permits multiple independent threads of execution to be used by an only one processor. With HTT, one processing core (or physical core) is used like two processors, permitting the scheduling of two tasks per core. For the operating system, one physical core appears as two different virtual (logical) cores, allowing concurrent scheduling of two processes per core that are enabled to use the same resources.

3.1.4 Parallelism

Task parallelism is the type of parallelism, where distributive tasks (set of instructions to carry out a unit of work) should be executed concurrently. The performed tasks are diverse and unrelated. For example, when a user might be reading an article on a website while playing music from his or her music library in the background, two processing units are working doing different works.

Data parallelism focusing on data collections/ structures like arrays and matrices. In contrast to the task parallelism, where different operations are performed on the same or different data in parallel, data parallelism makes concurrent the execution of the same operations, that are performed on different subsets of the same data structure. This type of programming have gained a lot of fame in the scientific community. Many applications from a great variety of fields have been developed in parallel systems and have used this kind of programming in order to reduce the computational cost.

C#-Task Parallel Library

In this thesis, the main programming language that was used is the C#. C# provides capabilities of task and parallel programming with the help of the Task Parallel Library (TPL) without worrying about thread scheduling. The main advantage of TPL is that provides scalability, thus the same program can be executed in two different systems considering the different hardware specifications and achieving the best utilization of resources. TPL gives the opportunity of effective data parallelism by using Parallel methods like Parallel.For. For example, the classical shape of a loop in C# would be:

```
for (int i = start_value; i < end_value; i++)
{
   //Instructions
}
```

It could be replaced by the:

```
Parallel.For(start_value, end_value, i =>
{
   //Instructions
});
```

In fact, Parallel.For is organizing the creation of parallel tasks. Each task will execute a specific number of iterations. TPL sets the optimal number of tasks based on the available resources and the workload per processor. Necessary condition for the successful execution of a Parallel.For is that each iteration should be independent from the others. It is important to note that the overhead for the scheduling of the parallel tasks is high and in order to achieve a speedup many instructions must be executed in each iteration.

3.2 Graphics Processing Units (GPUs)

Traditionally, GPUs are designed to handle and modify memory to accelerate the creation of images in a frame buffer intended for output to a display. Fundamental procedure of representation of a 3D model into a display is the graphics pipeline. Graphics pipeline refers to the sequence of steps used to create a 2D raster representation of a 3D scene. In terms of hardware, GPUs allow to manipulate simultaneously multiple data in a single stage of graphics pipeline by using parallel processing units. In response to commercial demand for real-time graphics rendering, the current generation of GPUs have evolved into many-core processors that are specifically designed to perform data-parallel computation. GPUs deliver tremendous computational performance (measured in GFLOPS) and high memory bandwidth.
The ratio of peak floating-point (double precision) calculation of GPUs is in the order of 1.5-3.0 TFLOPS versus 150 GFLOPS of CPUs. Similar difference between CPUs and GPUs is observed to the memory bandwidth, as is it shown in the following figures.

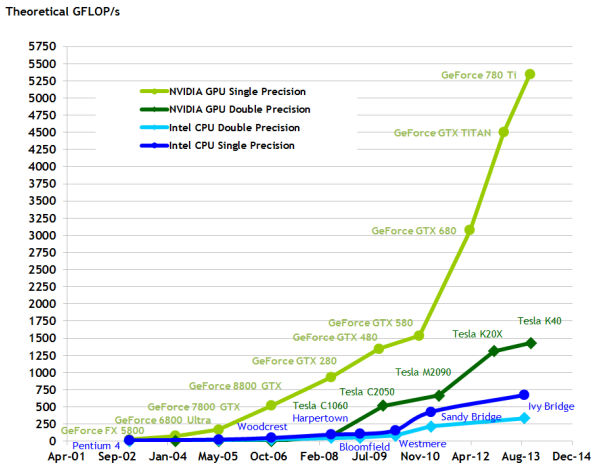


Figure 3.4. Floating-point operations per second for CPU and GPU.

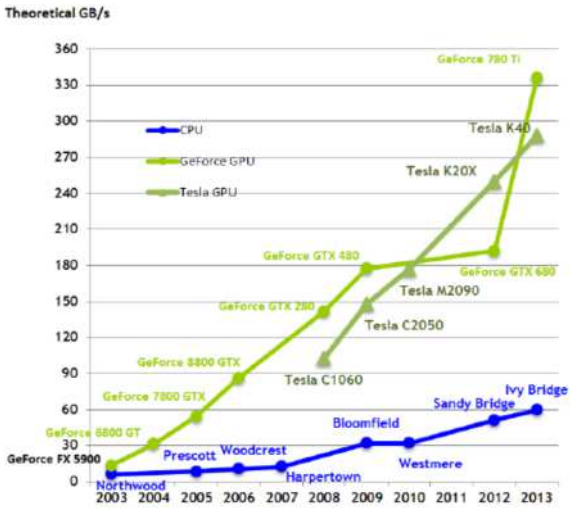


Figure 3.5. Memory bandwidth for CPU and GPU.

This difference is devoted on the different design between CPUs and GPUs. CPU design is oriented to optimize the implementation of serial code and minimize the execution of a single thread with large workload and with big divergent in the flow (if…else, loops). This approach is also called latency oriented, where latency is the time needed to perform a task. Thus, CPUs are equipped with structures, such as logic units and large cache memories, that reduce the latency by scheduling the execution of threads. That scheduling causes overhead and increases the total time for the procedures that combine the management of available resources.

On the other hand, GPUs as much as possible data, similarly separated across the processors, with small divergent in the flow. Since GPUs have independently been developed to perform data-parallel computation, allowing greater flexibility in graphics rendering as well as more generalized hardware, and permitting the same generalized hardware to perform not only different steps of the pipeline, like in fixed purpose hardware, but even in fields where data parallel implementation is applicable.

Last years, the scientific community has shown a great interest about the data-parallel processing capabilities that a GPU has. The quickly analysis of difficult and time consuming problems, that are objective in a data-parallel implementation with the help of GPUs now is a fact. Taking the advantage of multiple threads significantly speedup could be achieved. The usage of the breakthrough properties of GPUs for the solution of problems that concern a large variety of fields is referred to as general-purpose computation on GPUs (GPGPU).

3.2.1 GPU architecture

Modern GPUs and CPUs are based on the SIMD (single instruction, multiple data) architecture in order to perform data-parallel computations. In this model multiple processing elements/threads perform the same instructions on different pieces of data. This architecture, allows the fully exploitation of the most processing power as it lets the flow control to be shared among a specific number of threads. Note that in GPUs there are less flow control units for the scheduling of threads. This explains the reduced overhead that GPU threads have compared to CPU threads.

Figure 3.6. CPU vs. GPU. Architecture.
GPU devotes more cores to data processing.

In the **Figure 3.7**, it becomes obvious that GPUs and CPUs have many differences as regards their structure. CPUs devotes more circuits for the control flow and the cache memory, while is consisted of a few powerful cores. From the other hand, GPUs have more circuits devoted on data processing with many small cores. Each GPU core is much slower that the corresponding CPU, but it can works efficiently with the others, as the GPU architecture is optimized for massively parallel processing procedures.

It is generally approved that if for a problem a data-parallel algorithm could be found and the ratio of floating-point operations to memory accesses is quite large (≈ 10:1), then it is ideal for a GPU implementation.

A basic ingredient of the GPU architecture is that they are structured by many streaming multiprocessors (SM) who are consisted by streaming processors (SP). The number of SP per SM variates by the supported operations (or Compute Capability that will be analyzed later) that a GPU architecture provides.

A multithreaded program is partitioned into blocks of threads that execute independently from each other. This way GPUs break the main program to smaller "sub-problems" each of one is managed by the threads of a block. The control unit allocates thread blocks to Streaming Multiprocessors (SMs). This programming model provides scalability and efficient management of the available resources since a CUDA program can execute on any number of multiprocessors. It deserves to note that each GPU has its own independent memory different form the motherboard's RAM.

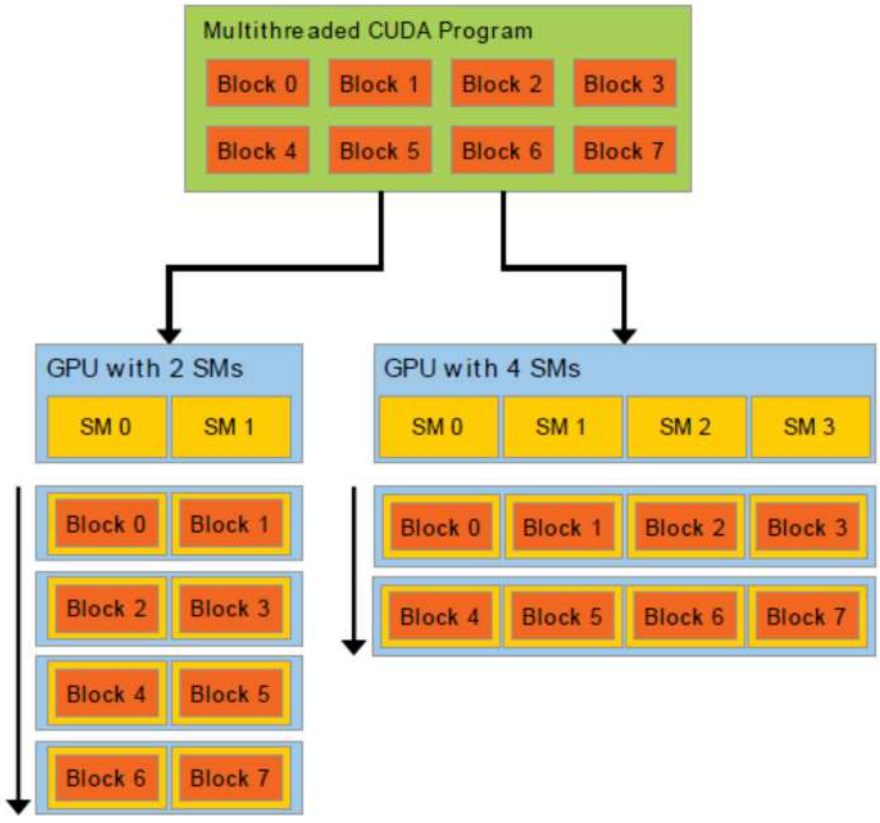


Figure 3.7. Automatic scalability provided by CUDA.

3.2.2 GPU programming

The usage of 3D computer graphics libraries like OpenGL and DirectX in order to implement general purpose parallel algorithms is extremely demanding and time consuming. That's because it is necessary to face any problems by using graphics operations. In November 2006, the initial release of CUDA-SDK gave a huge impulse for the development of general purpose computing. The name CUDA is the acronym for Compute Unified Device Architecture and it is used to describe the parallel computing platform and application programming interface (API) model created by Nvidia. It allows GPGPU to be performed on certain Nvidia GPUs. CUDA provides a few extensions to C and allows software developers to use C as a programming language. As mentioned earlier, in this thesis C# was selected as the main programming language for the implementations that will be follow. In order to procced to GPU computing on .NET framework, the community version of Alea GPU was used. Alea GPU is based on the CUDA technology and allows C# and F# programs to interact with GPUs. From now, the words CUDA and Alea GPU will be used as synonyms.

3.2.3 Compute capability

In order to achieve the maximum performance for a specific kernel the knowledge of the hardware specifications is necessary. For each CUDA compatible device, there are two numbers that feature its capabilities. The compute capability, which also sometimes called "SM version" comprises a major revision number *X* and a minor revision number *Y* and is denoted by *X.Y*. Devices with the same major revision number are of the same core architecture. The minor revision number corresponds to an incremental improvement to the core architecture, possibly including new features.

Technical specifications	Compute capability (version)						
	1.0	1.1	1.2	1.3	2.x	3.0	3.5
Maximum dimensionality of grid of threads blocks	2 (x, y)				3 (x, y, z)		
Maximum x-, y-, or z- dimension of a grid of threads blocks	65535					2^{31}-1	
Maximum dimensionality of thread block	3 (x, y, z)						
Maximum dimensions of a block	512 x 512 x 64				1024 x 1024 x 64		
Maximum number of threads per block	512				1024		
Warp size	32						
Maximum number of resident blocks per multiprocessor	8					16	
Maximum number of resident warps per multiprocessor	24		32		48	64	
Maximum number of resident threads per multiprocessor	768		1024		1536	2048	
Number of 32-bit registers per thread per multiprocessor	8 K		16 K		32 K	64 K	
Maximum number of 32-bit registers per thread	128				63		255
Maximum amount of shared memory per multiprocessor	16 KB				48 KB		
Constant memory size	64 KB						
Double precision	No			Yes			

Figure 3.8. Compute capabilities.

The GPU that used in this thesis is the NVIDIA GeForce GTX 670M, which has a 2.1 compute capability.

3.2.4 Threads organization

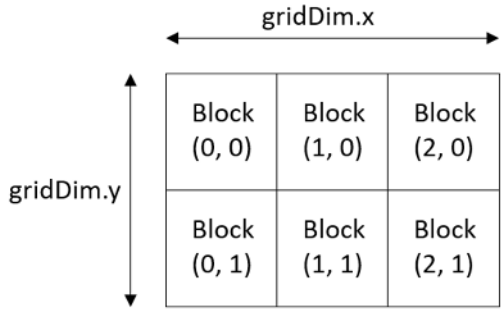


Figure 3.9. Grid organization in block of threads.

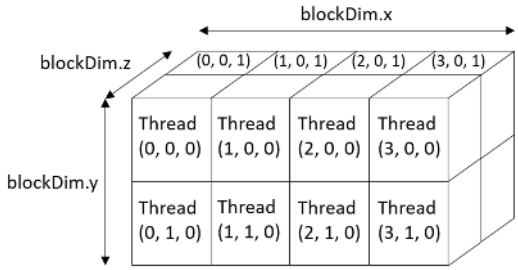


Figure 3.10. Block organization in threads.

CUDA allows the programmer to define functions, called *kernels* that contain the instructions that are executed on the GPU. Each time a CUDA kernel is launched parallel blocks of threads are called. These parallel blocks are organized in a structure called grid. Grid is organized as a 2D array of blocks, where any block could be referred by using two indexes blockIdx.x and blockIdx.y. Each of these blocks could have up to a specific number of threads which is limited by the compute capability of the device. As in blocks, in order to refer in a thread of a block, some indices should be called. The only difference is that threads of a block are organized in a 3D array and for each one there are threadIdx.x, threadIdx.y and threadIdx.z. It is important to note that, if the nature of the problem allows, blocks can be organized in 1D array and threads in a 2D or in a 1D.

Threads are organized in structures called warps. Warps only depend on the GPU architecture and their size is given and standard, thus, it is independent from the grid and the block size. Specifically, the size of each warp is equal to 32 threads.

Warps are used for thread scheduling in SMs. Once a block is assigned to a streaming multiprocessor, it is divided into 32-threads units called warps.

Supposing, three blocks assigned on a SM, that each of them has 256 threads. There will be 256/32 = 8 warps. Considering that the maximum number of threads per SM is 1024 there will be available 1024/32 = 32 warps. The homogeneity of the threads in a warp ensures high hardware utilization.

If the threads of the blocks, which are assigned on the same SM, are executing the same instruction, then all the warps can execute the same instruction in parallel. But, what happens in the case where one or more threads in a warp execute different instructions? CUDA programming model has predicted this case by partitioning the warp into groups of threads based on the instructions being executed. These groups are executed in a serial order.

In this way, there is always workload for the SM and while some warps make time consuming processes like memory accesses, others are selected for the execution. The selection of ready warps for execution is referred as zero-overhead scheduling because this procedure does not increase the execution time.

3.2.5 Structure of CUDA program

Usually CPU and GPU are referred as host and device. Thus, the code which amenable to data parallelism is implemented in the device. As mentioned earlier, the functions that contain code that will be executed on the GPU called kernels. Devices cannot read and write data directly in the computer‘s memory and vice versa. Data between host and device are transferred with some build in CUDA functions.
A typical structure of a CUDA program could be described by the following steps:

1. Allocate GPU memories.
2. Transfer data from CPU memory to GPU memory.
3. Definition of the grid’s features (number of blocks and threads).
4. Invoke or launch the CUDA kernel to implement the parallel code. In this step a large number of threads are generated and organized to structured called grid.
5. Copy data from GPU to CPU memory.
6. Deallocate data from GPU memory.

3.2.6 GPU memories

GPU devices have a great variety of memories each one with different scope, lifetime and caching behavior. The **Figure 3.11** represents the GPU memories. These memories can be sorted by their position on the device. The way they can be accessed can be seen with the help of the arrows which indicate the data flow.

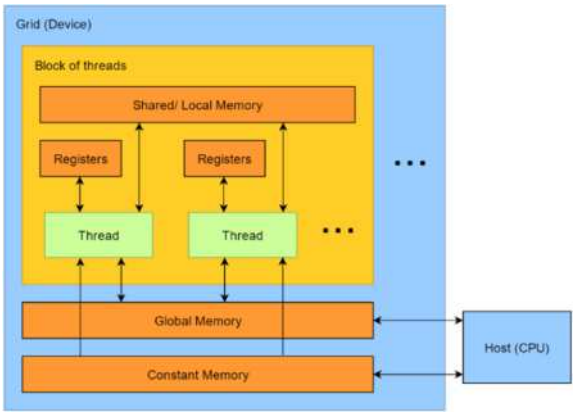


Figure 3.11. GPU Memories.

The table below presents the main types of memory, their access (R/ W) and their scope.

Memory	Location	Access	Scope
Register	On-chip	Read/Write	One thread
Local	Off-chip	Read/Write	One thread
Shared	On-chip	Read/Write	All threads in a block
Global	Off-chip	Read/Write	All threads + host
Constant	Off-chip	Read	All threads + host
Texture	Off-chip	Read (CUDA 2.1 and previous)	All threads + host

Figure 3.12. Block organization in threads.

3.2.6.1 Global memory

Global memory is responsible to perform the interaction between host and device. For the execution of a kernel on the device data from the host must be transferred to the device memory and when the execution of kernel finishes, the result data are transferred from GPU to CPU. Every GPU comes with her own dynamic random access memory (DRAM), which is usually large in size (GB). Part of GPU DRAM memory is the global memory. It is off-chip memory and it is much slower than the other GPU memories.

Memory access efficiency has a very important impact on the speed up, that can be achieved for a CUDA program. Often, global memory accesses limits the highest floating calculation bandwidth. For this reason, it is necessary to refer to the CGMA, which is the ratio of the number of floating point operations for each access to global memory. Supposing a GPU, that supports 100 GB/s of global memory accesses and the code demands 8 bytes to be load from the global memory. Then, CGMA becomes equal to 100/8 = 12, 5 gigaflops which is only a tiny part of the peak performance of 150 GFLOPS of a typical GPU.

Note that compared to the computer's RAM memory GPU's global memory is approximately 10 times faster (≈ 120 GB/s for Compute Capability1.3). However, global GPU memory is quite slow demanding a hundreds of clock cycles (≈500-700 clock cycles) to access memory addresses. This problem has a great effect on GPUs with oldest architectures, but, in newest GPU architectures like Fermi it seems to be disappeared since this architecture allows to global memory to be cached and as a result the number of clock cycles for reading data is reduced.

Global memory in CUDA is implemented with dynamic access memories (DRAMs). Data bits are stored in DRAM cells. Each time a location is accessed, many consecutive locations that includes the requested location are accessed. These consecutive locations also called burst. The size of each burst is 128 bytes and can be accessed by a warp in a single transaction. A best practice would provide only the access of the data that will be used. This is often is referred as coalesced memory access or memory coalescing. When the memory access is sequential and aligned then it is coalesced.

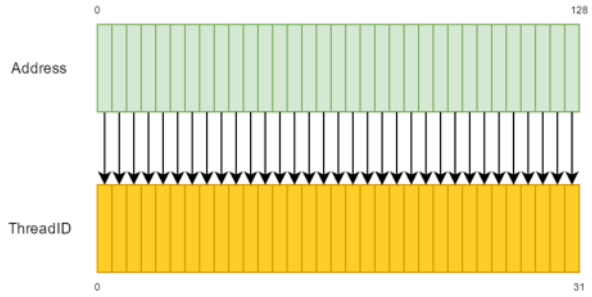


Figure 3.13. Aligned and Consecutive Access.

Sometimes, uncoalesced memory access is unavoidable. This could happen if the alignment is interrupted or the serial memory accesses are not possible (only for older GPUs). A classical case is when the memory access is sparse, this could drive to more than one transactions with a negative impact on the total time of execution.

For example, in the following figure the memory access is consecutive but misaligned. Two transactions are required. The first will load the first 31 DRAM cell, and the second will load the final which have been marked with red color.

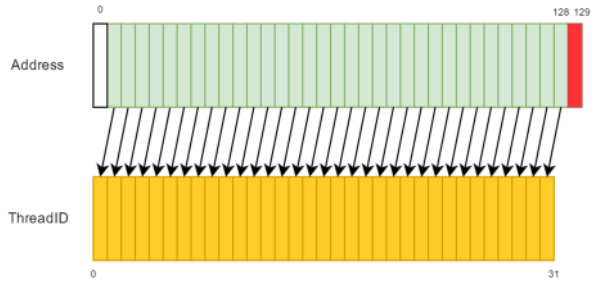


Figure 3.14. Unaligned and Consecutive Access.

3.2.6.2 Shared memory

Shared memory is located on the circuits of each SM (on-chip) and provides high memory bandwidth (TB/s) but it has a limited size (14-48 KB).
Every thread in a block can communicate and collaborate on computations with every thread located in the same block by using shared memory which is only visible from threads belonging on the same block. Shared memory is disturbed in an equivalent way for each block. Thus, it is possible to define the size of shared memory a block uses. Shared memory is ideal for caching data common for threads of a block. Also, a common technique, where all threads of a block need access to a subset of the values, is to have the threads cooperate in loading the values in the shared memory.

3.2.6.3 Constant memory

Constant memory is used only for reading data. Reading from constant memory the input data is more beneficial than reading from the global memory and can reduce memory bandwidth which many times occurs bottlenecks. The size of constant memory is small (64KB) but it is suitable for cashing data and the consecutive reads of the same address will not incur any additional memory traffic.

3.2.6.4 Registers

Registers are used in order to store automatic scalar (not arrays) variables. The scope of registers are within individual threads. They are the fastest memory type (TB/s) with shared memory in the GPU and they limited in number. Each SM can accommodate specific number of registers which is determined by hardware specifications. One SM can accommodate more blocks if they require few registers and fewer blocks if they require more registers. Thus, extended use of registers should be avoided because can result in a dramatic reduction in parallelism and performance achieved. For example, suppose a GPU of capability 1.0 that can accommodate 768 threads and 8192 registers per SM, the maximum number of registers per threads would be 8192/768 = 10. If the number of registers exceed this number, the number of concurrently executed threads will be decreased.

3.2.6.5 Other memories (texture & local)

Texture memory is another type of read-only memory that can improve performance and reduce global memory bandwidth. It was firstly used for traditional graphics operations but it usage have been extended for GPGPU computing.
Another type of memory which is also referred as local memory concerns the variables that cannot fit in the registers and they bound address space from the global memory and the kernel execution becomes slower.

3.2.7 Occupancy

The occupancy of a streaming multiprocessor is defined as:

$$\frac{Warps_{effective}}{Warps_{maximum}}$$

Where Warpseffective represents the number of effective warps and Warpsmaximum corresponds to the number of warps that could be potentially exploited. The maximum number of warps is specified by the hardware specifications of the GPU. The number of effective warps is based on the requirements of the program. Specifically it depends on:

- The compute capability.
- The number of registers per thread.
- The number of threads per block.
- The shared memory configuration (bytes per block).
- The maximum number of threads per single streaming multiprocessor.

The main aptitude of occupancy is to keep the streaming multiprocessors in actions and hiding the latency which is caused by arithmetic operations or memory accesses.
It should be stressed that the occupancy is not always identical to the performance of a code. Many times the key factor for the best performance is the optimization of the memory accesses which could be achieved by applying some optimization techniques.

3.2.8 Heterogeneous programming

Many CUDA functions are asynchronous to the CPU. This means that after the calling of these functions the control flow is returned immediately to the CPU which executes the rest host code. Usually, kernels which contain the device code, have such behavior. This could be easily understood in the **Figure 3.15** by imagining that the device code overlaps the host code, which follows after the launch of the kernel.

Except from kernels, the following commands have asynchronous behaviors:

- Memory copies between two addresses to the same device memory
- Memory copies performed by functions with the «Async» suffix
- Memory set functions calls

It is important to note that CUDA provides functions for the exact measurement of the execution time of a kernel since due to heterogeneous programming the usage of CPU timers sometimes is unreliable (except from the case where threads of CPU and GPU had been synchronized).

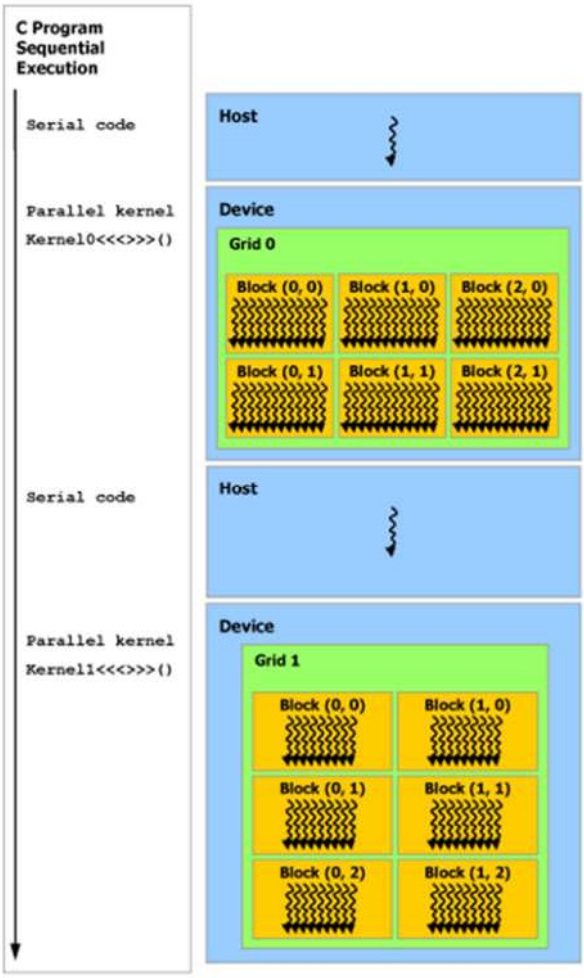


Figure 3.15. Heterogeneous Programming.

4 Memory Management

4.1 Virtual Memory

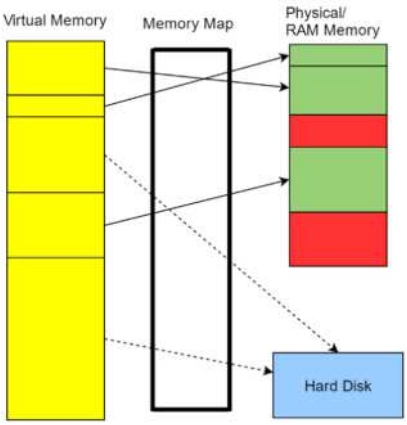


Figure 4.1. Virtual memory.
It combines RAM memory and hard disk's storage space
to form a large range of continuous addresses.

Necessary condition for the execution of a computer program is the existence of the memory. As it is known, each computer has at least one random access memory (RAM) (also called physical memory) that is used for the storage of temporary data needed through the execution. In contrast to hard disks, which are used for the definite saving of data, RAM memories are much faster. However, despite their capability to read and write data with high bandwidths, they have limited size compared to hard disks. This fact can lead the system to run out of memory. For this reason, a memory management technique, which uses a generalized memory, called virtual memory, have been developed. Virtual memory is implemented using both hardware (RAM Memory, Hard Disk) and software (Operating System).

In this technique, there are two different types of memory addresses, the physical and the virtual addresses. When the physical memory is full, the Operating System (OS) moves groups of data from RAM to hard disk through a process that maps the physical addresses to virtual ones. The temporary groups of data that will be stored in the hard disk are also called paged files and the corresponding process paging.

An important difference between 32-bit and 64-bit OS versions is that they allow different sizes of virtual address space (VAS). So, when a new application is executed on a 32-bit OS, the process has a 4 GB VAS. On a 32-bit Microsoft Windows installation, by default, only 2 GB are made available to processes for their own use. The other 2GB are used by the operating system. Contrary, on 64-bit executables which run on Microsoft Windows the limit of VAS is constrained up to 8 TB (for 64-bit executables in Windows 8) and up to 128 TB (for 64-bit executables in Windows 8.1 and later). From the previous analysis, it is obvious that the type of executable (32 or 64-bit) which will be used, is critical for the efficiency of a FEA/ IGA program and affects drastically its ability to solve large scale problems.

4.2 Time & Space Complexity

Time and space complexity of an algorithm quantifies the amount of time and storage space demanded by the algorithm to run in relation to the size of the input arguments. In order to represent these complicities big *O* Notation (*O()*) is used. The complexities are described by the higher terms of the input arguments supposing that their size tends to be infinite. For this reason, powers of the input argument are ignored as well as different logarithm's bases supposed to be equivalent.

Time complexity is measured in terms of number of operations rather than computing time in order to be independent from hardware specifications.

It is acceptable that:

$O(1)<O(logn)<O(n)<O(nlogn)<O(n^m)<O(m^n)<O(n!)<O(n^n)$

where m is some constant.

In **Figure 4.2**, classical algorithms and their time complexity are shown.

Notation	Name	Example algorithm
O(1)	Constant	Accesing a dense array element.
O(logn)	Logarithmic	Finding an item in a sorted array with a binary search.
O(n)	Linear	Finding an item in an unsorted array.

Figure 4.2. Time complexity of classical algorithms.

4.3 Handling of Matrices

Matrix storage and matrix operations are important performance factors for large scale simulations. Undoubtedly, the optimized storage of the stiffness matrix is a basic ingredient for the efficiency of a FEM/ IGM code. There is a large variety of different formats for the handling of stiffness matrices, that provide efficient storage and ease the solution of the equation system, which are based on their properties. Thus, a stiffness matrix is:

- Symmetric. The symmetric matrix structure allows storage of the symmetric half of the matrix and the handling of the coefficients of this part.
- Sparse. Sparseness of the matrix drastically decreases storage requirements and operation count in comparison to the dense matrix.
- Positive-definite. Positive definiteness means that larger coefficients are grouped near the main diagonal.

The structure grid that the isogeometric method provides, allows the numbering of nodes in a way that maximize the number of non-zero coefficients, that are located near the main diagonal of the matrix. Each format has different strengths and weaknesses. The dense format, for example, is the easiest format to implement and supports any kind of operation. However, it requires the most storage space since all the coefficients, zero and non-zero, are stored.

In order to overcome the problems that are generated from the storage of non-zero coefficients, another format called sparse, have been developed. Sparse matrices store the minimum possible entries. Nevertheless, they require higher indexing time and can perform only specific operations efficiently.

The choice of the right storage format is depended on many factors since it affects the performance of the code and its ability to solve problems with large number of unknowns.
It is a fact that many times the solution phase sets the storage format. For example, if the solution method requires factorization, skyline format is favorable. But, in case where a sparse solver have been chosen, sparse matrix format constitutes the optimized option.
It is worth to note that many formats provide the capability of transformation to other formats, fact which is very important and allows to use the most suitable shape for every part of the program (e.g. stiffness matrix assembly and solution phase).

4.3.1 Dense Matrix

Dense matrices store every entry in the matrix. If a dense matrix has m rows and n columns all the $m \cdot n$ coefficients will be stored or, in other words, the space complexity is $O(m \cdot n)$. This is the most classical format and supports all the operations. Note that dense matrix formats allow constant time-$O(1)$ access per element. The space that a dense matrix occupies is proportional to the number of coefficients it considers. Knowing the type of the computer number format which have been used for the definition of the matrix, the calculation of the total size in bytes could easily achieved by multiplying the number of total entries by the number of bytes that the computer number format occupies.
It is useful to know that some programming languages are zero-based (like C, Java), i.e. the indexes of an array of size n range from 0 to n−1, whereas others use one-based numbering (such as MATLAB, FORTRAN), i.e. the indexes of an array of size n range from 1 to n.
Many times it is desirable to transform the multiple dimensional dense matrices into vectors. This could be achieved by using either the row-major order or the column-major order.

4.3.1.1 Row-major & column-major orders

Row-major order and column-major order describe methods for arranging multidimensional arrays in linear memory.
In row-major order, sequential elements of the rows of the array are continuous in memory. This makes row major order ideal for row-oriented operations since due to

caching, the accessing of elements that are continuous in memory is faster than accessing other elements.
In column-major order, sequential elements of the columns of the array are continuous in memory. This format is more suitable for column-oriented operations.

Many programming languages in order to support multidimensional arrays automatically transform them to row-major or column-major order.
C-style programming languages (C, C++, C#) use the row-major order. Contrary, MATLAB, FORTRAN and OpenGL use the column-major order. The two different approaches are applied on the following example.
The matrix below will be stored, as follow, in the two orders.

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	A_{33}

(a) Storage.

	Row-major order															
Address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	A_{00}	A_{01}	A_{02}	A_{03}	A_{10}	A_{11}	A_{12}	A_{13}	A_{20}	A_{21}	A_{22}	A_{23}	A_{30}	A_{31}	A_{32}	A_{33}

(b) Row-major order.

	Column-major order															
Address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	A_{00}	A_{10}	A_{20}	A_{30}	A_{01}	A_{11}	A_{21}	A_{31}	A_{02}	A_{12}	A_{22}	A_{32}	A_{03}	A_{13}	A_{23}	A_{33}

(c) Column-major order.

Figure 4.3. Matrix storage.

It is easy to formulate the index of entry *[row, column]* for the two orders.

*index=column+row*ColumnNumber* for the Row-major order

and

*index=row+column*RowNumber* for the Column-major order

Note, that with the suitable transformation in the relationship between *index* and *[row, column]*, similar types can be produced for multiple dimensional arrays.

4.3.2 Symmetric skyline matrix

The skyline is an envelope that begins with the first nonzero coefficients in each column of the matrix. In skylining, only the coefficients between the main diagonal and the skyline are stored (normally by successive columns) in one-dimensional array. Symmetric Skyline matrix is also called variable band matrix storage. Reminded that, a matrix is banded if the nonzero terms of the matrix are gathered about the main diagonal.
Supposing a matrix shaped in the skyline format with *N* entries.
A matrix of size *N·N* could be fully described by two arrays:

- *ColumnPointer[N+1]* – pointer array for columns.
- *Values[ColumnPointer[N]]* – array containing the matrix coefficients.

The length of the *i-th* column is given by *ColumnPointer[i+1]-ColumnPointer[i]*. The length of the *Values* is equal to *ColumnPointer[N]*.

	0	1	2	3	4	5	6	7
0	●	●		●	●			
1	●	●	●	●	●	●		
2		●	●		●	●		
3	●	●		●	●			
4	●	●	●	●	●	●	●	●
5		●	●		●	●	●	●
6					●	●	●	●
7					●	●	●	●

(a) Fill of matrix.

0	1		5	9			
	2	3	6	10	14		
		4	7	11	15		
			8	12	16		
				13	17	19	22
					18	20	23
						21	24
							25

(b) Storage.

Figure 4.4. Skyline format.

For the shape of the matrix above the *ColumnPointer* would be:

ColumnPointer={0, 1, 3, 5, 9, 14, 19, 22, 26}

The first entry is always zero and the last entry contains the total number of coefficients. For the formulation of a stiffness matrix the *ColumnPointer* vector can be produced by using the connectivies matrices of finite elements.

Knowing the position *(row , column)* of a matrix coefficient, it is easy to define its value by using the *ColumnPointer*. The transition from two-index notation *(row, column)* to notation for a one-dimensional array could be achieved by the following type:

(row, column) → row - (ColumnPointer[column+1]- ColumnPointer[row]+1)

Since an analytic relation between two-index notation to one-dimension array have found, the time-complexity for accessing a single element is *O(1).*

4.3.3 Sparse matrix

Sparse matrices store only the non-zero entries. In order to take the advantage of sparse matrix format, it is necessary the zero entries of the corresponding dense matrix to constitute a great percentage of the total number of the entries. Beyond the fact that sparse matrix formats reduce storage requirements, they provide an excellent behavior in terms of the operations they support. Needless calculations between zero entries are avoided. Operations using standard dense-matrix structures and algorithms are slow and inefficient when applied to large sparse matrices. For this reason, many efficient algorithms that involve sparse matrices have been developed both for the solution phase (sparse solvers) and for operation processes. The configuration of sparse matrix includes more overhead than in dense matrices. That happens because there is the need to store indexes that will be used in order to indicate the positions of non-zero entries. The formats that are used for the sparse storage of a matrix can be divided into two groups:

- Those that support efficient modification, such as DOK (Dictionary of keys) or COO (Coordinate list). These are typically used to construct matrices.
- Those that support efficient access and matrix operations, such as CSR (Compressed Sparse Row) or CSC (Compressed Sparse Column).

4.3.3.1 Coordinate List (COO)

COO stores a list of (row, column, value) tuples. The entries can be added in any order. Also, duplicate entries are allowed. Generally, COO format does not allow lookups except from case that the entries are sorted (by row index, then column index). It is worth to note that COO format supports very fast conversion to and from CSR/CSC formats that support fast arithmetic and matrix vector operations. A matrix in COO format, with *N* entries, can be constructed by the following three arrays:

- *RowIndices [N]* - contains the rows where there is non-zero matrix coefficient
- *ColumnIndices [N]* - contains the columns where there is non-zero matrix coefficients.
- *Values [N]* - contains all non-zero entries.

COO format is ideal for incremental matrix construction. If the elements are placed one after another at the final matrix the access time will be constant and independent from the total size of the matrix.

	0	1	2	3	4
0	●	●			
1		●			
2	●			●	●
3			●		●

(a) Fill of matrix.

A_{00}	A_{01}	0	0	0
0	A_{11}	0	0	0
A_{20}	0	0	A_{23}	A_{24}
0	0	A_{32}	0	A_{34}

(b) Storage.

Figure 4.5. COO format.

For the shape of the matrix of Figure 4.5.a the three arrays become:

- *RowIndexes={2, 0, 0, 3, 2, 1, 3, 3}*
- *ColumnIndexes={3, 0, 1, 4, 0, 1, 2, 4 }*
- *Values={* A_{23}, A_{00}, A_{01}, A_{34}, A_{20}, A_{11}, A_{32}, A_{34}*}*

4.3.3.2 Dictionary of keys (DOK)

A Dictionary is a data structure that represents a collection of keys and values pair of data. The key is identical in a key-value pair and it can have at most one value in the dictionary, but a value can be associated with many different keys.
Operations associated with this data type allow:

- the addition of a pair to the collection
- the removal of a pair from the collection
- the modification of an existing pair
- the lookup of a value associated with a particular key

In the case it is desirable to implement a Dictionary in order store an array, the keys would be *(row, column)* that will correspond to a specific entry of the Values matrix.
The two major approaches to implement dictionaries are the hash tables or the search trees.

Hash tables implementation

Basic issue for the implementation of a hash table is the hash function which is responsible for the association of each index with a specific “address” (hash). This way, it is possible to find each value with only knowing an index. The main performance factor of a hash table is the choice of the hash function. The cost for each lookup is independent of the number of elements that are stored in the table, and as a result, they allows efficient constant time *O(1)* access of individual elements.

In most cases, programming languages, provide classes of dictionaries that have no need of initialization. They provide automatically increment of their size with the addition of extra elements. In this way the computational effort for the creation of indexing arrays is eliminated. For the description of the following table the key pairs are formed with syntax (x, y) as a table A which contains the corresponding values.

	0	1	2	3	4
0	●	●			
1		●			
2	●			●	●
3			●		●

(a) Fill of matrix.

A_{00}	A_{01}	0	0	0
0	A_{11}	0	0	0
A_{20}	0	0	A_{23}	A_{24}
0	0	A_{32}	0	A_{34}

(b) Storage.

$(0, 0) \rightarrow A_{00}$	$(2, 3) \rightarrow A_{23}$
$(0, 1) \rightarrow A_{01}$	$(2, 4) \rightarrow A_{24}$
$(1, 1) \rightarrow A_{11}$	$(3, 2) \rightarrow A_{32}$
$(2, 0) \rightarrow A_{20}$	$(3, 4) \rightarrow A_{34}$

(c)

Figure 4.6. DOK format.

