Εθνικό Μετσόβιο Πολυτεχνείο

Thesis: Efficient Isogeometric Analysis of Structures with Complicated Geometry

Extended Synopsis

The scope of this thesis is the actual implementation of Linear Static Isogeometric Analysis, (an innovative methodology of complete CAD – CAE integration introduced by J. Austin Cottrell, Thomas J.R. Hughes and Yuri Bazilevs) to real engineering structures with complicated geometry. NURBS models have been analyzed assuming linear elasticity. Finite Element Method and Non-Uniform Rational B-SPLines have been examined separately, as the two components of Isogeometric Analysis. The routines, which have been used for the linear static analysis of the presented applications, were developed in the high-level language "C++" and "OpenGL" on the interactive environment of well known Open Source Software. Additionally, geometrical representations were acquired through the NURBS-based software "Rhinoceros 3D" and "Maya". The topics considered are B-SPLine and NURBS entity properties, Refinement techniques, Stiffness Matrix Formulation, result post-processing (displacement, stress, and strain field) and linear 2D and 3D applications investigating models of various representations.

Σύνοψη

Ο στόχος της παρούσας διπλωματικής είναι η διερεύνηση και πραγματική εφαρμογή της Γραμμικής Ισογεωμετρικής Ανάλυσης, (που αποτελεί μια καινοτόμο μεθοδολογία πλήρους ενσωμάτωσης των τεχνολογιών CAD – CAE και πρωτοπαρουσιάστηκε από τους J. Austin Cottrell, Thomas J.R. Hughes και Yuri Bazilevs) σε αληθινές κατασκευές εξαιρετικά πολύπλοκης γεωμετρίας. Η ανάλυση πραγματοποιήθηκε σε προσομοιώματα NURBS με την παραδοχή γραμμικής ελαστικότητας. Η Μέθοδος των Πεπερασμένων Στοιχείων και τα Non-Uniform Rational B-SPLines μελετήθηκαν ξεχωριστά, ως τα δύο συστατικά της Ισογεωμετρικής Ανάλυσης. Οι ρουτίνες που χρησιμοποιήθηκαν για την γραμμική στατική ανάλυση των εφαρμογών αναπτύχθηκαν στην γλώσσα προγραμματισμού υψηλού επιπέδου “C++” και “OpenGL”, σε διαδραστικό περιβάλλον πολύ γνωστού, ανοιχτού κώδικος λογισμικού. Επιπρόσθετα, οι γεωμετρικές αναπαραστάσεις παράχθηκαν μέσω των NURBS-based λογισμικού “Rhinoceros 3D” και “Maya”. Τα θέματα που εξετάσθηκαν αφορούν ιδιότητες των οντοτήτων B-SPLine και NURBS, τεχνικές Διακριτοποίησης, Μόρφωση του Μητρώου Στιβαρότητας, επεξεργασία των αποτελεσμάτων (πεδία μετατοπίσεων, τάσεων, παραμορφώσεων) και γραμμικές εφαρμογές 2D και 3D με τη διερεύνηση διάφορων αναπαραστάσεων.

Acknowledgements

I would like to express my sincerest gratitude to all the people that contributed to the creation of this thesis. First of all, I would like to thank Professor **Manolis Papadrakakis** for the opportunity to get involved with Isogeometric Analysis and the research experience in NTUA. I would also like to acknowledge PhD Candidate **Panagiotis Karakitsios,** for his collaboration the last two years and his useful guidance and advice all this time. The result presented today would not be the same without him. Moreover I want to show my appreciation to the researcher **Christos Gritzalis** for the many hours he devoted for the final completion of the software with which all of my applications were tested. Last but not least, I would like to thank my parents for giving me the opportunity to study, but also for the constant guidance.

Ευχαριστίες

Θα ήθελα να εκφράσω την αμέριστη ευγνωμοσύνη μου σε όλους όσους συνέβαλλαν στη δημιουργία αυτής της διπλωματικής. Πρώτο θα ήθελα να ευχαριστήσω τον Καθηγητή **Μανόλη Παπαδρακάκη** για την ευκαιρία που μου προσέφερε να ασχοληθώ με την Ισογεωμετρική Ανάλυση και την εμπειρία της έρευνας στο ΕΜΠ. Θα ήθελα επίσης να ευχαριστήσω τον Υποψήφιο Διδάκτωρ **Παναγιώτη Καρακίτσιο** για την συνεργασία μας αυτά τα δύο χρόνια, την καθοδήγησή του και τις συμβουλές του. Το σημερινό αποτέλεσμα δεν θα ήταν το ίδιο χωρίς τη συνεισφορά του. Θερμές ευχαριστίες και στον ερευνητή **Χρήστο Γκρίτζαλη** για την αμέριστη προσπάθεια και επιμονή την οποία επέδειξε έτσι ώστε να ολοκληρώσουμε μαζί το λογισμικό στο οποίο πραγματοποιήθηκαν όλες μου οι εφαρμογές. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου για την ευκαιρία που μου προσέφεραν να σπουδάσω, καθώ και για την συνεχή καθοδήγηση τους.

Πάνω από όλους, θα ήθελα να ευχαριστήσω την Αξιότιμη Πρέσβειρα Γιάννα Αγγελοπούλου Δασκαλάκη, η οποία πίστεψε στις δυνατότητές μου και συνέβαλε στην εξέλιξη τόσο της προσωπικής όσο και της ίδιας της ερευνητικής μου σταδιοδρομίας.

Prologue

My daily routine as a student in School of Civil Engineering, National Technical University of Athens was very different two years ago. It all began in 2013, a few days after the end of the summer exam period. Professor Manolis Papadrakakis, who taught "Statics III – the Direct Stiffness Method" at this semester, distinguished me as one of the students with the best project for that year and proposed the beginning of a cooperation that would lead to the composition of my thesis.

After familiarizing with the FEM basics, I began my cooperation with PhD Candidate Panagiotis Karakitsios, who is involved in advanced computational methods, and more specifically Isogeometric Analysis. That was the time I was called to work on the challenging and intriguing topic of analyzing real structures of highly complicated geometry. I was actually called to implement and apply the method of Isogeometric Analysis into real projects proving that this revolutionary method could in fact fully integrate CAD with CAE and solve many of the problems that FEM could not deal with. Over the past two years, I was introduced into the research community, a different way of thinking and collaborators of remarkable intellect and extraordinary teamwork. I was provided the chance to use my creativity and give something in return to my University. Advance computational methods keep the answer to many of our everyday problems and it is our duty, as tomorrow's scientists to unlock its secrets.

Institute of Structural Analysis and Antiseismic Research at School of Civil Engineering provides an excellent starting point for young engineers. In the middle of a difficult economical situation, Centers of Excellence are a hopeful prospect for the aspiring researchers.

George Karaiskos
Athens, October 2015

Table of Contents

Table of Figures

1 The Concept of Isogeometric Analysis

1.1 Finite Element Method

1.1.1 Historical Overview

Isogeometric Analysis as a historic evolutionary computational mechanics achievement

Isogeometric Analysis is an innovative method, which integrates design and analysis in the greatest scale ever achieved. It was conceived by Thomas J. R. Hughes in 2003, Professor of Aerospace Engineering and Engineering Mechanics of the University of Texas at Austin.

Thomas J. R. Hughes is one of the greatest experts worldwide in computational and applied mathematics. He began his career as a mechanical design engineer at Grumman Aerospace and then went on to General Dynamics as a research and development engineer. After receiving his Ph.D. from the University of California at Berkley, he joined the faculty and moved on to California Institute of Technology. Afterwards, he was hired by Stanford University before joining the University of Texas at Austin.

Figure 1.1. Thomas Joseph Robert Hughes.

(http://users.ices.utexas.edu/~hughes/**)**

Isogeometric Analysis is set to bridge the gap that exists between Computer-Aided Design and Computer-Aided Engineering. Thomas J.R. Hughes published a book in 2009, along with J.A. Cottrell and Y. Bazilevs, explaining the fundamentals of this new method. The book, "Isogeometric Analysis: Towards Integration of CAD and FEA" is the first book to be issued on this new field of study and a trustworthy guide to the researchers that want to invest in this method. It contains a vast number of applications of Isogeometric Analysis, potential resources for the new researcher, advantages of the new method and future fields of study. The application of isogeometric methods can lead to results and improvements in computational mechanics, structural statics and dynamics and biomechanics.

The need for Isogeometric Analysis

The compelling 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 Finite Element Method instead of the traditional by-hand ways, design errors cannot be avoided. Since 2000, structural collapse cost humanity over 1500 lives; many of them could have been spared, had the engineer a more accurate tool for analysis and design. Unfortunately, present analysis technologies require a lot of man-hours for manual generation of approximated, FEM-suitable geometries and 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 transformation. Furthermore, the approximation of the geometry sometimes is clearly not enough for the desired convergence. These are the gaps that Isogeometric Analysis is set to fill. In order to fully understand Isogeometric Analysis, one has to acknowledge the evolution of analysis throughout its history and understand the principles this revolutionary method is based on. This introduction provides the historical review of the technologies and the requirements of structural analysis that led to the creation of this new method.

Before Finite Element Method

Engineers of the past had to meet the demands for structural analysis and representation of accurate results. Construction always needed the cost-efficiency provided by design. In the early years, the only weapon the engineer possessed was his mind. In order to solve a statically indeterminate structure, equilibrium, constitutive and compatibility laws had to be applied. Several methods existed in order to solve the problem.

The Force Method used the Betti-Maxwell Theorem in combination with virtual works in order to provide support forces and moments. The main idea is the preservation of equilibrium and calculation of forces, in order to ensure compatibility.

The Displacement Method, on the other hand, ensures 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, but its efficiency made it very popular among the engineers in the 1930s.

These methods had their limits; a vast amount of time was required for the solution and many errors would occur in the manual computational process. More complex problems could only be approximated and sometimes analysts 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 stress, strain and displacement conditions of the structure. Structural mechanics theorems were developed and used in order to solve the 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 birth of Finite Element Methods can be placed at 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 challenging 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 had studied Civil Engineering at the National Technical University of Athens.

Figure 1.2. John Argyris.
(http://www.nae.edu/27953.aspx**)**

Figure 1.3. City of Volos. Central Greece. John Argyris Birthplace.
(http://www.greekscapes.gr/~landscapesatlas**)**

John Argyris graduated from Technical University of Munich in 1936 and had begun to work in industrial applications of complex structures. He remained in Germany at 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 (FEM). All relative papers were at once labeled “secret”. This innovative method included a different measurement of stresses and strains, 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 Finite Element Method 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 Papardakakis Manolis has devoted his career in the evolution and outspread of FEM technologies. Nowadays, in the middle of the financial crisis in Greece, NTUA's Institute of Structural Analysis and Antiseismic Research 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, in National Technical University of Athens, continuing and improving the cycle of expanding the boundaries of human knowledge.

In the dawn of 2000, the structural engineering field 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 his field of study 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 [17]. Simulia Abaqus, originally released in 1978, was developed using an open-source language, Python and was initially intended for non-linear problems. It is particularly popular due to its wide range of modelling capabilities, both for linear and non-linear problems [16]. ADINA (Automatic Dynamic Incremental Non-Linear Analysis), first 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 [19]. FEMAP (Finite Element Modeling and Post-processing) 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 [20].

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 [18]. SOFiSTIK, first used in 1987, is directed towards bridge linear and non-linear analysis [22]. 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. StereoSTATIKA is suitable for reinforced concrete analysis in countries with dangerous seismic activity [21]. FESPA is a Greek software dedicated to analysis and design of reinforced concrete and steel buildings [23].

1.1.2 Basic Idea of Finite Element Method

According to [2], the basic idea behind Finite Element Method is the approximation of the solution field via piecewise polynomial functions, called the shape functions N. Displacement values U at any internal point of the element can be computed from displacements at nodes d:

$$\underset{(3\times1)}{\{U(X,Y,Z)\}} = \underset{(3\times3n_e)}{[N(X,Y,Z)]} \cdot \underset{(3n_e\times1)}{\{d\}}$$

where n_e is the number of the finite elements.