Sparse matrix formats for operations

For the solution face it is advisable to deal with a Compressed Sparse Row (CSR) or Compressed Sparse Column format (CSC). These formats allow fast row access and matrix-vector multiplications. CSR format represents a matrix by three (one-dimensional) arrays that respectively contain nonzero values, the extents of rows, and column indices. It is similar to COO, but compresses the row indices, hence the name. Contrary, the three arrays that determine a CRC format contain the non-zero values, the row indices and the extents of columns. The main difference of the two format concerns their performance on specific operations. Thus, the CSR format features efficient row slicing and fast matrix vector products but slow column slicing operations, where the CSC format excels and vice versa. Note that matrix-vector products are also efficient with CSC, though the CSR format may be faster.

4.3.3.3 Compressed sparse row (CSR)

In this format the matrix is represented by three arrays:

- *RowPointer[N+1]* - integer pointer array for rows
- *ColumnPointer[RowPointer[N]]*-contains the column numbers for nonzero matrix entries
- *Values[RowPointer [N]]* – array of doubles containing matrix coefficients.

The *i-th* element of *RowPointer* is the address of the first entry of the row. The number of entries in the *i-th* row is given by *RowPointer[i+1]-RowPointer[i]*. The length of arrays *ColumnPointer* and *Values* is equal to *RowPointer*.

	0	1	2	3	4	5	6	7
0	●	●		●	●			
1	●	●	●	●	●	●		
2		●	●		●	●		
3	●	●		●	●			
4	●	●	●	●	●	●	●	●
5		●	●		●	●	●	●
6					●	●	●	●
7					●	●	●	●

(a) Fill of matrix.

0	1		2	3			
4	5	6	7	8	9		
	10	11		12	13		
14	15		16	17			
18	19	20	21	22	23	24	25
	26	27		28	29	30	31
				32	33	34	35
				36	37	38	39

(b) Storage.

Figure 4.7. CSR format.

The matrix shown in Table 4.7. can be described with the following arrays:

RowPointer=[0 4 10 14 18 26 32 36 40]

ColumnPointer=[0 1 3 4; 0 1 2 3 4 5; 1 2 4 5; 0 1 3 4; 0 1 2 3; 4 5 6 7; 1 2 4 5 6 7;
4 5 6 7; 4 5 6 7]

Since one row of the *ColumnPointer* could be supposed as a sorted array, accessing an element which belongs in a CSR matrix type demands a binary search of the *ColumnPointer* for this. Thus, the row will be indexed by *O(1)* time, while the column will be indexed in *O(RowPointer[i+1]-RowPointer[i])* time.

Compressed sparse column (CSC)

In this format the matrix is represented by three arrays:

- *ColumnPointer[N+1]* - integer pointer array for columns
- *RowPointer[ColumnPointer[N]]*-contains the row numbers for nonzero matrix entries
- *Values[ColumnPointer [N]]* – array of doubles containing matrix coefficients.

The *i-th* element of *ColumnPointer* is the address of the first entry of the column. The number of entries in the *i-th* column is given by *ColumnPointer[i+1]- ColumnPointer[i]*. The length of arrays *RowPointer* and *Values* is equal to *ColumnPointer*.

	0	1	2	3	4	5	6	7
0	●	●		●	●			
1	●	●	●	●	●	●		
2		●	●		●	●		
3	●	●		●	●			
4	●	●	●	●	●	●	●	●
5		●	●		●	●	●	●
6					●	●	●	●
7					●	●	●	●

(a) Fill of matrix.

0	4		14	18			
1	5	10	15	19	26		
	6	11		20	27		
2	7		16	21			
3	8	12	17	22	28	32	36
	9	13		23	29	33	37
				24	30	34	38
				25	31	35	39

(b) Storage.

Figure 4.8. CSC format.

The matrix shown in Table 4.8. can be described with the following arrays:

ColumnPointer=[0 4 10 14 18 26 32 36 40]

RowPointer=[0 1 3 4; 0 1 2 3 4 5; 1 2 4 5; 0 1 3 4; 0 1 2 3; 4 5 6 7; 1 2 4 5 6 7;
4 5 6 7; 4 5 6 7]

Since one row of the *RowPointer* could be supposed as a sorted array, accessing an element which belongs in a CSC matrix type demands a binary search of the *RowPointer* for this column. Thus, the column will be indexed by *O(1)* time, while the row will be indexed in *O(ColumnPointer [i+1]- ColumnPointer [i])* time.

4.3.4 Time complexities per matrix format

The stiffness matrix format has a great impact on the performance of a code based on the isogeometric method.

It is worth mentioning that in this work the performance of the matrix formats is investigated only for the phase of assembling and no for the solution phase.

In the following table, there is a synopsis of time complexity of accessing an element for different matrix formats that can give a rough approximation for the efficiency of each type.

Matrix Format	Time complexity
Dense	*O(1)*
Symmetric Skyline	*O(1)*
Coordinate List (COO)	*O(1)*
Dictionary of Keys (DOK)	*O(1)*
Compressed Sparse Row (CSR)	*O(logn) where n is the number of non-zero elements of each row*
Compressed Sparse Column (CSC)	*O(logn) where n is the number of non-zero elements of each column*

Figure 4.9. Synopsis of the time complexity of accessing an element. Different matrix formats.

Thus, dense, symmetric skyline, COO and DOK have the same time complexities providing quicker accessing times for each element than CSR and CSC. Thus, they are ideal for stiffness matrix assembly. However, after the stiffness matrix assembly these formats should be converted to CSR or CSC since they are more efficient for the execution of arithmetic operations.

Note that the time complexity which concerns the coordinate list is referred only in the case of adding new elements in the total matrix. Contrary, in the case of looking up an element it will be equal to *O(N)*, where *N* is the size of the matrix.

5 Refinement

5.1 Introduction

NURBS geometries lie at the forefront of designing technology, as they provide numerous possibilities for representation, while supplying a reliable basis, which can be easily adjusted for the purpose of analysis. Thus, it is commonplace to use a more complex mesh to obtain better analysis results.

The transition from a coarse mesh to a fine one preserves the geometrical features of the model, while providing the designer with the ability to modify specific parts of the entity. In order to use efficiently refinement in analysis, the engineer has to understand the principles involved and the ways in which the basis is altered. Refinement is an automated process that requires minimal manual effort, but complicated nonetheless.

The combination of a limited number of control points with a low polynomial order is the optimum solution for a designer. This coarse mesh provides exact geometrical representation with reduced computational cost. It provides a certain level of understanding and control for the user, who can comprehend the simplified patterns of the basis and control net. The creation of stiffness matrix is also less time-consuming, due to the limited number of degrees of freedom and interconnections involved.

The coarse mesh, however, has flexibility issues. This means that each control point affects a rather large part of the model, so that small changes to control point variables reflect to significant changes in geometry. Moreover, basis function overlapping is reduced in coarse meshes. This makes a coarse mesh unsuitable for analysis. A feasible approach is the introduction of a coarse mesh for design, and afterwards refinement of this mesh for analysis purposes.

With the creation of a finer mesh, a detailed, flexible control net is introduced. Each control point affects a smaller portion of the model and the number of interconnections is usually increased. Mapping from parameter to physical space remains unchanged; this is very important in terms of analysis.

Stiffness matrix calculation is more time-consuming, but accuracy per degree of freedom is also improved.

There are three ways, in which a B-SPLine basis can be enriched. A different knot value vector may be selected, the polynomial order may be increased or a combination of both may occur; the raise of the polynomial order followed by the introduction of a richer knot vector. These techniques are referred to as h-, p- and k-refinement respectively.

5.2 Knot Insertion

Knot value insertion is the introduction of a finer mesh by enrichment of the existing knot value vector Ξ. A new knot value vector $\overline{\Xi}$ is created, such that $\Xi \subset \overline{\Xi}$. Only internal knot values can be added; the boundaries have to remain intact. A new set of B-SPLine basis functions is created. The coarse mesh representation is evaluated, as usual, by:

$$C(\xi) = \sum_{j=1}^{n} \left\{ N_{j,p}(\xi) \cdot X_j \right\} = \underset{1\times n}{\left\{ N(\xi) \right\}^T} \cdot \underset{n\times 1}{\left\{ X \right\}}$$

whereas the new representation

$$C(\xi) = \sum_{i=1}^{m} \left\{ \overline{N}_{i,p}(\xi) \cdot \overline{X}_i \right\} = \underset{(1\times m)}{\left\{ \overline{N}(\xi) \right\}^T} \cdot \underset{(m\times 1)}{\left\{ \overline{X} \right\}}$$

Therefore, the new set of control points $\left\{ \overline{X} \right\}$ has to be defined. This can be achieved with the evaluation of a transformation matrix, so that:

$$\underset{(mx1)}{\left\{ \overline{X} \right\}} = \underset{(m\times n)}{\left[T^p \right]} \cdot \underset{(n\times 1)}{\left\{ X \right\}}$$

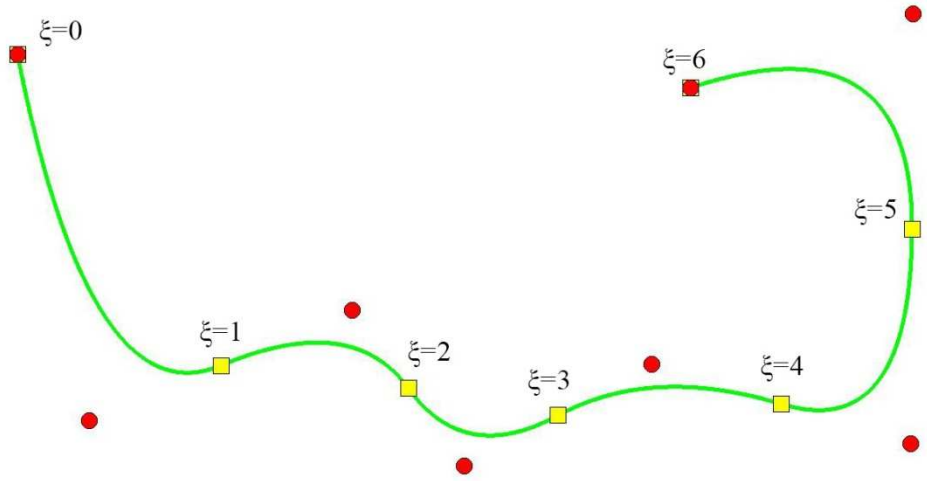
This matrix is formed recursively:

$$T_{ij}^0 = \begin{cases} 1, & \overline{\xi}_i \in \left[\xi_{j,} \xi_{j+1} \right) \\ 0, & \text{otherwise} \end{cases}$$

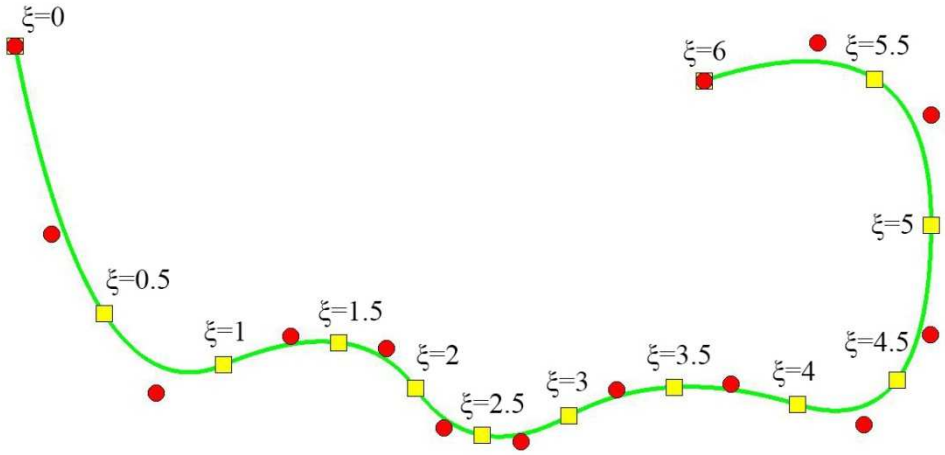
$$T_{i,j}^q = \frac{\overline{\xi}_{i+q} - \xi_j}{\xi_{j+q} - \xi} \cdot T_{i,j}^{q-1} + \frac{\xi_{j+q+1} - \overline{\xi}_{i+q}}{\xi_{j+q+1} - \xi_{j+1}} \cdot T_{i,j+1}^{q-1}, \text{ for } q = 1, 2, ..., p$$

This technique is called h-Refinement.

The B-SPLine curve, depicted in **Figure 5.1.a**, is refined by knot value insertion (**Figure 5.1.b**). Both geometry and parametric mapping remain intact. This can be confirmed by the fact that already existing knots have not been moved. For each new knot value, a basis function and a corresponding control point have been created. The support is still p+1 knot value spans, but their size has been reduced by the introduction of the new knot value vector. Therefore, each control point now affects a much smaller part of the curve. The limited area of effect for each control point leads to a more accurate approximation to the curve.



(a)



(b)

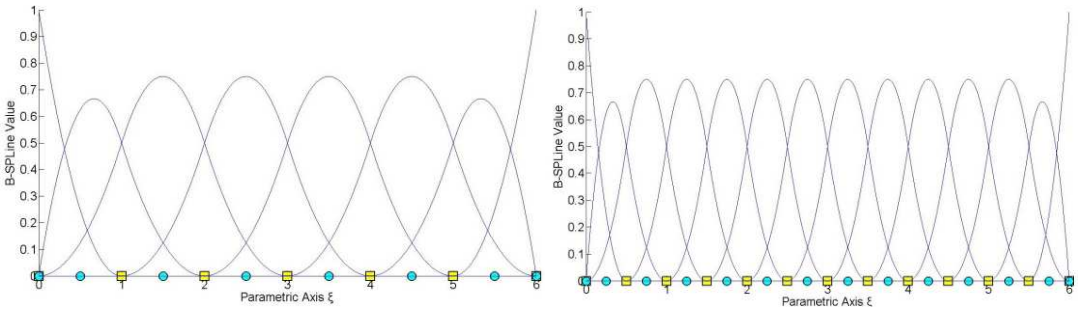
Figure 5.1. h-refinement applied on a B-SPLine curve.

(a) Coarse Mesh.

$$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6 \quad 6\}$$

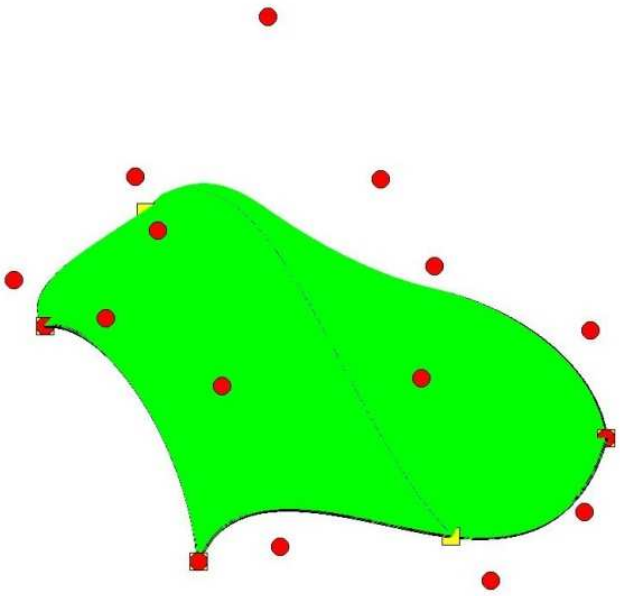
(b) Fine Mesh.

$$\overline{\Xi} = \{0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 1.5 \quad 2 \quad 2.5 \quad 3 \quad 3.5 \quad 4 \quad 4.5 \quad 5 \quad 5.5 \quad 6 \quad 6 \quad 6\}$$

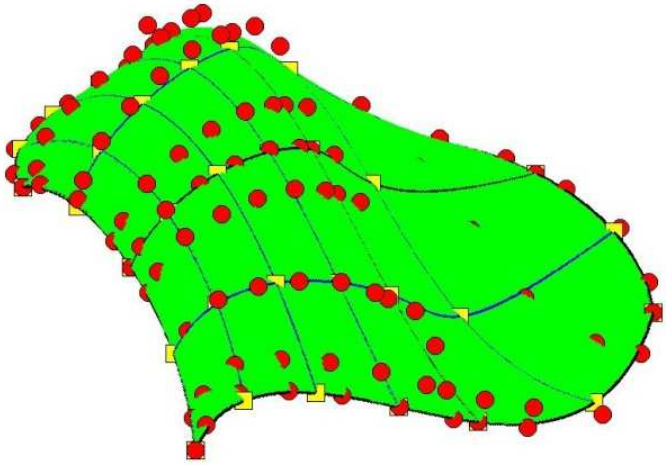


(a) Initial mesh. **(b)** Fine Mesh.

Figure 5.2. h-refinement. Knot insertion. Initial & fine mesh.



(a)



(b)

Figure 5.3. Knot insertion. Multivariate NURBS.

(a) Coarse mesh.

$$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2\}$$

$$H = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1\}$$

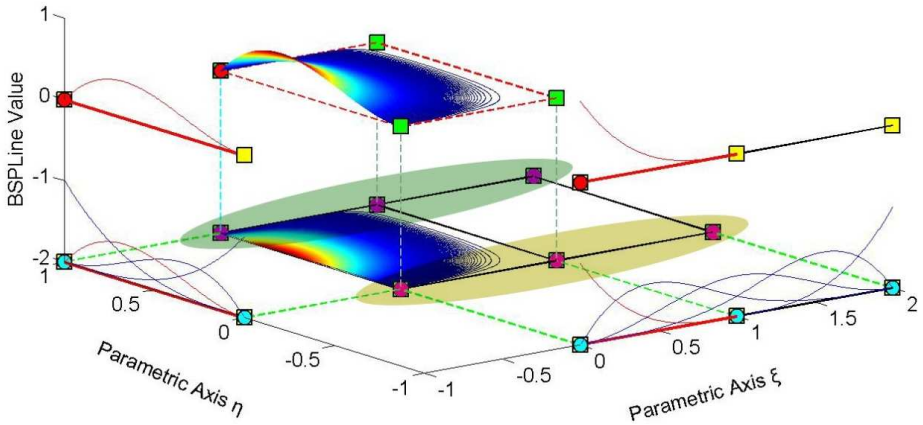
(b) Fine mesh.

$$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 0.25 \quad 0.5 \quad 0.5 \quad 0.75 \quad 1 \quad 1 \quad 1 \quad 1.5 \quad 2 \quad 2 \quad 2 \quad 2\}$$

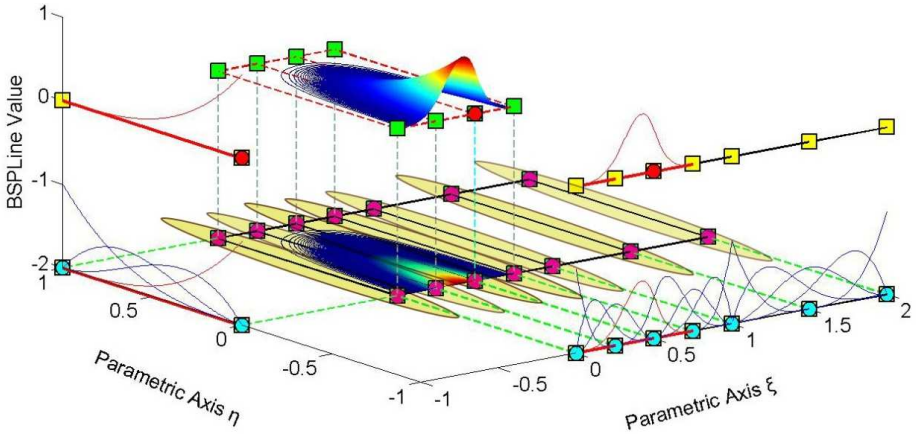
$$H = \{0 \quad 0 \quad 0 \quad 0 \quad 0.25 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.75 \quad 1 \quad 1 \quad 1 \quad 1\}$$

Refinement is applicable in multivariate problems as well. The surface in **Figure 5.3.a** is refined both per ξ and η. **Figure 5.4** depicts basis functions before and after refinement.

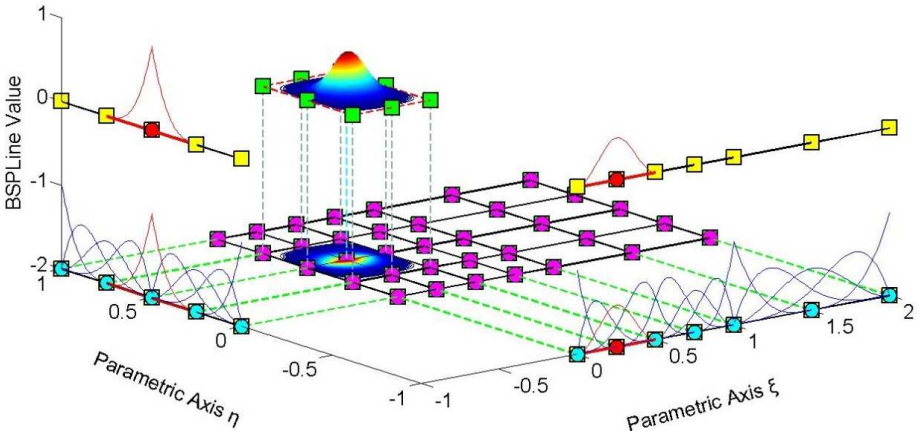
For every univariate (1D) control point per η, a whole set of control points per ξ are defined. Each set is refined individually, as shown in **Figures 5.4.a** & **5.4.b**. After the new control points per ξ are introduced, the same process is followed for refinement per η. The order, in which refinement is applied, is of no importance; due to tensor product properties, control points per η could have been refined first, but the result would be the same.



(a) Coarse mesh.



(b) Refinement (ξ).



(c) Refinement (η).

Figure 5.4. Knot insertion. Surface.

5.3 Degree Elevation

Instead of adding knot values to an existing knot value vector, the increment of the polynomial order can also enrich the basis. Apart from geometry, the mapping from parameter to physical space must also remain unchanged. This is achieved by keeping the same continuity per knot both for coarse and fine mesh.

In order to increase the degree of a B-SPLine curve from p to $\overline{p}$, the new knot value vector has to be defined first. No new knots are added, but every existing knot's multiplicity is increased by $\overline{p}-p$ times. The coarse mesh representation is defined as:

$$C(\xi)=\sum_{j=1}^{n}\left\{N_{j,p}(\xi)\cdot X_j\right\}=\underset{(1\times n)}{\left\{N(\xi)\right\}^T}\cdot\underset{(n\times 1)}{\left\{X\right\}}$$

whereas the fine mesh representation as:

$$C(\xi)=\sum_{i=1}^{m}\left\{\overline{N}_{i,p}(\xi)\cdot \overline{X}_i\right\}=\underset{(1\times m)}{\left\{\overline{N}(\xi)\right\}^T}\cdot\underset{(m\times 1)}{\left\{\overline{X}\right\}}$$

This leads to:

$$\underset{(1\times m)}{\left\{\overline{N}(\xi)\right\}^T}\cdot\underset{(m\times 1)}{\left\{\overline{X}\right\}}=\underset{(1\times n)}{\left\{N(\xi)\right\}^T}\cdot\underset{(n\times 1)}{\left\{X\right\}}$$

The new set of control points $\left\{\overline{X}\right\}$, necessary for the representation, will be defined through a Transformation matrix for p-Refinement. There are many efficient algorithms for degree elevation. The following is a quick, reliable approach we have been efficiently using:

At first, coarse mesh B-SPLine basis functions are evaluated at m points throughout the patch. Careful selection (such as no points of C^0 Continuity or other irregularities are involved) is preferred. This way, a B-SPLine function matrix is created, $\underset{(n\times m)}{\left[N\right]}$, which contains the values of the n basis functions for the m selected points.

The same points are used for the evaluation of new basis functions, with the refined degree and the new knot value vector. Thus, $\underset{(m\times m)}{\left[\overline{N}\right]}$ is created. Since the same points are evaluated and the parametric mapping remains the same, it applies that:

$$\underset{(m\times m)}{\left[\overline{N}\right]^T}\cdot\underset{(n\times 1)}{\left\{\overline{X}\right\}}=\underset{(m\times n)}{\left[N\right]^T}\cdot\underset{(n\times 1)}{\left\{X\right\}}\Rightarrow\underset{(n\times 1)}{\left\{\overline{X}\right\}}=\left(\underset{(m\times m)}{\left[\overline{N}\right]^T}\right)^{-1}\cdot\underset{(m\times n)}{\left[N\right]^T}\cdot\underset{(n\times 1)}{\left\{X\right\}}$$

Therefore,

$$\underset{(m\times n)}{[T]} = \left(\underset{(m\times m)}{\left[\overline{N}\right]^T} \right)^{-1} \cdot \underset{(m\times n)}{[N]^T}$$

If $\underset{(m\times m)}{\left[\overline{N}\right]^T}$ cannot be reversed, a different set of points have to be selected. This procedure can also be used for h-Refinement applications.

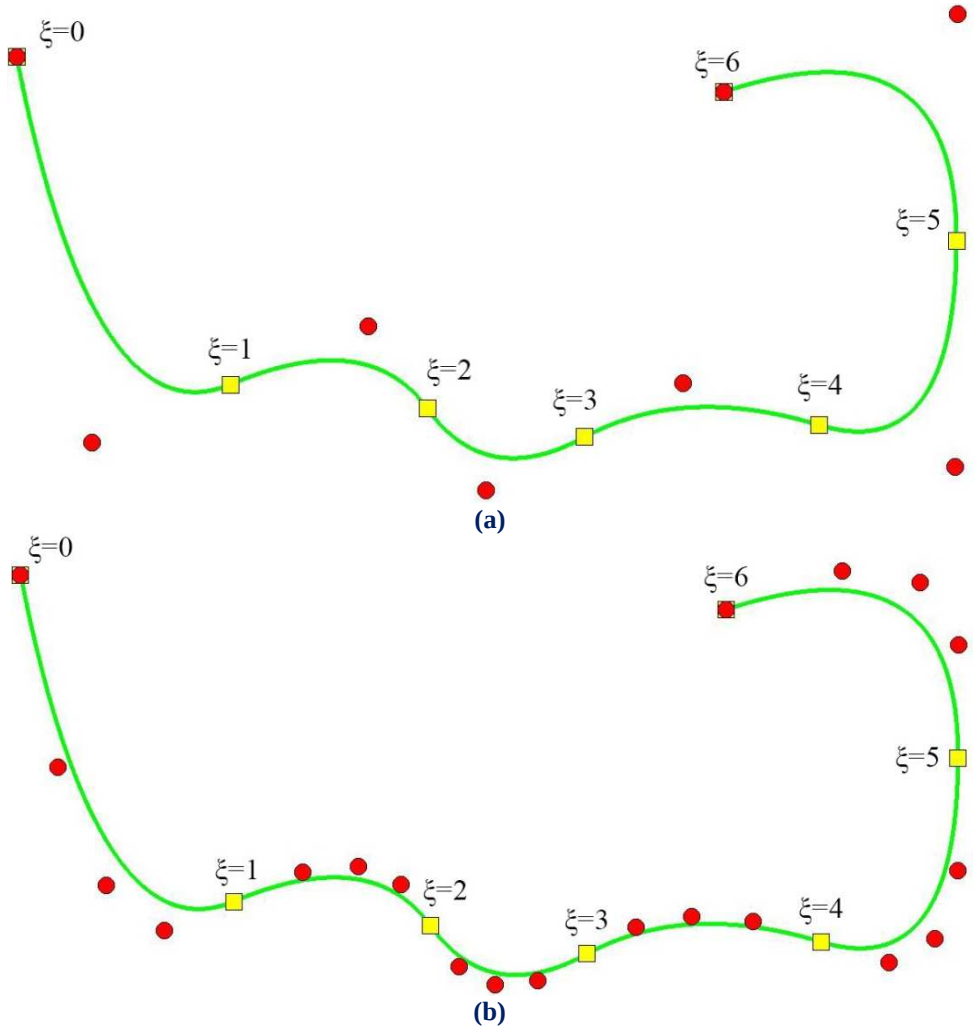


Figure 5.5. B-SPLine curve. Degree elevation.

(a) Coarse mesh (p=2)

$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6 \quad 6\}$

(b) Fine mesh (p=4)

$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad . \quad . \quad . \quad 6 \quad 6 \quad 6 \quad 6 \quad 6\}$

The curve in **Figure 5.5** is subjected to p-refinement. Knot spans remains intact. Geometry and mapping from parameter space remain intact in p-refinement as well. Continuity remains C^1 across every internal knot. Curve approximation by control points is also improved with p-refinement.

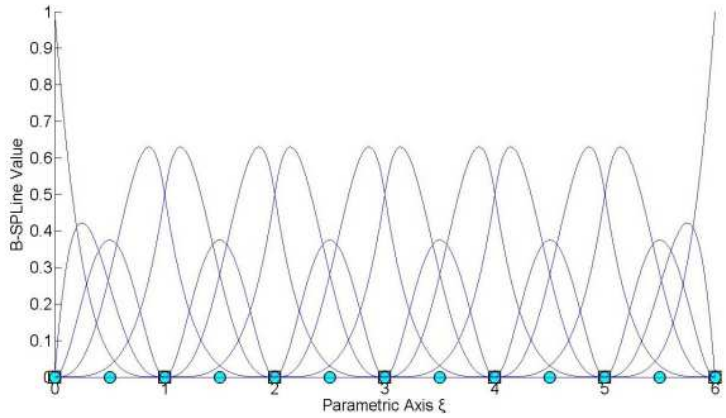
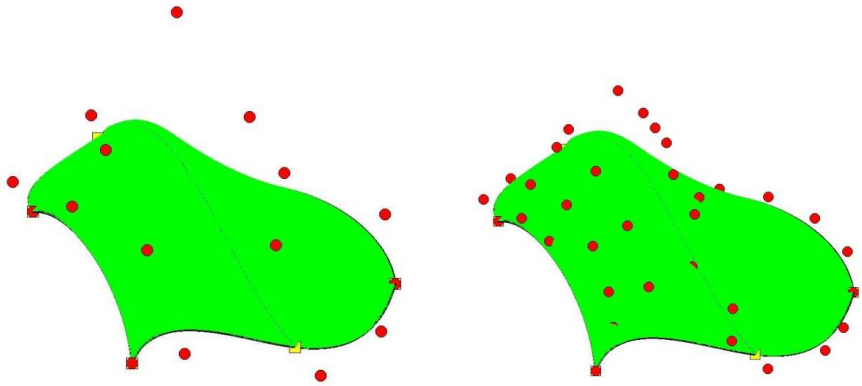


Figure 5.6. Basis. Degree elevation.
C^1 continuity across internal knots.



(a) **(b)**

Figure 5.7. Order elevation in multiple directions.
(a) Coarse mesh. (p,q)=(3,3).
(b) Fine mesh. (p,q)=(4,5).

Observe that knot rectangles have not changed in this example as well. The mesh is finer, but the continuity of the basis and the whole parameter space remains intact. Basis function interconnectivity, however, is improved.

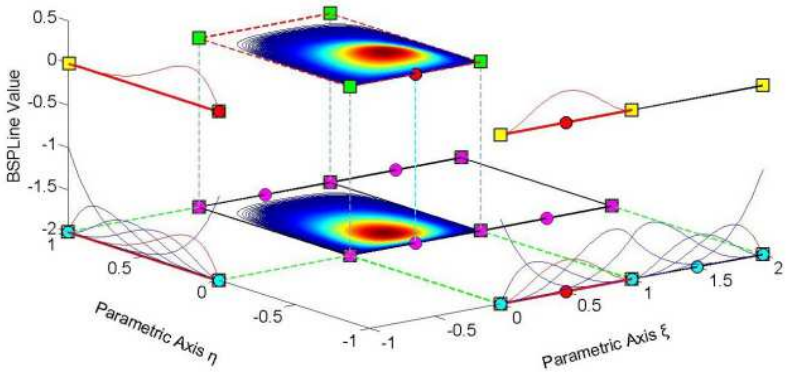
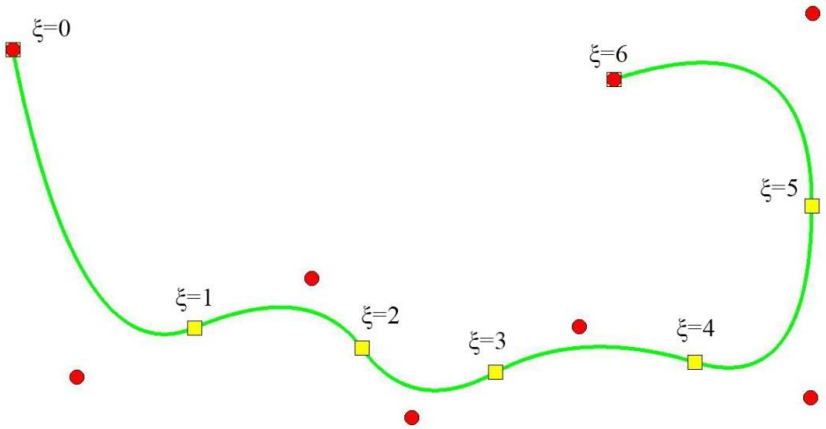


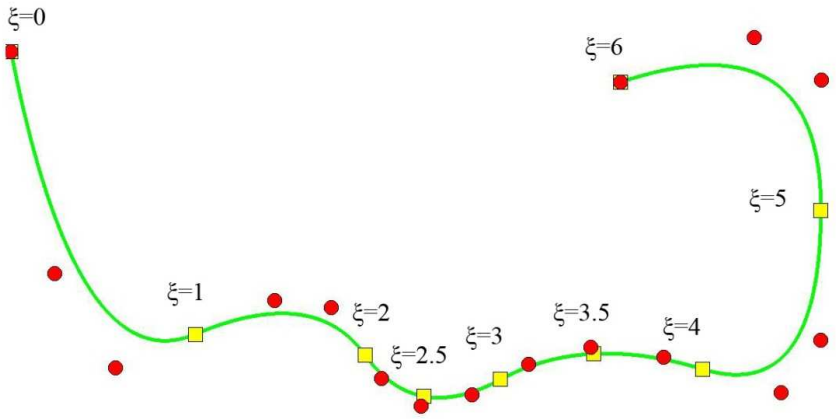
Figure 5.8. Degree elevation. Basis & shape functions.
Reduced continuity across internal knots.

5.4 Degree Elevation & Knot Insertion

Increasing polynomial degree with p-refinement is an improvement to the basis, but continuity remains intact as in the coarse mesh. In order to improve this aspect, k-refinement was introduced by Hughes. The basic idea is that, after p-refinement, h-refinement can be applied in order to create basis functions of C^{p-1}-continuity. This is a powerful tool that can lead to better convergence rates.



(a) Coarse mesh.

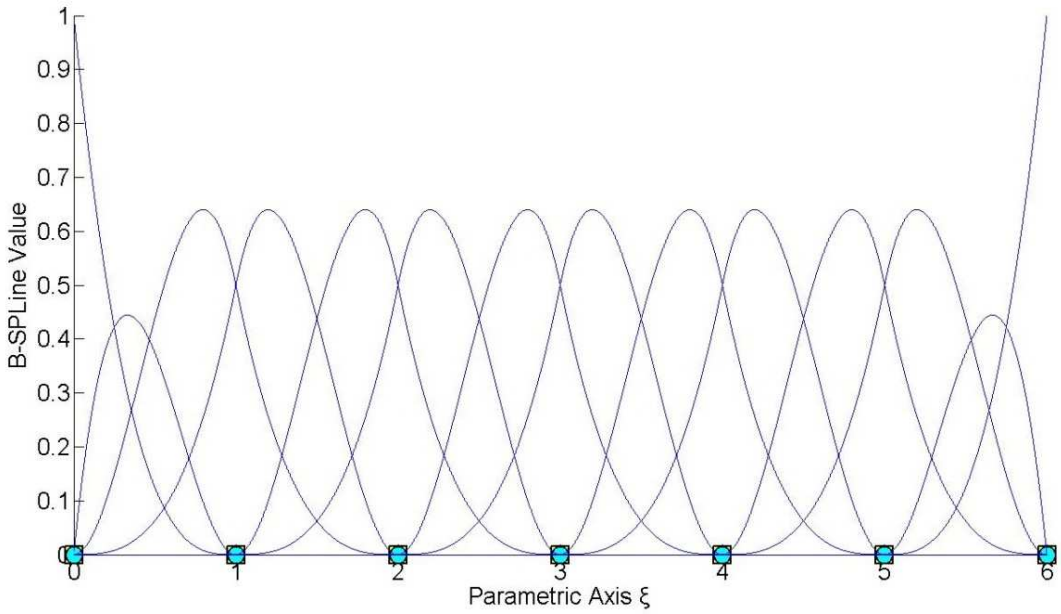


(b) Fine mesh. p=3.

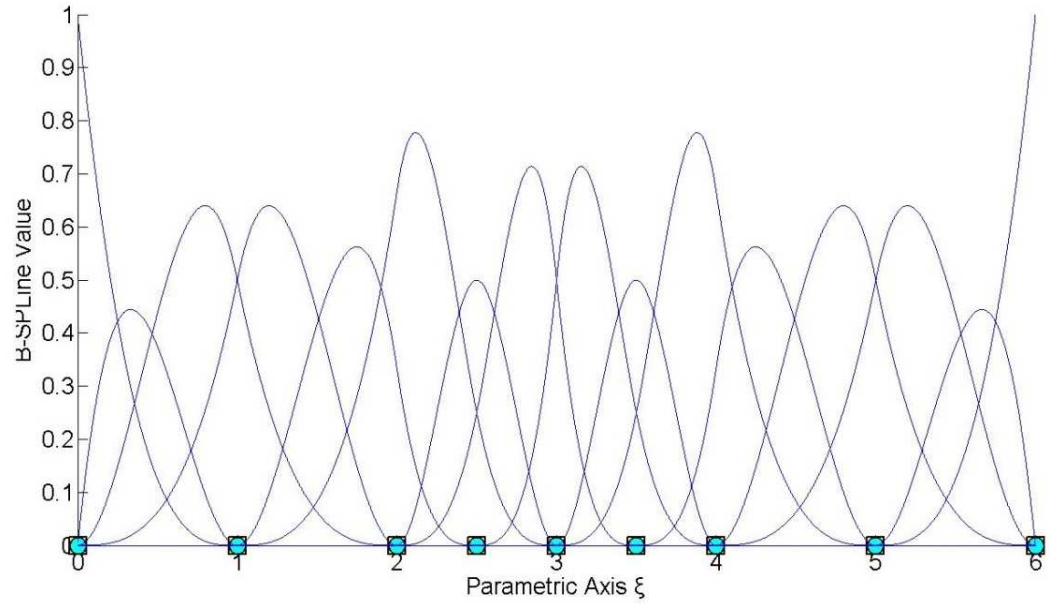
$$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2.5 \quad 3 \quad 3 \quad 3.5 \quad 4 \quad 4 \quad 5 \quad 5 \quad 6 \quad 6 \quad 6 \quad 6\}$$

Figure 5.9. k-refinement. B-SPLine curve.

In **Figure 5.9**, p-Refinement is firstly performed, so the degree is elevated to third. The C^1-continuity in the coarse mesh is maintained. Afterwards, h-refinement is applied, with the insertion of two extra knot values. Continuity at these knot values is C^2, thus taking advantage of the new order.
The combination of degree elevation and knot insertion drives us to a more efficient refinement type.



(a) Degree elevation to cubic.
Ξ={0,0,0,0,1,1,2,2,3,3,4,4,5,5,6,6,6,6}



(b) Knot insertion.
Ξ={0,0,0,0,1,1,2,2,2.5,3,3,3.5,4,4,5,5,6,6,6,6}

Figure 5.10. k-Refinement two steps (order elevation, knot insertion).

After degree elevation, internal knots still possess C^1 continuity. Only two basis functions have non-zero value at every knot. After knot insertion, two new C^2-continuity knots are inserted. Three B-SPLine basis functions are non-zero at these knots, ξ=2.5 and ξ=3.5.

In this way, increased continuity for the new degree is achieved.