A generalization for the whole structure leads to

$$\underset{(3\times1)}{\left\{U^S(X,Y,Z)\right\}} = \underset{(3\times3n)}{\left[N^S(X,Y,Z)\right]} \cdot \underset{(3n\times1)}{\left\{d^S\right\}}$$

The problem is directly downsized from infinite unknowns to a finite number of degrees of freedom. The next step is to define stress and strain matrices and their interconnection, which represents Hooke's Constitutive Law.

$$\{\sigma\} = \begin{bmatrix} \sigma_X \\ \sigma_Y \\ \sigma_Z \\ \sigma_{XY} \\ \sigma_{YZ} \\ \sigma_{ZX} \end{bmatrix} \quad \{\varepsilon\} = \begin{bmatrix} \varepsilon_X \\ \varepsilon_Y \\ \varepsilon_Z \\ \gamma_{XY} \\ \gamma_{YZ} \\ \gamma_{ZX} \end{bmatrix} \quad (\text{3D case})$$

$$\{\sigma\} = [E] \cdot \{\varepsilon\}$$

Deformation matrix [B] evaluates strains anywhere in the model from nodal displacements.

$$\{\varepsilon(X,Y,Z)\} = [B(X,Y,Z)] \cdot \{d\}$$

Using internal and external virtual work equilibrium, we can evaluate the stiffness matrix [k] for each element:

$$\underset{(3n_e\times3n_e)}{[k]} = \int_V \underset{(3n_e\times6)}{[B]^T} \cdot \underset{(6\times6)}{[E]} \cdot \underset{(6\times3n_e)}{[B]}\, dV$$

In the same manner, distributed loads can be transformed into equivalent nodal loads.

$$\underset{(3n_e\times1)}{\{r\}} = \int_V \underset{(3n_e\times3)}{[N]^T} \cdot \underset{(3\times1)}{\{f\}}\, dV$$

The contribution of each element is added to the global matrices, producing the global stiffness matrix [K] and the force vector {R}. The displacement matrix can now be calculated from:

$$\underset{(3n\times1)}{\{R\}} = \underset{(3n\times3n)}{[K]} \cdot \underset{(3n\times1)}{\{D\}}$$

where n is the number of the nodes.

We can observe that (naturally) shape functions [N] and deformation matrix [B] depend on the Cartesian coordinates. This leads to iteration problems when complex geometries (e.g. circles or conic sections in general) are involved. The solution, which contributed to the further generalization of FEM, along with a more efficient iteration algorithm, is isoparametric elements.

The basic idea is the existence of a parent element in the parameter space, which can be modeled as a regular shape (e.g. a cube, a square or an equilateral triangle). Each element in the physical space (the "real" modeling space) can be described using a linear combination of the parent element's shape functions. Hence, geometry can be approximated by the same functions used for the solution field. This explains the name "isoparametric".

Drawbacks

Despite the evolution of FEM all this time, some problems have yet to be solved. For one, even isoparametric elements can only produce an approximation of geometry. The most challenging tasks of the day often require exact geometrical representation in order to achieve the necessary accuracy. After the meshing has been completed, the initial geometry plays no more role in the analysis procedure. This is intuitively worrying, to begin with. Furthermore, it produces a vast number of problems. The inevitable geometrical approximation means there will be convergence errors by definition, regardless of the solution methods and the available computational power. This affects the efficiency of the computational methods used for the solution.

If a finer mesh is required, refinement algorithms will return to the initial geometry and produce a different approximation. The new, fine mesh cannot be directly produced from the coarse mesh. Efficiency is certainly at a low, as procedures already completed have to be repeated in order for the new mesh to be created. Precious analysis time is required and the geometrical differences between the coarse and fine mesh make it difficult to compare the results.

Hierarchical structures provide even greater challenges. Hierarchical refinement is considered an efficient refinement technique, as it focuses on the crucial areas of the model, but it is not easily applicable in FEM meshes.

The lack of integration between geometry and mesh generation is crucial. Computational geometry provides simply an input file for the meshing of the finite element model. 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 frustrating and pointless for the modern engineer; instead of devoting his time in creativity and design, he has to undergo the mundane task of regenerating a slightly different mesh over and over again. Many design errors are attributed to this task. In structural design, a vast amount of time is spent on these integration cycles between initial, preliminary and final design phases. The risk is even greater in complex and innovative structures, where the engineers cannot know beforehand what structural results to expect and have almost no way of checking the accuracy of the analysis.

Computational geometry has evolved since its birth; new and optimized geometrical 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.

Figure 1.4 depicts a car model. In order to create this object, a designer has to define the following variables:

- Degree of shape functions for each parametric axis
- Knot value vector for each parametric axis
- Control points (Cartesian coordinates and weights)

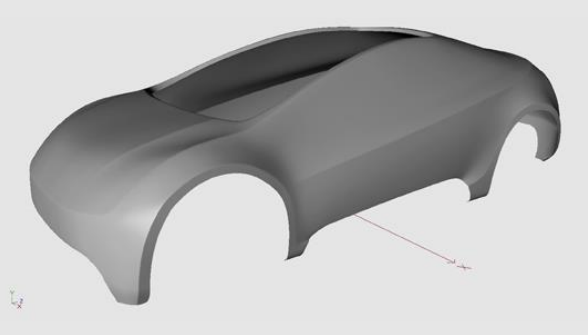


Figure 1.4. Car model. Geometry design.
(https://grabcad.com/library/tag/tutorial**)**

As a result, a mesh of finite elements is created, the “exact geometry mesh”, which surprisingly is not used by FEM software, which instead creates a new approximate one. **Figure 1.5** represents the initial geometry mesh of the car with the corresponding control net.

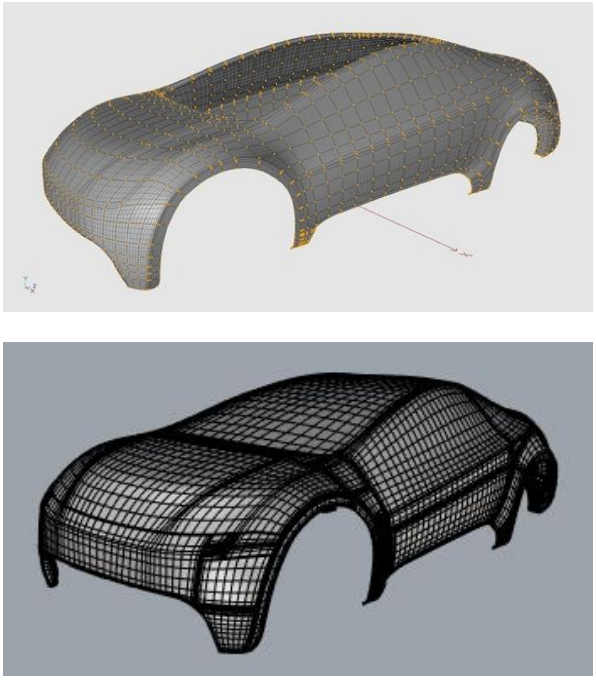
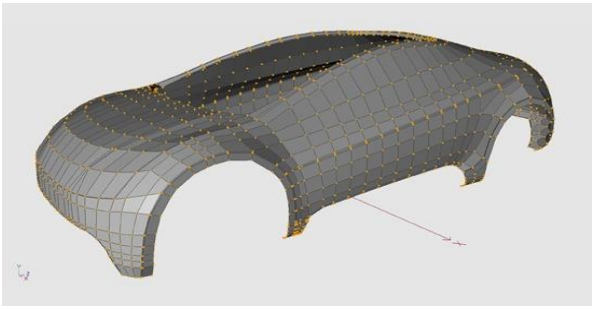
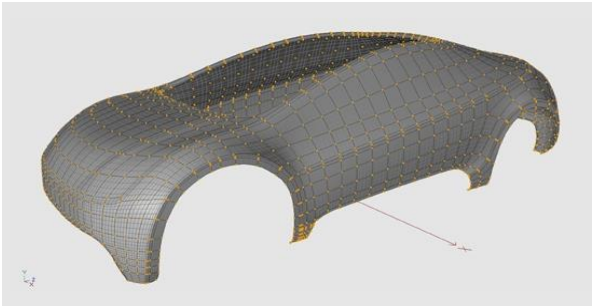


Figure 1.5. Car model.
Geometry initial mesh and control points.
(https://grabcad.com/library/tag/tutorial**)**

With the previous initial mesh, a designer can represent exactly the car model, but an engineer cannot analyze it accurately and has to apply refinement in order to increase the accuracy of the solution field. **Figure 1.6** shows a finer mesh of the car model for both cases. In FEA, the smooth surface of the car participates only as input in FEM software and is thereafter replaced by quad finite elements. In IGA, the geometry remains intact and the mesh is the exact geometrical model.



(a) Fine FEA mesh (top one).



(b) Fine IGA mesh.

Figure 1.6. Car model.

1.2 Computer Aided Design

1.2.1 Historical Overview

Isogeometric Analysis as a historic evolutionary computational design achievement

The evolution of computing systems made the option to 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, one piece at a time in a computer. 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 room for improvement. Computational geometry is involved in a vast number of engineering applications and should not be considered independently. Design entities are supposed to cooperate with finite element methods. Re-inventing and improving computational geometry structures is the first step in completing this task. In order to understand the modern and future world of Computer-Aided Design, one has to study the history, the creation and the necessities that led to the creation of computational geometry entities.

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. Anything ring a bell so far [24]?

Figure 1.7. Original splines and weights (called Ducks).
(http://www.boatdesign.net)

The development of NURBS arose from the need to effectively represent 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 finally introduced B-SPLines, which were similar to Bezier curves, meaning that the curve was defined by a set of points, called the "control points", but their number was independent of the polynomial order of the curve. B-SPLines were a generalization of Bezier curves, but they were also 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 [26].

After that, NURBS began to spread across the CAD industry. They possessed a lot of interesting attributes. The parameterization of the whole curve was downsized to a few control points coordinates, numerically stable mathematical procedures and easy modification. Cartoon characters, videogame graphics, ships, cars, airplanes were designed using NUBRS. 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 the increasing popularity of Non-Uniform Rational B-SPLines in the CAD industry [25].

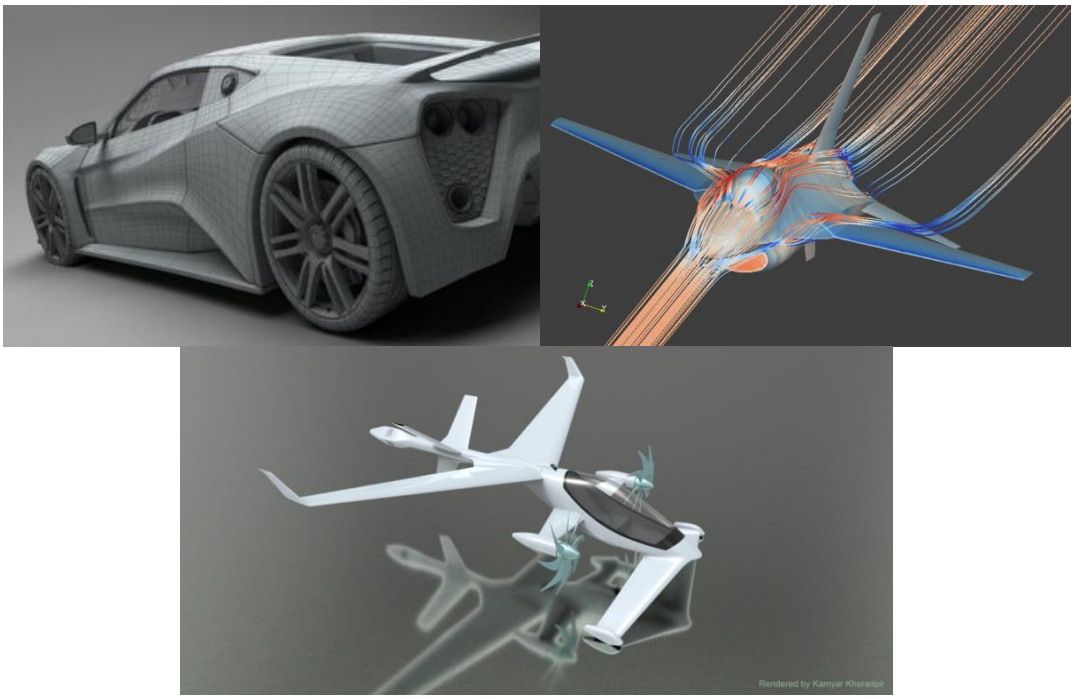
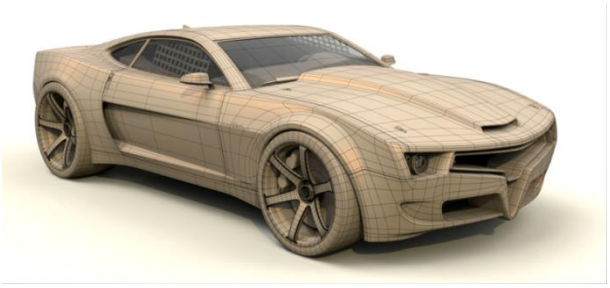
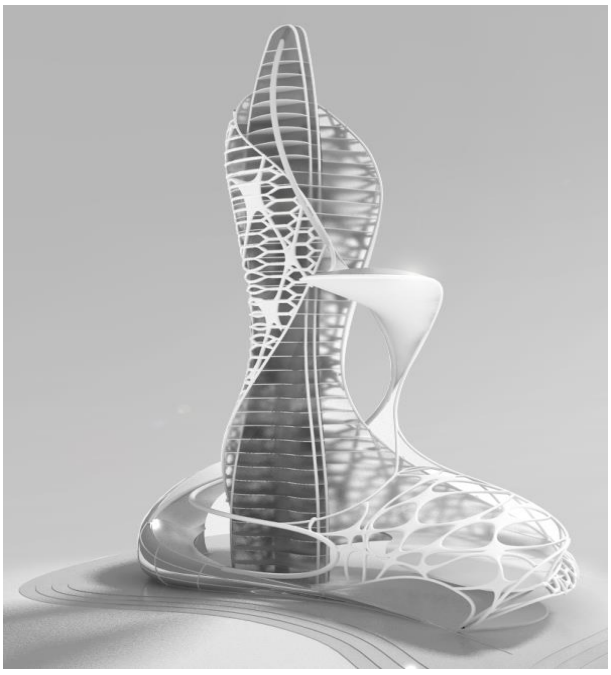


Figure 1.8. Entities created by NURBS.
(http://www.creativecrash.com/3d-model/zenvo-st1-super-car)
(https://team.inria.fr/opale/research/)
(https://www.cgtrader.com/free-3d-models/aircraft/other/vtol-concept)

Today, even though many other forms of more suitable representations exist, such as T-SPLines, polycube SPLines and subdivision surfaces, NURBS still hold a large share of the market. Many platforms exist for NURBS and designers still find it easier to use them. Many handful tools have been developed over the years for them (knot insertion, order elevation, curve fitting, patching). This makes them more attractive and easy-to-use than newer techniques with less industry experience behind them.



Car mesh
(http://engineervsdesigner.com/brad-peebler/)



Skyscraper
(http://archinect.com/potramihai/project/skyscraper-concept-m)

Motorbike
(http://matmoto.com/category/news/page/2)

Figure 1.9. Entities created with subdivision surfaces.

1.3 Isogeometric Analysis

The usual design process requires both exact geometrical representation of the model and accurate engineering results. Unfortunately, computational geometry and finite element method are represented in different file types and are not compatible to each other. The engineer has to create a model for FEM solution and the designer a model for CAD representation. Moreover, the typical design process is not straightforward. The designer produces CAD designs, which are transformed into FEM-compatible forms of representation by the analyst. After generating mesh and obtaining results, the analyst informs the designer of the appropriate changes in geometry. The designer then gives the new CAD model to the engineer, who has to regenerate the FEM model and the new mesh.

This cycle of CAD/CAE interaction can go on multiple times. In complex projects, each design consists of numerous CAD entities combined together and the integration process is estimated to take up at around 80% of the whole design time. Researchers around the world have been trying to achieve automatic CAD/CAE integration.

The main problem is that CAD and FEM, even though they refer to the same object, evolved differently. This incompatibility drove researchers into separate roads, building a wall between the two methods. Thomas J.R. Hughes, a Professor of Aerospace Engineering and Engineering Mechanics at the University of Texas at Austin, came up with a different point of view. Instead of trying to connect present CAD and CAE formulas, we should reinvent them in ways that enable the integration. This is the scope of isogeometric analysis.

The basic idea is to exploit the functions used for the exact geometrical representation in order to describe the solution field. Isogeometric analysis extends, in essence, isoparametric elements, but the process of altering geometry for the sake of the solution approximation is reversed.

This leads to the creation of a single model, capable both of exact representation and analysis. Designers and engineers will be working on the same platform. Time for meshing and entity translation will be eliminated in an instant. This direct contact between analysis and geometry means that every single change can be integrated as soon as it happens, with no risk of errors or timely tasks involved. Most importantly, the designer has to follow the engineer’s perspective and vice versa; the modern designer has to learn how to help the engineer and the modern engineer has to learn the methods the designer is using.

Isogeometric analysis brings together two very different technologies, combining their best points to one. This leads to a better adaptation both from engineers and designers. In order to understand and improve isogeometric analysis, it simply needs to improve its counterparts. Finite element and computational geometry codes need not change drastically. This makes the new technology even more attractive.

Understanding the basics of an innovation and implementation in the daily routine is usually a difficult and time-consuming task.

There are many geometrical forms of representation suitable for analysis, such as NURBS, T-SPLines, polycube SPLines and subdivision surfaces. Each entity has its own advantages and drawbacks, but the variety provided ensures a vast number of alternatives to use, depending on the case. This ensures the generalization of isogeometric method to even more complex geometries.

FEM's shape functions are defined only in the interior of the element. Each element has C^{-1} continuity in the edges. IGA's shape functions are not contained in one element. Most of the times, they are defined through many elements. This ensures a greater continuity and interconnectivity. This different approximation works better and leads to greater convergence than the classical methods.

Refinement by order elevation or knot insertion has always been important for computational geometry. Hierarchical adaptation has been developed for a vast number of entities. All these technologies can be exploited by IGA. Hierarchical structures can be easily developed, straight from the geometrical model. Meshing and refinement is also immediately accomplished.

2 Basic Ingredients of IGA

2.1 Introduction

Until recently, the majority of CAD software users had not realized that by designing a model, they simultaneously created its corresponding mesh of finite elements. This information, although redundant for designers, devoted to computational geometry, is a revolutionary remark for the engineering community.

Before Thomas J.R. Hughes' idea, known as isogeometric analysis (IGA), engineers used to create a new approximate mesh instead of taking advantage of the existing accurate one. The additional geometry error makes the process less accurate, though more time-consuming.

This observation seems now very obvious, but it took years of research until 2003, when Thomas J. R. Hughes and his research team succeeded to cut the Gordian knot of CAD – CAE integration.

The main idea is the elimination of the node mesh in the analysis process. The role and properties of the node mesh are inherited by two separate meshes, obtained directly from the geometrical representation:

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

- The knot mesh, which provides appropriate discretization for numerical integration and boundaries for shape function influence in the model.

For the scope of this thesis, I have worked exclusively with Non-Uniform Rational B-SPLines (NURBS), as they are the most commonly used computational geometry technology.

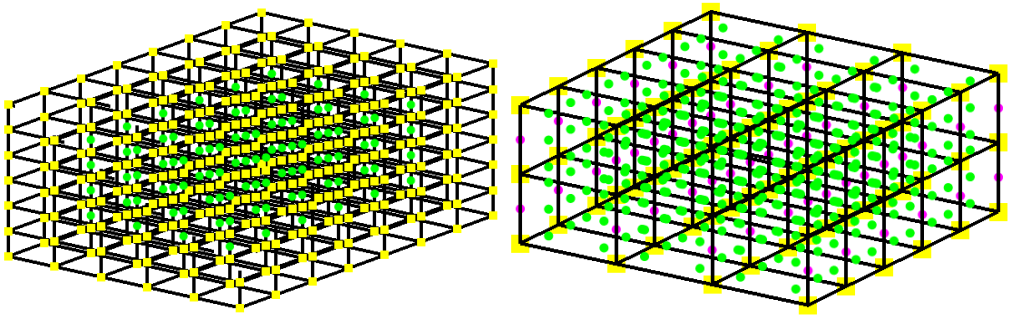
Despite the fact that quite more advanced SPLines have emerged, CAD industry still invests in NURBS. Since 1970, billions of dollars have been directed towards the outspread and evolution of NURBS, establishing them as a common tool for graphic representation around the globe.

Both professionals and amateurs still use NURBS despite their disadvantages, such as difficulties in patch connection and local refinement. The reason for this is that NURBS are not only much more simple in their definition and use, but also able to represent with accuracy smooth curves and all conic sections.

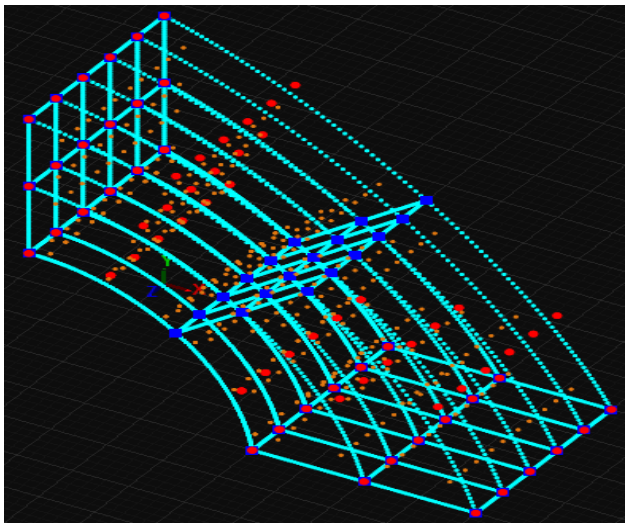
2.2 Index, Parameter and Physical Space

In most occasions, the exact solution of a natural 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 extraordinary geometries is a powerful asset for modern engineers, who are capable of facing surprisingly more complicated problems.

Accurate geometrical representations of the natural model are designed in the familiar Cartesian system, called physical space. Additionally, it is very helpful to envision a complex structure in an imaginary, basic space, where all geometries can be represented as lines, rectangles and cuboids. This is the parameter space. This approach is far from new; it is already known from the isoparametric concept in Finite Element Methods. The parameter space utilized in isogeometric analysis, however, holds some major differences. Furthermore, isogeometric analysis also introduces the index space. This additional space plays an important role for some kinds of SPLines, but it is only auxiliary for NURBS.



(a) **(b)**



(c)

Figure 2.1. B-SPLine solid.
(a) Index space.
(b) Parameter space.
(c) Physical space.

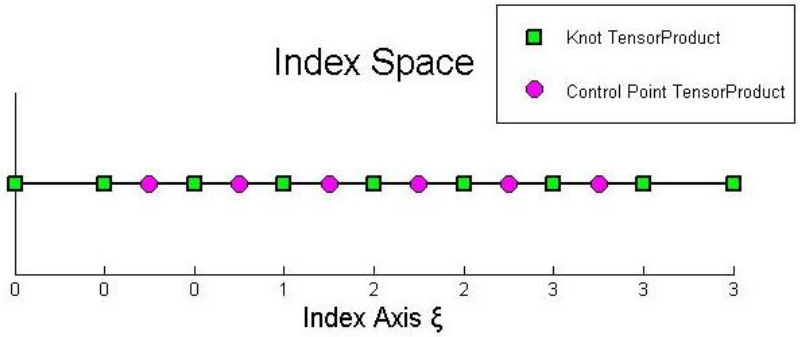
2.2.1 Index Space

Index space is a representation of the model with respect to knot values. It is a line in 1D, containing the corresponding knot values in equally spaced positions. This space focuses upon the sequence of knot values rather than their actual numerical content.

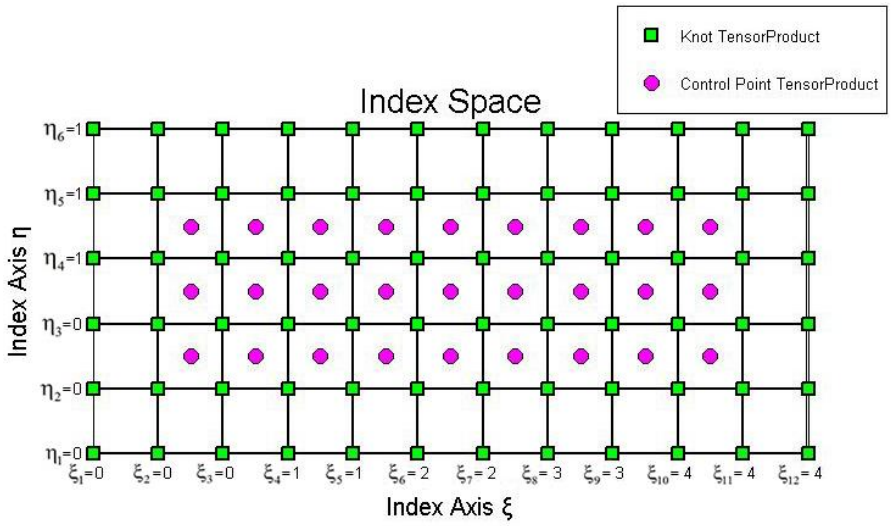
Index space describes the contribution of each knot value to the creation of a certain B-SPLine basis function. This helps identify the level of interconnection between basis functions and the knot value support of each function.

Control points are also evaluated in the index space. In fact, control points are defined as the center of the support of knot value spans.

Expansion to 2D or 3D leads to the creation of rectangles or cuboids respectively. Due to tensor product properties, everything mentioned about 1D extends and applies to both 2D and 3D. Thus, index space provides information that can contribute to the comprehension of a complex representation.