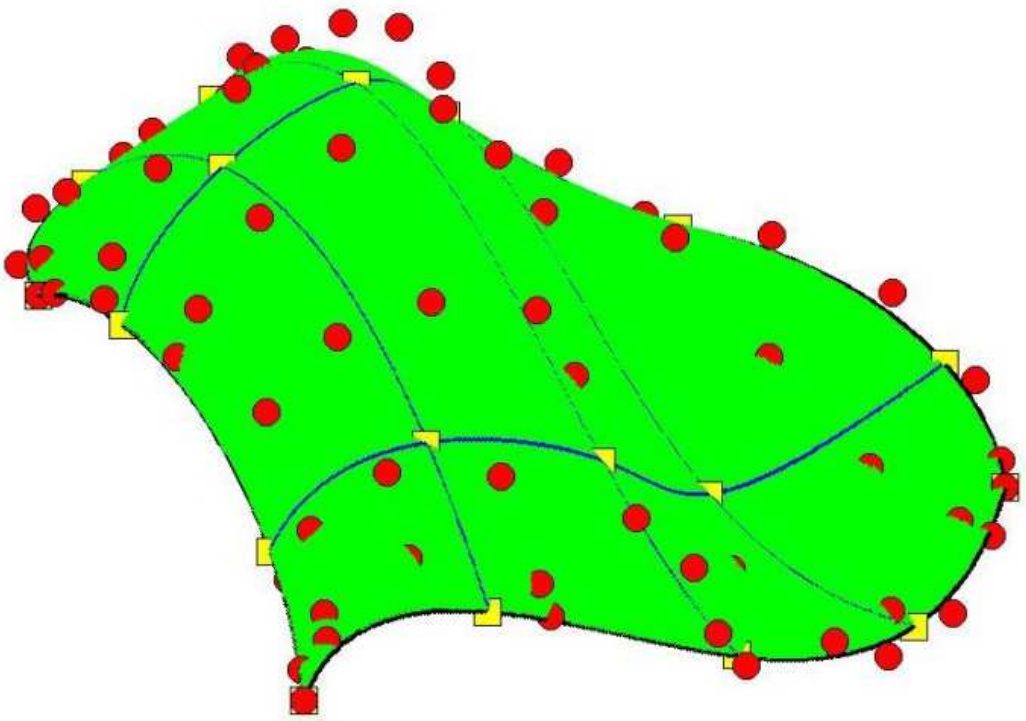


Figure 5.11. K-refinement. NURBS surface.

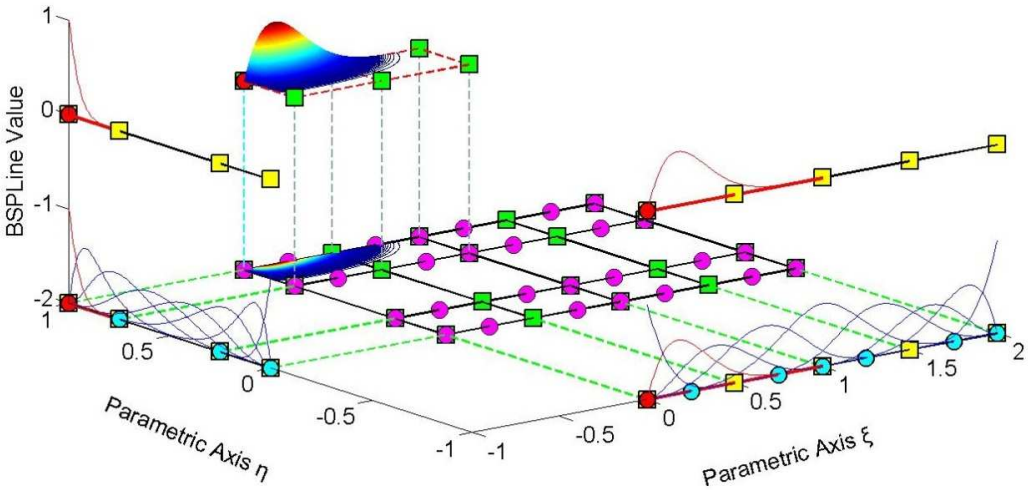


Figure 5.12. K-refinement. NURBS surface. Basis.

$$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 1 \quad 1.5 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2\}$$

$$H = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0.25 \quad 0.75 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\}$$

In **Figure 5.12**, the application of k-refinement for a surface is shown. When p-Refinement is applied, continuity of the internal knots remains C^2. After h-refinement, continuity of the additional control points in axes ξ, η is increased to C^3 and C^4 respectively.

5.5 Reverse Refinement

Transformation from a fine into a coarse mesh is also required for some applications. The ability to go back-and-forth between coarse and fine mesh is useful for certain aspects of the analysis and design. This process is called reverse refinement.

Reverse refinement does not always provide an accurate solution. Removal of specific knots or downgrading of polynomial degree may affect the geometry of the NURBS model. Reverse p-refinement, the so-called degree reduction, is not possible, if there are knots of C^{p-1} continuity. In general, reverse refinement is an overdetermined problem, which can only be solved in certain cases. It is applicable, when refinement has already been implemented and the engineer is willing to return to the initial mesh of the NURBS model. The procedure we follow is similar to refinement.

In order to calculate the coarse mesh's control points, a transformation matrix (from fine into initial) has to be introduced, so that:

$$\underset{(n\times 1)}{\{X\}} = \underset{(n\times m)}{[T_{CF}]} \cdot \underset{(mx1)}{\{\overline{X}\}}$$

This can be created from the transformation matrix $\underset{(m\times n)}{[T_{FC}]}$ (from initial to fine).

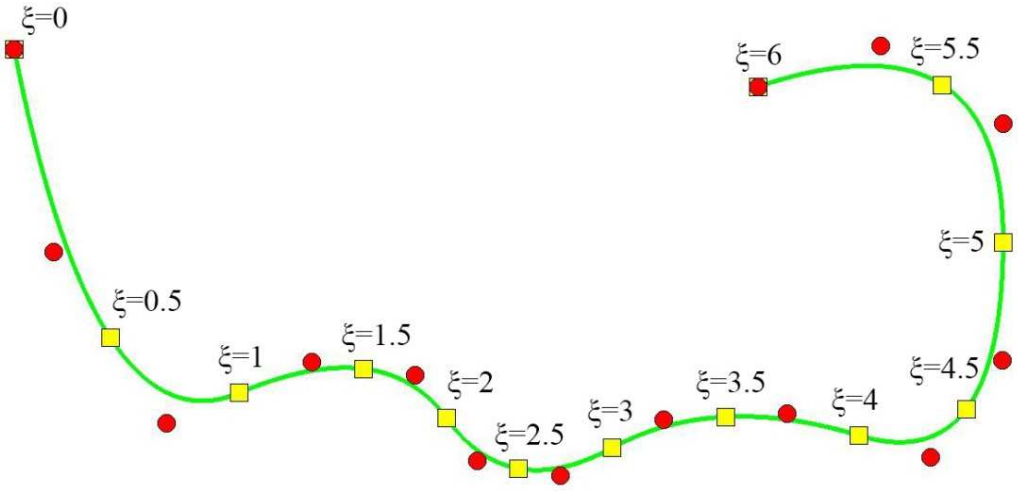
$$\underset{(mx1)}{\{\overline{X}\}} = \underset{(m\times n)}{[T_{FC}]} \cdot \underset{(n\times 1)}{\{X\}} \Rightarrow \underset{(n\times m)}{[T_{FC}]^T} \cdot \underset{(mx1)}{\{\overline{X}\}} = \underset{(n\times m)}{[T_{FC}]^T} \cdot \underset{(m\times n)}{[T_{FC}]} \cdot \underset{(n\times 1)}{\{X\}}$$

Therefore,

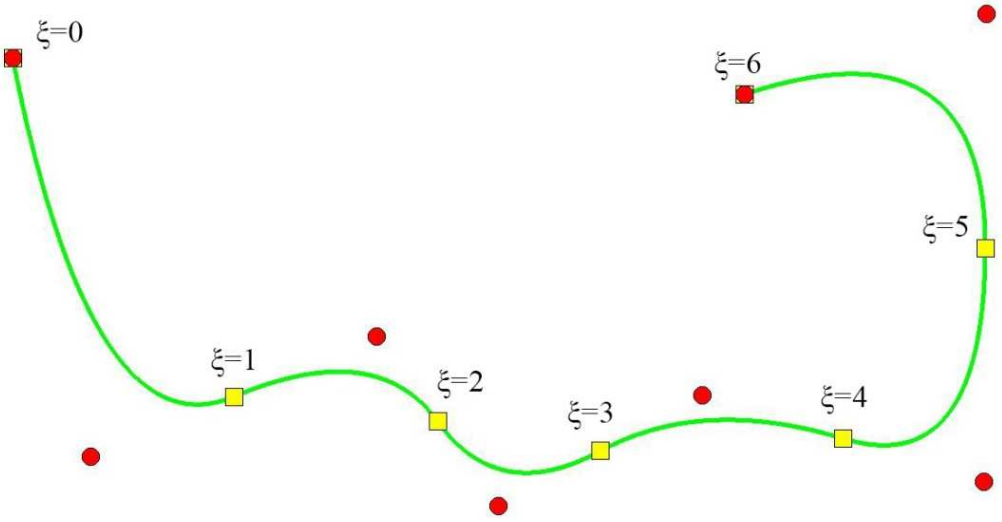
$$\underset{(n\times 1)}{\{X\}} = \left(\underset{(n\times m)}{[T_{FC}]^T} \cdot \underset{(m\times n)}{[T_{FC}]} \right)^{-1} \cdot \underset{(n\times m)}{[T_{FC}]^T} \cdot \underset{(mx1)}{\{\overline{X}\}}$$

and

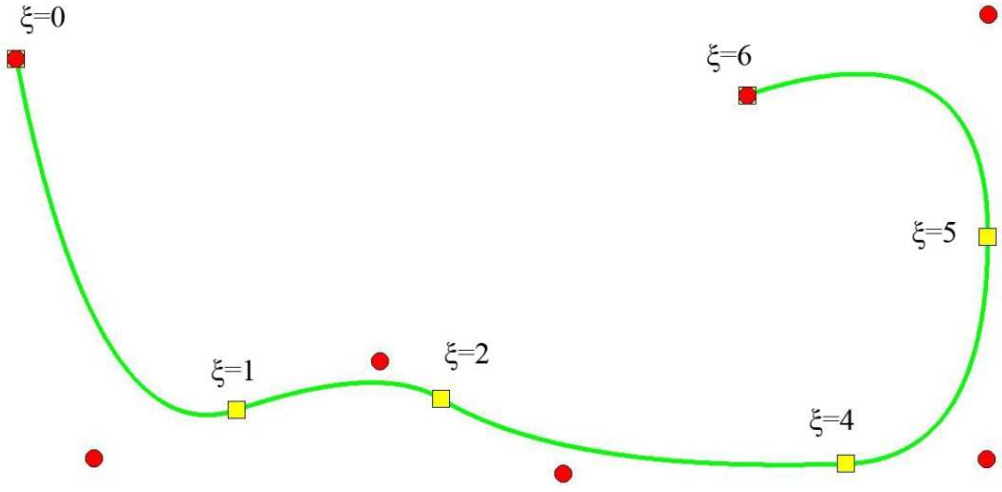
$$\underset{(n\times m)}{[T_{CF}]} = \left(\underset{(n\times m)}{[T_{FC}]^T} \cdot \underset{(m\times n)}{[T_{FC}]} \right)^{-1} \cdot \underset{(n\times m)}{[T_{FC}]^T}$$



(a) $\Xi = \{0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 1.5 \quad 2 \quad 2.5 \quad 3 \quad 3.5 \quad 4 \quad 4.5 \quad 5 \quad 5.5 \quad 6 \quad 6 \quad 6\}$



(b) $\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6 \quad 6\}$



(c) $\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 4 \quad 5 \quad 6 \quad 6 \quad 6\}$

Figure 5.13. Knot removal (reverse knot insertion).

5.6 Refinement

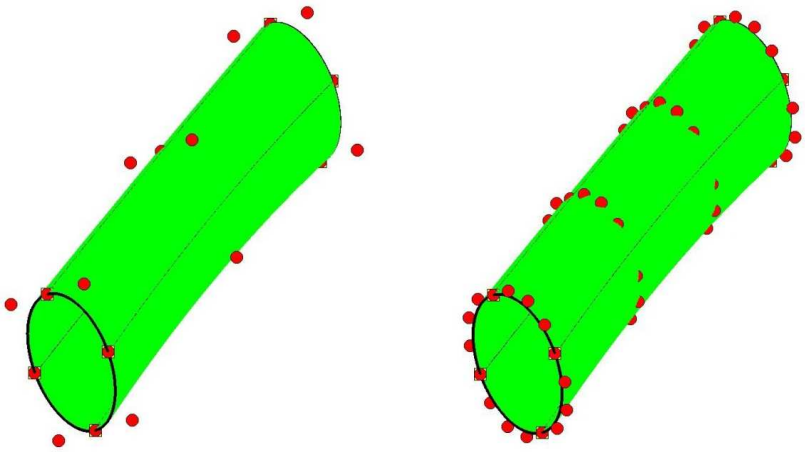
NURBS is created by the projection of a B-SPLine; therefore, NURBS refinement can be achieved by refining the corresponding projective B-SPLine curve. Weights are the fourth-coordinate of the projective curve.

The first step is to evaluate the projective control points $\{B^W\}$, by multiplying every coordinate with the corresponding weight.

$$\left\{ X_i^w \quad Y_i^w \quad Z_i^w \right\} = \left\{ X_i \cdot W_i \quad Y_i \cdot W_i \quad Z_i \cdot W_i \right\}$$

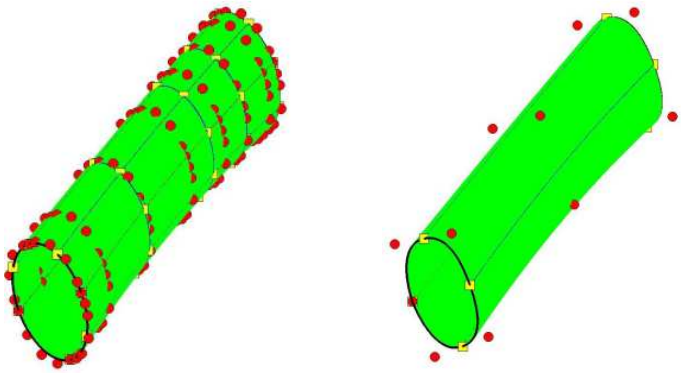
Control point coordinates and weights are defined by dividing the Cartesian coordinates with their respective weights.

$$\left\{ X_i \quad Y_i \quad Z_i \right\} = \left\{ \frac{X_i^w}{W_i} \quad \frac{Y_i^w}{W_i} \quad \frac{Z_i^w}{W_i} \right\}$$



(a) Coarse mesh. **(b)** Fine mesh

Figure 5.14. p-refinement. NURBS surface.



(a) Coarse mesh. **(b)** Fine mesh

Figure 5.15. k-refinement & reverse refinement. NURBS surface.

6 Stiffness Matrix

6.1 Preliminary Steps for Analysis

The main difference between isogeometric analysis and finite element analysis is the nature of the shape functions used for the approximation of the solution field. The replacement of Lagrange polynomials by NURBS (widely popular in the CAD community) provides additional valuable features to isogeometric analysis.

6.1.1 Shape functions

FEM shape functions are usually polynomials (e.g. Lagrange polynomials). However, they present major disadvantages. FEM shape functions are interpolatory at all nodes, both internal and external, and have C^{-1}-continuity at the edge. This results in the incompetence of FEM to define stresses and strains at the boundaries of the elements, needing iterative methods such as extrapolation, in order to achieve that.

NURBS, as shape functions, shows greater overlapping, as nearby elements are strongly connected, thus the simulation provides an improved approach to the real behavior of the engineering structure. Continuous shape function derivatives lead to a continuous stress and strain field, minimizing the need for application of corrective methods.

6.1.2 Control points

Classical FEM downsizes the natural problem of infinite unknowns to a finite number. These unknowns are the degrees of freedom of the nodes. The position of the nodes depends on the element type. As a general rule, the nodes are usually located at the corners and at the middle of the element sides. They belong to the element and therefore to the model.

Displacement field of the NURBS model can be approximated by a linear combination of the unknowns, the pseudo-displacements and the corresponding shape functions.

In isogeometric analysis, NURBS are chosen as shape functions. The isoparametric concept is applied with the geometrical mapping defining the solution approximation. The geometrical representation is achieved through a combination of control points and their corresponding shape functions. Degrees of freedom, connected to the control points are the unknowns.

6.1.3 Elements

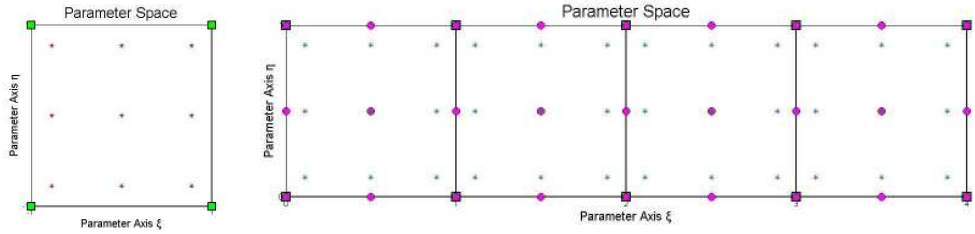
The isogeometric finite element could be either the patch or the knot span of the patch. In order to perform exact numerical integration, a certain number of Gauss points are chosen for the domain of every piece of the polynomial basis functions. This domain is the knot span, which resembles the finite element of FEM. This is the reason why, in this thesis, knot spans and not patches are considered as isogeometric elements. In IGA, shape functions are not restricted to the interior of each element (knot span). They are non-zero across p+1 knot spans and overlap with more shape functions. This overlapping results in a denser stiffness matrix than the classical FEM one. Apart from that, B-SPLine functions are defined in the whole domain, thus allowing for integration throughout the patch without formulating local element matrices separately.

6.1.4 Gauss points

6.1.4.1 Parametric coordinates

Gauss points are chosen for each knot span. Their coordinates and weights are located at the reference element [-1,1] as the roots of the Legendre polynomial. The next step is to transform the coordinates and weights of the desired knot span $[\xi_i, \xi_{i+1})$.

$$\xi = \frac{\left(\xi_{i+1} - \xi_i\right)\cdot \xi^R + \left(\xi_{i+1} + \xi_i\right)}{2},\ w^{GP\xi} = \frac{\left(\xi_{i+1} - \xi_i\right)}{2}\cdot w_\xi^R$$



(a) Reference knot span **(b)** Parameter Space.

Figure 6.1. Gauss points.

Full tensor product properties are applied here as well, leading in similar equations for the other two parametric axes.

$$\eta = \frac{\left(\eta_{j+1} - \eta_j\right)\cdot \eta^R + \left(\eta_{j+1} + \eta_j\right)}{2},\ \zeta = \frac{\left(\zeta_{k+1} - \zeta_k\right)\cdot \zeta^R + \left(\zeta_{k+1} + \zeta_k\right)}{2}$$

$$w^{GP\eta} = \frac{\left(\eta_{j+1} - \eta_j\right)}{2}\cdot w_\eta^R,\ w^{GP\zeta} = \frac{\left(\zeta_{k+1} - \zeta_k\right)}{2}\cdot w_\zeta^R$$

6.1.4.2 Gauss point number

Gauss point coordinates and weights are calculate for every patch. For the exact integration of a polynomial of degree q, $\frac{q+1}{2}$ or $\frac{q+2}{2}$ Gauss points are required per knot span for q odd and even respectively.

For 1D problems, the maximum degree of the deformation matrix is defined from the derivation of piecewise polynomial shape functions, therefore p-1.

$\left[B(\xi)\right]^T \cdot [E] \cdot \left[B(\xi)\right]$ yields to the product of polynomials of maximum order p-1 resulting in a polynomial of maximum order $(p-1)+(p-1)=2p-2$. Thus, the minimum number of Gauss points per knot span required for exact integration is:

$$\frac{(2p-2)+2}{2}=p$$

For 2D and 3D problems, the maximum order of the deformation matrix is determined by partial derivation of piecewise polynomial shape functions. Therefore, the maximum degree is p for derivation in the remaining directions.

The order of the product $\left[B(\xi)\right]^T \cdot [E] \cdot \left[B(\xi)\right]$ is p+p=2p.

In order to achieve exact integration, the minimum number of Gauss points per knot span is:

$$\frac{2p+2}{2}=p+1$$

Thus, per knot span:

- For 1D problems, **p** Gauss points per knot span are required.
- For 2D and 3D problems, **p+1** Gauss points per knot span are required.

6.1.5 Patches

Patches are used where a change in geometry type or material occurs. They can also be used in any case C^{-1} or C^0 continuity is required. If separate knot vectors are used for every patch, stiffness matrices will be formulated for every patch. The separate matrices are combined into one via connectivity arrays. If the connection is watertight, the procedure is the same as the one used in classical FEM to combine local element matrices to the total stiffness matrix.

6.1.6 Elasticity matrix

Elasticity matrices for 1D elasticity, plane strain, plane stress and 3D elasticity are defined by the following equations,

where:

- E is the Young's modulus
- ν is the Poisson's ratio

1D elasticity:

$$\underset{(1x1)}{[E]} = E$$

2D elasticity, plane stress:

$$\underset{(3x3)}{[E]} = \frac{E}{1-\nu^2} \cdot \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}$$

2D elasticity, plane strain:

$$\underset{(3x3)}{[E]} = \frac{E}{(1-\nu)\cdot(1-2\nu)} \cdot \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

3D elasticity:

$$\underset{(6x6)}{[E]} = \frac{E}{(1-\nu)\cdot(1-2\nu)} \cdot \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

The corresponding stress and strain vectors are:

1D elasticity:

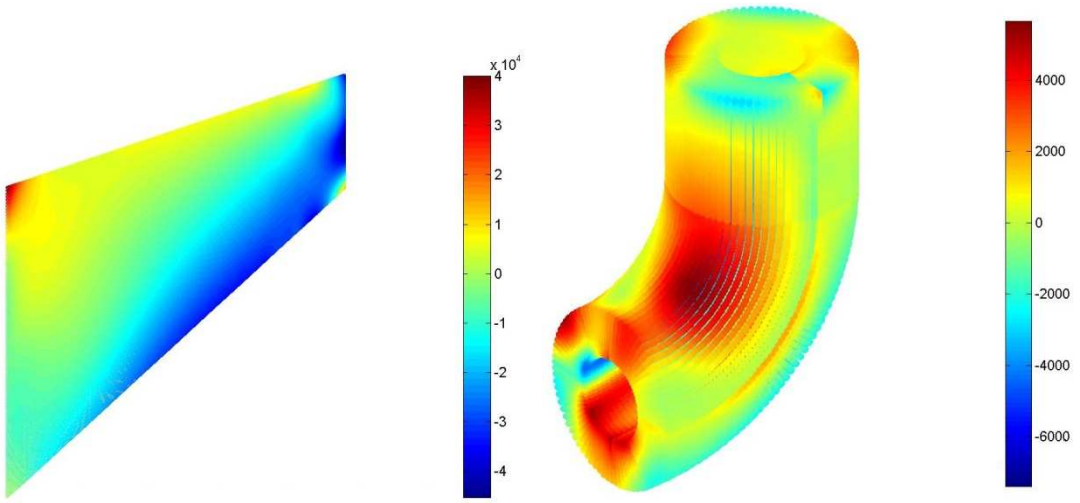
$$\underset{(1\times1)}{\{\sigma\}} = \{\sigma_X\}$$
$$\underset{(1\times1)}{\{\varepsilon\}} = \{\varepsilon_X\}$$

2D elasticity, plane stress and plane strain:

$$\underset{(3\times1)}{\{\sigma\}} = \begin{Bmatrix} \sigma_X \\ \sigma_Y \\ \tau_{XY} \end{Bmatrix}$$
$$\underset{(3\times1)}{\{\varepsilon\}} = \begin{Bmatrix} \varepsilon_X \\ \varepsilon_Y \\ \gamma_{XY} \end{Bmatrix}$$

3D elasticity:

$$\underset{(6\times1)}{\{\sigma\}} = \begin{bmatrix} \sigma_X \\ \sigma_Y \\ \sigma_Z \\ \sigma_{XY} \\ \sigma_{YZ} \\ \sigma_{ZX} \end{bmatrix}, \underset{(6\times1)}{\{\varepsilon\}} = \begin{bmatrix} \varepsilon_X \\ \varepsilon_Y \\ \varepsilon_Z \\ \gamma_{XY} \\ \gamma_{YZ} \\ \gamma_{ZX} \end{bmatrix}$$



(a) σ_{xx} (plane stress) **(b)** τ_{yz} (3D elasticity)

Figure 6.2. Stress contour distribution.

6.2 Assembly

6.2.1 Contribution-wise stiffness matrix formulation

Traditionally, in the classical finite element method, after the formulation of each local stiffness matrix, the influenced entries of the total stiffness matrix are updated by adding the contribution of each element. This procedure is applicable on IGA too.

However, instead of using the element entities, the total stiffness matrix can be formulated by using directly the Gauss point entities without any reference in elements. For each Gauss point, by taking into consideration the nodes with which it interacts, a local stiffness matrix is defined. Both of these procedures are based on the contribution of the Gauss points in the total stiffness matrix and they are the two variations of the contribution-wise (CW) method.

It is worth mentioning that the procedures, which will be described in this thesis, are applicable on NURBS models with more than one patch.

6.2.1.1 Gauss point-wise variant of the CW method

The stiffness contribution of each Gauss point is consequently added to the stiffness matrix of the patch. This is achieved by the direct addition of each Gauss point's contribution.

Because of the different domain of influence that a Gauss point has, the stiffness matrix, which corresponds to each quadrature point, has variable dimensions. Therefore, the workload needed for each Gauss point's iteration is not fixed. Connectivity or mapping, that will refer to the relation between Gauss points and control points, should be formulated. This procedure, which is a variation of the contribution-wise method, is called Gauss point-wise and can be described by the following expression:

$$K = \sum_{G} w_G B_G^T E B_G$$

where:

- w_G is the weight factor of the Gauss point
- E is the elasticity matrix (constitutive law)
- B_G is the deformation matrix computed at the corresponding Gauss point.

The global stiffness matrix assembly is presented in the following flow chart in **Figure 6.3**.

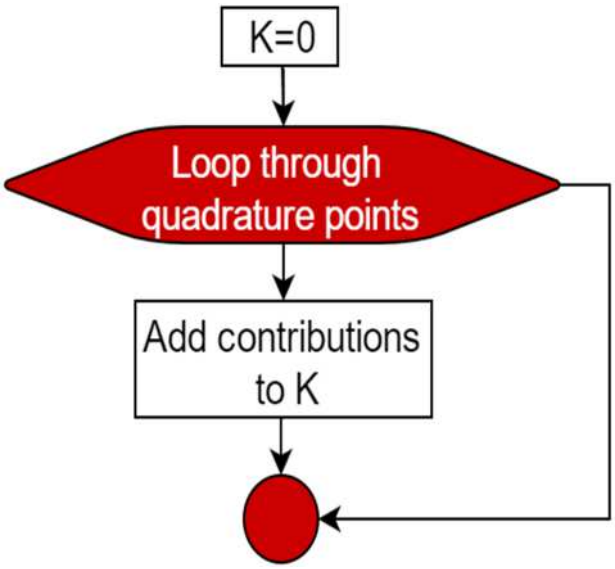


Figure 6.3. Stiffness matrix assembly.
Gauss point-wise variant of the CW method.

6.2.1.2 Element-wise variant of the CW method

It is more efficient to group Gauss points that affect the same degrees of freedom into elements. The stiffness matrix of each element is built by adding the contributions of all the Gauss points that are enclosed in it. Due to the fact that p+1 B-Spline basis functions of order p are non-zero for each knot span, the number of columns/ rows of the stiffness matrix is fixed, thus equals to the degree freedom number per point multiplied by the number of non-zero shape functions for each knot span. This approach is usually encountered in FEA, since it makes more evident the discretization of the geometric domain into elements. This procedure gives emphasis in elements. Therefore, it is known as element-wise variant of the contribution-wise method.

The stiffness matrix of an element is equal to:

$$K_E = \sum_{E_G} w_G B_G^T E B_G$$

After the calculation of the local stiffness matrix, the entries of the total matrix, which correspond to the control points influenced by an element, are updated:

$$K = \sum_{E} K_E$$

The implementation of the Gauss point-wise formulation demands the determination of correlations between Gauss points and control points. On the contrary, the implementation of the element-wise approach demands the determination of correlations between elements and control points. As a result, the occupying memory space for the storage of the demanded correlations is reduced.

6.2.2 Interaction-wise stiffness matrix formulation

In the contribution-wise method there is always at least one loop through the Gauss points. This causes the continual updating of the appropriate stiffness matrix coefficients. In order to avoid this updating, another method called interaction-wise has been developed. The interaction – wise approach is based on the computation of the final value of each stiffness matrix coefficient. For each node combination $i-j$, K_{ij} that describes the interaction between the two nodes is formed by adding the contributions of the Gauss points that both belong to i and j domains of influence. Note that two control points are interacting if there is at least one Gauss point that both influences them. In this case the corresponding K_{ij} would be non-zero. As in the contribution-wise, there are two variations of interaction-wise, one emphasizes on the Gauss points entities and the other emphasizes on the elements.

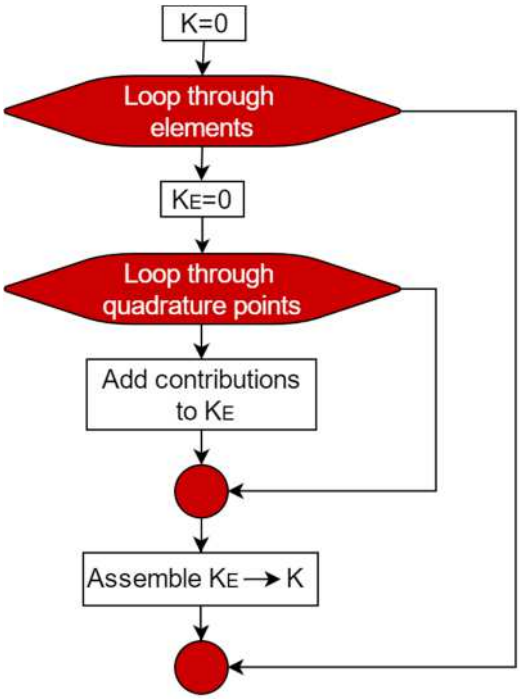


Figure 6.4. Stiffness matrix assembly. Element-wise variant of the CW method.

6.2.2.1 IW variant with individual Gauss Points

In this approach, each Gauss point needs to be handled individually and may affect different degrees of freedom. Firstly, all the interacting control point pairs and the Gauss points they share should be specified. K_{ij} is formed by the following expression:

$$K_{ij} = \sum_{Sh.G} w_G B_i^T E B_j$$

The summation of the $B_i^T E B_j$ is operated for every Gauss point, which both influences i and j control points. These control points are also referred as shared Gauss points. They have a fundamental importance for the implementation of IW approach.

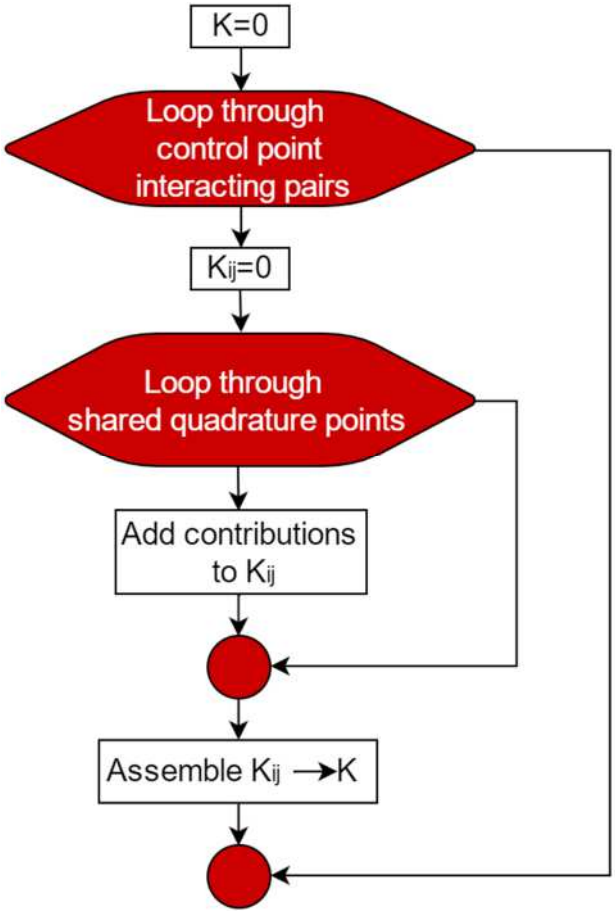


Figure 6.5. Stiffness matrix assembly. IW variant with individual Gauss points.

6.2.2.2 IW variant for element-driven applications

Instead of determining shared Gauss point for each interacting node pair, it is more beneficial to handle the Gauss points, which are enclosed within an element as a group. The summation of the $B_i^T E B_j$ is operated for every Gauss point of all shared elements, so for each interacting control point pair, the equation is the following:

$$K_{ij} = \sum_{Sh.G} w_G B_i^T E B_j = \sum_{Sh.E} \left(\sum_{G_E} w_G B_i^T E B_j \right)$$

For element driven application there is need for two loops. One operates iterations through the shared elements of an interacting control point pair and the other through the enclosed Gauss points of the element.

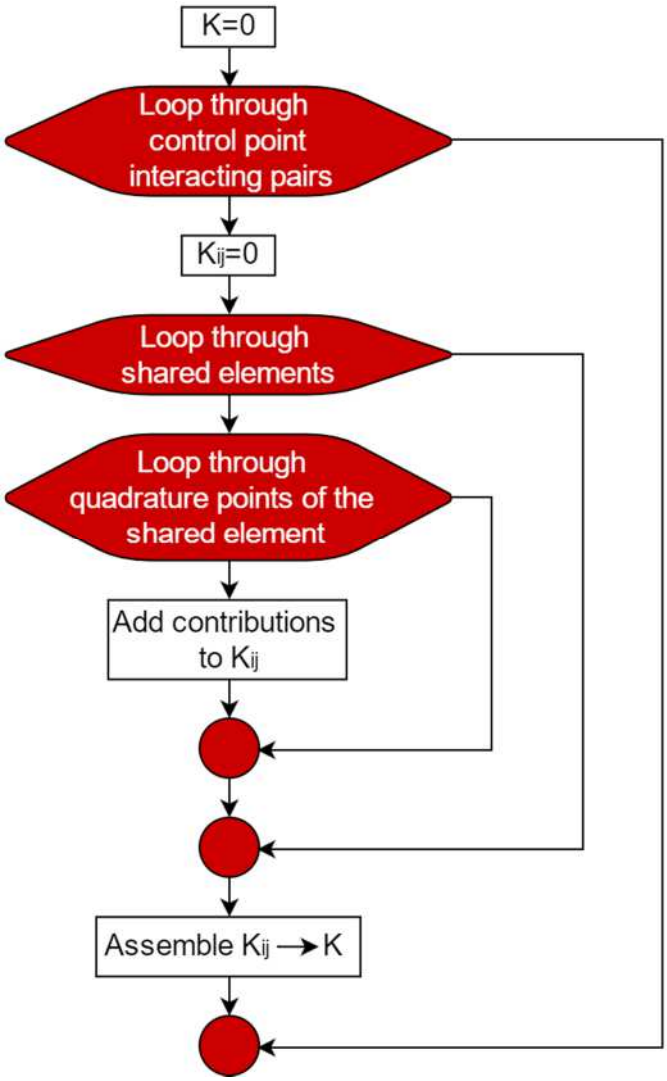


Figure 6.6. Stiffness matrix assembly in IW variant. Element-driven applications.

6.2.3 Input Data

The formulation of the stiffness matrix requires:

- Structural analysis and material
- Computational geometry

6.2.4 Best suited matrix format

Sparse formats store only non-zero entries, occupying as possible as less storage space. Usually GPUs DRAM memories are smaller than the DRAM of the total system. Thus, in this thesis for the implementations that concern GPUs (certainly in IW approach), sparse matrix formats have been used for the storage of the total stiffness matrix.

6.2.4.1 Interaction-wise approach

In the case of IW approach, the entries of the total stiffness matrix are updated only once. That means that there is no need of lookups. Each time new coefficients are added. As a result, the selected matrix format should allow the incremental construction of the total matrix. For these reasons, the selected format is the Coordinate List (COO). It is worth mentioning that this format allows the quick transformation to other compressed formats like CSR, providing higher performance during the solution phase.

6.2.4.2 Element-wise approach

In contrast to the IW approach, where there is no need of the constantly updating of some coefficients, in the case of EW approach, the matrix type must allow lookups. As a result, COO format is unsuitable. A basic factor, which is important for the optimized selection of the stiffness matrix format, is the time spent for the initialization of the indices that each format requires. In this thesis, three different storage matrix formats have been implemented: the skyline, the CSR and the DOK. The last two formats store only the non-zero stiffness matrix coefficients and as a subsequent they demand less storage space.

On the other hand, skyline format forces the storage of needless, for the equation system solution, entries. Thus, it does not take the advantage of the computer's memory. It is a common situation the low performance of this format when the fast physical memory of the system have been totally filled and the operating system need to transfer data to the slow hard disk.

2D Problem Type	Example	DOF	Stiffness Matrix Initiliazation Time (ms)		
			Skyline	CSR	DOK
	P1-1	2.420.000	100.021	2.379	2
	P1-2	2.000.000	56.941	1.939	2
	P1-3	1.620.000	33.128	1.580	2
	P1-4	1.280.000	18.466	1.266	2
	P2-1	768.800	14.934	4.886	2
	P2-2	500.000	4.372	3.099	2
	P2-3	245.000	1.488	1.529	2
	P2-4	101.250	528	618	2
	P3-1	301.088	3.346	7.029	2
	P3-2	204.800	1.413	4.824	2
	P3-3	151.250	1.294	3.577	2
	P3-4	101.250	538	2.377	2
	P4-1	151.250	1.387	12.319	2
	P4-2	101.250	821	8.172	2
	P4-3	51.200	325	3.998	2
	P4-4	20.000	91	1.553	2

Table 6.1. Computing time for the initialization of the stiffness matrix. Element-wise implementation for 2D problems.

3D Problem Type	Example	DOF	Stiffness Matrix Initiliazation Time (ms)		
			Skyline	CSR	DOK
	P1-1	648.000	8.640	2.569	2
	P1-2	375.000	7.532	1.484	2
	P1-3	192.000	3.499	750	2
	P1-4	81.000	1.028	314	2
	P2-1	117.912	3.467	8.661	2
	P2-2	107.811	3.225	7.883	2
	P2-3	52.728	815	3.706	2
	P2-4	20.577	178	1.333	2
	P3-1	36.501	808	24.216	2
	P3-2	27.783	533	16.934	2
	P3-3	20.577	591	11.759	2
	P3-4	14.739	343	8.101	2
	P4-1	14.739	413	47.185	2
	P4-2	10.125	239	27.773	2
	P4-3	6.591	122	14.331	2
	P4-4	3.993	55	6.142	2

Table 6.2. Computing time for the initialization of the stiffness matrix. Element-wise implementation for 3D problems.

In the case of problems with numerous unknowns, the initialization time of the stiffness matrix stored in skyline format is higher than in CSR and DOK formats. This is because the size of the skyline stiffness matrix is over than the 16 GB computers' RAM and, as a result, part of the matrix is stored in the hard disk, which has lower performance and transfer rate than the RAM memory.

2D Problem Type	Example	DOF	Stiffness Matrix Assembly Time (ms)			Stiffness Matrix Assembly Ratio		
			Skyline	CSR	DOK	CSR/Skyline	DOK/Skyline	CSR/DOK
	P1-1	2.420.000	52.866	40.312	45.534	0,76	0,86	0,89
	P1-2	2.000.000	41.164	33.351	37.552	0,81	0,91	0,89
	P1-3	1.620.000	33.976	27.192	30.355	0,80	0,89	0,90
	P1-4	1.280.000	25.100	21.360	23.831	0,85	0,95	0,90
	P2-1	768.800	114.494	102.968	100.561	0,90	0,88	1,02
	P2-2	500.000	65.000	67.172	65.653	1,03	1,01	1,02
	P2-3	245.000	31.357	32.643	32.231	1,04	1,03	1,01
	P2-4	101.250	12.801	13.534	13.147	1,06	1,03	1,03
	P3-1	301.088	196.856	213.360	195.998	1,08	1,00	1,09
	P3-2	204.800	128.161	144.114	134.285	1,12	1,05	1,07
	P3-3	151.250	96.376	106.173	96.129	1,10	1,00	1,10
	P3-4	101.250	62.808	69.687	65.165	1,11	1,04	1,07
	P4-1	151.250	338.474	380.396	336.900	1,12	1,00	1,13
	P4-2	101.250	227.473	253.125	226.824	1,11	1,00	1,12
	P4-3	51.200	108.623	119.615	106.994	1,10	0,99	1,12
	P4-4	20.000	41.036	45.098	40.580	1,10	0,99	1,11

Table 6.3. Computing time ratio for the assembly of the total stiffness matrix. Different matrix formats (Skyline, CSR, DOK) for 2D problems. Element-wise implementation.