(a) Curve



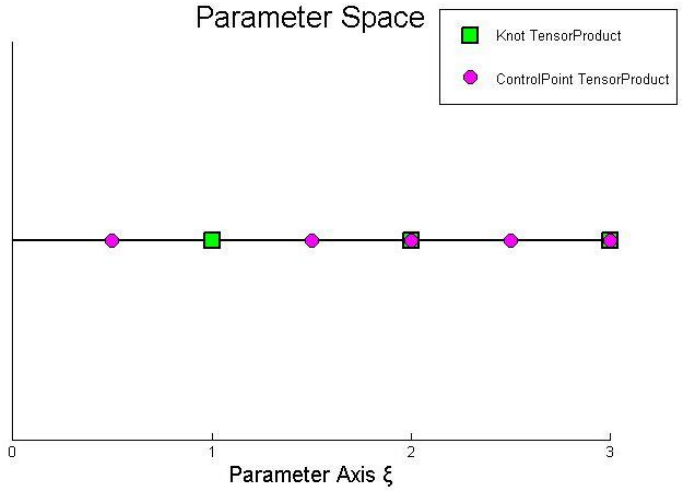
(b) Surface

Figure 2.2. Curve and surface represented in index space.

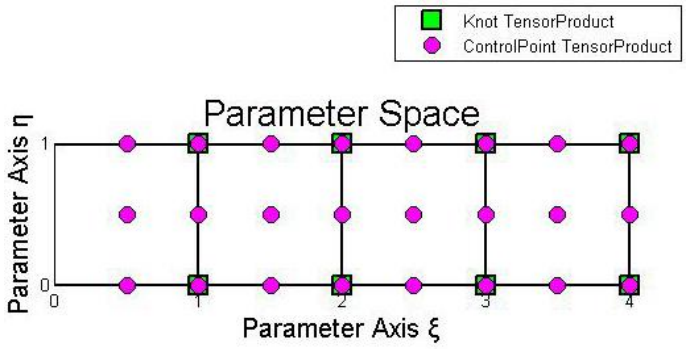
2.2.2 Parameter Space

Parameter space is a representation of the model with respect to knots. SPLine entities are always represented as orthogonal shapes in parameter space. Only lines, rectangles and cuboids exist here. In order to transform those simple patterns to virtually unlimited, complex geometries, the application of a mapping from parameter to physical space is required. Hence, parameter space is a primitive, abstract representation of physical space. The mapping between parameter space and physical space is achieved through the jacobian matrix and its inverse. This is something widely utilized in FEM as well.

The illustration of basis functions in the parameter space allows for a better understanding of concepts such as support, control point coordinates and the role of knots in basis function creation. Each knot marks the beginning and the end of a basis function domain. By "domain" we mean the area in which the basis function is non-zero, as all basis functions are defined throughout the parameter space, but are non-zero only in specific knot spans. Basis functions sharing the same domain are overlapping in parameter space and controlling a common part of the entity in the physical space.



(a) Curve



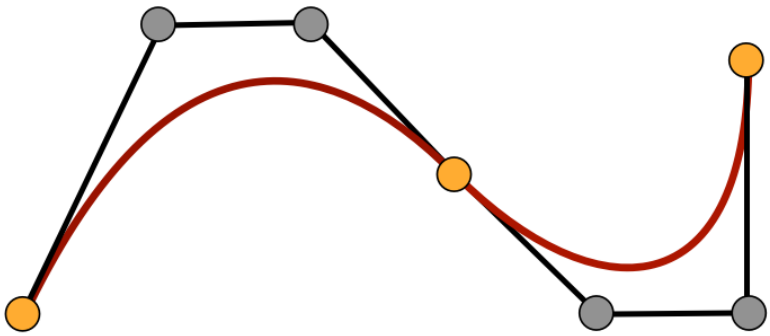
(b) Surface

Figure 2.3. Curve and surface represented in parameter space.

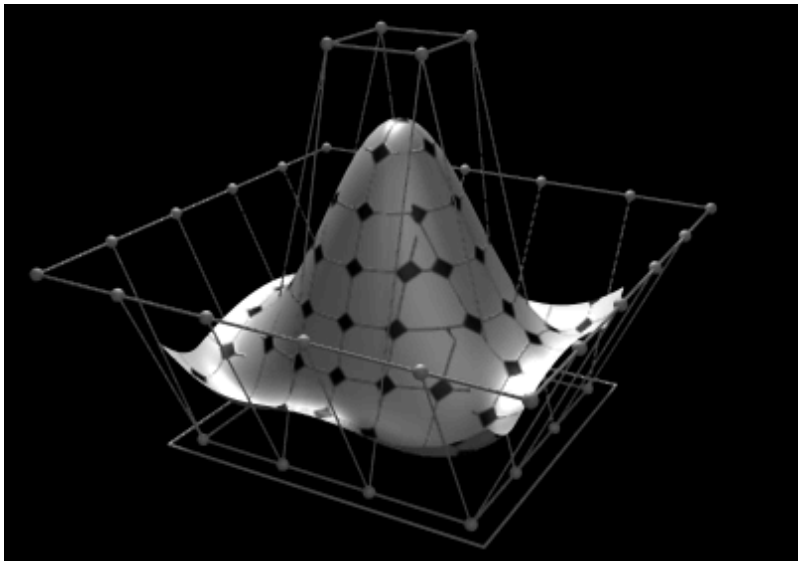
2.2.3 Physical Space

Physical space is the already known Cartesian space, where the real model is represented. Simple orthogonal shapes from parameter space are transformed into complex entities in the physical space. Physical coordinates of the control points play a major role in the aforementioned mapping, but an equally drastic role is set upon basis functions. In fact, for a given set of control points, only a single set of basis functions can lead to the same geometry. We will examine this thoroughly later.

Control points can often be seen outside the model in physical space in contrast to FEM's nodes, which always belong to the mesh. It is one of the reasons NURBS and SPLine entities in general can accurately represent multiple types of geometries and the understanding of this peculiarity is one of the many challenges of isogeometric analysis.



(a) Curve.
(http://graphics.cs.ucdavis.edu/~joy/ecs178/Unit-4/Unit4.html)



(b) Surface.
(https://commons.wikimedia.org/wiki/File:NURBS_3-D_surface_frame.png)

Figure 2.4. Curve and surface represented in physical space.

2.3 B-SPLine Geometries

2.3.1 B-SPLine Basis Functions

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 B-SPLine basis functions at $\xi \in \left[\xi_1, \xi_{n+p+1} \right]$ using the Cox-de Boor recursive formula [1].

First, for degree $p = 0$ (piecewise constant, Box B-SPLine), we have that:

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

The piecewise constant does not include the right edge ξ_{i+1} in order to ensure partition of unity, as the next basis function begins at that edge.

The last function, however, includes both left and right edge, in order to be defined for the whole knot span.

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

Afterwards, for degree $p = 1, 2, \dots$:

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

with the assumption of:

$$\frac{0}{0} \doteq 0$$

We keep the same symbols with [1], which is quite popular, because we want the reader to browse through this thesis conveniently.

In the software implementation of this method, however, we discourage the use of symbolizations and instead represent the variables with full names.

2.3.2 Knots as Boundaries of Basis Functions

Knots define the boundaries of the model's basis functions. They represent "switches", which turn "on" or "off" a certain piece of a B-SPLine basis function. In order to acquire the knots and the basis functions, a knot vector must be defined.

A knot vector Ξ is usually defined in bibliography as a set of coordinates ξ_i, with $\xi_i \leq \xi_{i+1}$. It can contain the same number multiple times and generates the basis in a unique way.

In order to improve efficiency and communication between members of GiGA Team, we define as:

- "knot value vector": the whole set of non-decreasing coordinates ("knot values")
- "knot vector": the set of unique coordinates ("knots")

For example, a knot value vector could be:

$$\{0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3\}$$

where $0, 1, 1, 1, 2, 2, 3, 3$ are the separate knot values.

The corresponding knot vector is:

$$\{0 \quad 1 \quad 2 \quad 3\}$$

where $0, 1, 2, 3$ are the separate knots.

Let p be the polynomial degree of the basis function. If the first and the last knots are repeated $p+1$ times, the knot value vector is considered "Open", because it has C^{-1} continuity on the edges, creating an open curve that is interpolatory at these points. If knot values are equally spaced, the knot value vector is considered "uniform". In CAD community, non-uniform, open knot value vectors are widely used. An example of such a knot value vector with $p = 2$ is:

$$\{0 \quad 0 \quad 0 \quad 0.5 \quad 1 \quad 1.5 \quad 2 \quad 2.5 \quad 3 \quad 3 \quad 3\}$$

A knot value vector may contain integers or decimals. In fact, the actual numerical content of knot values is of no importance. What matters is the relative distance between them. This means a knot value vector can be multiplied by any number, or have a number added to every knot value and the resulting basis would still be the same.

In GiGA Team, we generally prefer to use knot value vectors that start from 0 and span by 1, as it is more convenient for the human mind to use an integer system.

2.3.3 Control Points as the Center of the Support

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

For even degrees, the center of the support in the index space lies between two sequential knot values, $i+\frac{p}{2}$ and $i+\frac{p}{2}+1$. As a result, the coordinate of the control point is defined as the average of these knot values.

$$\xi_{CP} = 0.5 \cdot \left(\xi_{i+\frac{p}{2}} + \xi_{i+\frac{p}{2}+1} \right)$$

which means that a control point of even degree can either be on a knot, or in the middle of a knot span.

For odd degrees, the center of the support is the knot value $i+\frac{p+1}{2}$. Therefore, for odd degrees, control points are always coincident with knots.

2.3.4 Full Tensor Product Nature

B-SPLine basis functions are of full-tensor product nature. Consequently, it is easy to combine B-SPLines across different directions, in order to evaluate a multi-directional B-SPLine shape function.

2D B-SPLine shape functions can be evaluated as tensor product of basis functions $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$.

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

3D B-SPLine shape functions are a tensor product of basis functions in three directions, $N_{i,p}(\xi)$, $M_{j,q}(\eta)$ and $L_{k,r}(\zeta)$.

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

It is understood that all properties of B-SPLine basis functions in one direction are inherited by the multi-directional shape functions.
As a result, thorough adaptation to one-directional B-SPLine basis function properties and techniques is very important for understanding multi-directional complex geometries.

2.3.5 B-SPLine Basis Function Properties

According to "The NURBS Book" [3], B-SPLine basis functions possess the following important 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 that produces n functions of order p has $n+p+1$ knot values.

8. Every B-SPLine basis function shares support with $2p$ B-SPLines.

2.3.5.1 Local Support

Local support means that basis functions are non-zero only in certain knot spans in parameter space. This can be expressed by:

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

Local support is a result of the recursive character of B-SPLines. For the creation of a B-SPLine function of degree p, two consecutive B-SPLine functions of order $p-1$ are used. For the creation of those consecutive basis functions, three consecutive functions of order $p-2$ are needed. Inductively, $p+1$ consecutive box basis functions are required.

Each box function has a support of one knot value 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 value spans.

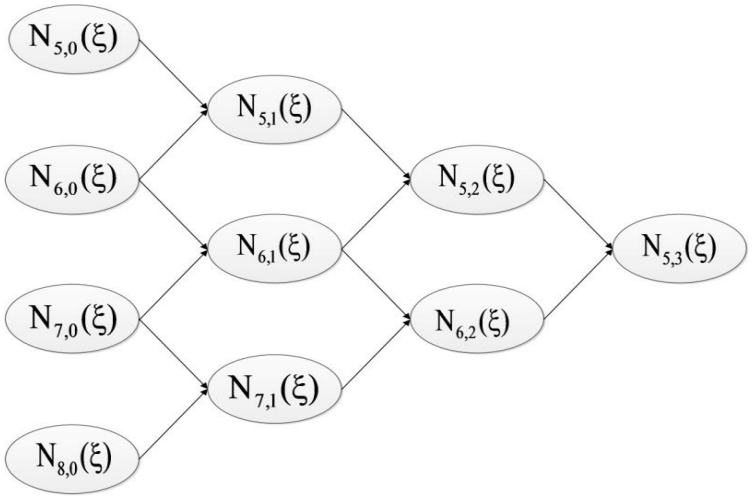


Figure 2.5. Lower-order basis functions required for the creation of $N_{5,3}(\xi)$.

In **Figure 2.5**, the recursive character of B-SPLines is represented. The box functions, drawn in red, are required in order to build the linear basis functions, drawn in green. Linear functions are combined for the evaluation of quadratic $(p=2)$ basis functions. Bear in mind that some box and linear functions are zero across the entire domain, but still contribute to the evaluation of the next-order B-SPLines.

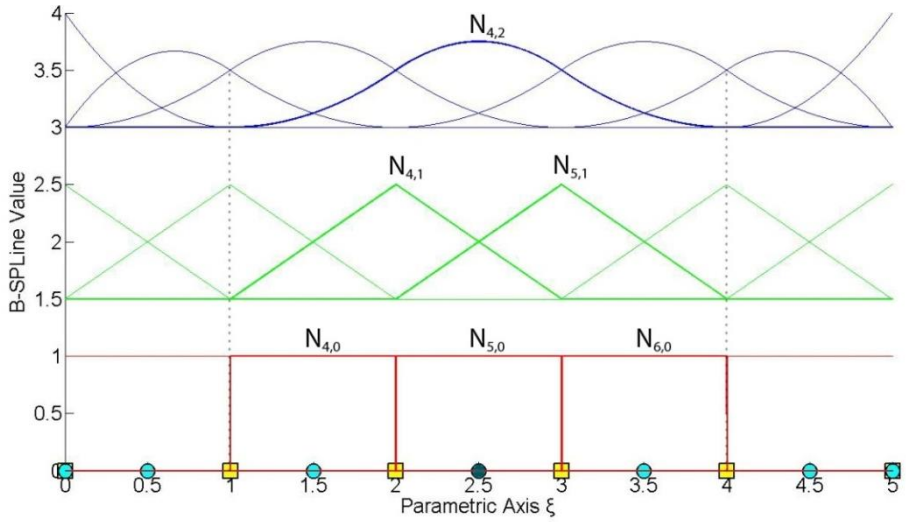


Figure 2.6. B-SPLine recursive character.
Basis functions required for the evaluation of a quadratic B-SPLine function.

For example, $N_{4,2}(\xi)$ in **Figure 2.6** is created from $N_{4,1}(\xi)$ and $N_{5,1}(\xi)$. They, 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 $N_{4,2}(\xi)$.

In a similar fashion, a support can be defined with respect to knot values contributing to the creation of a B-SPLine basis function. These are the $p+2$ knot values that are contained in the knot value span support of the function. Both knot value span support and knot value support are necessary for the understanding of the isogeometric method.

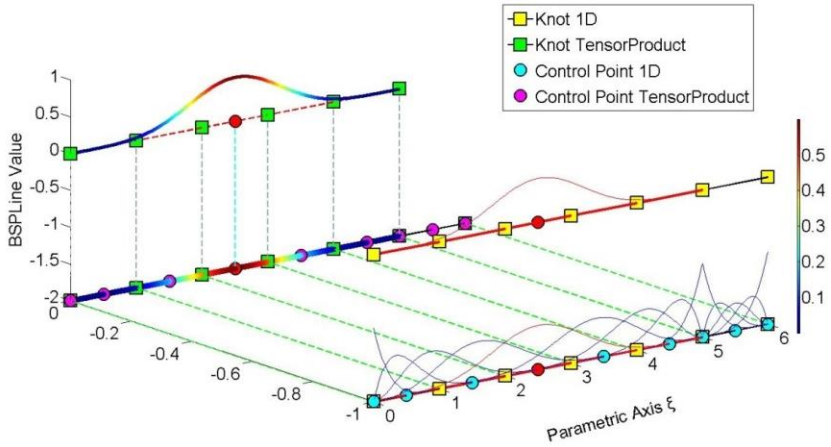


Figure 2.7. Quadric B-SPLine basis functions (C^3/ C^0-continuity).
$\Xi=\{0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 5\ \ 5\ \ 5\ \ 6\ \ 6\ \ 6\ \ 6\ \ 6\}$
1D shape function $N_{5,4}(\xi)$.

In **Figure 2.7**, the degree is $p=4$, so each basis function has a support of $p+1=4+1=5$ knot value spans. The selected function $N_{5,4}(\xi)$, drawn in red, is non-zero only in 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 $\{0,1,2,3,4,5\}$ shown in green. There are no trivial spans in this B-SPLine function, so it is considered a fully developed basis function.

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

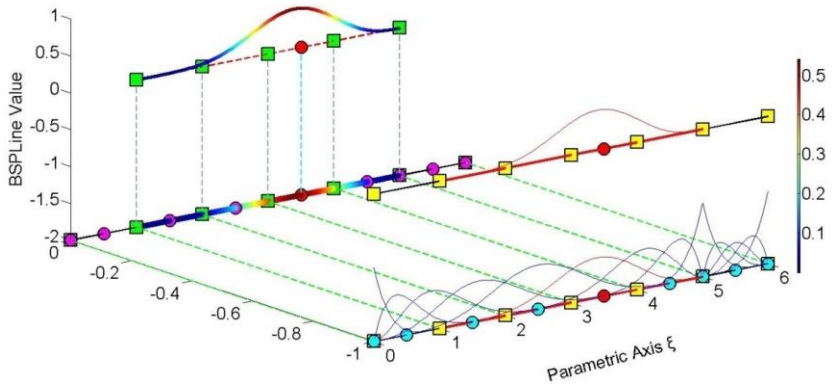


Figure 2.8. 1D shape function $N_{6,4}(\xi)$.
$\Xi=\{0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 5\ \ 5\ \ 5\ \ 6\ \ 6\ \ 6\ \ 6\}$

In **Figure 2.8**, $N_{6,4}(\xi)$ is highlighted. This function is non-zero in the knot value spans $[1,2)$, $[2,3)$, $[3,4)$, $[4,5)$ and $[5,5)$. The corresponding knot values shown in green are $\{1,2,3,4,5,5\}$. The support is five knot value spans, but this leads to only four knot spans, as a trivial span is contained.

Observe the next basis functions in this example; as more trivial spans are contained, the knot span support of each function is reduced. Still, the knot value span support and the corresponding knot value support follow the rules already mentioned.

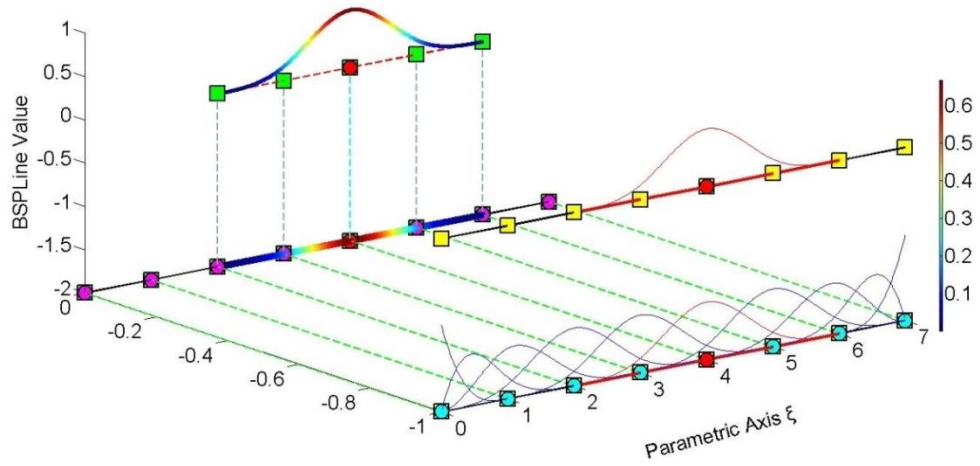


Figure 2.9. B-SPLine basis functions for knot value vector.
$\Xi = \{0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 7\ \ 7\ \ 7\}$.
1D shape function $N_{6,3}(\xi)$ **.**

In the above figure, the polynomial degree of the basis functions is $p = 3$. Knot value span support is $p+1 = 4$ knot value spans, which requires knot value support of $p+2 = 5$ knot values.

$N_{6,3}(\xi)$ is non-zero in the knot value spans $[2,3)$, $[3,4)$, $[4,5)$ and $[5,6)$. No trivial spans are involved, so this is another fully developed B-SPLine. The knot value support consists of the knot values $\{2,3,4,5,6\}$.

B-SPLine tensor product properties enable the immediate expansion of 1D properties to 2D and 3D B-SPLine shape functions.

$N_{3,2}(\xi)$ in **Figure 2.10** has a support of 3 knot value spans per ξ, $[0,1)$, $[1,2)$ and $[2,3)$. It is created by knot values $\{0,1,2,3\}$.

$M_{4,2}(\eta)$ has a support of 3 knot value spans per η, $[1,2)$, $[2,3)$ and $[3,4)$. It is created by knot values $\{1,2,3,4\}$.

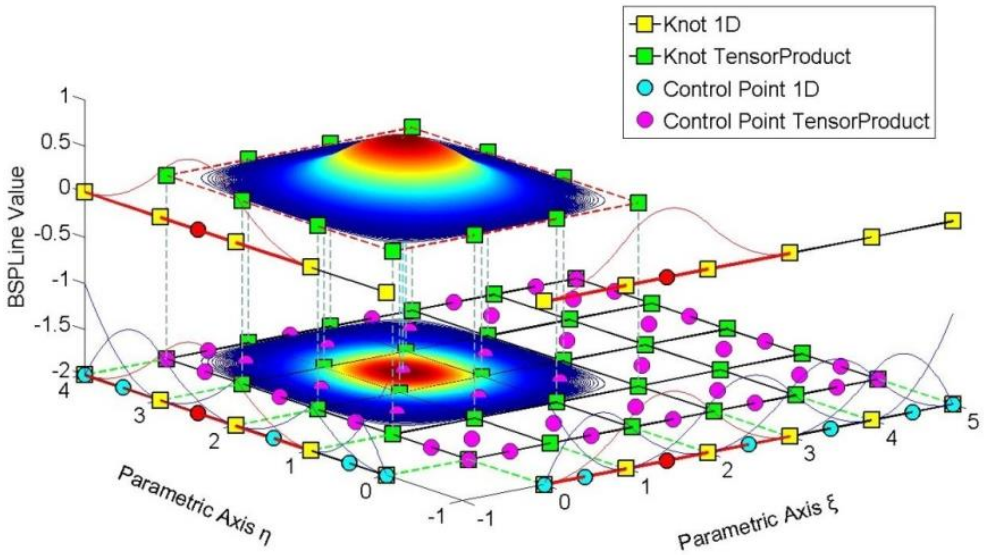


Figure 2.10. Shape function $R_{i,j}^{p,q} = R_{3,4}^{2,2}(\xi,\eta)$ as a tensor product of $N_{3,2}(\xi)$ and $M_{4,2}(\eta)$.

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

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

The 2D shape function $R_{3,4}^{2,2}(\xi,\eta)$ is the full tensor product of $N_{3,2}(\xi)$ and $M_{4,2}(\eta)$. The support of the shape function is a rectangle created from the supports of the respective basis functions. It has a total area of $3 \cdot 3 = 9$ knot value rectangles.

Observe that the value of the bidirectional B-SPLine is represented both in the third axis of the graph and by projection of the contour in the 2D plane. This is useful for representation of 3D basis functions.

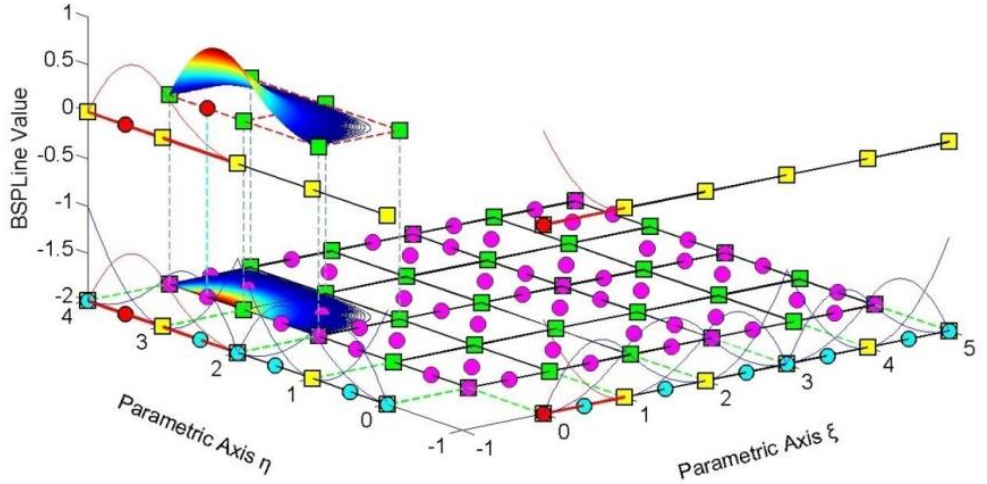


Figure 2.11. Shape function $R_{i,j}^{p,q} = R_{1,6}^{2,2}(\xi,\eta)$ as a tensor product of $N_{1,2}(\xi)$ and $M_{6,2}(\eta)$.

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

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

Both 1D basis functions have reduced continuity in **Figure 2.11**.

$N_{1,2}(\xi)$ has a knot value span support of $[0,0)$, $[0,0)$ and $[0,1)$, leading to a support of a single knot span, $[0,1)$. The corresponding knot value support is $\{0,0,0,1\}$.

$M_{6,2}(\eta)$ has a knot value support of $\{2,3,4,4\}$, which leads to a knot value span support of $[2,3)$, $[3,4)$, $[4,4)$. One trivial and two non-trivial spans are contained.

Support for the shape function $R_{1,6}^{2,2}(\xi,\eta)$ is defined as the tensor product of the supports of the basis functions, namely $1\cdot 2=2$ knot rectangles.

3D B-SPLine shape functions can be represented as in the **Figure 2.12**.