3D Problem Type	Example	DOF	Stiffness Matrix Assembly Time (ms)			Stiffness Matrix Assembly Ratio		
			Skyline	CSR	DOK	CSR/Skyline	DOK/Skyline	CSR/DOK
	P1-1	648.000	191.590	200.908	192.356	1,05	1,00	1,04
	P1-2	375.000	108.868	116.048	109.343	1,07	1,00	1,06
	P1-3	192.000	52.688	58.721	55.167	1,11	1,05	1,06
	P1-4	81.000	20.957	24.616	21.987	1,17	1,05	1,12
	P2-1	117.912	859.969	1.016.016	880.786	1,18	1,02	1,15
	P2-2	107.811	784.517	946.694	799.103	1,21	1,02	1,18
	P2-3	52.728	378.312	427.336	371.135	1,13	0,98	1,15
	P2-4	20.577	132.636	154.648	132.522	1,17	1,00	1,17
	P3-1	36.501	2.756.318	3.242.222	2.658.773	1,18	0,96	1,22
	P3-2	27.783	2.020.185	2.281.756	1.991.082	1,13	0,99	1,15
	P3-3	20.577	1.334.847	1.546.266	1.432.565	1,16	1,07	1,08
	P3-4	14.739	890.229	1.038.591	915.342	1,17	1,03	1,13
	P4-1	14.739	5.520.876	6.356.664	5.372.734	1,15	0,97	1,18
	P4-2	10.125	3.342.265	3.833.656	3.295.937	1,15	0,99	1,16
	P4-3	6.591	1.818.298	2.080.640	1.924.829	1,14	1,06	1,08
	P4-4	3.993	860.041	961.577	853.529	1,12	0,99	1,13

Table 6.4. Computing time ration for the assembly of the total stiffness matrix. Different matrix formats (Skyline, CSR, DOK) for 3D problems. Element-wise implementation.

As in the previous case, where the initialization time of the stiffness matrix depends on the relation between its size and the computers' RAM size, the stiffness matrix assembly time seems to be influenced in the same way.

More specifically, in demanding 2D problems, where the computational cost per Gauss point is lower than the corresponding in 3D problems, the computer cannot adopt this fast alternation of memory accesses and it leads to higher stiffness matrix assembly times.

2D Problem Type	Example	DOF	Stiffness Matrix Coefficients		Stiffness Matrix Size (GB)		Ratio
			Skyline	Sparse	Skyline	Sparse	
	P1-1	2.420.000	5.327.625.600	43.507.216	42,62	0,35	122,45
	P1-2	2.000.000	4.002.996.000	35.952.016	32,02	0,29	111,34
	P1-3	1.620.000	2.918.426.400	29.116.816	23,35	0,23	100,23
	P1-4	1.280.000	2.049.916.800	23.001.616	16,40	0,18	89,12
	P2-1	768.800	1.906.232.160	38.291.344	15,25	0,31	49,78
	P2-2	500.000	999.744.000	24.880.144	8,00	0,20	40,18
	P2-3	245.000	342.873.300	12.166.144	2,74	0,10	28,18
	P2-4	101.250	91.071.675	5.008.644	0,73	0,04	18,18
	P3-1	301.088	699.568.656	29.246.464	5,60	0,23	23,92
	P3-2	204.800	392.286.720	19.855.936	3,14	0,16	19,76
	P3-3	151.250	248.875.275	14.638.276	1,99	0,12	17,00
	P3-4	101.250	136.226.475	9.771.876	1,09	0,08	13,94
	P4-1	151.250	331.150.875	24.108.100	2,65	0,19	13,74
	P4-2	101.250	181.177.875	16.080.100	1,45	0,13	11,27
	P4-3	51.200	64.992.000	8.065.600	0,52	0,06	8,06
	P4-4	20.000	15.786.000	3.097.600	0,13	0,02	5,10

Table 6.5. Number of stored stiffness matrix coefficients and the storage space (in GB). Skyline and sparse formats for 2D problems.

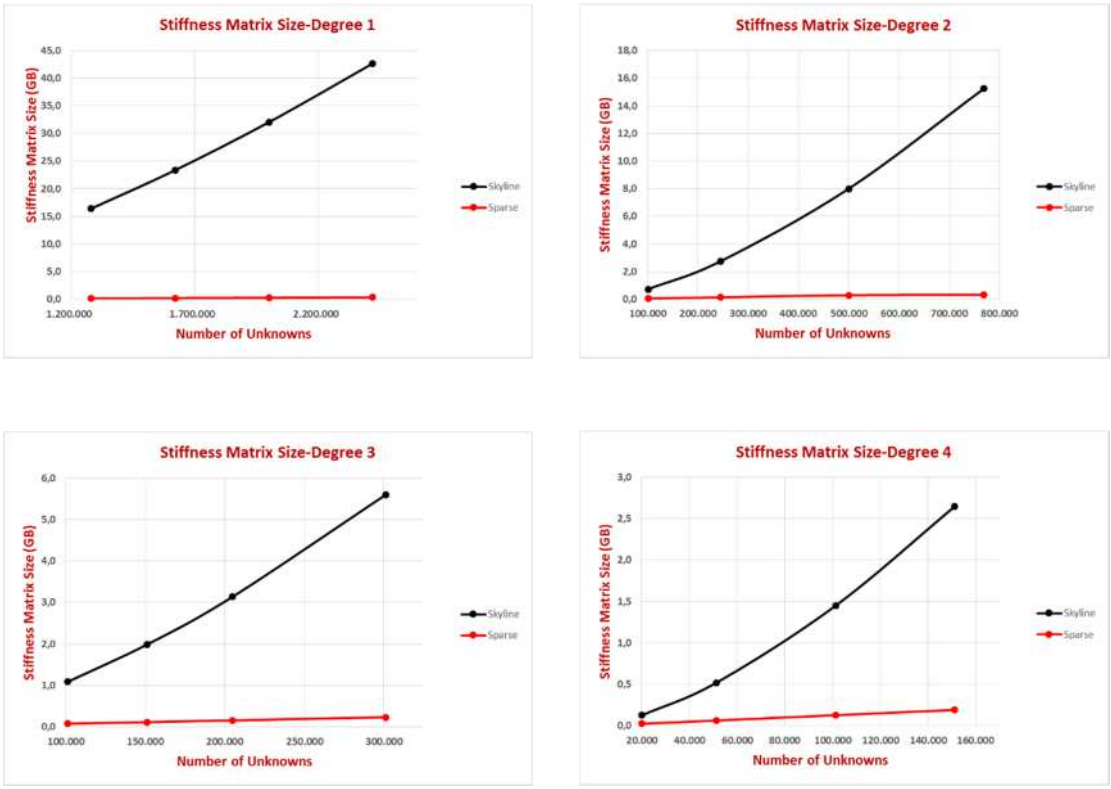


Figure 6.7. Stiffness matrix size (in GB) for 2D problems.

3D Problem Type	Example	DOF	Stiffness Matrix Coefficients		Stiffness Matrix Size (GB)		Ratio
			Skyline	Sparse	Skyline	Sparse	
	P1-1	648.000	6.999.663.600	50.757.768	56,00	0,41	137,90
	P1-2	375.000	2.813.227.500	29.176.128	22,51	0,23	96,42
	P1-3	192.000	921.969.600	14.787.288	7,38	0,12	62,35
	P1-4	81.000	218.853.900	6.133.248	1,75	0,05	35,68
	P2-1	117.912	805.661.484	39.698.496	6,45	0,32	20,29
	P2-2	107.811	693.626.571	36.177.111	5,55	0,29	19,17
	P2-3	52.728	209.681.004	17.159.616	1,68	0,14	12,22
	P2-4	20.577	43.366.569	6.344.721	0,35	0,05	6,84
	P3-1	36.501	165.941.481	29.771.541	1,33	0,24	5,57
	P3-2	27.783	104.801.445	22.143.375	0,84	0,18	4,73
	P3-3	20.577	63.172.473	15.944.049	0,51	0,13	3,96
	P3-4	14.739	35.962.293	11.025.387	0,29	0,09	3,26
	P4-1	14.739	46.342.884	21.173.733	0,37	0,17	2,19
	P4-2	10.125	24.421.500	13.687.875	0,20	0,11	1,78
	P4-3	6.591	11.703.588	8.214.057	0,09	0,07	1,42
	P4-4	3.993	4.932.444	4.437.351	0,04	0,04	1,11

Table 6.6. Number of stored stiffness matrix coefficients and the storage space (in GB). Skyline and sparse formats for 3D problems.

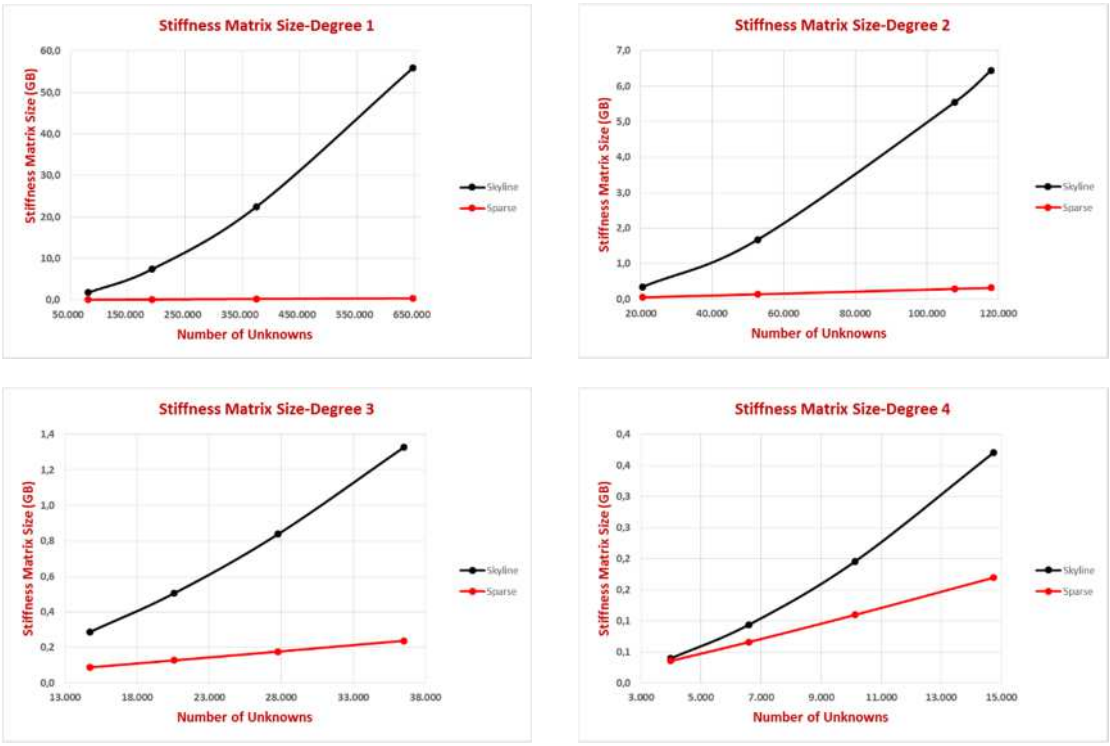


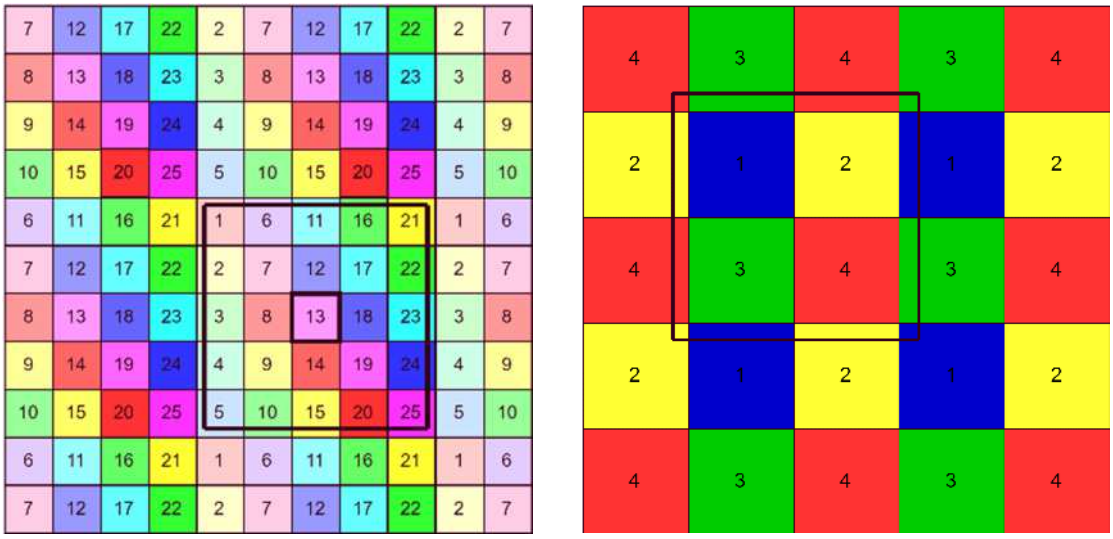
Figure 6.8. Stiffness matrix size (in GB) for 3D problems.

Undoubtedly, the benefit of using sparse format instead of skyline matrix format is obvious. This benefit tends to be increased as the number of unknowns grows for a specific B-SPLine basis function degree. Note, that the increasing polynomial degree leads to increasing B-SPLine basis function that results to denser stiffness matrices. However, due to the element overlapping of B-SPLine and NURBS functions this does not result to increased bandwidth of the stiffness matrix. Thus, for the examples P3-4 and P4-1 presented in the Appendix A, which have the same number of degrees of freedom but different degrees of the basis functions, p=3 for P3-4 and p=4 for P4-1, the size of sparse format is 0,17/ 0,09=1,89 times bigger in P4-1 problem than in P3-4. As far it concerns the skyline format the ratio is smaller and equal to 0,37/0,29=1,27.

6.2.5 Parallelization features of assembly methods

In contribution-wise method the entries of the total stiffness matrix are repetitively updated since more than one Gauss point should influence the same degrees of freedom. If the contribution of more than one interacting Gauss points is added on the total stiffness matrix at the same time, a race condition would be occurred. In order to prevent this situation from ever happening, conflicting updates of the stiffness matrix must be avoided. In GPUs this could be achieved by using atomic operations that synchronize the execution of threads that may be working concurrently. However, these atomics operations have a negative impact on the performance since they are much slower than the register access.

Another approach is to divide the elements/ Gauss points into different groups such that in every group there will be non-interacting elements. The elements/ Gauss points of a group can be computed concurrently since they update different values of the global stiffness matrix. Due to the fact that these groups can be visualized in different colors, i.e. non-interacting elements share the same color, this technique is called coloring. This procedure was firstly applied on finite elements where each element interacts only with its adjacent elements with whom it shares at least one node. In IGA due to the increased number of the interactions between the elements, the number of independent groups is decreased and as a result there is less number of data for parallelization compared to FEA. In the Figure 6.9, which correspond to the meshes of Figures 2.55.a and 2.55.b, each element has a specific color and number that indicates the group that it belongs.



(a) IGA (b) FEA

Figure 6.9. Coloring the elements in an (a) IGA (b) FEA structured mesh.

In Interaction-Wise method each K_{ij} is calculated directly and is located on the global stiffness matrix at a time. This procedure does not require the updating of the stiffness matrix coefficients. K_{ij} is calculated by taking into consideration the contribution of the Gauss points that i and j control points share. Because of the fact that the calculation and the arrangement of each K_{ij} to the global stiffness matrix becomes independently per control point pair, the interaction wise method seems ideal for parallel implementation. Thus, its thread could be assigned in every pair without need of atomic operations and synchronization points. Another great difference between CW and IW is that in IW the stiffness matrix which corresponds to a pair $i-j$ has limited dimensions. For 2D problems its dimensions is 2x2 and for 3D 3x3. This results to use directly the analytic type for the calculation of K_{ij}. In this way, needleless operations of the zero values of deformation matrix, which will be performed during matrix multiplication, are avoided. In addition, the CGMA ratio is increased since the loops that are necessary for matrix multiplication have been unrolled providing higher performance.

6.2.6 GPU implementation of the interaction-wise approach

The assembly of the stiffness matrix in interaction-wise approach could be broken in two phases. The calculation of the shape function values in the parameter space and the computation of Jacobian matrix and determinant could be done firstly. In this way the deformation matrices, which transfer the nodal displacement of the corresponding node to the strains of an element, are formatted.

Both of these phases are amenable to parallelization each one with different respect on IGA entities. Each phase is calculated with its own different kernel.

The results of the first kernel that concerns the calculation of the shape functions should be used by the second kernel which is responsible for the assembly of the stiffness matrix. In this way, data that have be used by the GPU from the execution of the first kernel and are necessary for the execution of the second kernel should not be transfer back in the host and then to the device. They can engage space of the GPU memory until the termination of the final kernel. In this way, there is time saving by the memory transfers.

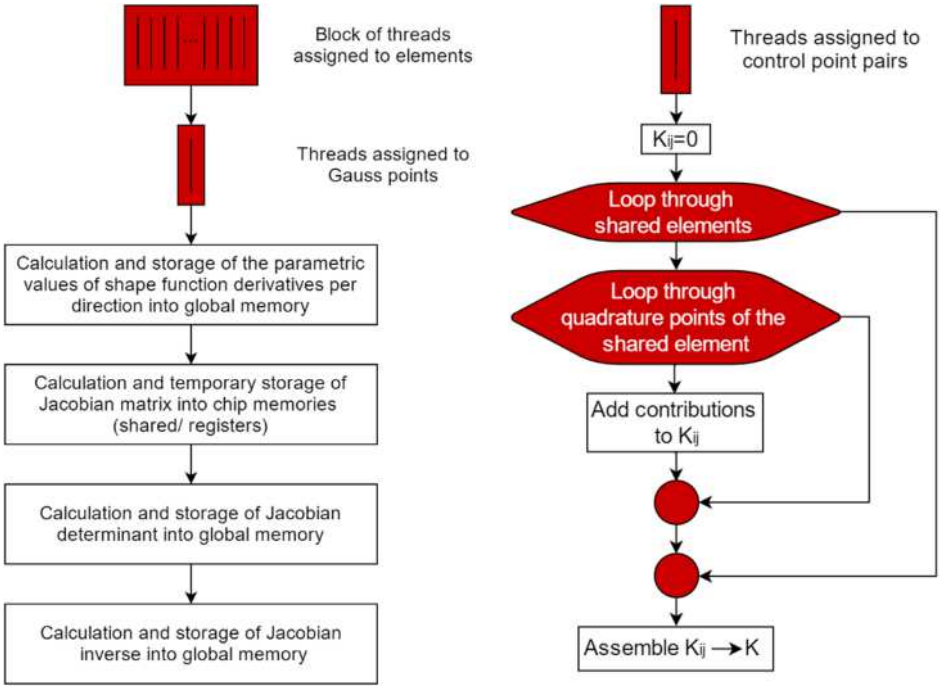
6.2.6.1 Phase 1 - calculation of derivatives & Jacobian matrix

In this phase, the shape functions derivatives, the Jacobian matrices and determinants are calculated for each influenced control point. Due to the fact that the shape functions have non-zero values only for the Gauss points that they influence, it is useful to group all the Gauss points into elements.

By identifying the control points that an element interacts with, it is easy to calculate the non-zero values of the shape functions derivatives, which correspond to these control points, on each Gauss point of the element. After that, the Jacobian matrix and determinant are computed.

The calculation of the data above, can be performed independently for each Gauss point of each element. This provides in this phase two levels of parallelization: the major over the elements and the minor over the Gauss points. It is reminded that CUDA threads are organized into a two-level hierarchy which is consisted by blocks and threads. Thus, a block of threads is assigned to each element while each thread handles only one Gauss point. Note that there is no need of using if-then-else statements that would cause the threads to follow different control flow paths and reduce the performance of the code.

This CUDA threads set-up does not come without disadvantages. More specifically, in order to ensure high hardware utilization it is recommended that the number of threads per block to be chosen as a multiple of the warp size (equal to 32 threads). However, the number of Gauss points per element is not necessarily a power of two, fact that leads to the underutilization of resources.



(a) Phase 1 **(b)** Phase 2

Figure 6.10. IW approach.
Phase 1: Blocks assigned to elements and threads to Gauss points.
Phase 2: Each thread assigned to each control point pair.

All the values of the basis functions are calculated on the CPU before the execution of the kernel. The assignment of each element to its own block of threads allows the effective handling of different memories that CUDA provides. Thus, data that have relation between different Gauss points of the same element can be stored into the shared memory. Also, the use of the constant memory for variables that do not change for the duration of the kernel like the number knot spans per axes, the number of Gauss points per knot span, etc. can have a positive impact on the performance.

6.2.6.2 Phase 2 - calculation of the total stiffness coefficients

In the second phase, each stiffness matrix coefficient, which is formulated per control point pair, can be calculated independently providing the capability of parallelization across different control point pairs. Thus, each thread is assigned to each control point pair. After that, two sequential loops iterate through the shared elements of the node pair and their enclosed Gauss points in order to collectively calculate the Kij. Since the Kij describes the interactions between i and j control point pair, the size of the Kij array is standard both for 2D (2x2) and 3D problems (3x3). This devotes the computation of the Kij extremely efficient as an analytic type can be produced and in that way the ratio of number of floating point operations for each access to the global memory is increased which is very beneficial for GPGPU computations.
The efficiency of GPU implementation is evaluated through speedup defined as the ratio:

$$S = \frac{t^{GPU}}{t^{CPU_{1CORE}}} \tag{0.1}$$

where t^{GPU} is the time left for the execution of the examined part of the code implemented on the GPU, while $t^{CPU_{1CORE}}$ is the corresponding execution time in 1 Core of the CPU.

2D Problem Type	Example	DOF	Time (ms)		
			Phase 1	Phase 2	Total
	P1-1	2.420.000	3.186	20.215	23.401
	P1-2	2.000.000	2.702	17.035	19.737
	P1-3	1.620.000	2.082	13.731	15.813
	P1-4	1.280.000	1.633	10.834	12.467
	P2-1	768.800	4.804	72.505	77.309
	P2-2	500.000	3.178	47.313	50.491
	P2-3	245.000	1.519	22.985	24.504
	P2-4	101.250	641	9.232	9.873
	P3-1	301.088	5.858	157.057	162.915
	P3-2	204.800	3.928	106.635	110.563
	P3-3	151.250	2.893	78.599	81.492
	P3-4	101.250	1.938	51.915	53.853
	P4-1	151.250	6.841	294.491	301.332
	P4-2	101.250	4.405	189.588	193.993
	P4-3	51.200	2.192	92.534	94.726
	P4-4	20.000	842	35.528	36.370

Table 6.7. Computing time for the formulation of the stiffness matrix of 2D geometric domains. CPU implementation of the interaction-wise (IW) approach with a Core i7-3610QM at 2.30 GHz.

2D Problem Type	Example	DOF	Time (ms)		
			Phase 1	Phase 2	Total
	P1-1	2.420.000	218	348	566
	P1-2	2.000.000	195	295	490
	P1-3	1.620.000	150	242	392
	P1-4	1.280.000	135	195	330
	P2-1	768.800	206	1.859	2.065
	P2-2	500.000	150	1.222	1.372
	P2-3	245.000	70	583	653
	P2-4	101.250	39	237	276
	P3-1	301.088	125	4.908	5.033
	P3-2	204.800	81	3.504	3.585
	P3-3	151.250	77	2.600	2.677
	P3-4	101.250	49	1.637	1.686
	P4-1	151.250	112	10.907	11.019
	P4-2	101.250	82	7.334	7.416
	P4-3	51.200	60	3.347	3.407
	P4-4	20.000	28	1.250	1.278

Table 6.8. Computing time. Stiffness matrix formulation. 2D problems. GPU implementation of the interaction-wise (IW) approach with a GeForce GTX 670M.

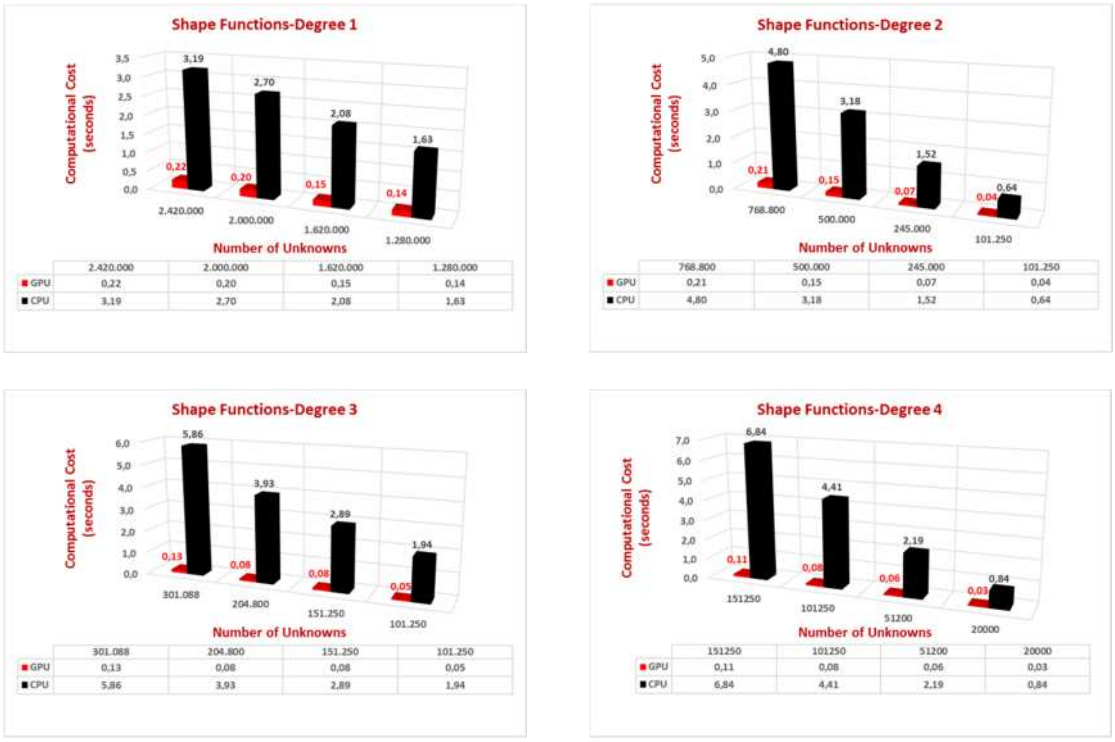


Figure 6.11. Computing time. Phase 1. 2D problems. CPU & GPU implementations.

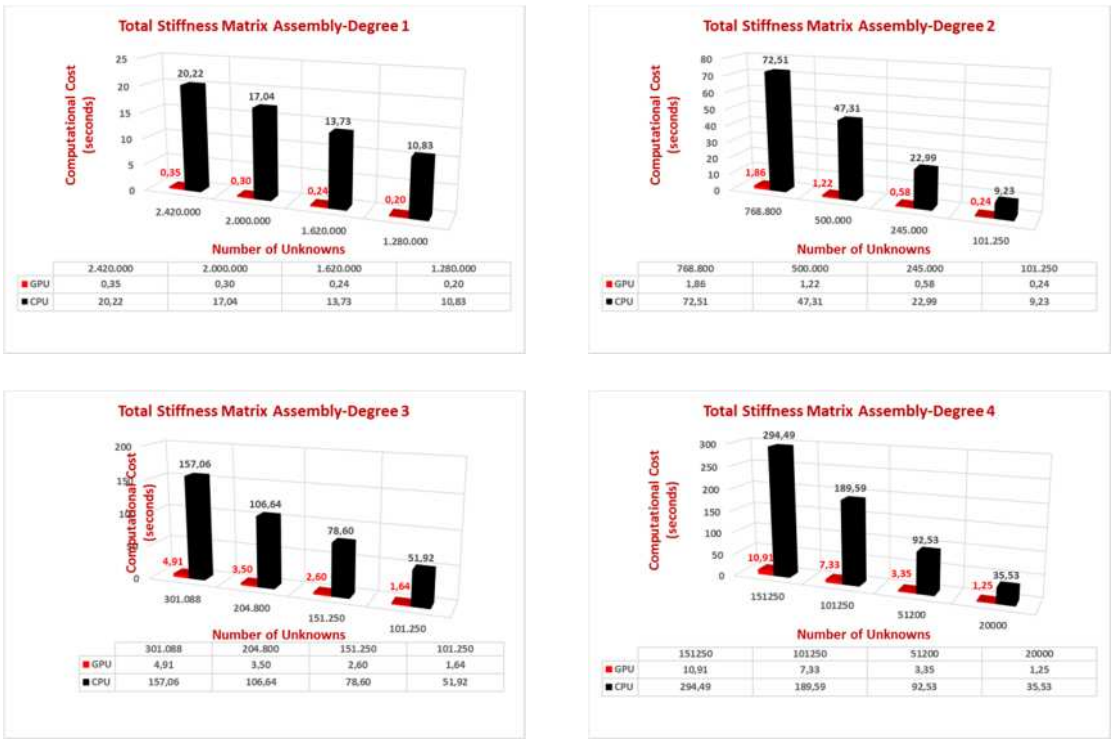


Figure 6.12. Computing time. Phase 2. 2D problems.
CPU (one Core i7-3610QM) and GPU (GTX 670M) implementation of the interaction-wise (IW) approach.

2D Problem Type	Example	DOF	Speedup Ratios of GPU impl.		
			Phase 1	Phase 2	Total
	P1-1	2.420.000	15	58	41
	P1-2	2.000.000	14	58	40
	P1-3	1.620.000	14	57	40
	P1-4	1.280.000	12	56	38
	P2-1	768.800	23	39	37
	P2-2	500.000	21	39	37
	P2-3	245.000	22	39	38
	P2-4	101.250	16	39	36
	P3-1	301.088	47	32	32
	P3-2	204.800	48	30	31
	P3-3	151.250	38	30	30
	P3-4	101.250	40	32	32
	P4-1	151.250	61	27	27
	P4-2	101.250	54	26	26
	P4-3	51.200	37	28	28
	P4-4	20.000	30	28	28

Table 6.9. Speedup ratios for the formulation of the stiffness matrix of 2D geometric domains of the GPU (GTX 670M) implementation compared to the single core CPU (Core i7-3610QM) implementation of the IW approach.

2D Problem Type	Example	DOF	GPU Reserved Memory per Phase (GB)	
			Phase 1	Phase 2
	P1-1	2.420.000	0,61	1,45
	P1-2	2.000.000	0,50	1,19
	P1-3	1.620.000	0,41	0,97
	P1-4	1.280.000	0,32	0,76
	P2-1	768.800	0,68	1,49
	P2-2	500.000	0,44	0,98
	P2-3	245.000	0,22	0,48
	P2-4	101.250	0,09	0,20
	P3-1	301.088	0,72	1,43
	P3-2	204.800	0,49	0,97
	P3-3	151.250	0,36	0,71
	P3-4	101.250	0,24	0,48
	P4-1	151.250	0,82	1,46
	P4-2	101.250	0,55	0,97
	P4-3	51.200	0,27	0,49
	P4-4	20.000	0,10	0,19

Table 6.10. Demand of GPU's DRAM memory for both parallel phases for 2D problems.

3D Problem Type	Example	DOF	Time (ms)		
			Phase 1	Phase 2	Total
	P1-1	648.000	2.747	66.174	68.921
	P1-2	375.000	1.552	39.128	40.680
	P1-3	192.000	797	19.826	20.623
	P1-4	81.000	314	7.789	8.103
	P2-1	117.912	4.779	390.236	395.015
	P2-2	107.811	4.309	355.109	359.418
	P2-3	52.728	2.038	167.717	169.755
	P2-4	20.577	690	59.724	60.414
	P3-1	36.501	6.394	1.276.194	1.282.588
	P3-2	27.783	4.738	947.512	952.250
	P3-3	20.577	3.346	660.692	664.038
	P3-4	14.739	2.212	433.917	436.129
	P4-1	14.739	6.682	2.770.993	2.777.675
	P4-2	10.125	4.093	1.701.021	1.705.114
	P4-3	6.591	2.181	881.820	884.001
	P4-4	3.993	1.017	398.875	399.892

Table 6.11. Computing time for the formulation of the stiffness matrix of 3D domains.
CPU implementation of the interaction-wise (IW) approach with a Core i7-3610QM at 2.30 GHz.

3D Problem Type	Example	DOF	Time (ms)		
			Phase 1	Phase 2	Total
	P1-1	648.000	219	2.057	2.276
	P1-2	375.000	140	1.239	1.379
	P1-3	192.000	91	623	714
	P1-4	81.000	40	272	312
	P2-1	117.912	135	13.937	14.072
	P2-2	107.811	126	12.682	12.808
	P2-3	52.728	62	6.283	6.345
	P2-4	20.577	34	2.185	2.219
	P3-1	36.501	111	47.266	47.377
	P3-2	27.783	90	36.613	36.703
	P3-3	20.577	69	25.907	25.976
	P3-4	14.739	52	17.018	17.070
	P4-1	14.739	113	95.551	95.664
	P4-2	10.125	91	60.049	60.140
	P4-3	6.591	63	31.629	31.692
	P4-4	3.993	44	14.026	14.070

Table 6.12. Computing time for the formulation of the stiffness matrix of 2D geometric domains.
GPU implementation of the interaction-wise (IW) approach with a GeForce GTX 670M.

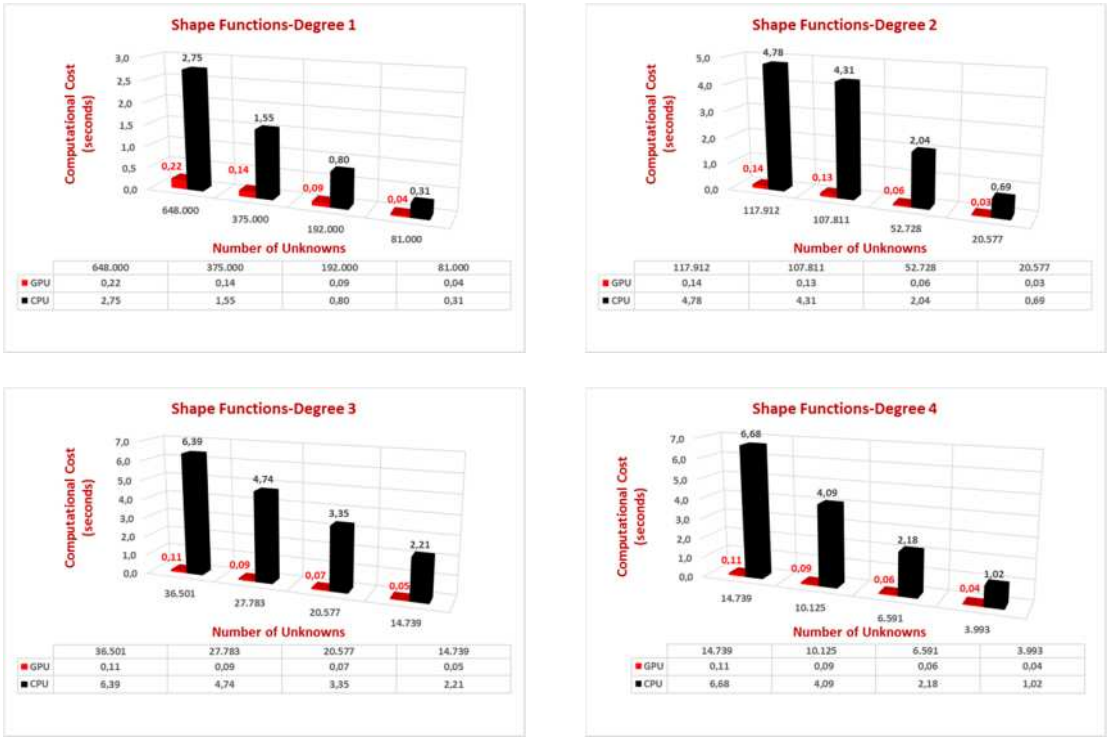


Figure 6.13. Computing time. Phase 1. 3D problems.
CPU & GPU implementations.

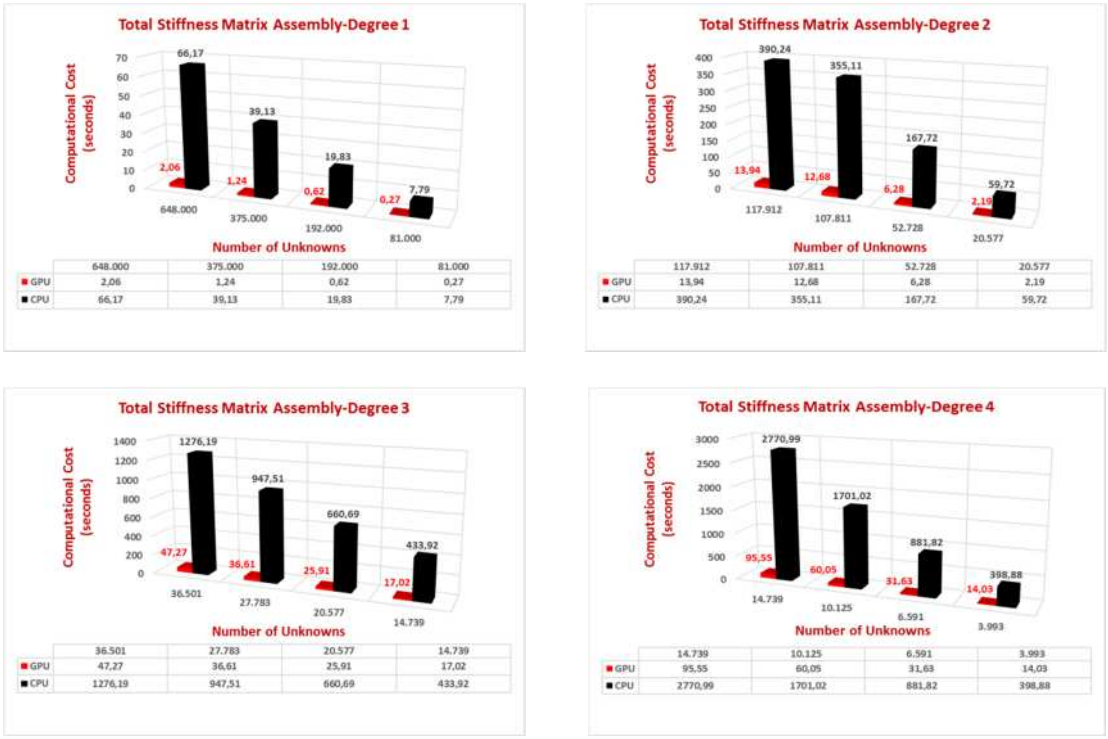


Figure 6.14. Computing time. Phase 2. 3D problems.
CPU (one Core i7-3610QM) & GPU (GTX 670M) implementation
of the interaction-wise approach.

3D Problem Type	Example	DOF	Speedup Ratios of GPU impl.		
			Phase 1	Phase 2	Total
	P1-1	648.000	13	32	30
	P1-2	375.000	11	32	29
	P1-3	192.000	9	32	29
	P1-4	81.000	8	29	26
	P2-1	117.912	35	28	28
	P2-2	107.811	34	28	28
	P2-3	52.728	33	27	27
	P2-4	20.577	20	27	27
	P3-1	36.501	58	27	27
	P3-2	27.783	53	26	26
	P3-3	20.577	48	26	26
	P3-4	14.739	43	25	26
	P4-1	14.739	59	29	29
	P4-2	10.125	45	28	28
	P4-3	6.591	35	28	28
	P4-4	3.993	23	28	28

Table 6.13. Speedup ratios for the formulation of the stiffness matrix of 3D geometric domains of the GPU (GTX 670M) implementation compared to the single core CPU (Core i7-3610QM) implementation of the IW approach.