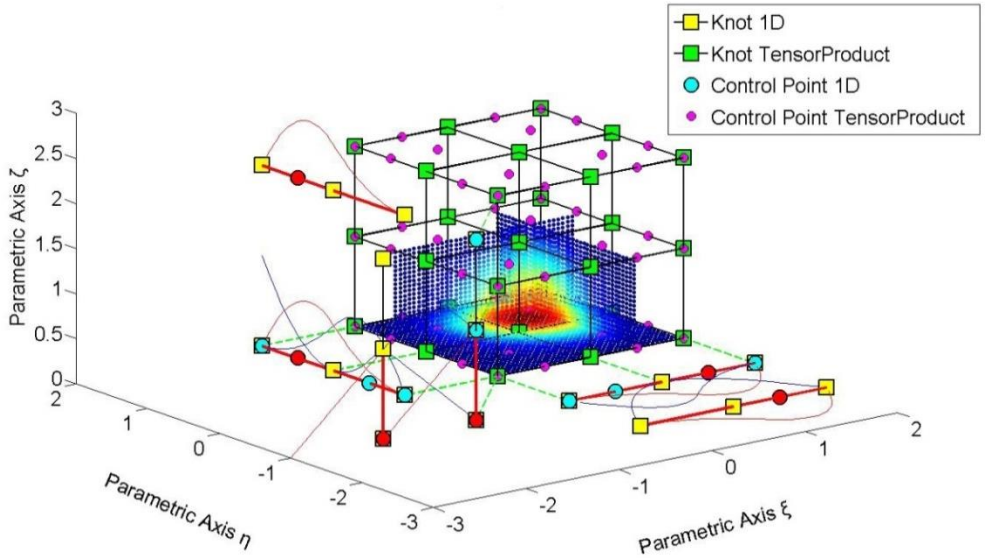


Figure 2.12. Shape function $R_{i,j,k}^{p,q,r}=R_{3,3,1}^{2,2,1}(\xi,\eta,\zeta)$ as tensor product of $N_{3,2}(\xi)$, $M_{3,2}(\eta)$, $L_{1,1}(\zeta)$.

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

The three axes represent the three parametric directions ξ, η, ζ. B-SPLine basis functions for ξ are drawn in the $\xi-\eta$ plane, functions for η in the $\eta-\zeta$ plane and functions for ζ in the $\zeta-\xi$ plane.

$N_{3,2}(\xi)$ and $M_{3,2}(\eta)$ both have a support of 3 knot value spans, $[0,1)$, $[1,2)$ and $[2,2)$ in their respective directions. Knot value support for those functions is $\{0,1,2,2\}$. Knot span support is restrained at 2 knot spans for each function.

$L_{1,1}(\zeta)$ has a knot value span support of $[0,0)$ and $[0,1)$, leading to the support of 1 knot span. Knot value support is defined as $\{0,0,1\}$.

Their tensor product value has been calculated as a function of two parametric directions at the control point coordinate of the remaining direction. The resulting contour is projected on a plane that is parallel to the two directions and intersects with the tensor product control point.

This process is repeated for all three possible combinations, thus creating contours at $\xi-\eta$, $\eta-\zeta$ and $\zeta-\xi$ planes.

For example, control point $(3,3,1)$ has coordinates $(\xi,\eta,\zeta)=(1.5,1.5,0)$.

$R^{2,2,1}_{3,3,1}(\xi,\eta,0)$ has been calculated and projected in $\zeta=0$ plane. The support of the projection shows the support of the 3D shape function per ξ, η, namely $2\cdot 2=4$ knot cells.

$R^{2,2,1}_{3,3,1}(\xi,1.5,\zeta)$ is represented in the plane $\eta=1.5$. Support of the 3D shape function per ξ, ζ is $2\cdot 1=2$ knot cells.

$R^{2,2,1}_{3,3,1}(1.5,\eta,\zeta)$ is projected in $\xi=1.5$ plane. The support of the 3D shape function per η, ζ is $2\cdot 1=2$ knot cells.

The support of the 3D shape function across the entire domain is the tensor product of the 1D supports. In this particular case, it is $2\cdot 2\cdot 1=4$ knot cuboids. The support is represented by the tensor product of the projections in **Figure 2.12**.

2.3.5.2 Maximum Number of Non-Zero Functions per Knot Span

A box function that is non-zero in one knot span contributes to the evaluation of 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 $p+1$ B-SPLine basis functions of order p.

It applies from Cox de Boor recursive formula that only one box function is non-zero across a selected knot span. As a result, only the corresponding $p+1$ B-SPLine 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 in stiffness matrix formulation, in order to reduce computational cost.

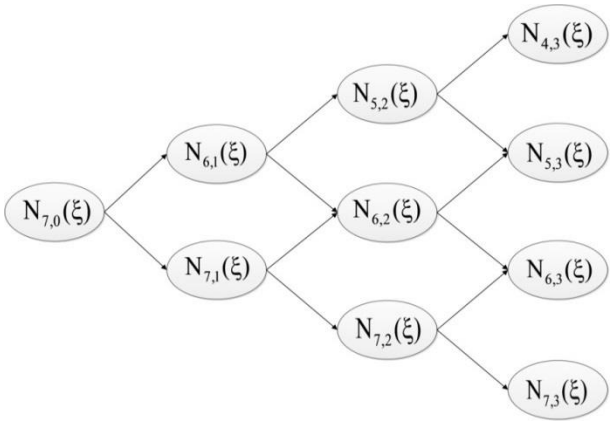


Figure 2.13. Contribution of box function to the creation of higher-order B-SPLine basis functions.

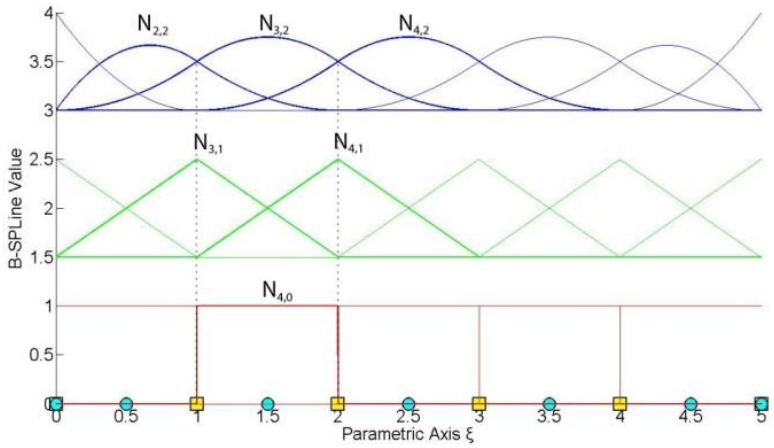


Figure 2.14. Contribution of one box function to non-zero higher-order B-SPLine basis functions across knot span $[1,2)$.

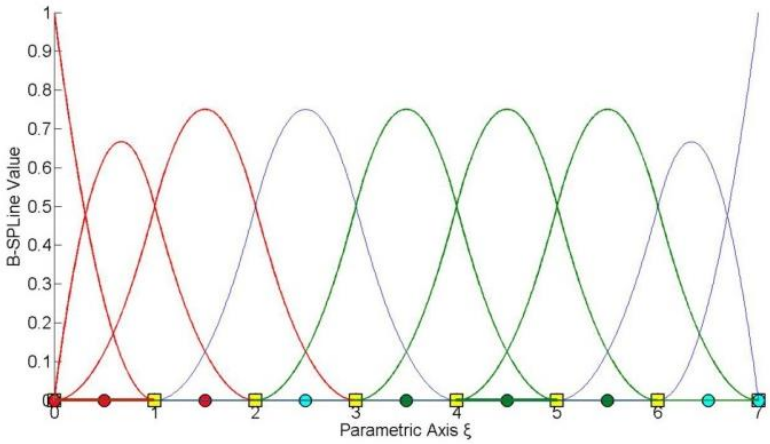


Figure 2.15. B-SPLine basis functions.
$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 7 \quad 7\}$.
Basis functions, that are non-zero in knot span [0,1), are drawn in red.
Basis functions, that are non-zero in knot span [4,5), are drawn in green.

In **Figure 2.15**, two separate knot spans are examined. For every knot span, $p+1=3$ basis functions are non-zero.

For the first knot span, $[0,1)=[\xi_3,\xi_4)$, basis functions $N_{1,p}(\xi)$, $N_{2,p}(\xi)$, $N_{3,p}(\xi)$ are non-zero.

For the second knot span, $[4,5)=[\xi_7,\xi_8)$, basis functions $N_{5,p}(\xi)$, $N_{6,p}(\xi)$, $N_{7,p}(\xi)$ are non-zero.

As a result, in 2D, at a given knot rectangle, only the tensor products of the respective non-zero basis functions are non-zero, namely $(p+1)\cdot(q+1)$ shape functions.

Inductively, in 3D, at a given knot cuboid, the tensor products of the corresponding non-zero basis functions create $(p+1)\cdot(q+1)\cdot(r+1)$ non-zero shape functions.

2.3.5.3 Non-negativity

It has been established that:

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

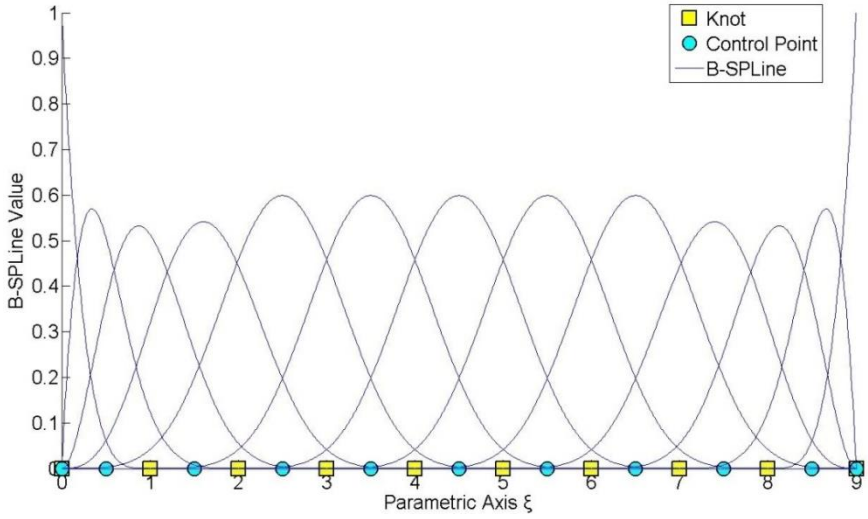


Figure 2.16. Quadric B-SPLine basis functions.
$\Xi=\{0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ \ 9\ \ 9\ \ 9\ \ 9\}$

The degree in **Figure 2.16** is $p=4$. As we see in the picture above, all the basis functions are positive for every ξ, i. This property is very important for isogeometric analysis and it does not apply in the classical shape functions of finite element analysis.

Naturally, it is easier for the human mind to work with only positive values, so this attribute simplifies and encourages the understanding of B-SPLines. Non-negativity applies for 2D and 3D shape functions as well.

2.3.5.4 Partition of Unity

Partition of unity is expressed by

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

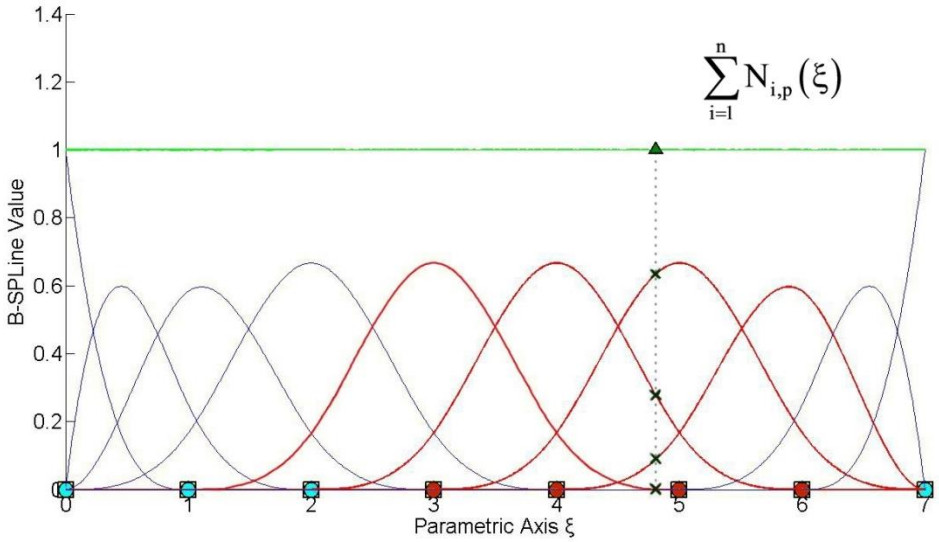


Figure 2.17. B-SPLine basis functions. Partition of unity.
$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 7 \quad 7 \quad 7\}$
Sum of B-SPLine function value at every ξ.

The degree for the B-SPLine basis functions in **Figure 2.17** is $p = 3$. The green horizontal line represents the sum of B-SPLine values at the corresponding ξ, which, of course, is equal to 1 for every ξ.

For example, B-SPLine values have been evaluated for $\xi = 4.81$, with the following results for the four non-zero basis functions.

$$N_{5,3}(4.81) = 0.0011$$
$$N_{6,3}(4.81) = 0.2763$$
$$N_{7,3}(4.81) = 0.6340$$
$$N_{8,3}(4.81) = 0.0886$$

The above yields:

$$\sum_{i=1}^{10} N_{i,3}(4.81) = 1$$

as expected.

Partition of unity applies for multi-dimensional shape functions as well. In 2D, partition of unity is expressed as $\sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q} = \sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\xi)\cdot M_{j,q}(\eta) = 1$ which is a result of tensor product nature [1].

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

In full analogy, 3D shape functions also possess partition of unity.

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

Partition of unity is also a property of great importance for the shape functions that are used in finite element analysis.

2.3.5.5 C^{p-m} Continuity

At a knot with multiplicity m, $N_{i,p}(\xi)$ produces $p-m$ continuous derivatives or, in other words, it has C^{p-m} continuity. Continuity less than C^0 is not acceptable for internal knots, which means they can be repeated up to p times. Bear in mind that as continuity decreases, B-SPLine basis functions tend to be more "steep".

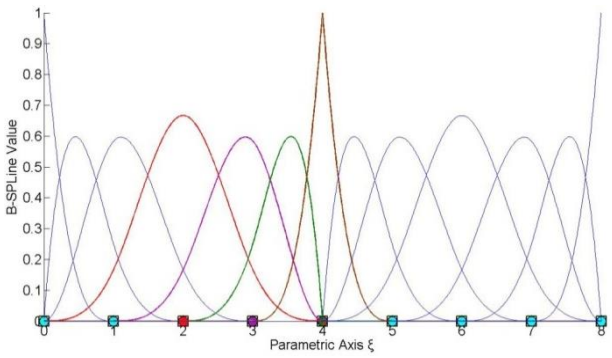


Figure 2.18. B-SPLine basis functions for knot value vector.
$\Xi = \{0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 4\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 8\ \ 8\ \ 8\}$

The polynomial order in **Figure 2.18** is $p = 3$. Continuity for a basis function is affected only by the corresponding knot values. Knot $\xi = 4$ has multiplicity $m = 3$ with respect to the knot vector. Its multiplicity with respect to the basis functions depends on each function's knot value support.

$N_{4,3}(\xi)$, a fully developed basis function, drawn in red, has a knot value support of $\{0,1,2,3,4\}$. The knot $\xi = 4$ has multiplicity $m = 1$ with respect to the basis function $N_{4,3}(\xi)$, so this function is $C^{p-m} = C^{3-1} = C^2$ continuous in the entire domain.

The purple basis function $N_{5,3}(\xi)$ has a knot value support of $\{1,2,3,4,4\}$. The knot in question has multiplicity $m=2$ with respect to the basis function, which makes $N_{5,3}(\xi)$ $C^{p-m}=C^{3-2}=C^{1}$ continuous across $\xi=4$. 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. The multiplicity of the knot is $m=3$ for both basis functions. Therefore, these functions have $C^{p-m}=C^{3-3}=C^{0}$ continuity across $\xi=4$.

Observe that when a B-SPLine basis function $N_{j,p}(\xi)$ has a knot ξ_m with multiplicity p in the center of the knot value support, it applies that $N_{j,p}(\xi_m)=1$.

In 2D and 3D cases, continuity per direction is obtained straight from the continuity of the corresponding one-directional B-SPLine basis functions.

In **Figure 2.19.a**, $R_{3,4}^{2,2}(\xi,\eta)$ is tensor product of $N_{3,2}(\xi)$ and $M_{4,2}(\eta)$. $N_{3,2}(\xi)$ has $C^{p-m}=C^{2-1}=C^{1}$ continuity across $\xi=3$ and $M_{4,2}(\eta)$ has $C^{p-m}=C^{2-2}=C^{0}$ continuity across $\eta=2$. Therefore, $R_{3,4}^{2,2}(\xi,\eta)$ has C^{1} continuity with respect to ξ and C^{0} continuity with respect to η across $(\xi,\eta)=(3,2)$.

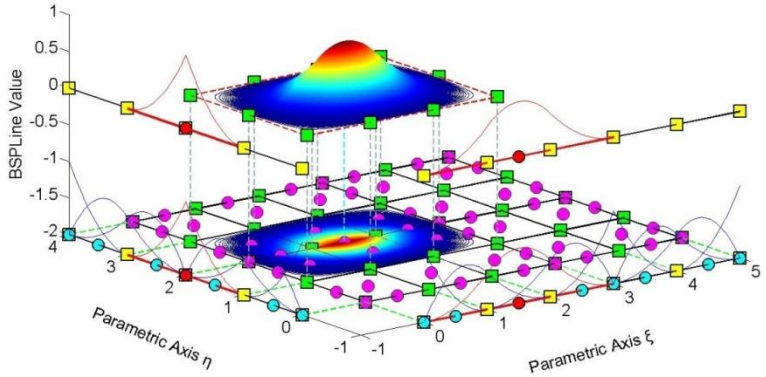
In **Figure 2.19.b**, $R_{5,4}^{2,2}(\xi,\eta)$ is tensor product of $N_{5,2}(\xi)$ and $M_{4,2}(\eta)$. Both B-SPLine basis functions have C^{0} continuity across $\xi=3$, $\eta=2$ respectively. As a result, $R_{5,4}^{2,2}(\xi,\eta)$ has C^{0} continuity across $(\xi,\eta)=(3,2)$. This particular case of C^{0} continuity for both directions will be utilized in future applications.

Note that C^{0} continuity per both directions leads to:

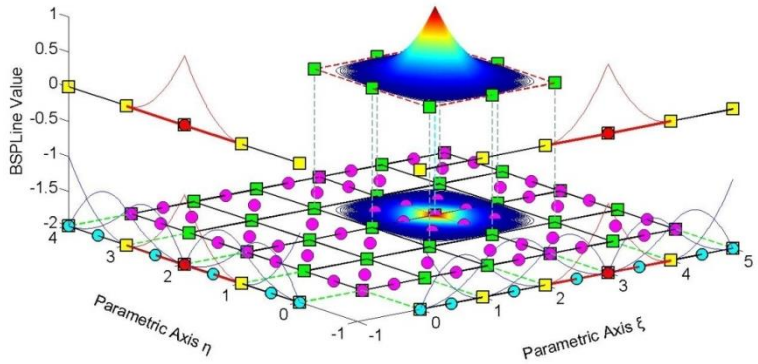
$$R_{5,4}^{2,2}(3,2)=N_{5.2}(3)\cdot M_{4.2}(2)=1$$

Non-negativity & partition of unity require that:

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



(a) **Shape function $R_{i,j}^{p,q}=R_{3,4}^{2,2}(\xi,\eta)$ as tensor product of $N_{3,2}(\xi)$ and $M_{4,2}(\eta)$.**



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

Figure 2.19. B-SPLine basis functions for directions ξ, η.
$\Xi = \{0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 3\ \ 3\ \ 4\ \ 5\ \ 5\ \ 5\}$, $H = \{0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 2\ \ 3\ \ 4\ \ 4\ \ 4\}$

Figure 2.20 represents $R_{4,3,3}^{2,2,2}(\xi,\eta,\zeta)$, a shape function formed as a tensor product of three C^0 continuous functions across $(2,1,1)$. The corresponding basis functions are $N_{4,2}(\xi)$, $M_{3,2}(\eta)$ and $L_{3,2}(\zeta)$. Naturally, at the point of C^0 continuity for all three directions, it applies that $R_{4,3,3}^{2,2,2}(2,1,1) = N_{4,2}(2)\cdot M_{3,2}(1)\cdot L_{3,2}(1) = 1$.

Observe that all functions of reduced continuity tend to be more "steep". This happens because they contain multiple knot values of the same knot, and therefore develop across trivial spans. This "steepness" is the first indication that a basis function has reduced continuity. Moreover, internal C^0 continuity produced exactly the same B-SPLine basis functions as C^{-1} continuity on the edge of the knot vector. This means that at every knot with multiplicity $m = p$, only one function remains non-zero. Due to partition of unity, that function's value at that knot is equal to 1.

In multi-dimensional functions, C^0 continuity across a knot requires the basis functions per all directions to be C^0 continuous at that point. In this case, the value of the multi-dimensional shape function across this knot is equal to 1.

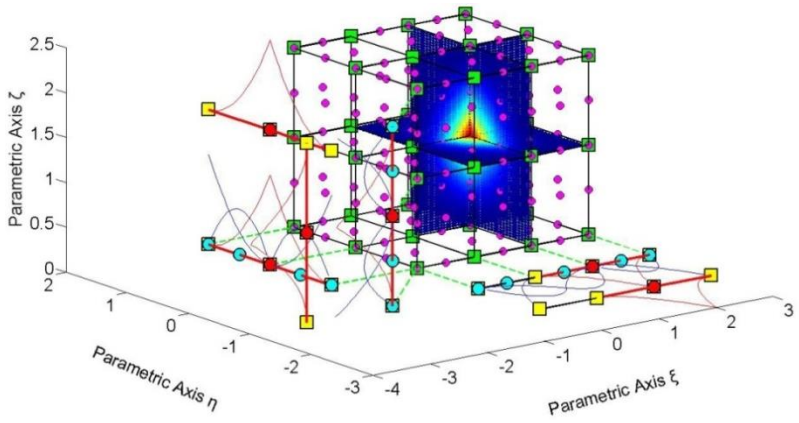


Figure 2.20. Shape function $R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) = R_{4,3,3}^{2,2,2}(\xi,\eta,\zeta)$.
$\Xi = \{0\ \ 0\ \ 0\ \ 1\ \ 2\ \ 2\ \ 3\ \ 3\ \ 3\}$, $H = Z = \{0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 2\ \ 2\ \ 2\}$

2.3.5.6 Linear Independence

Given a finite number of distinct vectors $\{u_1 \quad u_2 \quad ... \quad u_n\}$ and scalars $\{a_1 \quad a_2 \quad ... \quad a_n\}$, the subset S of a vector space V is called linearly independent if the equation $a_1u_1 + a_2u_2 + ... + a_nu_n = 0$ leads to the unique solution $a_1 = a_2 = ... = a_n = 0$. Thus, the subset is linearly independent if a linear combination of the vectors is the zero vector, only for $a_1 = a_2 = ... = a_n = 0$. The linear independence in isogeometric analysis applies for the basis functions $\{N_{1,p}(\xi) \quad N_{2,p}(\xi) \quad ... \quad N_{n,p}(\xi)\}$

The equation

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

leads to $a_1 = a_2 = ... = a_n = 0$. We can reach the conclusion that 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. Some 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 B-SPLine 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 isogeometric analysis, $\dim(V)$ is equal to the number of control points and, consequently, the number of basis functions used.

- Let there be a mass of vectors $\{u_1 \quad u_2 \quad ... \quad u_n\}$, which form the basis of a vector space with $\dim(V) = n$ and the numbers $\{a_1 \quad a_2 \quad ... \quad a_n\}$, called the coordinates of u. In isogeometric analysis, the basis of the vector space is the set of B-SPLine basis functions $\{N_{1,p}(\xi) \quad N_{2,p}(\xi) \quad ... \quad N_{n,p}(\xi)\}$ and the numbers are the coordinates $\{X_1 \quad X_2 \quad ... \quad X_n\}$ 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 in 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.

The vector space V in $\mathbb{R}$ is a mass of vectors with the properties below:

1. Commutativity of addition

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

In isogeometric analysis, 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 isogeometric analysis,

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

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 isogeometric analysis, 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 isogeometric analysis 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 isogeometric analysis:

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

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 isogeometric analysis, this equation applies as:

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

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

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

which is implemented in isogeometric analysis as:

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

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

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

Its equivalent in isogeometric analysis is:

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

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 $C(\xi)$. 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, it is understood that for a specific set of basis functions, only one set of control points can yield the appropriate geometry. If one wants to change the basis functions, the control points have to be shifted accordingly.