3D Problem Type	Example	DOF	GPU Reserved Memory per Phase (GB)	
			Phase 1	Phase 2
	P1-1	648.000	0,45	1,39
	P1-2	375.000	0,26	0,80
	P1-3	192.000	0,13	0,40
	P1-4	81.000	0,05	0,17
	P2-1	117.912	0,65	1,43
	P2-2	107.811	0,59	1,31
	P2-3	52.728	0,27	0,61
	P2-4	20.577	0,10	0,22
	P3-1	36.501	0,83	1,48
	P3-2	27.783	0,60	1,09
	P3-3	20.577	0,42	0,77
	P3-4	14.739	0,28	0,52
	P4-1	14.739	0,85	1,35
	P4-2	10.125	0,51	0,83
	P4-3	6.591	0,28	0,47
	P4-4	3.993	0,13	0,23

Table 6.14. Demand of GPU's DRAM memory for both parallel phases for 3D problems.

6.3 Equations

Gauss point-wise variant of the contribution-wise method is the most generic approach of formulating the stiffness matrix of a structure. All the other ways of stiffness matrix formulation are based on it. So, let us to formulate the stiffness matrix and present the flow of the computations that are determined by this method.

6.3.1 Stiffness matrix 1D

Consider a simple 1D application as an example. These applications are utilized only in truss systems with axial tension, but their significance is mostly academic. Simplifying the problem, without loss of generality, can lead to a better understanding of the principles involved. In our research career, we often address 1D analogies for the solution of complex problems and it has always been extremely helpful.

In such a case, only axial deformation for each point of the truss exists. This deformation is $u(x) = u(C(\xi)) = u(\xi)$. The respective strain matrix consists of one value:

$$\underset{(1\times1)}{\{\varepsilon\}} = [\varepsilon_x] = \left[\frac{\partial u}{\partial x}\right]$$

In order to calculate the derivative of u, we must first establish a transition from Physical Space to Parameter Space:

$$\frac{\partial \varphi}{\partial x} = \frac{\partial \varphi}{\partial \xi}\frac{\partial \xi}{\partial x}, \quad \frac{\partial \varphi}{\partial \xi} = \frac{\partial \varphi}{\partial x}\frac{\partial x}{\partial \xi}$$

$$\left[\frac{\partial \varphi}{\partial \xi}\right] = [J]\cdot\left[\frac{\partial \varphi}{\partial x}\right]$$

where $[J]$ is the Jacobian matrix enabling transition from Physical to Parameter Space and vice-versa. It can be evaluated with the help of basis functions $R_i(\xi)$ and the control points' Cartesian coordinates X_i as shown:

$$\underset{(1\times1)}{\left[J(\xi)\right]} = \underset{(1\times n)}{\left[R_{1,\xi}(\xi) \quad R_{2,\xi}(\xi) \quad \dots \quad \dots \quad \dots \quad R_{n,\xi}(\xi)\right]}\cdot\underset{(n\times1)}{\begin{bmatrix} X_1 \\ X_2 \\ \cdot \\ \cdot \\ \cdot \\ X_n \end{bmatrix}}$$

where $R_{i,\xi}(\xi) = \frac{\partial}{\partial \xi}R_i(\xi)$.

In Finite Element Analysis, the reverse transformation is utilized. This is why the inverse matrix $\underset{(1x1)}{[J]^{-1}}$ is needed.

$$\underset{(1x1)}{[J]^{-1}} = \frac{1}{\underset{(1\times1)}{[J]}}$$

Special care has to be taken in order for the Jacobian to be correct. The positive direction of the axes in Parameter and Physical Space must coincide, or the determinant of the Jacobian will be negative and the matrix $[J]$ irreversible. Numerical integration on points of singularity, such as two points on Parameter Space mapped into the same point on Physical Space, has to be avoided as well.

The next step is to calculate the matrices $[B_1]$ and $[B_2]$. The matrix $[B_1]$ transfers the strains of the element from Parameter to Physical Space and the matrix $[B_2]$ transfers the nodal displacements of the elements to the strains at the Parameter Space. Therefore, the matrices $[B_1]$and $[B_2]$ can be calculated from the equations below:

$$\underset{(1\times1)}{\{\varepsilon\}} = \underset{(1\times1)}{\left[\frac{\partial u}{\partial x}\right]} = \underset{(1\times1)}{[J]^{-1}} \cdot \left[\frac{\partial u}{\partial \xi}\right]$$

$$\underset{(1\times1)}{\left[B_1(\xi)\right]} = \left[\frac{1}{J_{11}}\right]$$

$$\underset{(1\times1)}{\{\varepsilon\}} = \underset{(1\times1)}{[B_1]} \cdot \underset{(1\times1)}{\{u_\xi\}}$$

$$\left[\frac{\partial u}{\partial \xi}\right] = \begin{bmatrix} R_{1,\xi}(\xi) & R_{2,\xi}(\xi) & \dots & \dots & \dots & R_{n,\xi}(\xi) \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_n \end{bmatrix}$$

$$\underset{(1\times n)}{\left[B_2(\xi)\right]} = \begin{bmatrix} R_{1,\xi}(\xi) & R_{2,\xi}(\xi) & \dots & \dots & \dots & R_{n,\xi}(\xi) \end{bmatrix}$$

$$\underset{(1\times1)}{\{u\}} = \underset{(1\times n)}{[B_2]} \cdot \underset{(n\times1)}{\{d\}}$$

After that, the deformation matrix is evaluated.

$$\underset{(1\times n)}{\left[B(\xi)\right]} = \underset{(1\times1)}{\left[B_1(\xi)\right]} \cdot \underset{(1\times n)}{\left[B_2(\xi)\right]}$$

Deformation matrix produces strain values anywhere in the model, by utilizing nodal displacements.

$$\underset{(1\times1)}{\{\varepsilon(\xi)\}} = \underset{(1\times n)}{[B(\xi)]} \cdot \underset{(n\times1)}{\{d\}}$$

The stiffness matrix for the patch is evaluated as shown:

$$\underset{(n\times n)}{[K]} = \int_{\xi_0}^{\xi_{n+p+1}} \underset{(n\times1)}{[B(\xi)]}^{T} \cdot \underset{(1x1)}{[E]} \cdot \underset{(1\times n)}{[B(\xi)]} \cdot A \cdot \det[J]\, d\xi$$

Direct integration is almost never applicable. Numerical integration is used instead, looping through all the Gauss points of a patch and their respective weights:

$$\underset{(n\times n)}{[K]} = \sum_{i=1}^{GP\xi} \left\{ \underset{(n\times1)}{[B(\xi_i)]}^{T} \cdot \underset{(1x1)}{[E]} \cdot \underset{(1\times n)}{[B(\xi_i)]} \cdot A \cdot \det[J] \cdot w_i^{GP\xi} \right\}$$

where

- A : the area of the cross-section
- GP_ξ : the total number of Gauss points for the specific patch
- ξ_i : the coordinates of the Gauss points
- $w_i^{GP\xi}$: the corresponding weights.

6.3.2 Stiffness matrix 2D

2D elasticity problems have many applications in modern analysis. The logic is exactly the same as in 1D problems. The main difference is, obviously, the utilization of one more dimension. Parameter Space is defined on (ξ,η) and Physical Space on (x,y). Displacements per x, y at any point in the entire domain are defined as $u(x,y) = u(S(\xi,\eta)) = u(\xi,\eta)$ and $v(x,y) = v(\xi,\eta)$ respectively.

The strain vector is defined as:

$$\underset{(3\times1)}{\{\varepsilon\}} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{bmatrix} \Rightarrow \underset{(3\times1)}{\{\varepsilon\}} = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$

The transformation of a function φ between Parameter and Physical Space yields:

$$\frac{\partial \varphi}{\partial x} = \frac{\partial \varphi}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial \varphi}{\partial \eta}\frac{\partial \eta}{\partial x}$$

$$\frac{\partial \varphi}{\partial y} = \frac{\partial \varphi}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial \varphi}{\partial \eta}\frac{\partial \eta}{\partial y}$$

$$\frac{\partial \varphi}{\partial \xi} = \frac{\partial \varphi}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial \varphi}{\partial y}\frac{\partial y}{\partial \xi}$$

$$\frac{\partial \varphi}{\partial \eta} = \frac{\partial \varphi}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial \varphi}{\partial y}\frac{\partial y}{\partial \eta}$$

Thus, the 2D Jacobian matrix can be defined as:

$$\begin{bmatrix} \frac{\partial \varphi}{\partial \xi} \\ \frac{\partial \varphi}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{\partial \varphi}{\partial \xi} \\ \frac{\partial \varphi}{\partial \eta} \end{bmatrix} = \underset{(2\times 2)}{[J]} \cdot \begin{bmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \end{bmatrix}$$

and the inverse mapping:

$$\begin{bmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \end{bmatrix} = \underset{(2\times 2)}{[J]}^{-1} \cdot \begin{bmatrix} \frac{\partial \varphi}{\partial \xi} \\ \frac{\partial \varphi}{\partial \eta} \end{bmatrix}$$

The Jacobian matrix can be evaluated as shown:

$$\underset{(2\times 2)}{[J]} = \begin{bmatrix} R_{1,\xi}(\xi,\eta) & R_{2,\xi}(\xi,\eta) & \dots & \dots & \dots & R_{N,\xi}(\xi,\eta) \\ R_{1,\eta}(\xi,\eta) & R_{2.\eta}(\xi,\eta) & \dots & \dots & \dots & R_{N,\eta}(\xi,\eta) \end{bmatrix} \cdot \begin{bmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ . & . \\ . & . \\ . & . \\ X_N & Y_N \end{bmatrix}$$

where $N = n \cdot m$ is the total number of control points.

The inverse Jacobian matrix is used in stiffness matrix calculation:

$$\underset{(2\times 2)}{[J]}^{-1} = \begin{bmatrix} J^*_{11} & J^*_{12} \\ J^*_{21} & J^*_{22} \end{bmatrix} = \frac{1}{\det[J]} \cdot \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix}$$

The determinant of the Jacobian matrix is also required and is equal to:

$$\det[J] = J_{11} \cdot J_{22} - J_{21} \cdot J_{12}$$

In order to calculate the deformation matrix for 2D problems, $[B_1]$ and $[B_2]$ have to be evaluated as usual.

To obtain matrix $[B_1]$:

$$\underset{(3\times1)}{\{\varepsilon\}} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{bmatrix} = \frac{1}{\det[J]} \cdot \begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ 0 & 0 & -J_{21} & J_{11} \\ -J_{21} & J_{11} & J_{22} & -J_{12} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{bmatrix}$$

Hence,

$$\underset{(3\times4)}{[B_1(\xi,\eta)]} = \frac{1}{\det[J]} \cdot \begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ 0 & 0 & -J_{21} & J_{11} \\ -J_{21} & J_{11} & J_{22} & -J_{12} \end{bmatrix}$$

To calculate matrix $[B_2]$:

$$\begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{bmatrix} = \begin{bmatrix} R_{1,\xi} & 0 & R_{2,\xi} & 0 & \dots & \dots & \dots & R_{N,\xi} & 0 \\ R_{1,\eta} & 0 & R_{2,\eta} & 0 & \dots & \dots & \dots & R_{N,\eta} & 0 \\ 0 & R_{1,\xi} & 0 & R_{2,\xi} & \dots & \dots & \dots & 0 & R_{N,\xi} \\ 0 & R_{1,\eta} & 0 & R_{2,\eta} & \dots & \dots & \dots & 0 & R_{N,\eta} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ . \\ . \\ . \\ u_N \\ v_N \end{bmatrix}$$

Hence,

$$\underset{(4\times2N)}{[B_2(\xi,\eta)]} = \begin{bmatrix} R_{1,\xi} & 0 & R_{2,\xi} & 0 & \dots & \dots & \dots & R_{N,\xi} & 0 \\ R_{1,\eta} & 0 & R_{2,\eta} & 0 & \dots & \dots & \dots & R_{N,\eta} & 0 \\ 0 & R_{1,\xi} & 0 & R_{2,\xi} & \dots & \dots & \dots & 0 & R_{N,\xi} \\ 0 & R_{1,\eta} & 0 & R_{2,\eta} & \dots & \dots & \dots & 0 & R_{N,\eta} \end{bmatrix}$$

Having determined $[B_1]$ and $[B_2]$, the deformation matrix is calculated as:

$$\underset{(3\times 2N)}{[B(\xi,\eta)]} = \underset{(3\times 4)}{[B_1(\xi,\eta)]} \cdot \underset{(4\times 2N)}{[B_2(\xi,\eta)]}$$

In order to evaluate the stiffness matrix, integration is required.

$$\underset{(2N\times 2N)}{[K]} = \int_{\xi_0}^{\xi_{n+p+1}} \int_{\eta_0}^{\eta_{m+q+1}} \underset{(2N\times 3)}{[B(\xi,\eta)]^T} \cdot \underset{(3x3)}{[E]} \cdot \underset{(3\times 2N)}{[B(\xi,\eta)]} \cdot t \cdot \det[J]\, d\eta d\xi$$

Numerical integration procedures for ξ,η lead to integration for tensor product Gauss points.

$$\underset{(2N\times 2N)}{[K]} = \sum_{i=1}^{GP\xi} \sum_{j=1}^{GP\eta} \underset{(2N\times 3)}{[B(\xi_i,\eta_j)]^T} \cdot \underset{(3x3)}{[E]} \cdot \underset{(3\times 2N)}{[B(\xi_i,\eta_j)]} \cdot t \cdot \det[J] \cdot w_i^{GP\xi} \cdot w_j^{GP\eta}$$

where:

- t: the thickness of the cross-section
- GP_ξ: the total number of Gauss points per ξ for the specific patch
- GP_η: the total number of Gauss points per η for the specific patch
- ξ_i, η_j: the coordinates of the tensor product Gauss point i, j
- $w_i^{GP\xi}$, $w_j^{GP\eta}$: the corresponding weights

The only difference, at this point, between plane stress and plane strain is the elasticity matrix which is the result of the utilized Constitutive Law.

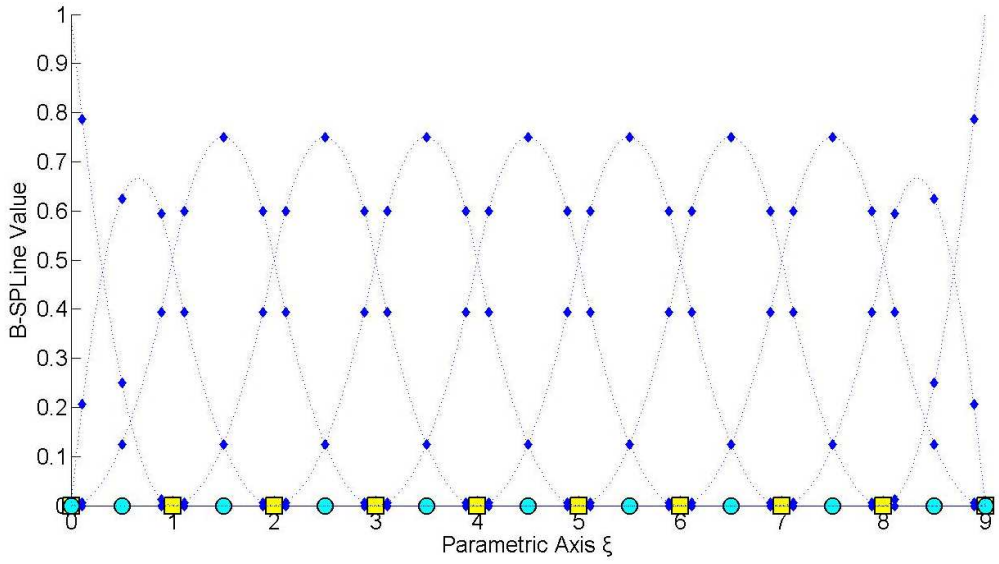


Figure 6.15. Quadratic basis functions calculated at Gauss points.
Ξ={0,0,0,1,2,3,4,5,6,7,8,9,9,9}

6.3.3 Stiffness matrix 3D

3D elasticity is merely the extension of 2D elasticity in all directions, with a complete stress field. Every other problem can be created by downgrading 3D problems into 2D and 1D problems.

The displacement field for each point in Physical Space is now defined for x, y, z by $u(x,y,z) = u(S(\xi,\eta,\zeta)) = u(\xi,\eta,\zeta)$, $v(\xi,\eta,\zeta)$, $w(\xi,\eta,\zeta)$ respectively. The strain field can now be defined as:

$$\underset{(6\times1)}{\{\varepsilon\}} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial w}{\partial z} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

which leads to the definition of the Jacobian matrix for 3D:

$$\begin{bmatrix} \frac{\partial \varphi}{\partial \xi} \\ \frac{\partial \varphi}{\partial \eta} \\ \frac{\partial \varphi}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \\ \frac{\partial \varphi}{\partial z} \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{\partial \varphi}{\partial \xi} \\ \frac{\partial \varphi}{\partial \eta} \\ \frac{\partial \varphi}{\partial \zeta} \end{bmatrix} = \underset{3\times3}{[J]} \cdot \begin{bmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \\ \frac{\partial \varphi}{\partial z} \end{bmatrix}$$

and the inverse Jacobian matrix as well:

$$\begin{bmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \\ \frac{\partial \varphi}{\partial z} \end{bmatrix} = \underset{(3\times3)}{[J]^{-1}} \cdot \begin{bmatrix} \frac{\partial \varphi}{\partial \xi} \\ \frac{\partial \varphi}{\partial \eta} \\ \frac{\partial \varphi}{\partial \zeta} \end{bmatrix}$$

Jacobian matrix can be calculated from the derivatives of the shape functions.

$$\underset{(3\times3)}{[J]} = \begin{bmatrix} R_{1,\xi}(\xi,\eta,\zeta) & R_{2,\xi}(\xi,\eta,\zeta) & \dots & \dots & \dots & R_{N,\xi}(\xi,\eta,\zeta) \\ R_{1,\eta}(\xi,\eta,\zeta) & R_{2.\eta}(\xi,\eta,\zeta) & \dots & \dots & \dots & R_{N,\eta}(\xi,\eta,\zeta) \\ R_{1,\zeta}(\xi,\eta,\zeta) & R_{2,\zeta}(\xi,\eta,\zeta) & \dots & \dots & \dots & R_{N,\zeta}(\xi,\eta,\zeta) \end{bmatrix} \begin{bmatrix} X_1 & Y_1 & Z_1 \\ X_2 & Y_2 & Z_2 \\ . & . & . \\ . & . & . \\ . & . & . \\ X_N & Y_N & Z_N \end{bmatrix}$$

The inverse of the Jacobian matrix is:

$$\underset{(3\times3)}{[J]^{-1}} = \begin{bmatrix} J^*_{11} & J^*_{12} & J^*_{13} \\ J^*_{21} & J^*_{22} & J^*_{23} \\ J^*_{31} & J^*_{32} & J^*_{33} \end{bmatrix}$$

$[B_1]$ is evaluated as:

$$\underset{(6\times9)}{\left[B_1(\xi,\eta,\zeta)\right]} = \begin{bmatrix} J^*_{11} & J^*_{12} & J^*_{13} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & J^*_{21} & J^*_{22} & J^*_{23} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & J^*_{31} & J^*_{32} & J^*_{33} \\ J^*_{21} & J^*_{22} & J^*_{23} & J^*_{11} & J^*_{12} & J^*_{13} & 0 & 0 & 0 \\ 0 & 0 & 0 & J^*_{31} & J^*_{32} & J^*_{33} & J^*_{21} & J^*_{22} & J^*_{23} \\ J^*_{31} & J^*_{32} & J^*_{33} & 0 & 0 & 0 & J^*_{11} & J^*_{12} & J^*_{13} \end{bmatrix}$$

As for $[B_2]$:

$$\underset{(9\times3N)}{\left[B_2(\xi,\eta,\zeta)\right]} = \begin{bmatrix} R_{1,\xi} & 0 & 0 & R_{2,\xi} & 0 & 0 & \dots & \dots & \dots & R_{N,\xi} & 0 & 0 \\ R_{1,\eta} & 0 & 0 & R_{2,\eta} & 0 & 0 & \dots & \dots & \dots & R_{N,\eta} & 0 & 0 \\ R_{1,\zeta} & 0 & 0 & R_{2,\zeta} & 0 & 0 & \dots & \dots & \dots & R_{N,\zeta} & 0 & 0 \\ 0 & R_{1,\xi} & 0 & 0 & R_{2,\xi} & 0 & \dots & \dots & \dots & 0 & R_{N,\xi} & 0 \\ 0 & R_{1,\eta} & 0 & 0 & R_{2,\eta} & 0 & \dots & \dots & \dots & 0 & R_{N,\eta} & 0 \\ 0 & R_{1,\zeta} & 0 & 0 & R_{2,\zeta} & 0 & \dots & \dots & \dots & 0 & R_{N,\zeta} & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & R_{1,\xi} & 0 & 0 & R_{2,\xi} & \dots & \dots & \dots & 0 & 0 & R_{N,\xi} \\ 0 & 0 & R_{1,\eta} & 0 & 0 & R_{2,\eta} & \dots & \dots & \dots & 0 & 0 & R_{N,\eta} \\ 0 & 0 & R_{1,\zeta} & 0 & 0 & R_{2,\zeta} & \dots & \dots & \dots & 0 & 0 & R_{N,\zeta} \end{bmatrix}$$

As a result, the deformation matrix for 3D elasticity is calculated as:

$$\underset{(6\times 3N)}{\left[B(\xi,\eta,\zeta)\right]} = \underset{(6\times 9)}{\left[B_1(\xi,\eta,\zeta)\right]} \cdot \underset{(9\times 3N)}{\left[B_2(\xi,\eta,\zeta)\right]}$$

The corresponding stiffness matrix is produced by integration

$$\underset{(3N\times 3N)}{[K]} = \int_{\xi_0}^{\xi_{n+p+1}} \int_{\eta_0}^{\eta_{m+q+1}} \int_{\zeta_0}^{\zeta_{l+r+1}} \underset{(3N\times 6)}{\left[B(\xi,\eta,\zeta)\right]^T} \cdot \underset{(6x6)}{[E]} \cdot \underset{(6\times 3N)}{\left[B(\xi,\eta,\zeta)\right]} \cdot \det[J]\, d\zeta d\eta d\xi$$

Numerical integration is used in 3D as well

$$\underset{(3N\times 3N)}{[K]} = \sum_{i=1}^{GP\xi} \sum_{j=1}^{GP\eta} \sum_{k=1}^{GP\zeta} \underset{(3N\times 6)}{\left[B(\xi_i,\eta_j,\zeta_k)\right]^T} \cdot \underset{(6x6)}{[E]} \cdot \underset{(6\times 3N)}{\left[B(\xi_i,\eta_j,\zeta_k)\right]} \cdot \det[J] \cdot w_i^{GP\xi} \cdot w_j^{GP\eta} \cdot w_k^{GP\zeta}$$

where:

- GP_ξ: the total number of Gauss points per ξ for the specific patch
- GP_η: the total number of Gauss points per η for the specific patch
- GP_ζ: the total number of Gauss points per ζ for the specific patch
- ξ_i, η_j, ζ_k: the coordinates of the tensor product Gauss point ijk
- $w_i^{GP\xi}$, $w_j^{GP\eta}$, $w_k^{GP\zeta}$: the corresponding weights of Gauss points

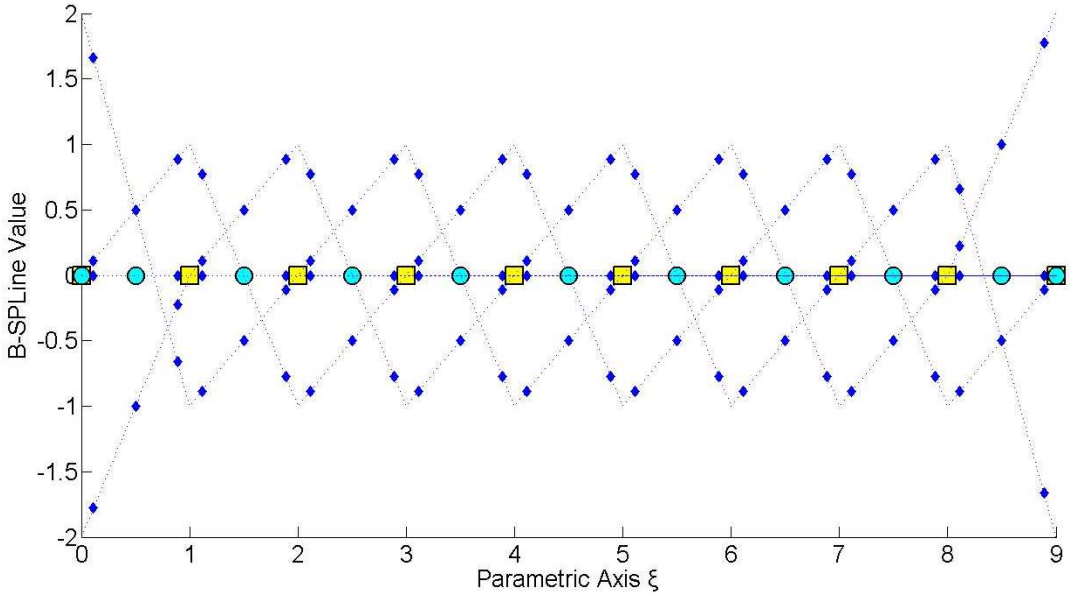
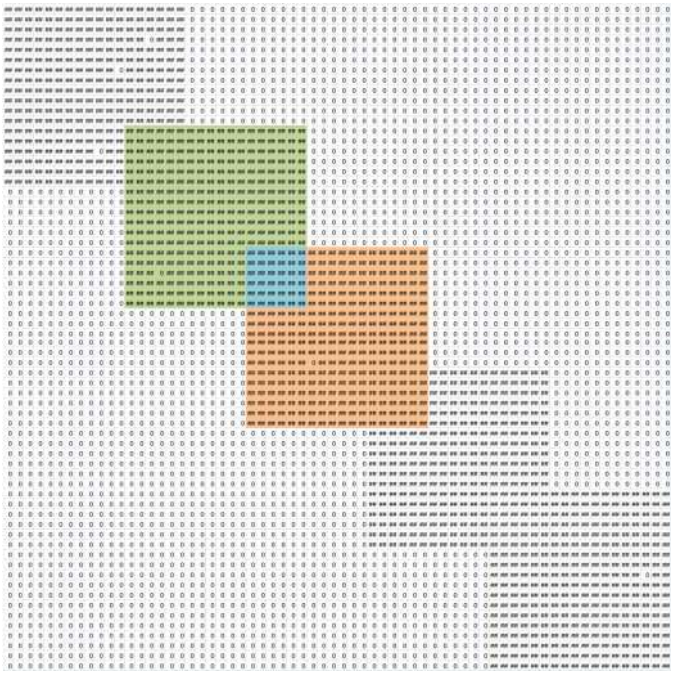
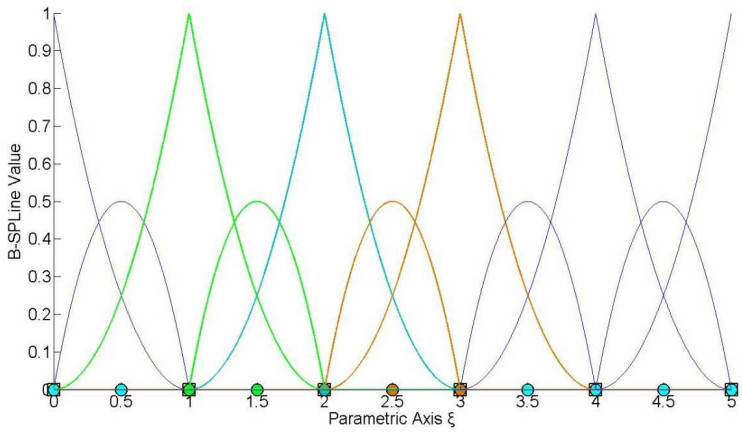


Figure 6.16. Quadratic basis function derivatives evaluated at Gauss points.
Ξ={0,0,0,1,2,3,4,5,6,7,8,9,9,9}

6.3.4 Examples



(a) Stiffness matrix.

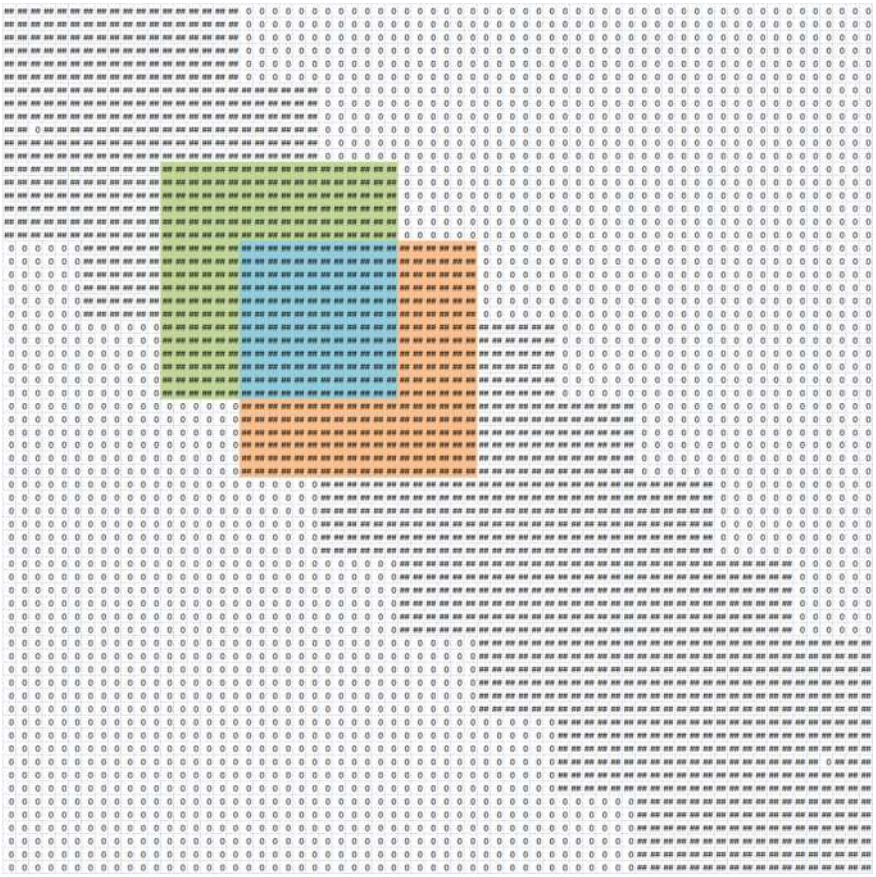


(b) Quadratic basis.

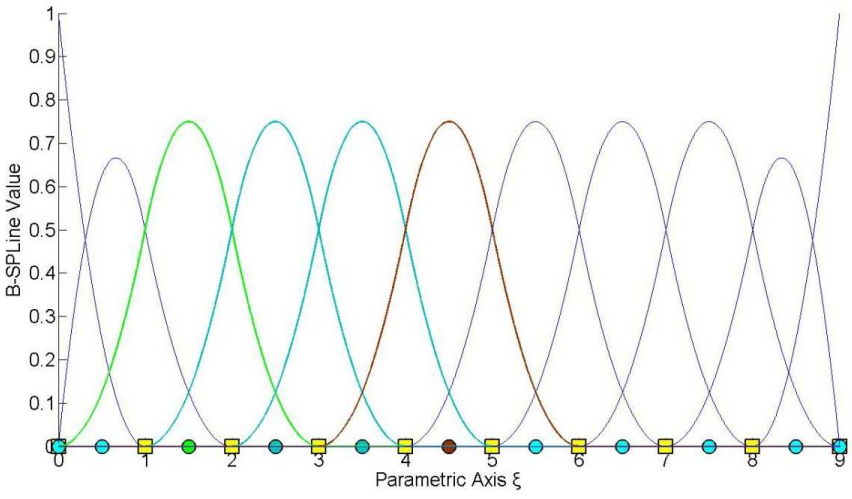
Figure 6.17. Stiffness matrix (1.470 non-zero coefficients) and basis.

In **Figure 6.17**, the degree is equal to p=2, continuity is C^0, and the model has 66 degrees of freedom. The non-zero elements of the matrix are 1.470. The interconnectivity between the elements applies in a small part of the matrix, as it is shown in **Figure 6.17.a**. This occurs because the overlapping between the B-SPLines exists only for $N_{5,2}(\xi)$.

The non-zero coefficients are the result of the shared support between the B-SPLines, which exists for 2p B-SPLines. $N_{5,2}(\xi)$ shares support with the B-SPLines $N_{3,2}(\xi)$, $N_{5,2}(\xi)$, $N_{6,2}(\xi)$ and $N_{7,2}(\xi)$. For these B-SPLines, the coefficients of the matrix are non-zero. This process requires less composition time, because of the limited number of interconnections.



(a) Stiffness matrix.



(b) Quadratic basis.

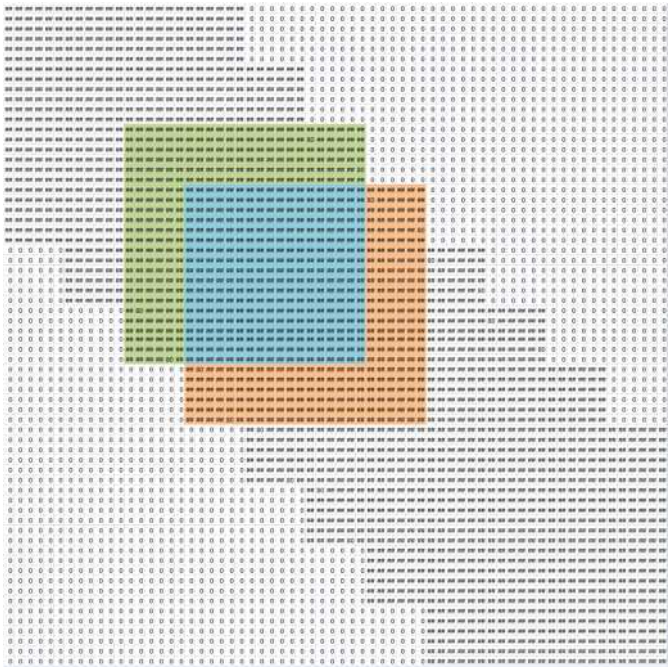
Figure 6.18. Stiffness matrix (1.762 non-zero coefficients) and basis.

In **Figure 6.18**, the model is analyzed for a degree of p=2 and C^1 continuity with the same 66 degrees of freedom. The interconnectivity between the elements has increased and therefore provides a better approximation of the model than the one in **Figure 6.17**. In this case, the formulation of the matrix requires more time, but the overlapping between the B-SPLines leads to better results.

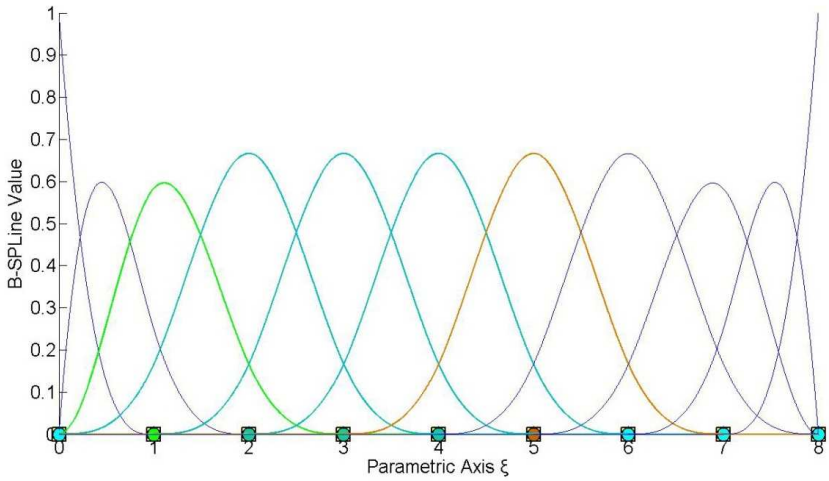
A stiffness matrix coefficient is non-zero, when the corresponding control points have intersected shared support.

A basis function shares support with other 2p ones. Across knot span [2,3) of **Figure 6.15**, basis functions $N_{3,2}(\xi)$, $N_{4,2}(\xi)$ and $N_{5,2}(\xi)$ have non-zero value and across knot span [3,4), $N_{4,2}(\xi)$, $N_{5,2}(\xi)$ and $N_{6,2}(\xi)$. The support of both $N_{4,2}(\xi)$ and $N_{5,2}(\xi)$ contains these two knot spans.

The basis function $N_{4,2}(\xi)$ shares support with other p basis functions left and right, in particular the functions $N_{2,2}(\xi)$, $N_{3,2}(\xi)$, $N_{5,2}(\xi)$ and $N_{6,2}(\xi)$. The same applies for the $N_{5,2}(\xi)$, as it shares support with $N_{3,2}(\xi)$, $N_{4,2}(\xi)$, $N_{6,2}(\xi)$ and $N_{7,2}(\xi)$.



(a) Stiffness matrix.



(b) Cubic basis.

Figure 6.19. Stiffness matrix (2.340 non-zero coefficients) and basis.

In **Figure 6.19**, the model is analyzed for p=3 and C^2-continuity and for the same 66 degrees of freedom.

In knot span [2,3) $N_{3,3}(\xi)$, $N_{4,3}(\xi)$, $N_{5,3}(\xi)$ and $N_{6,3}(\xi)$ are non-zero and in knot span [3,4) the B-SPLines $N_{4,3}(\xi)$, $N_{5,3}(\xi)$, $N_{6,3}(\xi)$ and $N_{7,3}(\xi)$ respectively. In this case, the interconnectivity between the elements expands in almost all the degrees of freedom, because the overlapping between basis functions exists for $N_{4,3}(\xi)$, $N_{5,3}(\xi)$ and $N_{6,3}(\xi)$.

The B-SPLine $N_{4,3}(\xi)$ shares support with 2p B-SPLines as it is previously mentioned, three B-SPLines on the left and three B-SPLines on the right, as it is shown in Figure 6.9. In particular, $N_{4,3}(\xi)$ shares support with $N_{1,3}(\xi)$, $N_{2,3}(\xi)$, $N_{3,3}(\xi)$, $N_{5,3}(\xi)$, $N_{6,3}(\xi)$, $N_{7,3}(\xi)$.

For these B-SPLines, the corresponding coefficients at the stiffness matrix are non-zero. The same applies for $N_{5,3}(\xi)$ and $N_{6,3}(\xi)$. Therefore, the number of the non-zero elements has increased even more, thus the approximation in this solution is more close to the real problem.

As the continuity increases, the interconnectivity of the elements affects more degrees of freedom. This leads to a better approximation of the physical problem, but also the creation of the stiffness matrix becomes more time-consuming. For the same polynomial order, more non-zero elements are created at the stiffness matrix.

This is a result of the shared support of the B-SPLine basis functions that exists for 2p B-SPLines, which leads to overlapping between the B-SPLines and interconnectivity of the degrees of freedom in the stiffness matrix. Therefore, as Continuity increases, the number of non-zero elements increases as well.

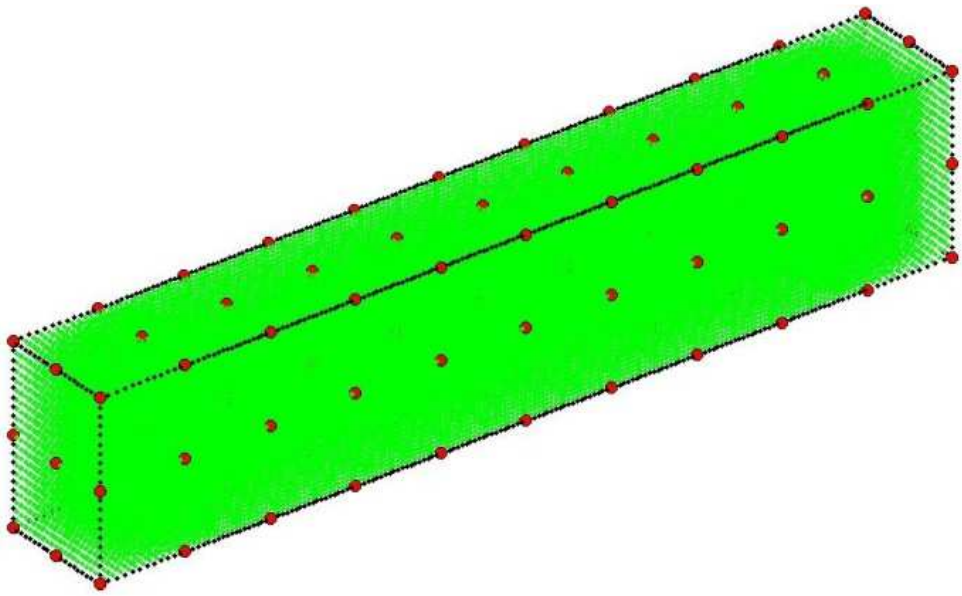
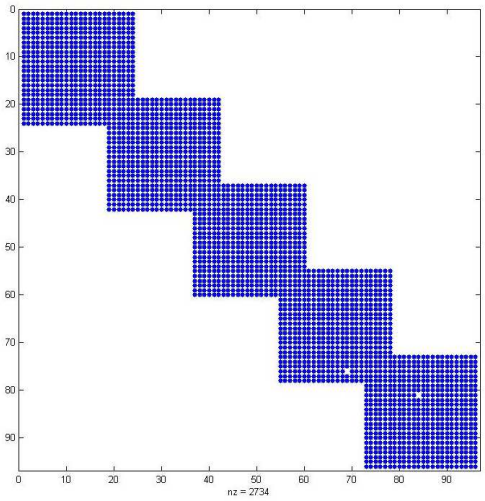
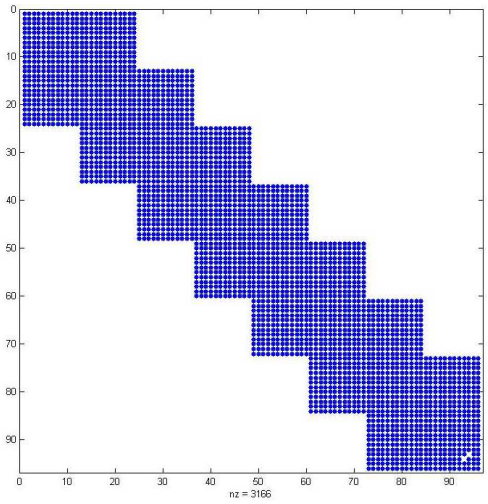


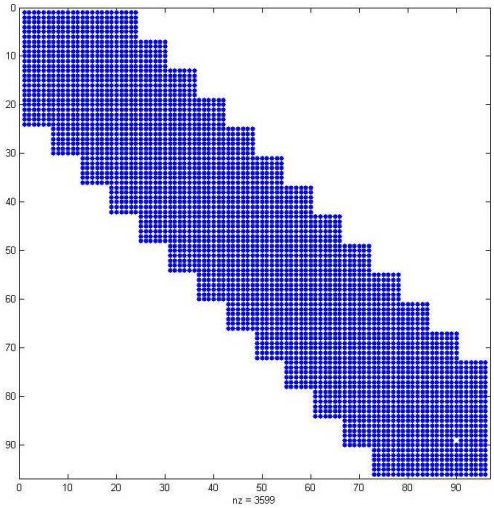
Figure 6.20. 3D NURBS model.



(a) C^0-continuity.



(b) C^1-continuity.



(c) C^2-continuity.

Figure 6.21. Stiffness matrix.
Cubic basis and 99 degrees of freedom.

7 Loads, Constraints & Results

7.1 Analysis

7.1.1 External load

After the stiffness matrix assembly, the external load vector has to be calculated. Concentrated loads can be assigned directly to the degrees of freedom. Distributed loads f(ξ,η,ζ) have to be transformed into equivalent concentrated loads by the following integration:

$$\underset{(N\times 1)}{\{F\}} = \int_{\xi_0}^{\xi_{n+p+1}} \int_{\eta_0}^{\eta_{m+q+1}} \int_{\zeta_0}^{\zeta_{l+r+1}} \underset{(N\times 1)}{\{R(\xi,\eta,\zeta)\}} \cdot \underset{(1\times 1)}{f(\xi,\eta,\zeta)} \cdot \det[J]\, d\zeta d\eta d\xi$$

More specifically, for each case:

1D:

$$\underset{(N\times 1)}{\{F\}} = \int_{\xi_0}^{\xi_{n+p+1}} \underset{(N\times 1)}{\{R(\xi)\}} \cdot \underset{(1\times 1)}{f(\xi)} \cdot \det[J] d\xi$$

2D:

$$\underset{(N\times 1)}{\{F\}} = \int_{\xi_0}^{\xi_{n+p+1}} \int_{\eta_0}^{\eta_{m+q+1}} \underset{(N\times 1)}{\{R(\xi,\eta)\}} \cdot \underset{(1\times 1)}{f(\xi,\eta)} \cdot \det[J] d\eta d\xi$$

3D:

$$\underset{(N\times 1)}{\{F\}} = \int_{\xi_0}^{\xi_{n+p+1}} \int_{\eta_0}^{\eta_{m+q+1}} \int_{\zeta_0}^{\zeta_{l+r+1}} \underset{(N\times 1)}{\{R(\xi,\eta,\zeta)\}} \cdot \underset{(1\times 1)}{f(\xi,\eta,\zeta)} \cdot \det[J] d\zeta d\eta d\xi$$

7.1.2 Boundary conditions

Certain degrees of freedom are fixed, in that their displacements are zero. These are called stationary and the corresponding rows and columns are deleted from the stiffness matrix and the load vector. Thus, both the stiffness matrix and the load vector end up with only the free degrees of dreedom, $[K_{ff}]$ and $\{F_f\}$ respectively. The solution of this matrix system is calculated as follows:

$$\{F_f\} = [K_{ff}] \cdot \{D_f\} \Rightarrow \{D_f\} = [K_{ff}]^{-1} \cdot \{F_f\}$$

7.2 Displacement, Strain & Stress Field

7.2.1 Displacement field

After the solution of the equation, control point Displacements are obtained. Unlike classical FEM, control points are usually placed outside the area of the model. In general, displacements of the model differ from the displacements of the corresponding control points. Conclusively, analysis results are considered “Pseudo-Displacements” and play an auxiliary role in calculating the physical model ones. As mentioned before, the distribution of the displacement field is achieved via Shape Functions [2]:

1D:

$$d(\xi) = \sum_{i=1}^{N} \left\{ R_i(\xi) \cdot D_i \right\} = \underset{(1\times N)}{\left\{ R_i(\xi) \right\}^T} \cdot \underset{(N\times 1)}{\left\{ D \right\}}$$

2D:

$$d(\xi,\eta) = \sum_{i=1}^{N} \left\{ R_i(\xi,\eta) \cdot D_i \right\} = \underset{(1\times N)}{\left\{ R_i(\xi,\eta) \right\}^T} \cdot \underset{(N\times 1)}{\left\{ D \right\}}$$

3D:

$$d(\xi,\eta,\zeta) = \sum_{i=1}^{N} \left\{ R_i(\xi,\eta,\zeta) \cdot D_i \right\} = \underset{(1\times N)}{\left\{ R_i(\xi,\eta,\zeta) \right\}^T} \cdot \underset{(N\times 1)}{\left\{ D \right\}}$$

where:

- R_i is the shape function i
- D_i is the displacement of the corresponding control point
- N is the total number of control points

If a control point c is interpolatory to the curve at (ξ_c, η_c, ζ_c), it follows that:

$$d(\xi_c,\eta_c,\zeta_c) = \sum_{i=1}^{N} \left\{ R_i(\xi_c,\eta_c,\zeta_c) \cdot D_i \right\} = 1 \cdot D_c = D_c \,.$$

7.2.2 Strain & stress field

The strain vector can be evaluated at any point in the field with the help of control point displacements and the deformation matrix [B].

1D

$$\underset{(1\times1)}{\{\varepsilon(\xi)\}} = \underset{(1\times N)}{[B(\xi)]} \cdot \underset{(N\times1)}{\{D\}}$$

2D

$$\underset{(3\times1)}{\{\varepsilon(\xi,\eta)\}} = \underset{(3\times2N)}{[B(\xi,\eta)]} \cdot \underset{(2N\times1)}{\{D\}}$$

3D

$$\underset{(6\times1)}{\{\varepsilon(\xi,\eta,\zeta)\}} = \underset{(6\times3N)}{[B(\xi,\eta,\zeta)]} \cdot \underset{(3N\times1)}{\{D\}}$$

Applying Hooke's Constitutive Law leads to:

1D

$$\underset{(1\times1)}{\{\sigma(\xi)\}} = \underset{(1\times1)}{[E]} \cdot \underset{(1\times1)}{\{\varepsilon(\xi)\}} = \underset{(1\times1)}{[E]} \cdot \underset{(1\times N)}{[B(\xi)]} \cdot \underset{(N\times1)}{\{D\}}$$

2D

$$\underset{(3\times1)}{\{\sigma(\xi,\eta)\}} = \underset{(3\times3)}{[E]} \cdot \underset{(3\times1)}{\{\varepsilon(\xi,\eta)\}} = \underset{(3\times3)}{[E]} \cdot \underset{(3\times2N)}{[B(\xi,\eta)]} \cdot \underset{(2N\times1)}{\{D\}}$$

3D

$$\underset{(6\times1)}{\{\sigma(\xi,\eta,\zeta)\}} = \underset{(6\times6)}{[E]} \cdot \underset{(6\times1)}{\{\varepsilon(\xi,\eta,\zeta)\}} = \underset{(6\times6)}{[E]} \cdot \underset{(6\times3N)}{[B(\xi,\eta,\zeta)]} \cdot \underset{(3N\times1)}{\{D\}}$$

Note that stress and strain vectors are evaluated via the derivatives of the shape functions. This means that their distribution is going to be one order less than the displacement distribution. This is why stress and strain continuity cannot be achieved in FEM models, where shape functions are always C^{-1} continuous. This problem is solved when the derivatives of the shape functions are also continuous, which means using shape functions with C^1 continuity or higher.

8 Applications

8.1 Annulus 2D

The annulus is subjected to plane stress. The third direction (thickness) is significantly smaller than the other two. Only a quarter of the annulus has to be analyzed, as there is symmetry in both axes. The initial mesh has been designed with the minimum required number of control points, shown in **Figure 8.1**.

Quadratic basis functions are selected for both the parametric axes ξ, η. The NURBS model is discretized into non-zero knot spans. A quadratic rule of p+1=3 Gauss points per 1D knot span is chosen. Thus, each finite element contains 3x3=9 Gauss points.

There are n=3 control points (axis ξ) and m=3 control points (axis η), means 9 control points and 18 degrees of freedom.

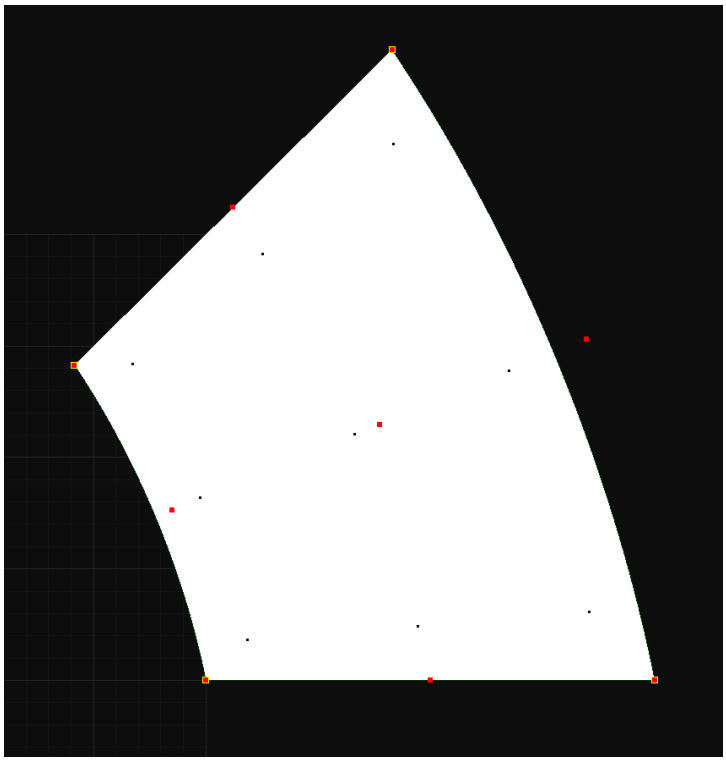
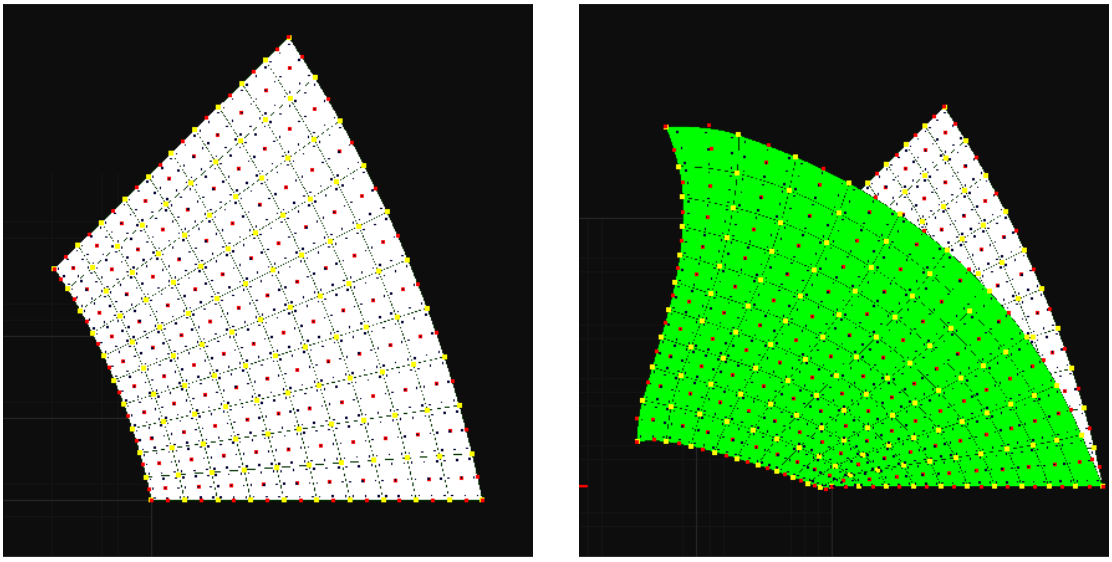


Figure 8.1. Annulus. Coarse mesh.
Ξ={0,0,0,1,1,1}, H={0,0,0,1,1,1}

In order to increase the accuracy of the analysis results, greater discretization is needed. For this reason, h-refinement is applied, enriching the basis by adding basis functions of the same order. The fine mesh is depicted in **Figure 8.2**.



(a) Undeformed shape **(b)** Deformed shape

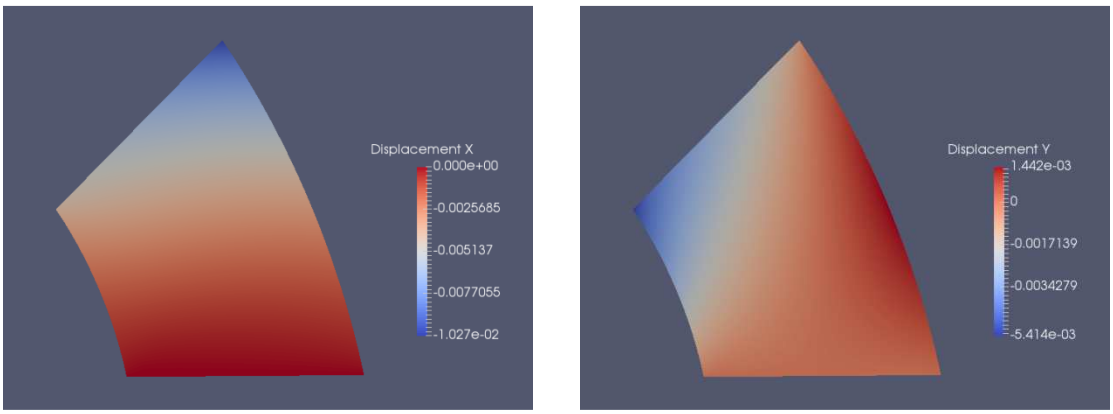
Figure 8.2. The quarter of an annulus.

$$\Xi = \{0 \quad 0 \quad 0 \quad 0.1 \quad 0.2 \quad 0.3 \quad 0.4 \quad 0.5 \quad 0.6 \quad 0.7 \quad 0.8 \quad 0.9 \quad 1 \quad 1 \quad 1\}$$

$$\mathrm{H} = \{0 \quad 0 \quad 0 \quad 0.1 \quad 0.2 \quad 0.3 \quad 0.4 \quad 0.5 \quad 0.6 \quad 0.7 \quad 0.8 \quad 0.9 \quad 1 \quad 1 \quad 1\}$$

The degrees of freedom at the bottom edger are fixed. Loads are applied on the opposite edge with radial and tangential direction to knot lines.

In **Figure 8.3**, contour for the displacement field are presented for the undeformed shape.

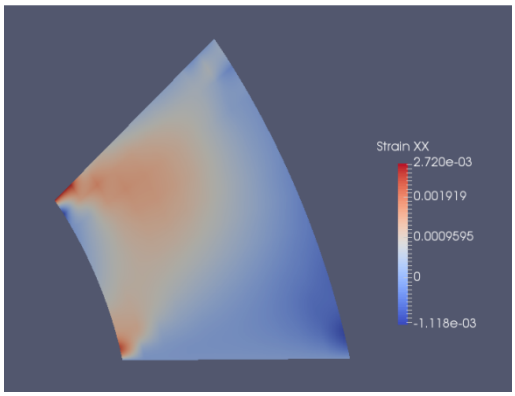


(a) Displacement X **(b) Displacement Y**

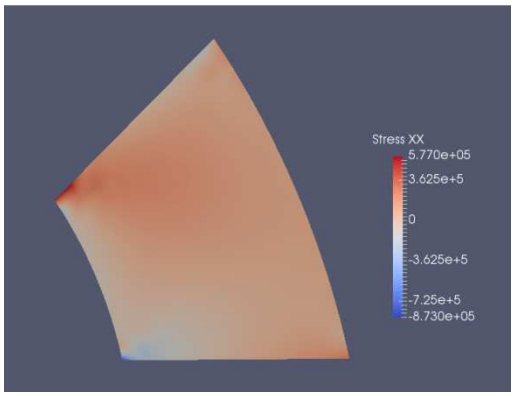
Figure 8.3. Contour. Displacement field.

In the case of contour (displacement X), the blue area declares that the top side of the annulus has negative (left) displacement, while the bottom one has zero values. For displacement Y, negative displacement is apparent, just as expected for the chosen loading case.

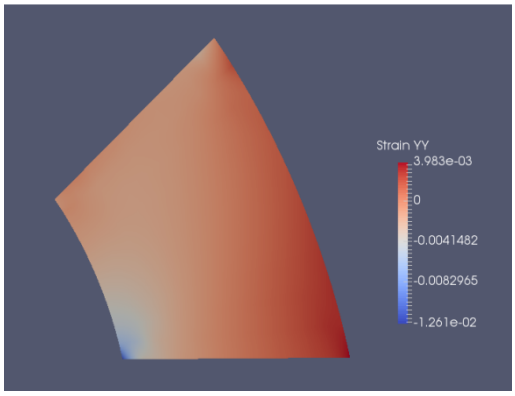
In **Figure 8.4**, contours for strain X, Y, XY and the corresponding stresses are presented. As expected strain and stress contours have similar shape, since in linear elasticity stresses are calculated by multiplying the elasticity matrix with the strains.



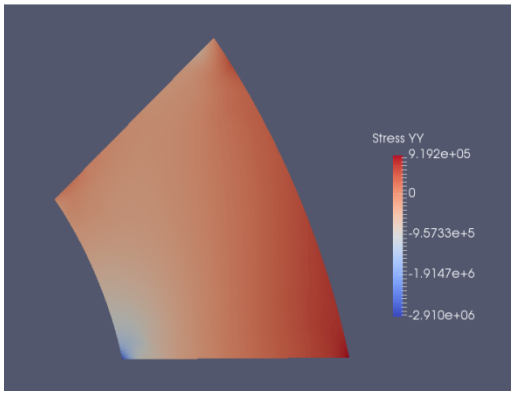
(a) Strain XX



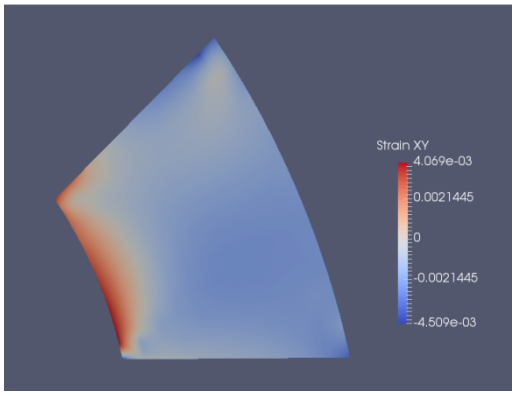
(d) Stress XX



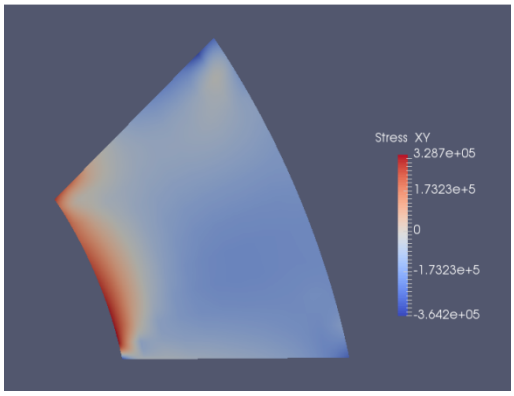
(a) Strain YY



(d) Stress YY



(a) Strain XY



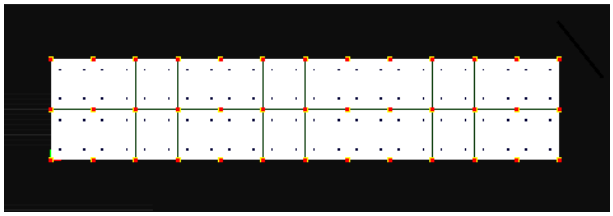
(d) Stress XY

Figure 8.4. Contour. Strain & stress field.

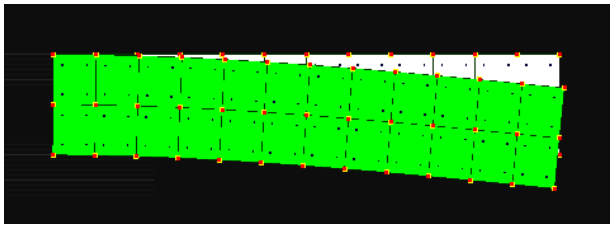
Since the loads in Y direction are opposite to the boundary conditions, they stretch the annulus, and as a result, tensile stresses are noted through this direction. The distribution of the shear stresses can be foreseeable by just observing the shape of deformation of each knot span.

8.2 Cantilever 2D

A cantilever beam with a rectangular cross-section is studied. The beam is fixed at one end and free at the other. The load force is chosen to be 3x1000 kN. The beam is made of steel. The beam is subjected to a load at its free edge. The cantilever s analyzed as a plane stress problem. The basis is linear for both axes ξ, η. There are n=13 control points (axis ξ) and m=3 control points (axis η), thus there is a total amount of 39 control points and 78 degrees of freedom.



(a)

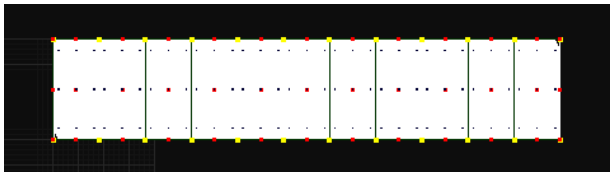


(b)

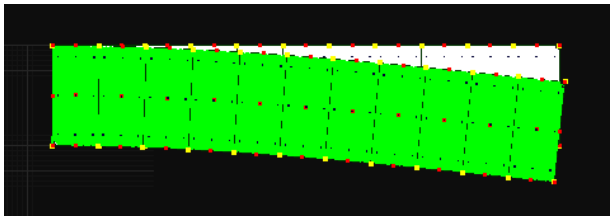
Figure 8.5. Cantilever beam. Initial mesh.

$$\Xi = \{0 \quad 0 \quad 0.083 \quad 0.166 \quad 0.25 \quad 0.333 \quad 0.416 \quad 0.5 \quad 0.583 \quad 0.666 \quad 0.75 \quad 0.833 \quad 0.916 \quad 1 \quad 1\}$$

$$H = \{0 \quad 0 \quad 0.5 \quad 1 \quad 1\}$$



(a)

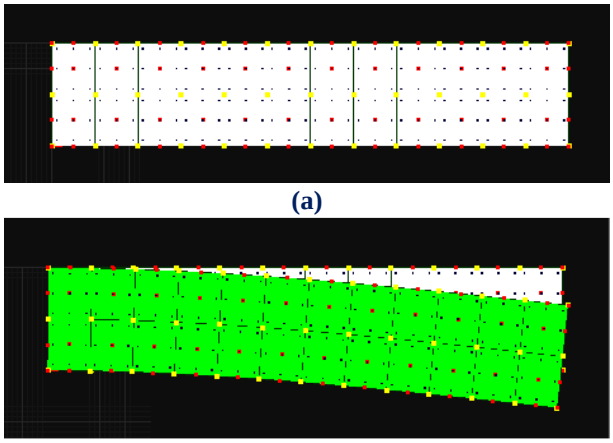


(b)

Figure 8.6. Cantilever beam. Fine mesh.

$$\Xi = \{0 \quad 0 \quad 0 \quad 0.09 \quad 0.181 \quad 0.272 \quad 0.363 \quad 0.454 \quad 0.545 \quad 0.636 \quad 0.727 \quad 0.818 \quad 0.909 \quad 1 \quad 1 \quad 1\}$$

$$H = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$$



(a)

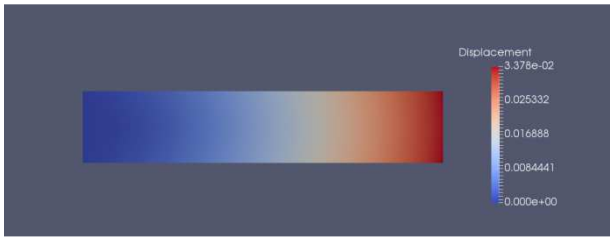
(b)

Figure 8.7. Cantilever beam. New fine mesh.

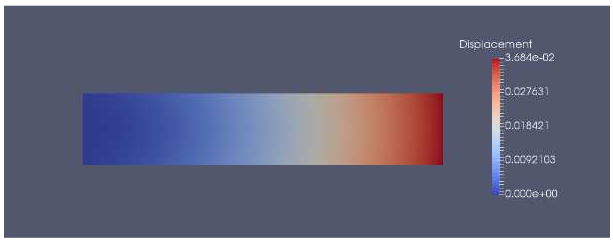
$$\Xi = \{0 \quad 0 \quad 0 \quad 0.083 \quad 0.166 \quad 0.25 \quad 0.333 \quad 0.416 \quad 0.5 \quad 0.583 \quad 0.666 \quad 0.75 \quad 0.833 \quad 0.916 \quad 1 \quad 1 \quad 1\}$$

$$H = \{0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 1 \quad 1\}$$

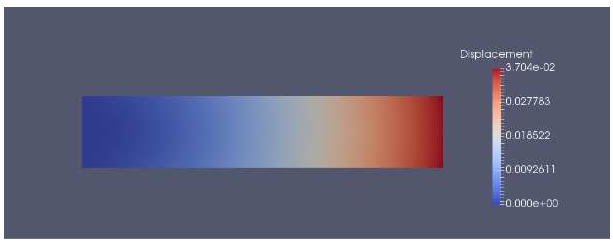
This mesh has been created by implementing degree elevation (p-refinement). As the continuity per knot is keeping the same, the total number of knot spans remains intact. In **Figure 8.8**, contour for displacement field is presented.



(a) 13x3 control points, p=1

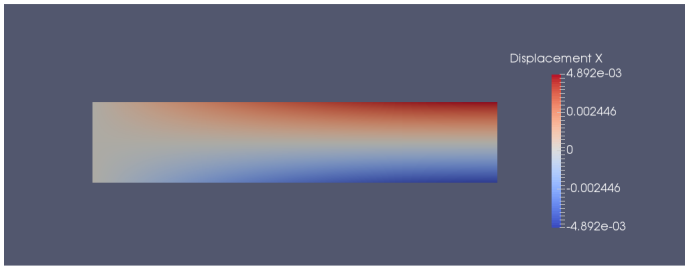


(b) 13x3 control points, p=2

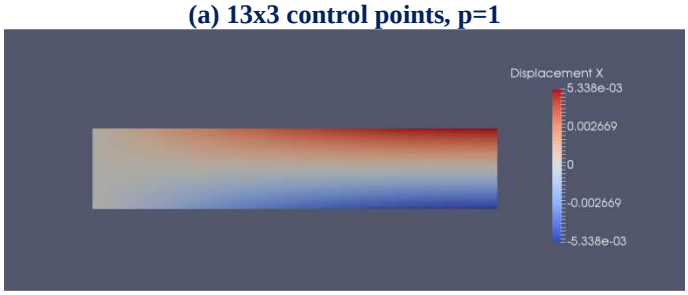


(c) 14x4 control points, p=2

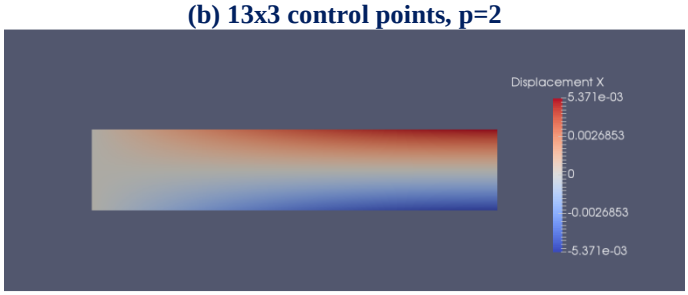
Figure 8.8. Contour. Displacement field.



(a) 13x3 control points, p=1

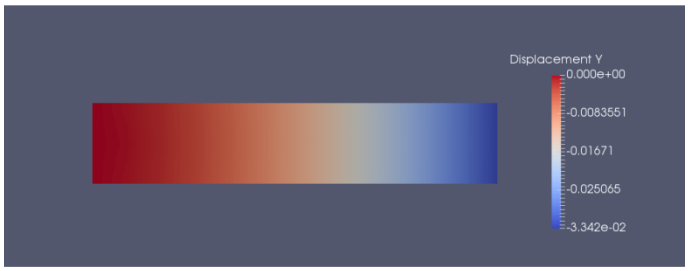


(b) 13x3 control points, p=2

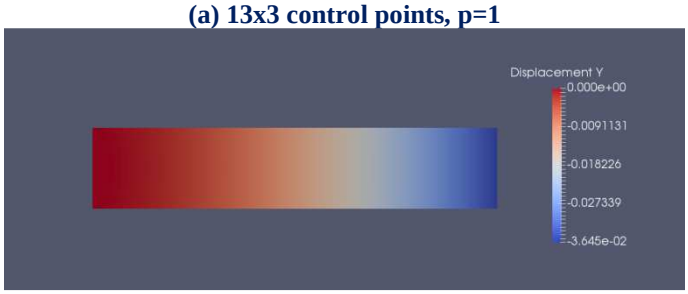


(c) 14x4 control points, p=2

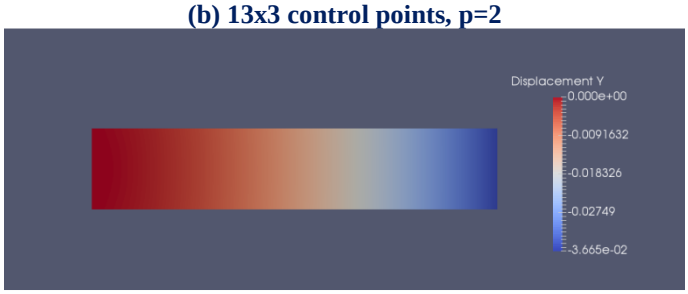
Figure 8.9. Contour. Displacement X.



(a) 13x3 control points, p=1



(b) 13x3 control points, p=2



(c) 14x4 control points, p=2

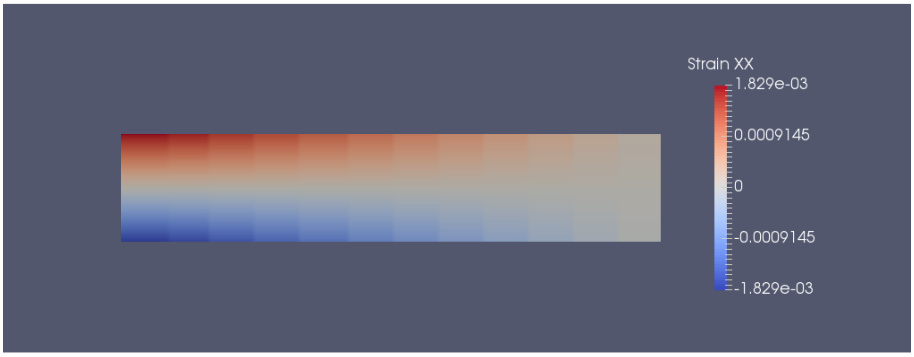
Figure 8.10. Contour. Displacement Y.

In **Figure 8.11, 8.12, 8.13**, contours for strain X, Y and XY are presented. The components of the strain tensor are calculated by the following equation:

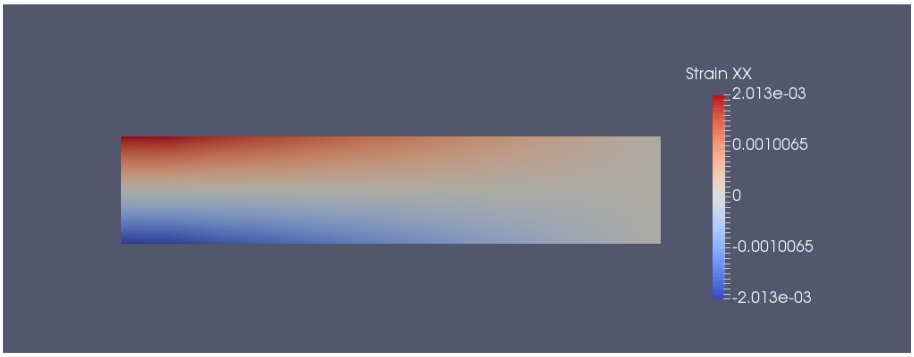
$$\underset{(3\times1)}{\{\varepsilon(\xi,\eta)\}} = \underset{(3\times2N)}{[B(\xi,\eta)]} \cdot \underset{(2N\times1)}{\{D\}}$$

where $\underset{(3\times2N)}{[B(\xi,\eta)]}$ is the deformation matrix and $\underset{(2N\times1)}{\{D\}}$ the pseudo-displacements.

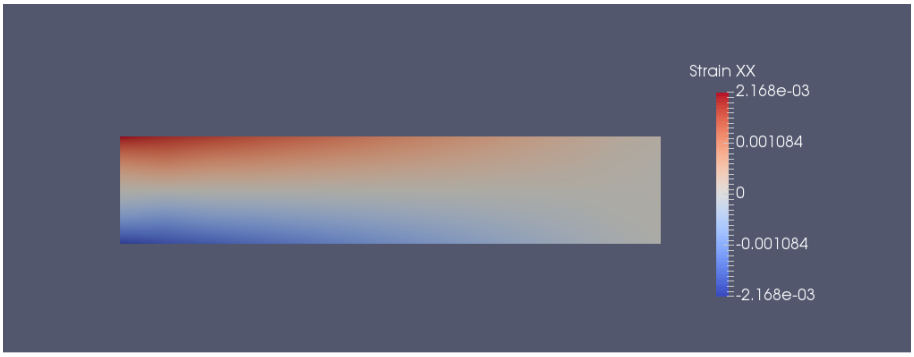
Discontinuities are observed across the boundaries of the knot spans. The finer parametrization leads to smoother contours.



(a) 13x3 control points, p=1



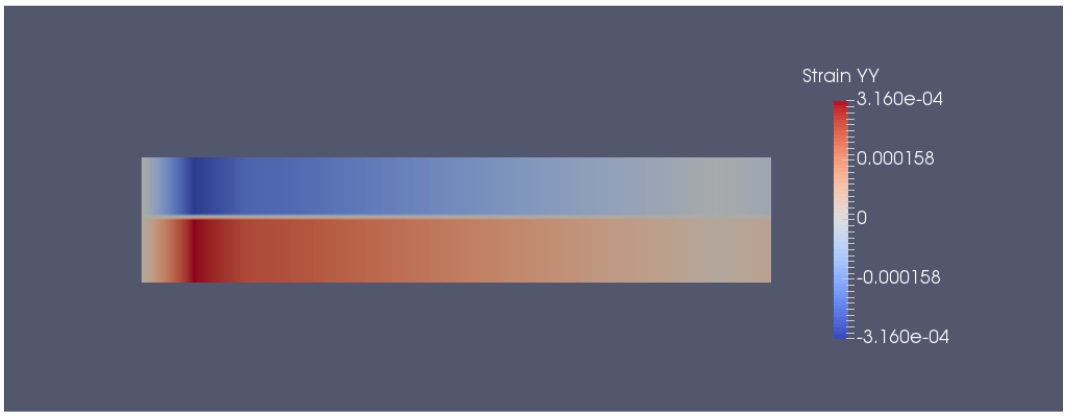
(b) 13x3 control points, p=2



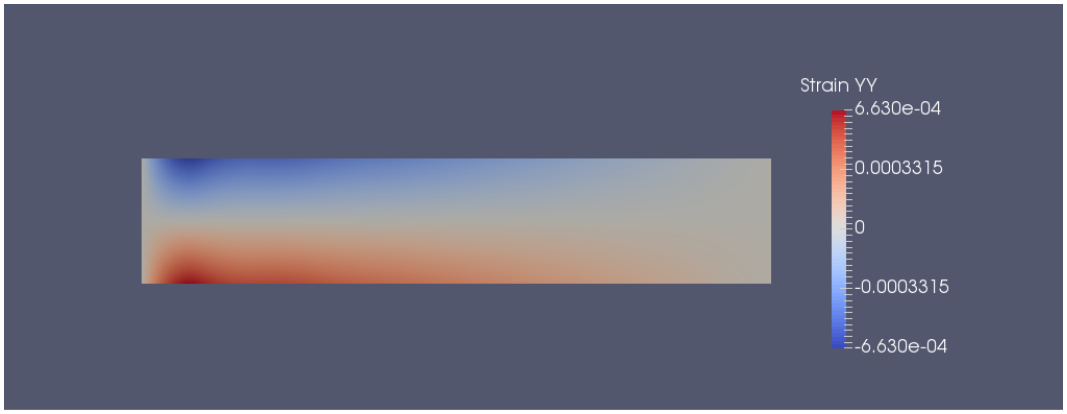
(c) 14x4 control points, p=2

Figure 8.11. Contour. Strain XX.

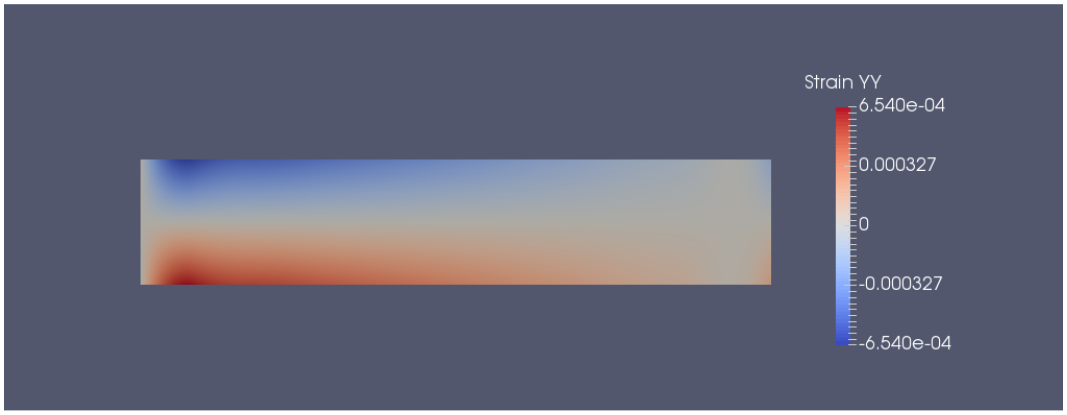
The linear basis leads to the reduced accuracy. The maximum value of strain XX is about 3,5 times bigger than the maximum one of strain YY. Due to Poisson's phenomenon, maximum strain YY appears in these points, where maximum strain XX is located. However, at the left edge, where displacements X and Y are fixed, strain YY has zero value. The same happens at the right free edge.



(a) 13x3 control points, p=1



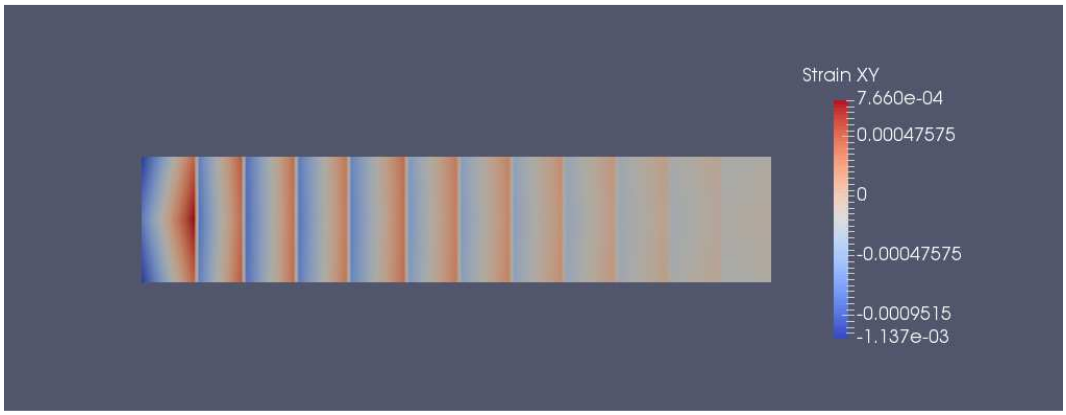
(b) 13x3 control points, p=2



(c) 14x4 control points, p=2

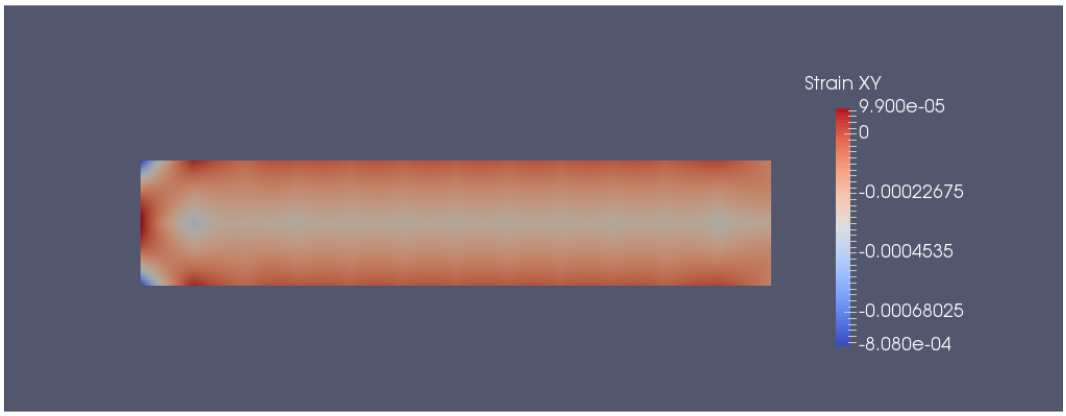
Figure 8.12. Contour. Strain YY.

The distribution of shear strains is different for the two cases of the quadratic basis. This indicates the inability of coarse meshes to simulate problems of bending. Thus, for the exact evaluation, the thickening of the isogeometric mesh seems to have primitive importance.



(a) 13x3 control points, p=1

(b) 13x3 control points, p=2



(c) 14x4 control points, p=2

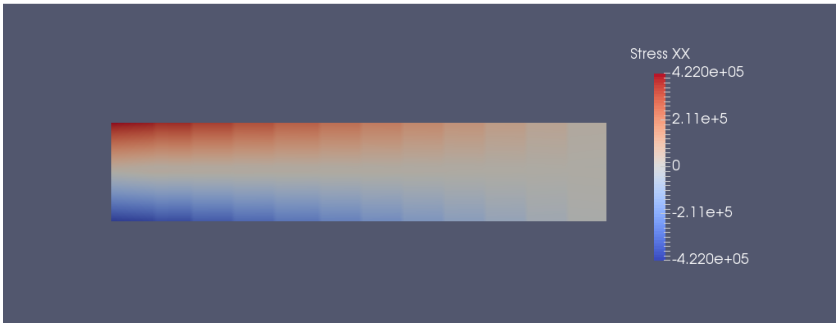
Figure 8.13. Contour. Strain XY.

Figure 8.14, 8.15 and **8.16** depict contours for strain X, Y and XY. More specifically, the components of the stress tensor (plane elasticity) are calculated by the equation.

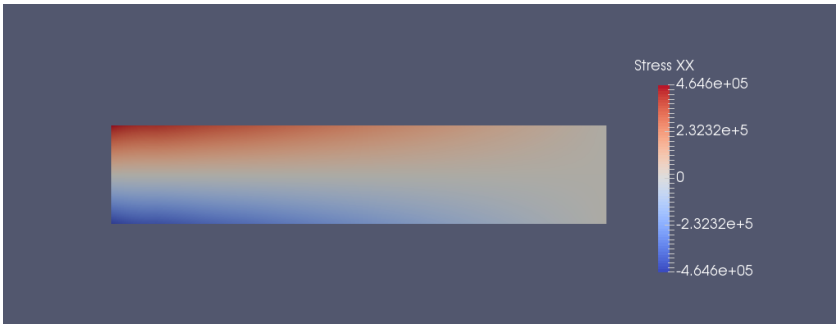
$$\underset{(3\times1)}{\{\sigma(\xi,\eta)\}} = \underset{(3\times3)}{[E]} \cdot \underset{(3\times1)}{\{\varepsilon(\xi,\eta)\}} = \underset{(3\times3)}{[E]} \cdot \underset{(3\times2N)}{[B(\xi,\eta)]} \cdot \underset{(2N\times1)}{\{D\}}$$

where

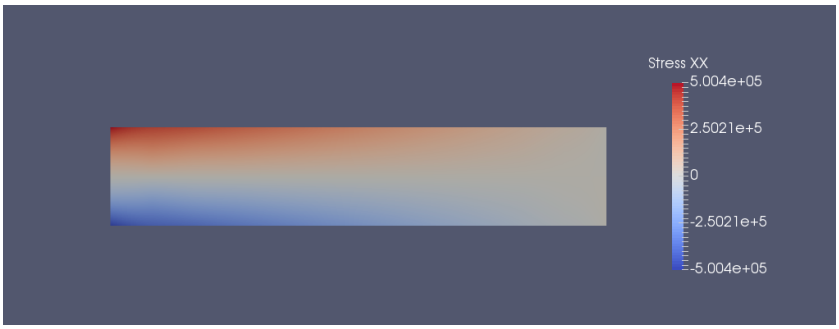
- $\underset{(3\times3)}{[E]}$ is the elasticity matrix
- $\underset{(3\times1)}{\{\varepsilon(\xi,\eta)\}}$ is the strain vector



(a) 13x3 control points, p=1



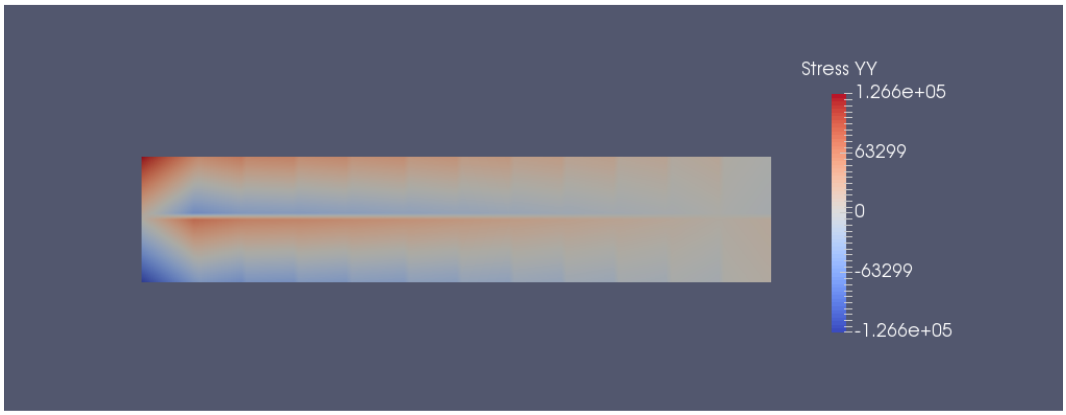
(b) 13x3 control points, p=2



(c) 14x4 control points, p=2

Figure 8.14. Contour. Stress XX.

Strain and stress field depends on the derivatives. That's why strain and stress field are less accurate than the displacement one. Stress YY can be neglected for the major part of the cantilever. As noted before, at the left fixed edge, stress YY has its maximum value.



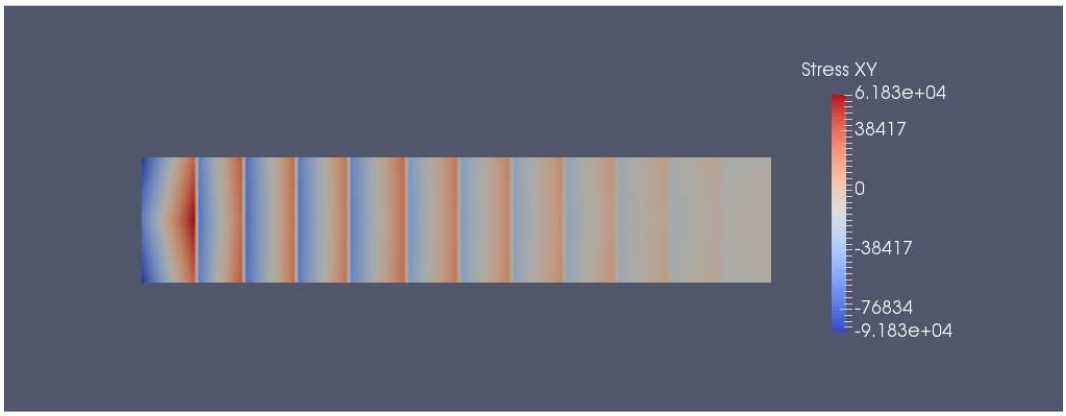
(a) 13x3 control points, p=1

(b) 13x3 control points, p=2

(c) 14x4 control points, p=2

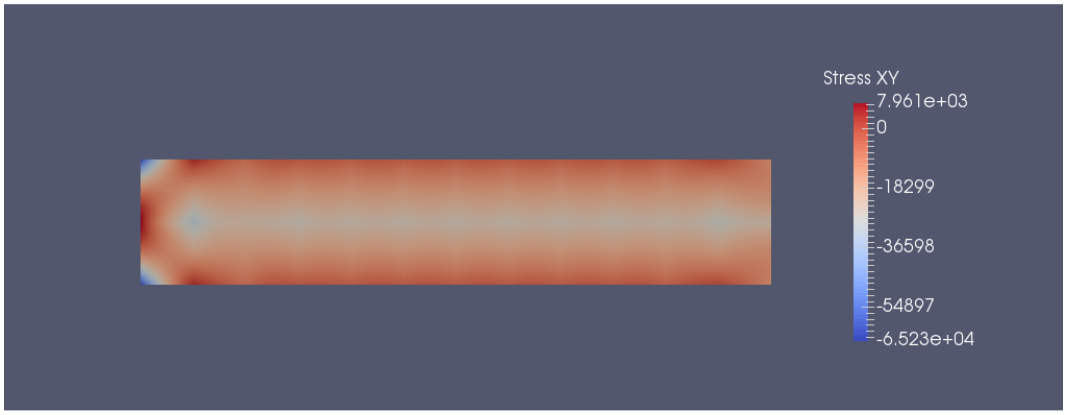
Figure 8.15. Contour. Stress YY.

The linear basis is unsuitable for pure bending problems. This results in the overestimation of the shear stress field, that seems to be zero at the middle of the elements. This state changes in the case of more accurate elements of higher degree. The distribution of the shear stresses becomes smoother, especially in the case of the third representation.



(a) 13x3 control points, p=1

(b) 13x3 control points, p=2



(c) 14x4 control points, p=2

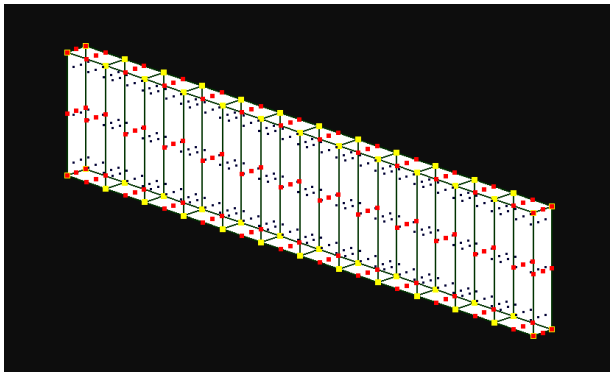
Figure 8.16. Contour. Stress field. XY.

8.3 Cantilever 3D

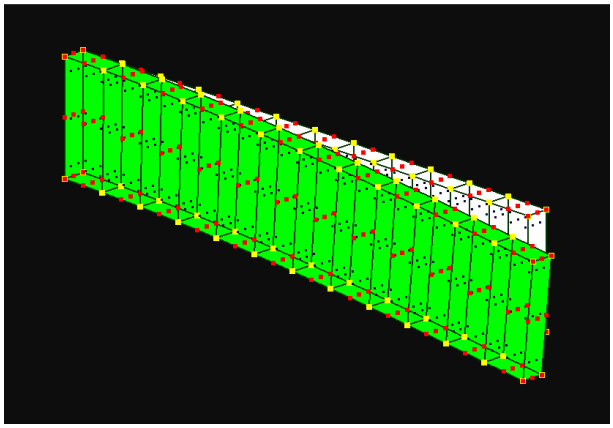
The first 3D application is the cantilever. The initial mesh is similar to the 2D cantilever with the difference that one knot span in the parametric direction ζ is added. The boundary and loading conditions are peculiar to the 2D cantilever. Specifically, the beam is fixed at the left edge and free at its right. The load force (free right edge) is shared among the nine control points, means 333,33 kN per control point.

The basis is quadratic (axes ξ, η, ζ). There are n=14 control points (axis ξ), m=3 (axis η) and l=3 (axis ζ), thus a total amount of 126 control points and 378 degrees of freedom.

Gauss points' coordinates are evaluated for every knot span. Specifically, p+1=3 Gauss points are required per each 1D knot span. Thus, the number of Gauss points of every finite is equal to 3x3x3=27 and the total number for the whole cantilever is equal to 27x12=324.



(a) Undeformed configuration.

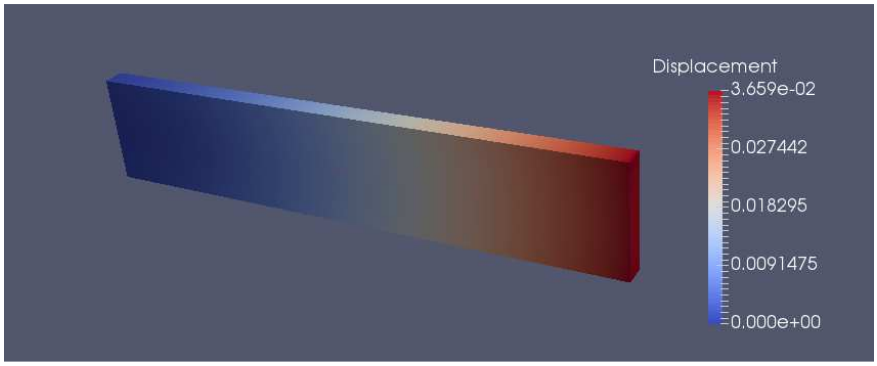


(b) Deformed configuration.

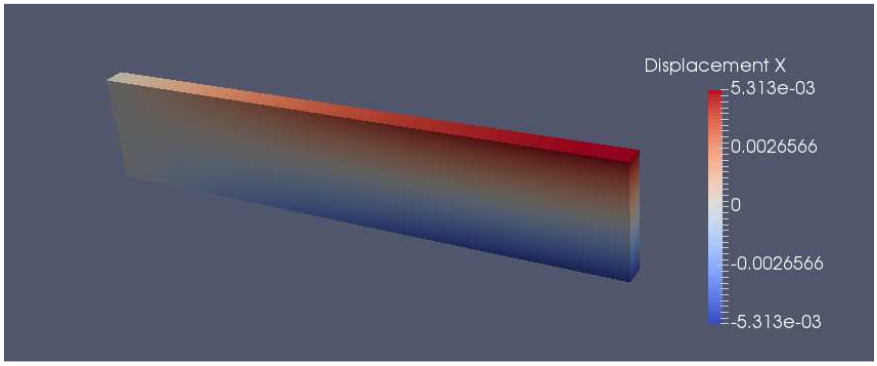
Figure 8.17. Cantilever 3D. Mesh 14x3x3.

$$\Xi = \{0 \quad 0 \quad 0 \quad 0.083 \quad 0.166 \quad 0.25 \quad 0.333 \quad 0.416 \quad 0.5 \quad 0.583 \quad 0.666 \quad 0.75 \quad 0.833 \quad 0.916 \quad 1 \quad 1 \quad 1\}$$

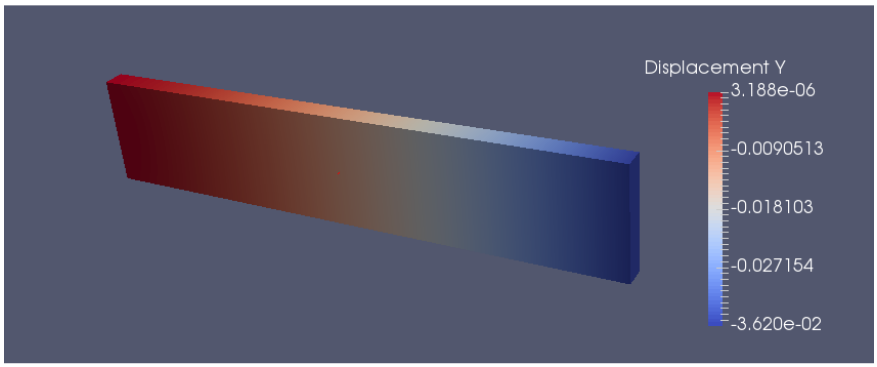
$$H = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}, \; Z = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$$



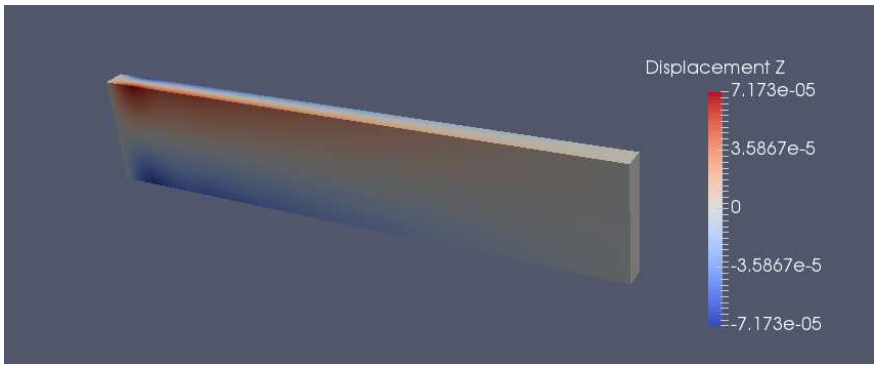
(a) Displacement.



(b) Displacement X.

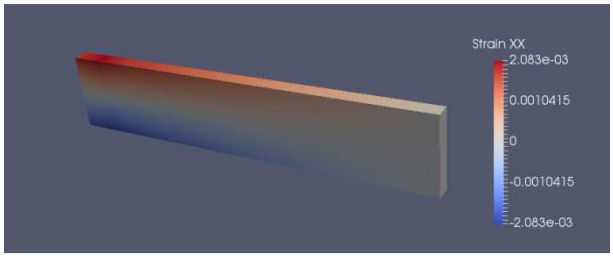


(c) Displacement Y.

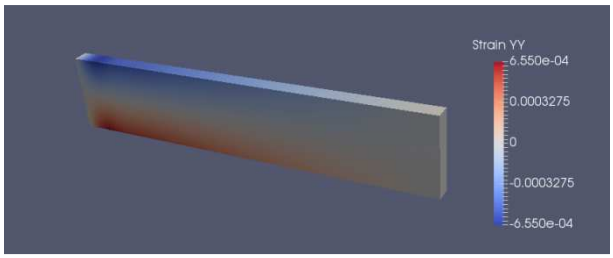


(d) Displacement Z.

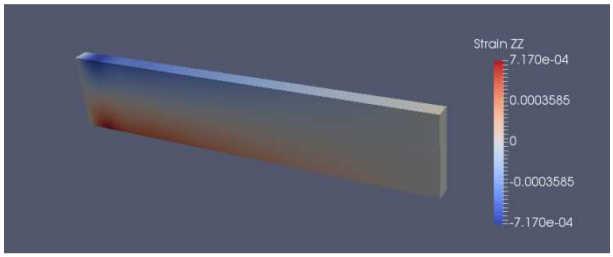
Figure 8.18. Contour. Displacement field.



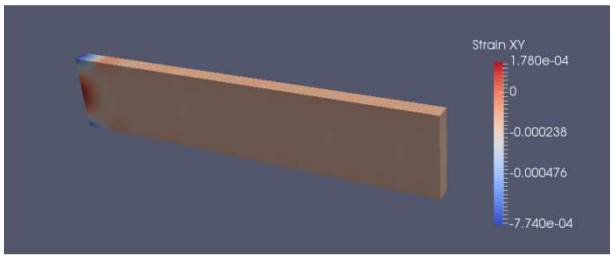
(a) XX.



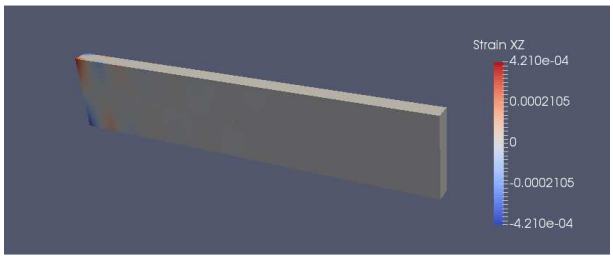
(b) YY.



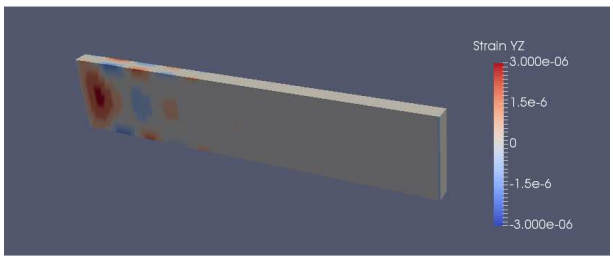
(c) ZZ.



(d) XY.

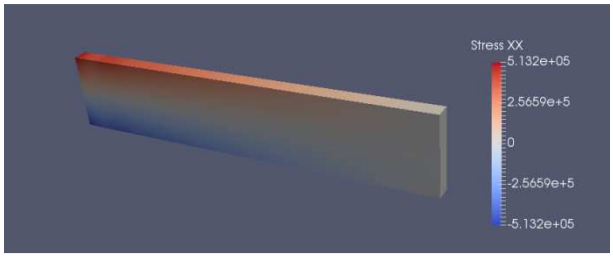


(e) YZ.

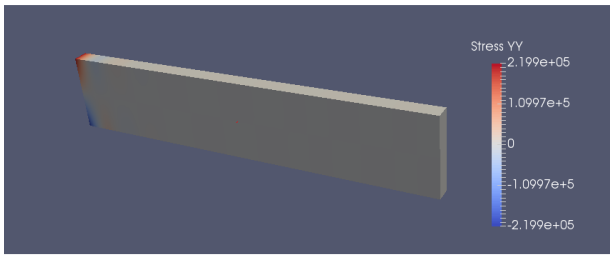


(f) ZX.

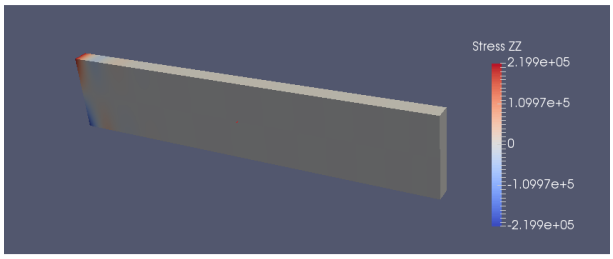
Figure 8.19. Contour. Strain field.



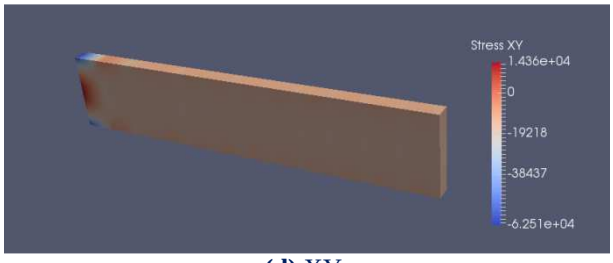
(a) XX.



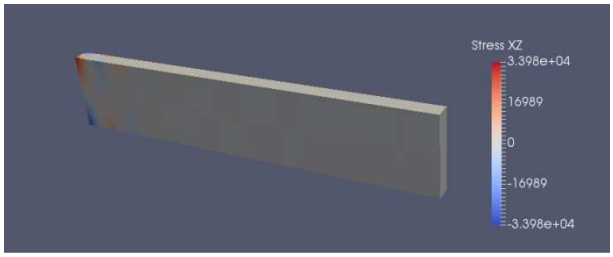
(b) YY.



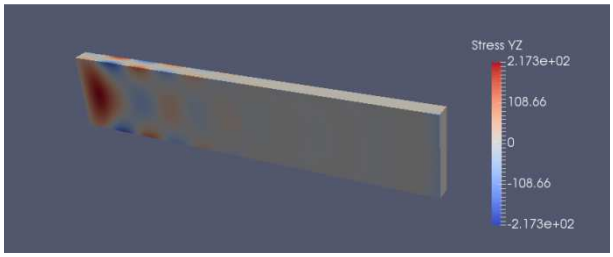
(c) ZZ.



(d) XY.



(e) YZ.



(f) ZX.

Figure 8.20. Contour. Stress field.

8.4 Bent Pipe

The bent pipe consists of a straight and a circular section, joined together. Parametric direction ζ represents the longitudinal direction. Parametric direction η represents the pipe thickness and ξ the creation of the circular shape.

The bent pipe is fixed at the top edge. The annulus of the pipe was created by using two cycles, one internal and one external. Note that each cycle is created by nine control points and contains four knot spans with a C^0-continuity at their boundary. These elements constitute a patch.

Initially, the bent pipe has been designed by using the minimum possible number of control points. Specifically, the basis is quadratic (axes ξ, ζ) and linear (axis η).

There are n=9 control points (axis ξ), m=3 (axis η) and l=5 (axis ζ), means a total amount of 90 control points and 270 degrees of freedom.

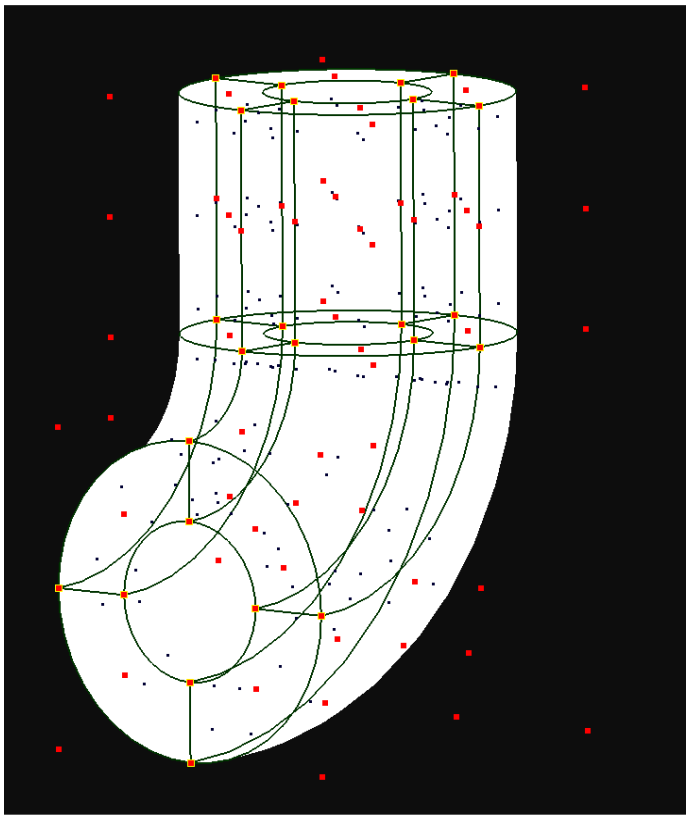
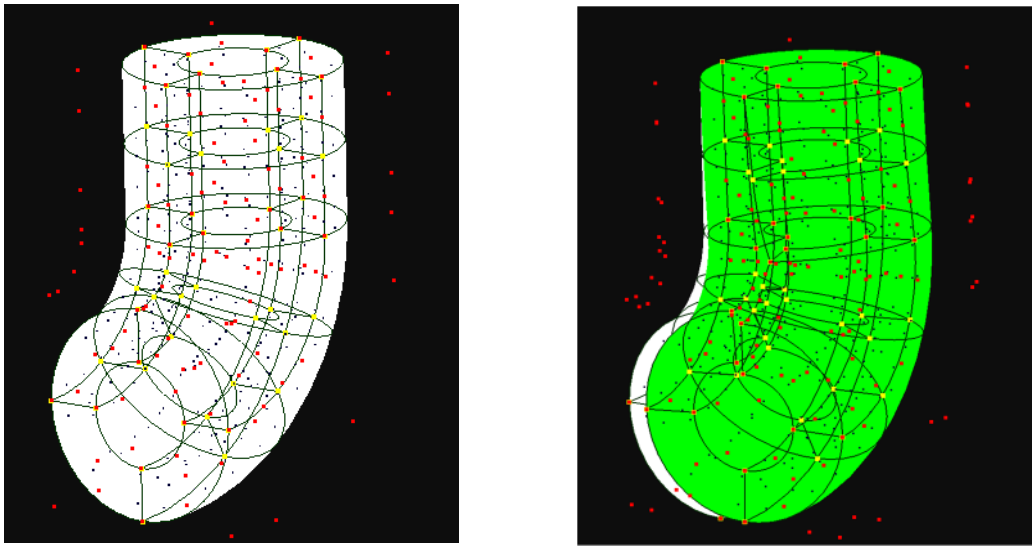


Figure 8.21. Bent pipe. Coarse mesh.
Ξ={0,0,0,1,1,2,2,3,3,4,4,4}, H={0,0,1,1}

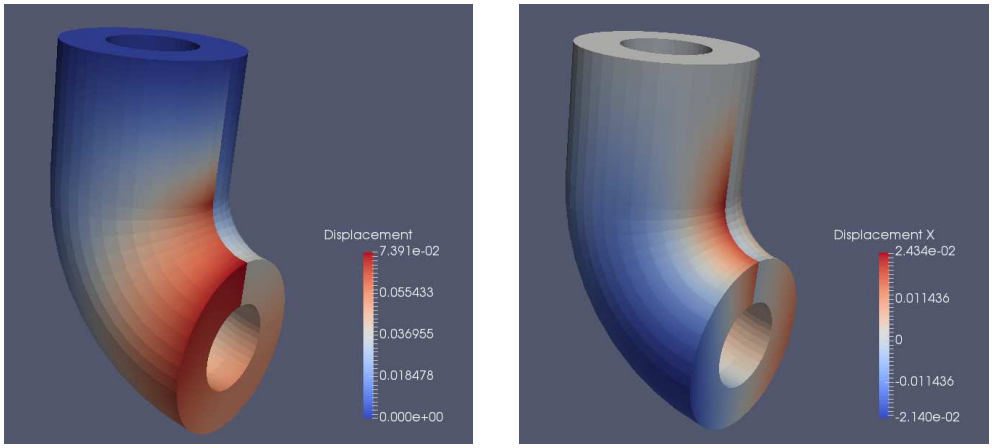
The coarse mesh cannot ensure accurate results. These can be achieved by using a finer mesh. For this reason, h-refinement (knot insertion) is applied. Thus, the degree of the basis functions remains the same. The number of control points is increased to 8 on parametric axis ζ.



(a) Undeformed **(b) Deformed**

Figure 8.22. Bent pipe. Fine mesh.
Ξ={0,0,0,1,1,2,2,3,3,4,4,4}, H={0,0,1,1}, Z={0,0,0,0.5,1,1,1.33,1.66,2,2,2}

Figure 8.23 depicts contour for displacement field.



(a) Displacement **(b)** Displacement X

(c) Displacement Y **(d)** Displacement Z

Figure 8.23. Contour. Displacement field.

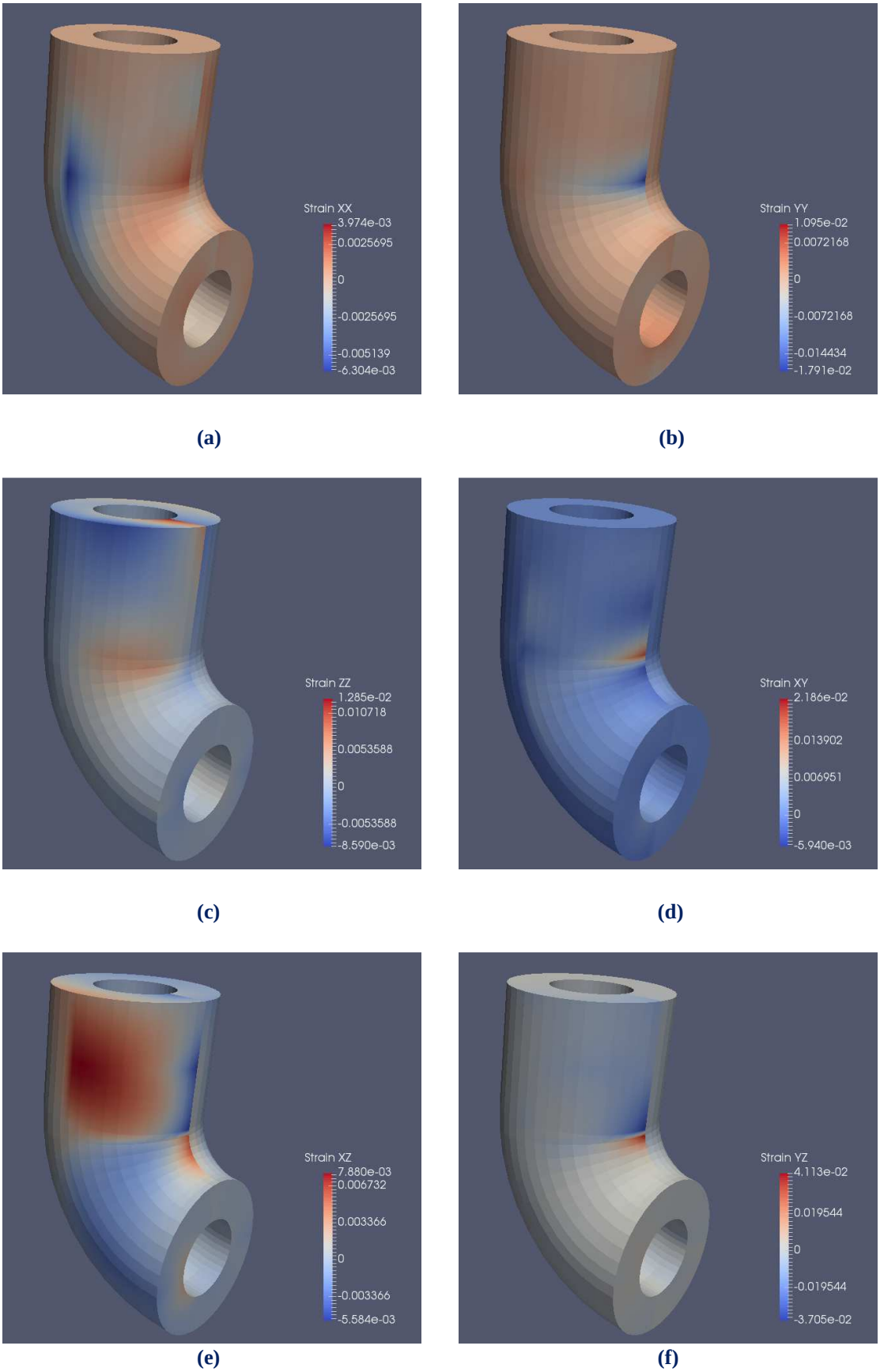


Figure 8.24. Contour. Strain field.

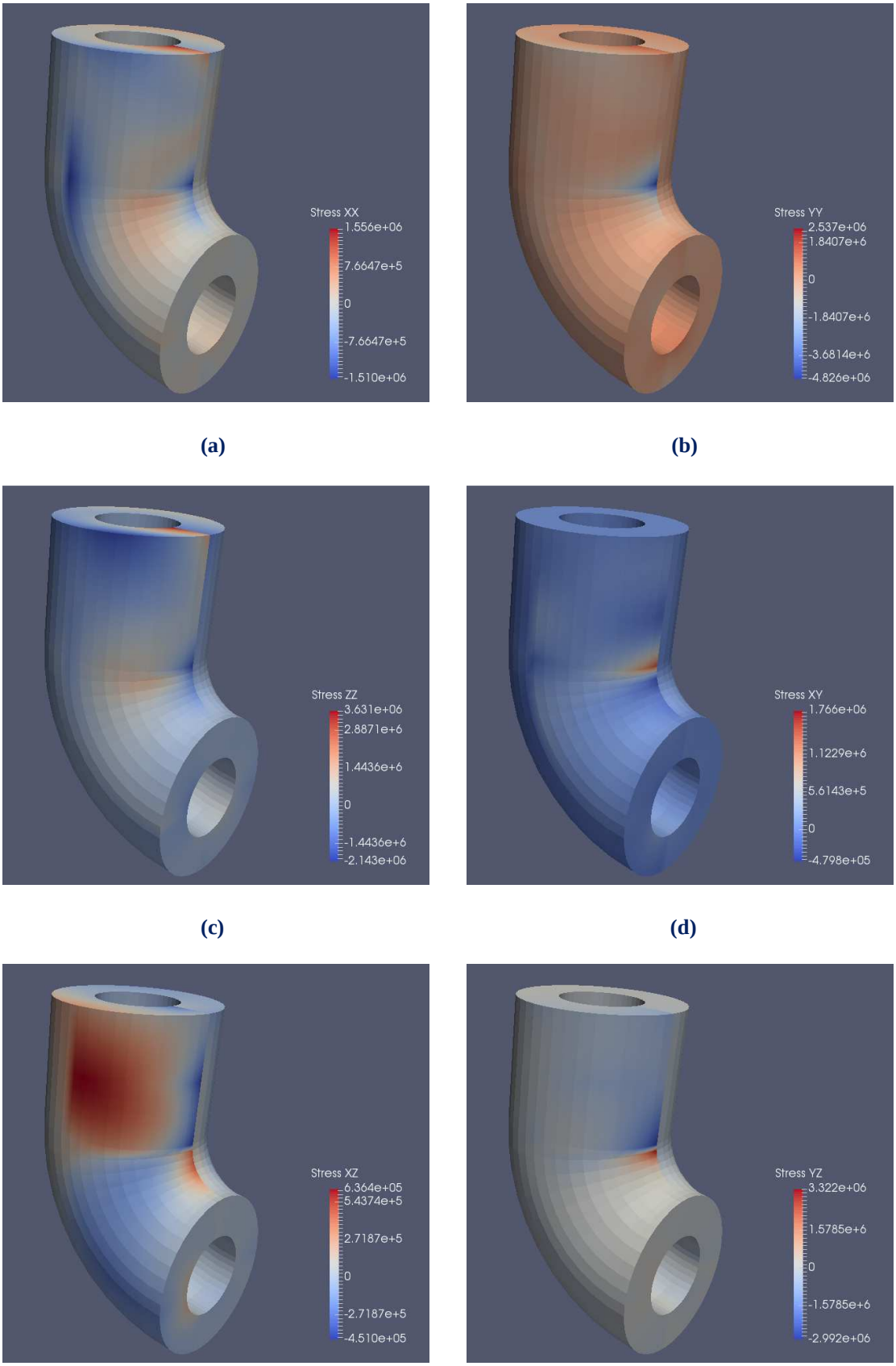


Figure 8.25. Contour. Stress field.

8.5 Thick Cylinder

The next model is the half of a thick cylinder. Parametric direction ξ represents the longitudinal direction of the pipe. Parametric direction η represents the pipe thickness and ξ the creation of the circular shape.

Initially, the cylinder has been designed by using the minimum possible number of control points. Specifically, the basis is quadratic for all axes. There is only one quadratic element along the thickness.

There are n=3 control points (axis ξ), m=3 (axis η) and l=5 (axis ζ), means a total amount of 45 control points and 135 degrees of freedom. However, this coarse mesh cannot ensure accurate results. These can be achieved by using a fine mesh. For this reason, h-refinement is applied.

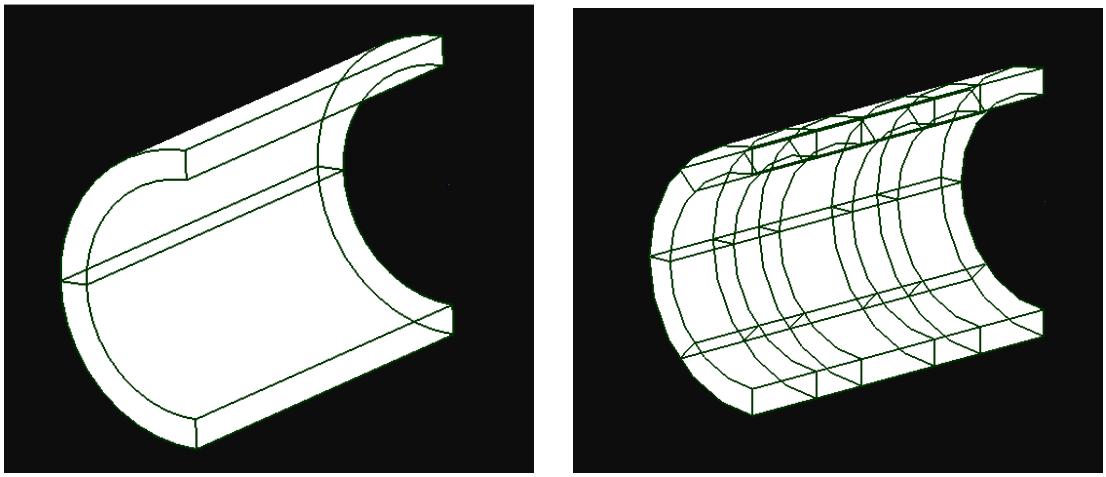


Figure 8.26. Cylinder. NURBS model.

Coarse Mesh: Knot value vector

$$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$$

$$H = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$$

$$Z = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2\}$$

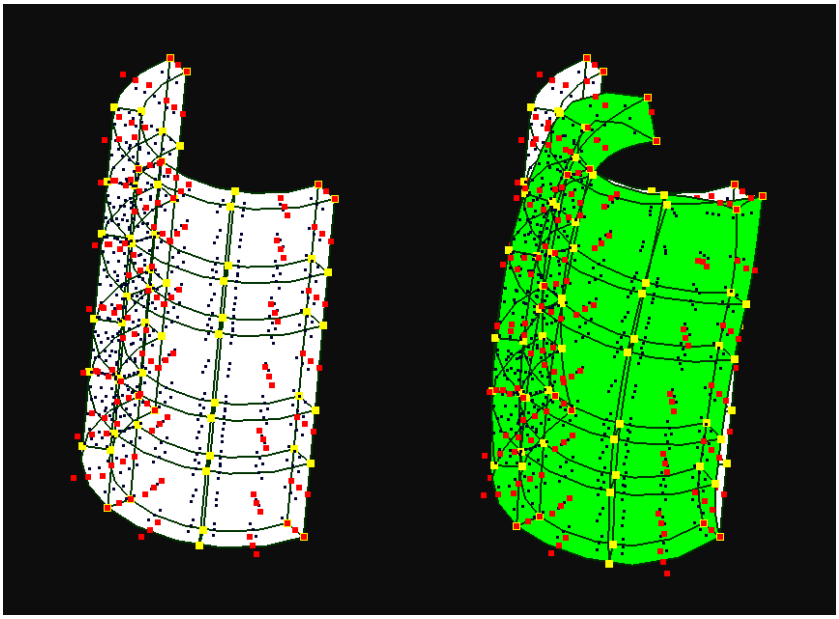
Fine Mesh: Knot value vector

$$\Xi = \{0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 2 \quad 2.5 \quad 3 \quad 3 \quad 3\}$$

$$H = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$$

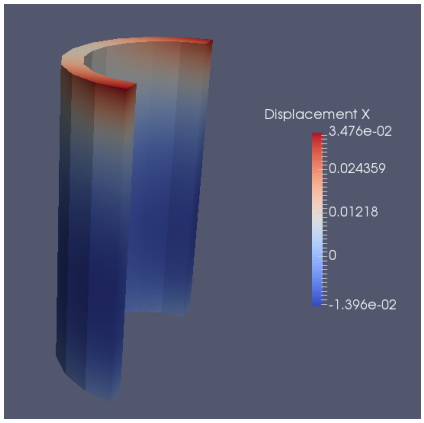
$$Z = \{0 \quad 0 \quad 0 \quad 0.25 \quad 0.5 \quad 0.5 \quad 0.75 \quad 1 \quad 1 \quad 1\}$$

The cylinder is fixed at the bottom.

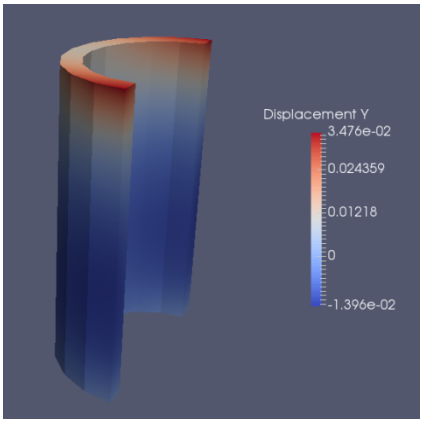


(a) Undeformed **(b)** Deformed Shape.

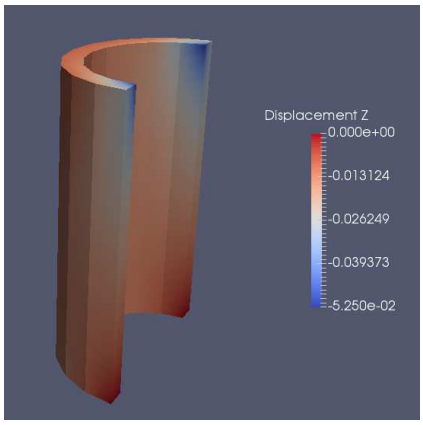
Figure 8.27. Bent pipe. Fine mesh.



(a) X

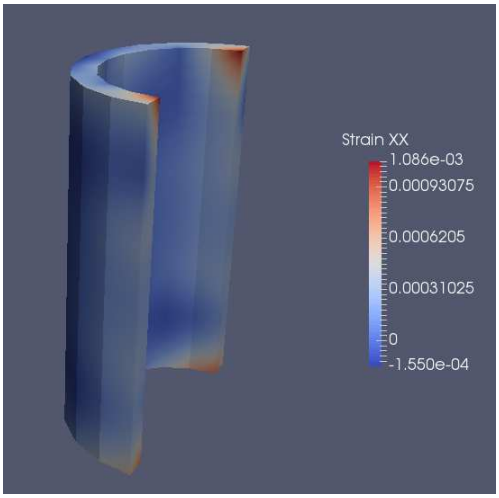


(b) Y

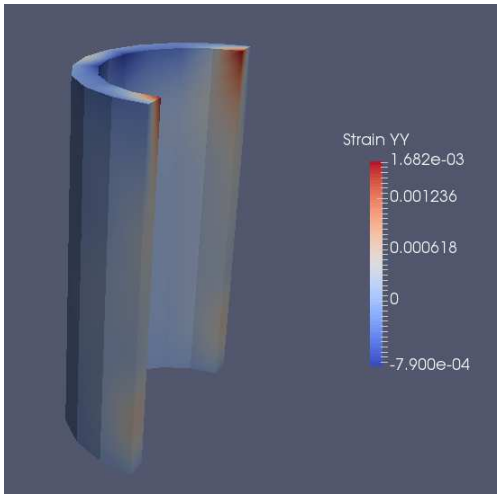


(c) Z

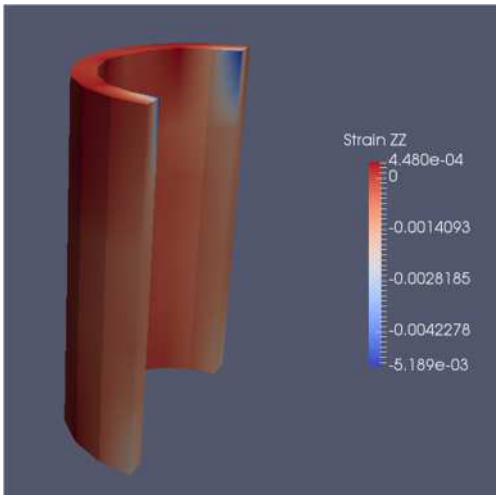
Figure 8.28. Contour. Displacement field.



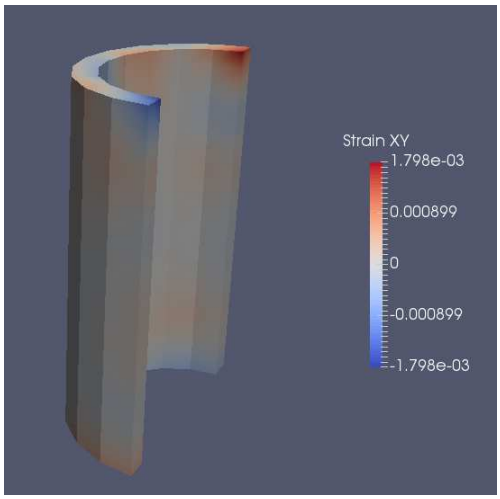
(a)



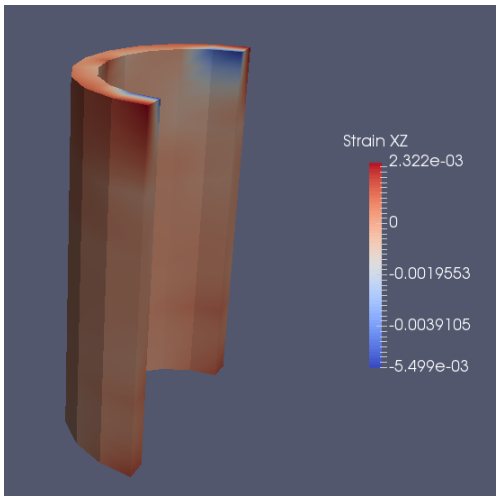
(b)



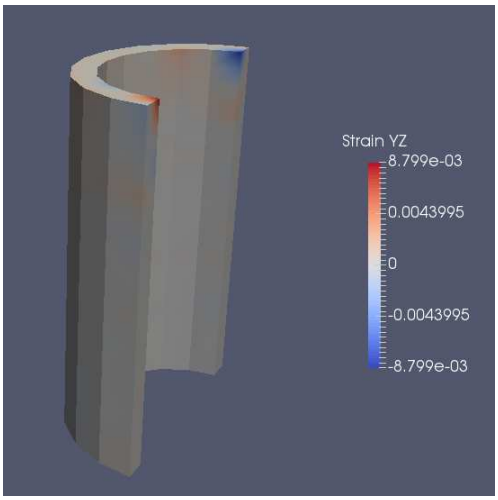
(c)



(d)

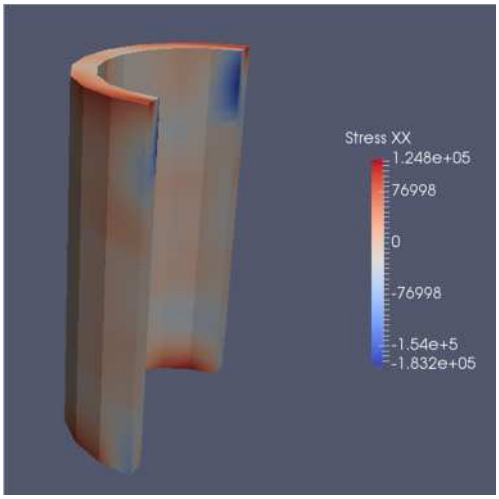


(e)

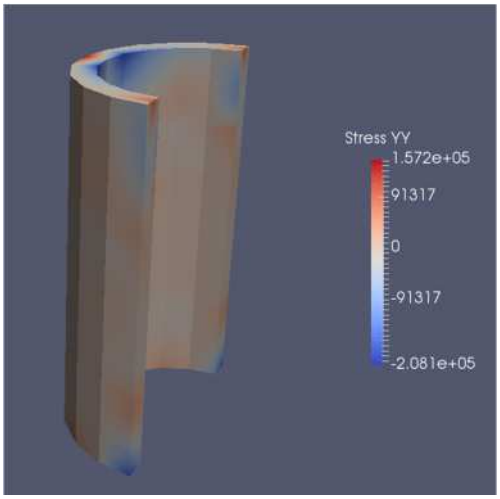


(f)

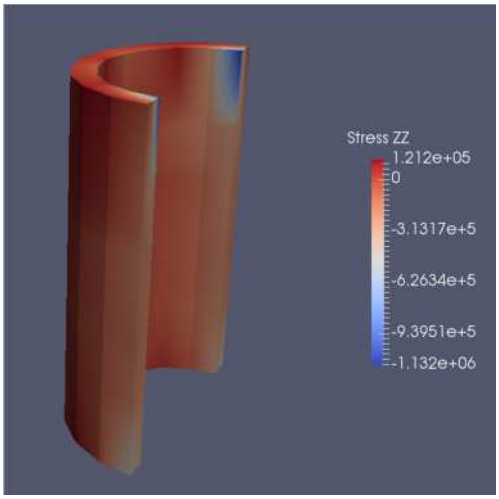
Figure 8.29. Contour. Strain field.



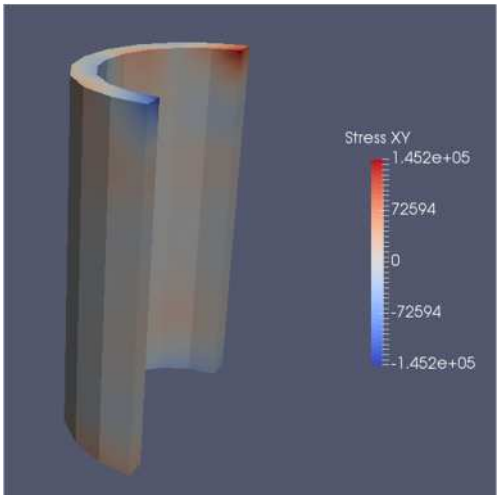
(a)



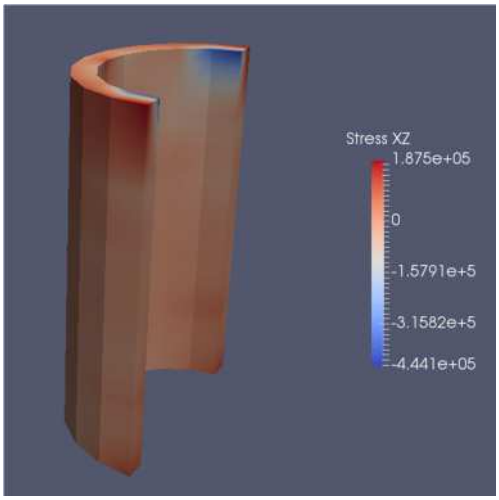
(b)



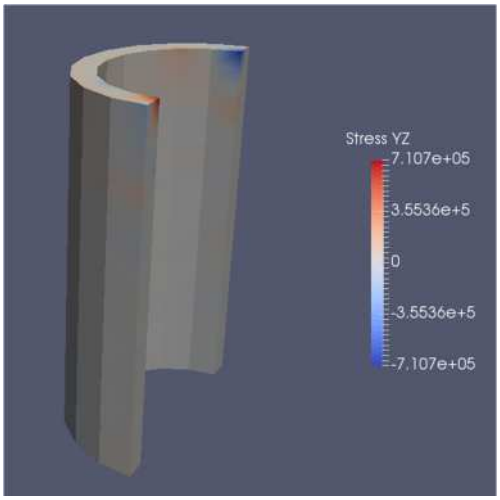
(c)



(d)



(e)



(f)

Figure 8.30. Contour. Stress field.

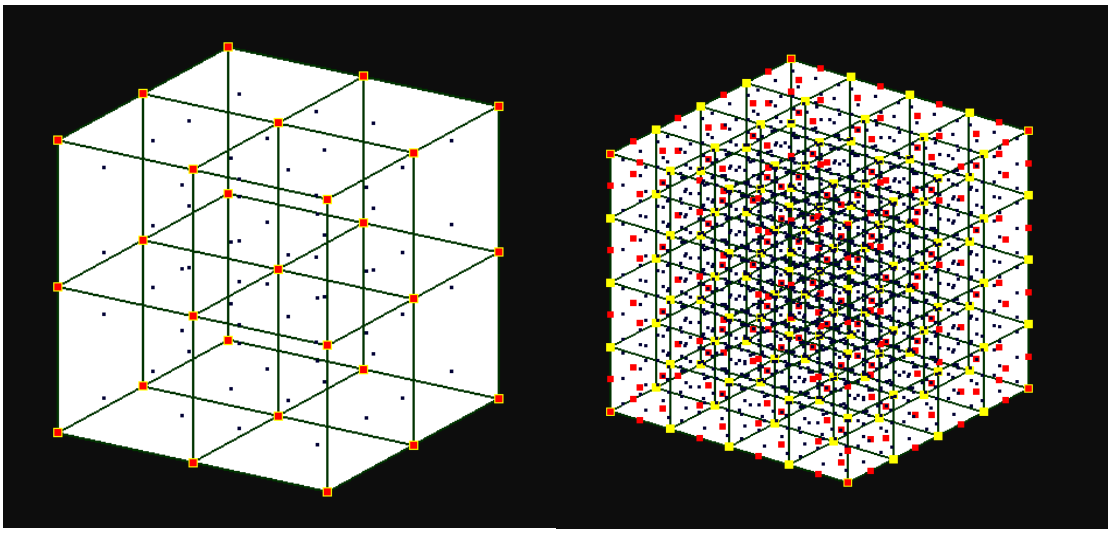
9 Conclusions

Exact Geometry

Isogeometric analysis is a computational approach that manages to bridge the gap between CAD and CAE technologies. The geometry design coincides with the mesh generation. The geometrically exact mesh is adapted for the analysis. Any abrupt change noted to the considered structure is taken into account for the generation of the initial mesh. In this way, it enables the exact representation for the analysis process. The mathematical model has the exact same features as the design model. Therefore, transition from design to analysis does not introduce geometric errors and the accuracy is instantly increased.

Improved refinement schemes

Computational geometry allows more efficient refinement techniques, that are utilized in the analysis process. Analysis is carried out without the creation of an approximate mesh. The analysis mesh is directly connected with the accurate geometrical representation of the model. Refinement preserves the exact geometry and parametric mapping, while providing greater accuracy for analysis. FEM, on the contrary, defines a new approximation with each refinement; thus, IGA refinement is much faster and more efficient. In addition to the classical h and p refinement, a new refinement type is available in IGA, the so-called k-refinement. K-refinement is proved to be the most efficient one.



(a) Coarse mesh **(b) Fine mesh**

Figure 9.1. k-refinement. 3D.

Element interconnectivity - Computational consequences

NURBS shape functions hold an overlapping, which leads to larger interconnectivity between elements. Derivatives are continuous across element boundaries, therefore stress and strain fields are continuous as well.

Higher interconnectivity, that IGA provides, significantly increases the accuracy of the analysis, but it leads to denser stiffness matrices, which demand greater memory resources for the storage of the stiffness matrix. As a result, the computational cost is increased compared to classical FEM for the same number of degrees of freedom. However, this cost can be minimized by extensively applying parallelization to some parts of the code like the phase of the stiffness matrix assembly.

CPUs & GPUs

The scope of using GPUs for scientific computing is not to replace the CPUs. They support asynchronous operations, means they can work on different tasks. In other words, CPUs and GPUs can offer their processing power in order to achieve the maximum possible performance. Nowadays, it is easy someone to find two or four CPU processing cores and one GPU. The optimized choice is to direct all the computational resources of the system to work concurrently for the execution of the same algorithm. In addition to this, last years have been created hybrid supercomputers, which are consisted of many CPUs and GPUs. Their final scope is the cooperation of these processing units, by sharing the workload among the aggregate of processing units for the achievement of a specific task.

State of the art

Isogeometric analysis is a very promising computational method with many potential benefits. The parallelization features, that can be applicable on IGA, cannot be examined in just one thesis. For this reason, some of the following subjects will be investigated in the near future.

- Multi-GPU implementation of formulation of the total stiffness matrix.
- Implementation of the existing code in new GPU's architectures with better hardware specifications than the NVIDIA GeForce GTX 670M which have been used for the execution.
- Implementation of algorithms for solving systems of linear equations in GPUs in order to leverage their performance.
- GPU implementation of IGA with more efficient integration rules or other approaches like collocation, which can reduce the cost of the stiffness matrix assembly, especially for higher-order approximations.
- Exploitation of CPUs' and GPUs' computing capabilities for the concurrent assembly of the same stiffness matrix.
- Exploitation of CPUs' and GPUs' computing capabilities for a hybrid solution of linear system equations.
- Solution of IGA dynamic and non-linear structural problems in parallel programming environments.

Appendix A

IGA Test Examples

The tested examples are presented in Tables A.1 and A.2. In order to maximize the number of calculations, no trivial knot spans are included. The examples are considered to be squares or cubes in the physical space.

For increased accuracy, double-precision floating point format have been used.

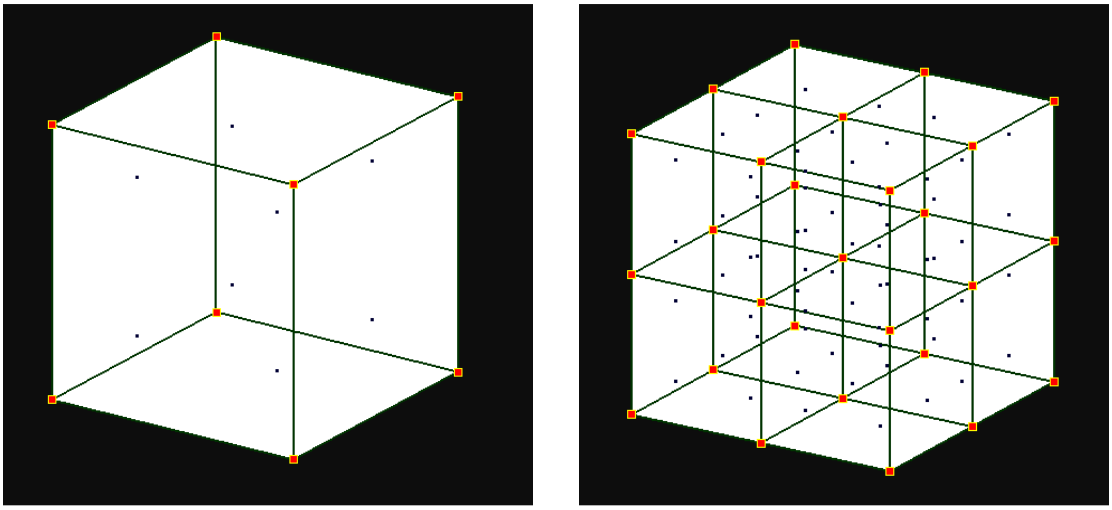
All the examples have been run on a CPU, Core i7-3610QM with 4 cores (8 threads) at 2.30 GHz, 6MB cache and on a GPU, GeForce GTX 670M which comes with 336 CUDA cores (or Shader Processing Units) each one at 598 MHz and 1.5 GB GDDR5 memory with 72 GB/s bandwidth.

	Example	Control Point Number		Degree		Knot Span Number		Finite Element Number	Gauss Point Number	DOF
		ξ	η	ξ	η	ξ	η			
2D Problem Type	P1-1	1.100	1.100	1	1	1.099	1.099	1.207.801	4.831.204	2.420.000
	P1-2	1.000	1.000	1	1	999	999	998.001	3.992.004	2.000.000
	P1-3	900	900	1	1	899	899	808.201	3.232.804	1.620.000
	P1-4	800	800	1	1	799	799	638.401	2.553.604	1.280.000
	P2-1	620	620	2	2	618	618	381.924	3.437.316	768.800
	P2-2	500	500	2	2	498	498	248.004	2.232.036	500.000
	P2-3	350	350	2	2	348	348	121.104	1.089.936	245.000
	P2-4	225	225	2	2	223	223	49.729	447.561	101.250
	P3-1	388	388	3	3	385	385	148.225	2.371.600	301.088
	P3-2	320	320	3	3	317	317	100.489	1.607.824	204.800
	P3-3	275	275	3	3	272	272	73.984	1.183.744	151.250
	P3-4	225	225	3	3	222	222	49.284	788.544	101.250
	P4-1	275	275	4	4	271	271	73.441	1.836.025	151.250
	P4-2	225	225	4	4	221	221	48.841	1.221.025	101.250
	P4-3	160	160	4	4	156	156	24.336	608.400	51.200
	P4-4	100	100	4	4	96	96	9.216	230.400	20.000

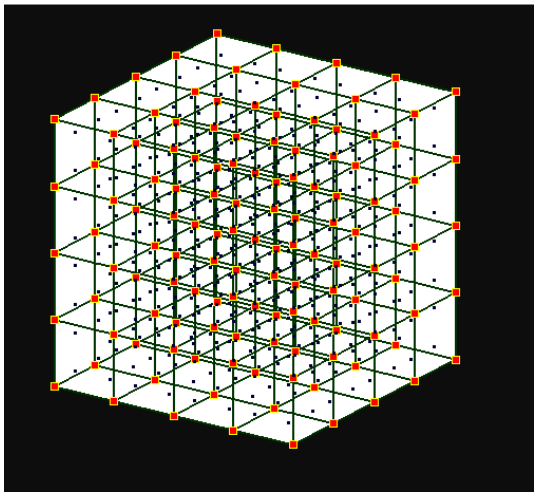
Table A.1. 2D examples.

	Example	Control Point Number			Degree			Knot Span Number			Finite Element Number	Gauss Point Number	DOF
		ξ	η	ζ	ξ	η	ζ	ξ	η	ζ			
3D Problem Type	P1-1	60	60	60	1	1	1	59	59	59	205.379	1.643.032	648.000
	P1-2	50	50	50	1	1	1	49	49	49	117.649	941.192	375.000
	P1-3	40	40	40	1	1	1	39	39	39	59.319	474.552	192.000
	P1-4	30	30	30	1	1	1	29	29	29	24.389	195.112	81.000
	P2-1	34	34	34	2	2	2	32	32	32	32.768	884.736	117.912
	P2-2	33	33	33	2	2	2	31	31	31	29.791	804.357	107.811
	P2-3	26	26	26	2	2	2	24	24	24	13.824	373.248	52.728
	P2-4	19	19	19	2	2	2	17	17	17	4.913	132.651	20.577
	P3-1	23	23	23	3	3	3	20	20	20	8.000	512.000	36.501
	P3-2	21	21	21	3	3	3	18	18	18	5.832	373.248	27.783
	P3-3	19	19	19	3	3	3	16	16	16	4.096	262.144	20.577
	P3-4	17	17	17	3	3	3	14	14	14	2.744	175.616	14.739
	P4-1	17	17	17	4	4	4	13	13	13	2.197	274.625	14.739
	P4-2	15	15	15	4	4	4	11	11	11	1.331	166.375	10.125
	P4-3	13	13	13	4	4	4	9	9	9	729	91.125	6.591
	P4-4	11	11	11	4	4	4	7	7	7	343	42.875	3.993

Table A.2. 3D examples.



(a) **(b)**



(c)

Figure A.1. h-refinement applied on a cubic domain.

(a) Coarse Mesh.

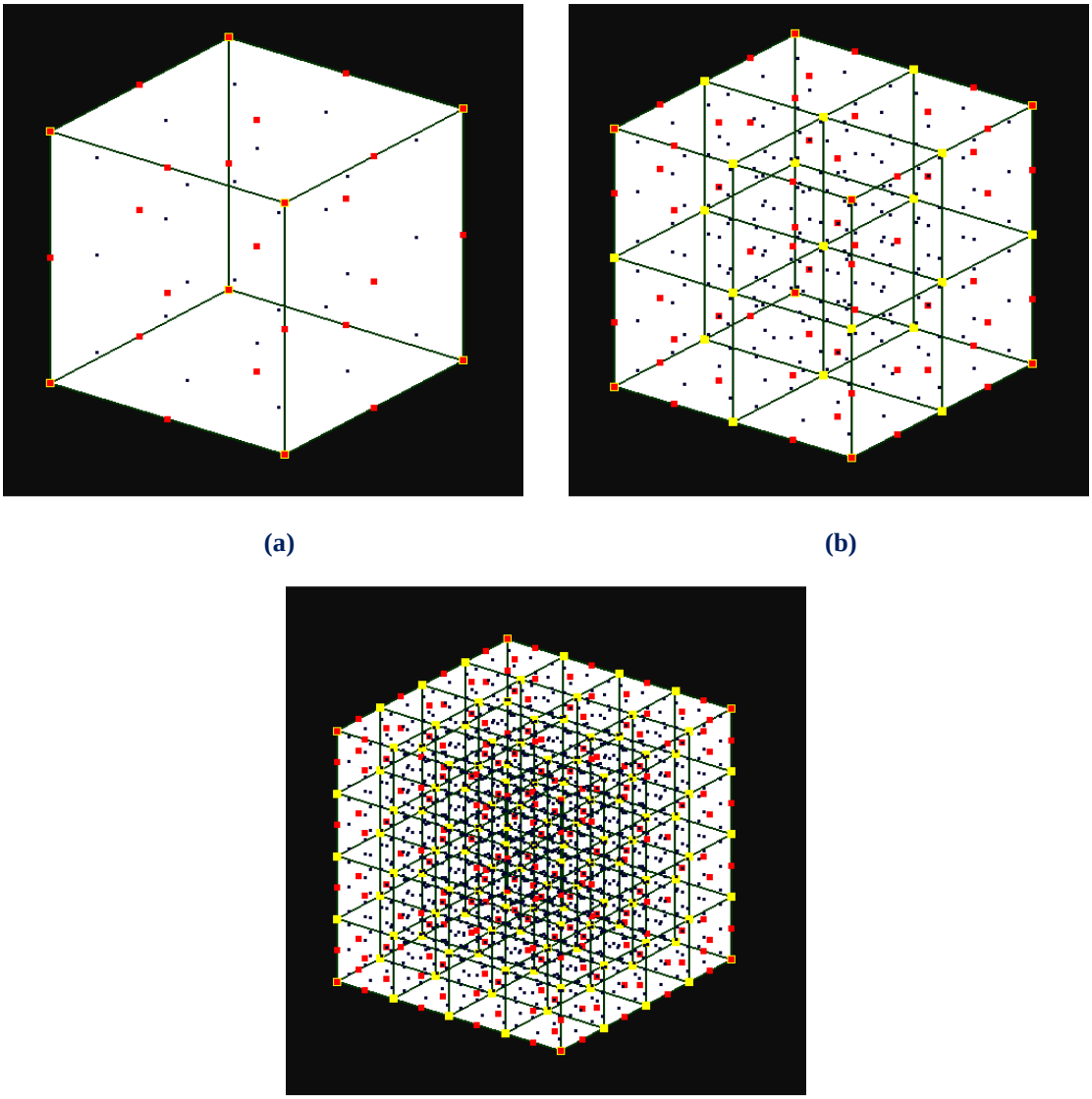
$$\Xi = \{0 \quad 0 \quad 1 \quad 1\}$$
$$H = \{0 \quad 0 \quad 1 \quad 1\}$$
$$Z = \{0 \quad 0 \quad 1 \quad 1\}$$

(b) Fine Mesh.

$$\bar{\Xi} = \{0 \quad 0 \quad 0.5 \quad 1 \quad 1\}$$
$$\bar{H} = \{0 \quad 0 \quad 0.5 \quad 1 \quad 1\}$$
$$\bar{Z} = \{0 \quad 0 \quad 0.5 \quad 1 \quad 1\}$$

(c) New Fine Mesh:

$$\bar{\bar{\Xi}} = \{0 \quad 0 \quad 0.125 \quad 0.25 \quad 0.375 \quad 0.5 \quad 0.625 \quad 0.75 \quad 0.875 \quad 1\}$$
$$\bar{\bar{H}} = \{0 \quad 0 \quad 0.125 \quad 0.25 \quad 0.375 \quad 0.5 \quad 0.625 \quad 0.75 \quad 0.875 \quad 1\}$$
$$\bar{\bar{Z}} = \{0 \quad 0 \quad 0.125 \quad 0.25 \quad 0.375 \quad 0.5 \quad 0.625 \quad 0.75 \quad 0.875 \quad 1\}$$



(a) **(b)**

(c)

Figure A.2. h-Refinement applied on a cubic domain.

(a) Coarse Mesh.

$$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$$
$$H = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$$
$$Z = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$$

(b) Fine Mesh.

$$\bar{\Xi} = \{0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 1 \quad 1\}$$
$$\bar{H} = \{0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 1 \quad 1\}$$
$$\bar{Z} = \{0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 1 \quad 1\}$$

(c) Very Fine Mesh.

$$\bar{\bar{\Xi}} = \{0 \quad 0 \quad 0 \quad 0.125 \quad 0.25 \quad 0.375 \quad 0.5 \quad 0.625 \quad 0.75 \quad 0.875 \quad 1 \quad 1 \quad 1\}$$
$$\bar{\bar{H}} = \{0 \quad 0 \quad 0 \quad 0.125 \quad 0.25 \quad 0.375 \quad 0.5 \quad 0.625 \quad 0.75 \quad 0.875 \quad 1 \quad 1 \quad 1\}$$
$$\bar{\bar{Z}} = \{0 \quad 0 \quad 0 \quad 0.125 \quad 0.25 \quad 0.375 \quad 0.5 \quad 0.625 \quad 0.75 \quad 0.875 \quad 1 \quad 1 \quad 1\}$$

Numerical Results for 2D and 3D Elasticity Problems

Throughout this work, the two approaches of the computation of the stiffness matrix are tested in 2D and 3D elasticity problems. Results of the single core element-wise variant of the CW method and IW variant for element-driven applications in the CPU have been presented in Chapter 7. The comparison of the two approaches in a single core CPU is shown in Table A.3 and A.4.

	Example	DOF	Assembly Time (ms)		Ratio
			Element-wise	Interaction-wise	
2D Problem Type	P1-1	2.420.000	45.534	20.215	2,25
	P1-2	2.000.000	37.552	17.035	2,20
	P1-3	1.620.000	30.355	13.731	2,21
	P1-4	1.280.000	23.831	10.834	2,20
	P2-1	768.800	100.561	72.505	1,39
	P2-2	500.000	65.653	47.313	1,39
	P2-3	245.000	32.231	22.985	1,40
	P2-4	101.250	13.147	9.232	1,42
	P3-1	301.088	195.998	157.057	1,25
	P3-2	204.800	134.285	106.635	1,26
	P3-3	151.250	96.129	78.599	1,22
	P3-4	101.250	65.165	51.915	1,26
	P4-1	151.250	336.900	294.491	1,14
	P4-2	101.250	226.824	189.588	1,20
	P4-3	51.200	106.994	92.534	1,16
	P4-4	20.000	40.580	35.528	1,14

Table A.3. Overview of the numerical results obtained with the element-wise (EW) and interaction-wise (IW) approaches in the CPU (single core). 2D problem type.

	Example	DOF	Assembly Time (ms)		Ratio
			Element-wise	Interaction-wise	
3D Problem Type	P1-1	648.000	191.590	66.174	2,90
	P1-2	375.000	109.343	39.128	2,79
	P1-3	192.000	55.167	19.826	2,78
	P1-4	81.000	21.987	7.789	2,82
	P2-1	117.912	880.786	390.236	2,26
	P2-2	107.811	799.103	355.109	2,25
	P2-3	52.728	371.135	167.717	2,21
	P2-4	20.577	132.522	59.724	2,22
	P3-1	36.501	2.658.773	1.276.194	2,08
	P3-2	27.783	1.991.082	947.512	2,10
	P3-3	20.577	1.432.565	660.692	2,17
	P3-4	14.739	915.342	433.917	2,11
	P4-1	14.739	5.372.734	2.770.993	1,94
	P4-2	10.125	3.295.937	1.701.021	1,94
	P4-3	6.591	1.924.829	881.820	2,18
	P4-4	3.993	853.529	398.875	2,14

Table A.4. Overview of the numerical results obtained with the element-wise (EW) and interaction-wise (IW) approaches in the CPU (single core). 3D problem type.

References

[1] Isogeometric Analysis: Toward Integration of CAD and FEA - J. Austin Cottrell, Thomas J. R. Hughes, Yuri Bazilevs - Wiley, 2009

[2] Analysis of Structures with the Finite Element Method - M. Papadrakakis - Papasotiriou, 2001

[3] The NURBS Book - L. Piegl, W. Tiller - Springer, 1997

[4] Efficient quadrature for NURBS- based Isogeometric Analysis - T. J. R. Hughes, A. Realli, G. Sangalli - 2008

[5] A filly "locking-free" isogeometric approach for plane linear elasticity problems – Auricchio, Veig, Buffa, Lovadina, Reali, Sangalli – 2007

[6] A Practical Guide to Splines - de Boor - 1st Revised Edition 2001, Springer

[7] An introduction to IGA with Matlab implementation, FEM and XFEM Formulations - Nguyen, Simpson, Bordas, Rabczuk - 2012

[8] An Introduction to NURBS With Historical Perspective - Rogers - 2000, The Morgan Kaufmann Series in Computer Graphics

[9] Comparison of Numerical Quadrature Schemes in IGA - Rypl, Patzak – 2011

[10] Curves and Surfaces for CAGD, A Practical Guide - Farin - 5th Edition 2001

[11] Efficient quadrature for NURBS-based IGA - Hughes, Reali, Sangalli - 2008, Elsevier

[12] Enhancing IGA by a FEA-based local refinement strategy - Kleiss, Jüttler, Zulehner – 2011

[13] Isogeometric Analysis, CAD, finite elements, NURBS, exact geometry and mesh refinement - Hughes, Cottrell, Bazilevs – 2004

[14] The cost of continuity, Performance of using direct solvers in IGA - Collier, Pardo, Dalcin, Paszynski, Calo - 2011

[15] GPU accelerated computation of the isogeometric analysis stiffness matrix - Karatarakis A., Karakitsios P., Papadrakakis M. – 2013

[16] Massively Parallel Implementation of Finite Element, Meshless and Isogeometric Analysis Methods in Computational Mechanics – Alexander Karatarakis – PhD Dissertation, National Technical University of Athens (NTUA), 2014

[17] Isogeometric Methods for Numerical Simulation – Gernot Beer, Stephane Bordas – Springer, 2015

[18] Programming Finite Elements in Java – Gennadiy Nikishkov – Springer, 2010

[19] Parallel Scientific Computing. Theory, Algorithms, and Applications of Mesh Based and Meshless Methods – Roman Trobec, Gregor Kosec- Springer, 2015

[20] C# Multithreaded and Parallel Programming – Rodney Ringler – Packt Publishing

[21] Microsoft Visual C# 2013 Step by Step – John Sharp – Microsoft Press – 2013

[22] Programming Massively Parallel Processors. A Hands-on Approach – David B. Kirk, Wen-mei W. Hwu – Elsevier Inc. – 2010

[23] The CUDA HANDBOOK. A Comprehensive Guide to GPU Programming – Nicolas Wilt – Pearson Education – 2013

[24] CUDA C Programming Guide Version 7.5

[25] CUDA C Best Practices Guide Version 7.5

[26] Cornell Virtual Workshop, Introduction to GPGPU and CUDA Programming, https://cvw.cac.cornell.edu

[27] geomiso.com