2.3.5.7 Fixed Number of Knot Values

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

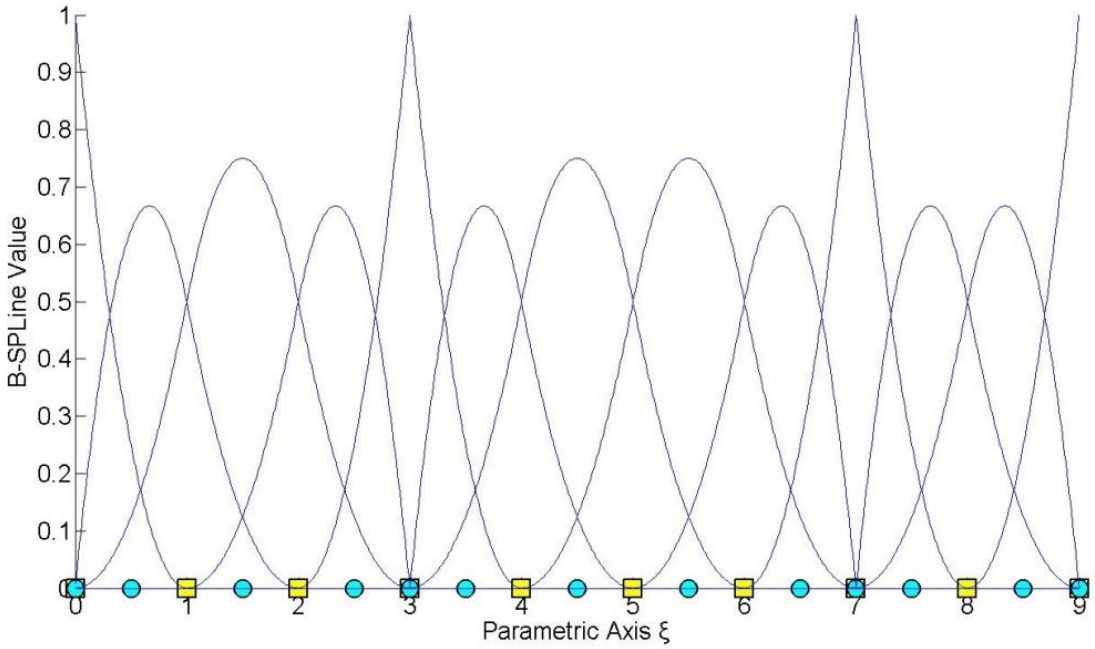


Figure 2.21. 1D B-SPLine basis functions.
$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 7 \quad 8 \quad 9 \quad 9 \quad 9\}$

In this example, the degree is equal to $p = 2$ and we have thirteen basis functions produced, so $n = 13$.

We observe that the knot value number is $k = n + p + 1 = 13 + 2 + 1 = 16$. This occurs because for the creation of n basis functions of degree p, needed to construct the curve, $n + p$ basis functions of order $p = 0$ are used, hence $n + p + 1$ knot values are needed.

In another approach, 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. More specifically, each basis function shares support with at most p basis functions on each side. This results in higher interconnection, compared to equivalent finite element method's shape functions. Basis function overlapping leads to interconnectivity between control points as well.

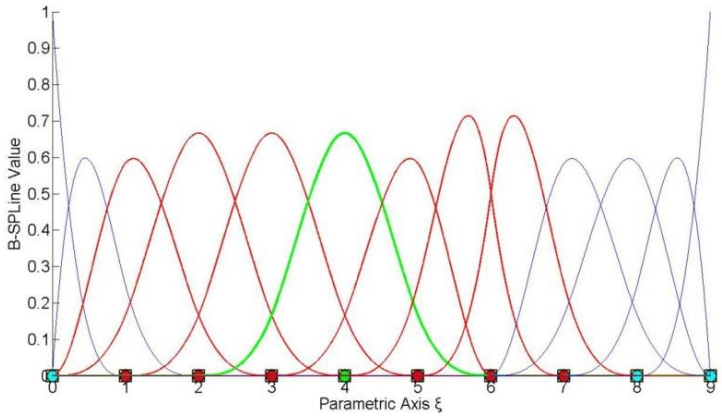


Figure 2.22. Shared support for $N_{6,3}(\xi)$.

$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6 \quad 7 \quad 8 \quad 9 \quad 9 \quad 9 \quad 9\}$

$N_{6,3}(\xi)$ 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 respective positions in the stiffness matrix will be non-zero.

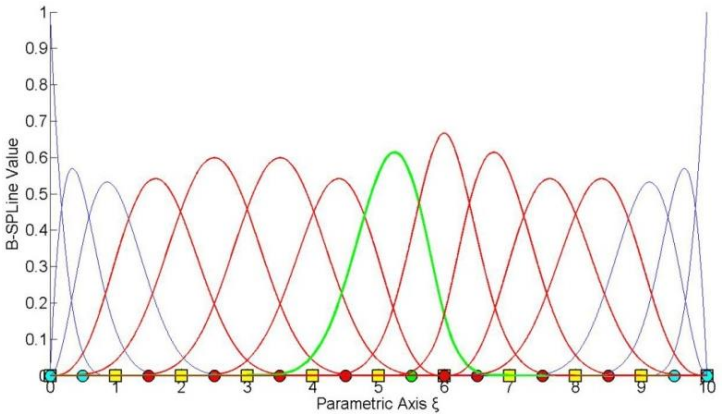


Figure 2.23. Shared support for $N_{8,4}(\xi)$.

$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 10 \quad 10 \quad 10 \quad 10\}$

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

2.3.6 B-SPLine Basis Function Derivatives

Basis function derivatives are widely used in isogeometric analysis. Deformation and stiffness matrices are built upon the derivatives of shape functions. As a result, the distribution of stresses and strains across the model is based on those derivatives. The derivatives of B-SPLines, as obtained from the recursive formula, are represented as a linear combination of previous polynomial order basis functions [1].

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

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

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

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

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

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

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

The partial derivatives of two-directional B-SPLine shape functions can be easily obtained by application of the quotient rule.

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

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

3D shape function derivatives per direction can be obtained in the same manner.

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

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

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

2.3.7 B-SPLine Curves, Surfaces and Solids

Given a knot value vector Ξ and a polynomial order p, we can evaluate the B-SPLine functions at every ξ. In order to create the B-SPLine curve, we also need a vector of coordinates for each basis function, the control points $X_i = \{X_i \quad Y_i \quad Z_i\}$. The curve is evaluated for every ξ as a linear combination of basis functions.

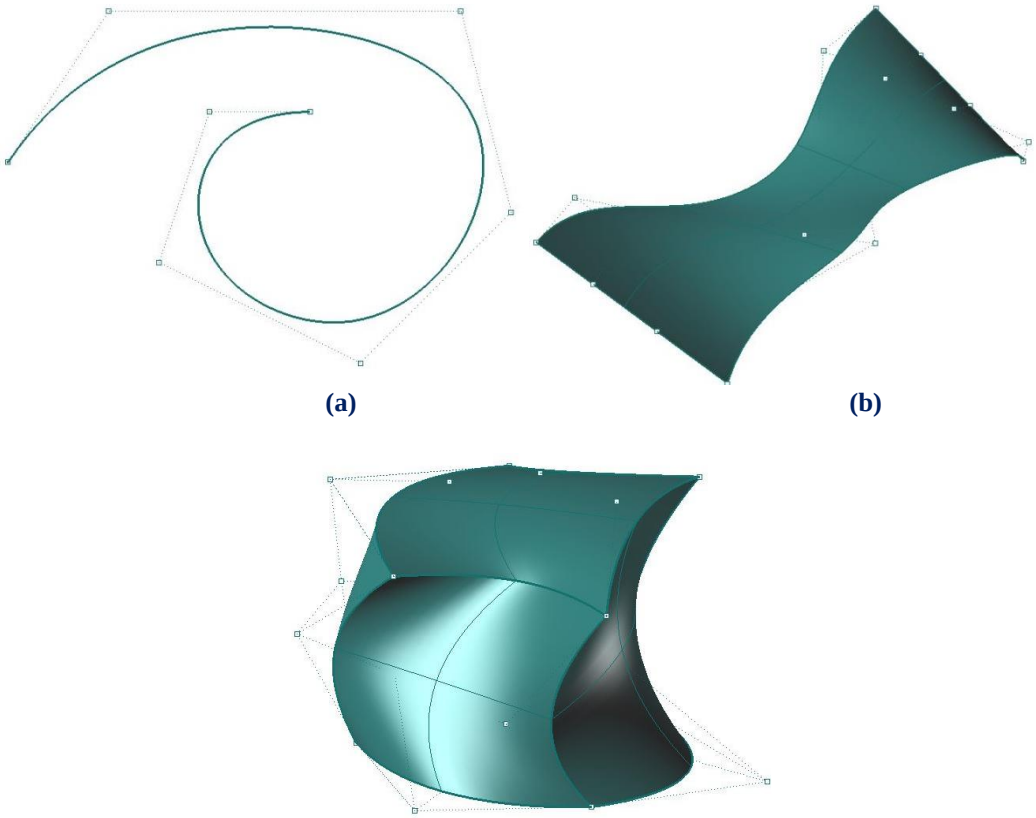
$$C(\xi) = \sum_{i=1}^{n} \left\{ N_{i,p}(\xi) \cdot X_i \right\}$$

After evaluating the shape functions, the B-SPLine surface is defined in analogy to the B-SPLine curve.

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

Using the tensor product properties, we can also evaluate the solid function.

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



(a) **(b)**

(c)

Figure 2.24. B-SPLine entities. (a) Curve, (b) Surface and (c) Solid.

2.3.8 B-SPLine Curve Properties

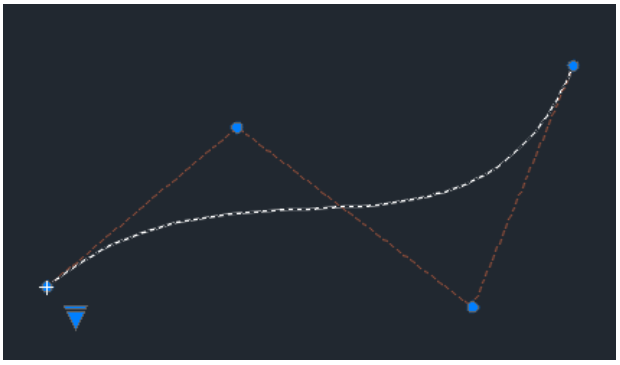
B-SPLine curve entities have the following properties, as obtained from Piegl, Tiller [3]:

1. B-SPLine curves are a generalization of Bezier curves.
2. $C(\xi)$ is a piecewise polynomial curve.
3. Each basis function corresponds to a certain control point.
4. The first and last control point as well as internal control points corresponding 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 at 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 $C(\xi)$ is a linear combination of $N_{i,p}(\xi)$, curve’s continuity and differentiability are obtained straight from the basis functions.
12. No line has more intersections with the curve, than with the control Polygon.

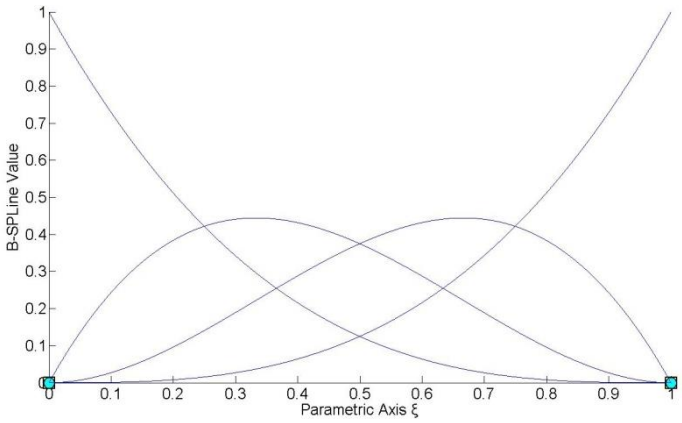
2.3.8.1 Generalization of Bezier Curves

B-SPLine curves are a generalization of Bezier curves. Given an open knot vector and $n = p+1$ control points, a Bezier curve is produced.

Bezier curves were utilized in CAD methods before B-SPLines. **Figure 2.25** presents 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)



(b)

Figure 2.25. B-SPLine as generalization of Bezier curves.
(a) Physical space, (b) Basis functions

2.3.8.2 Piecewise Polynomial Curve

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

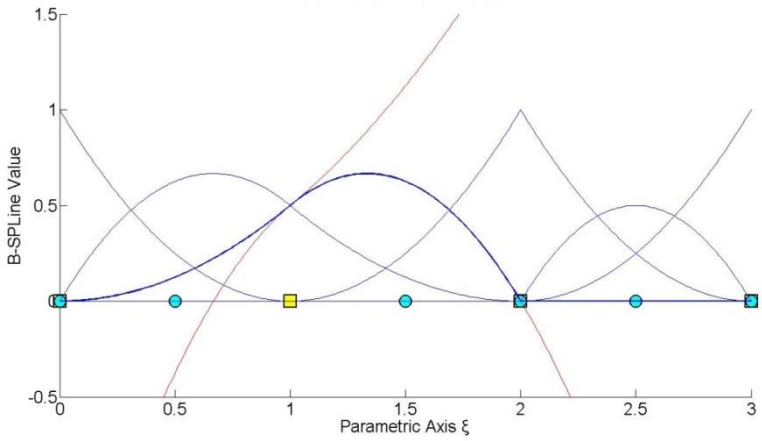


Figure 2.26. Piecewise polynomials that form a B-SPLine basis function.
Knots represent the boundaries of the pieces.

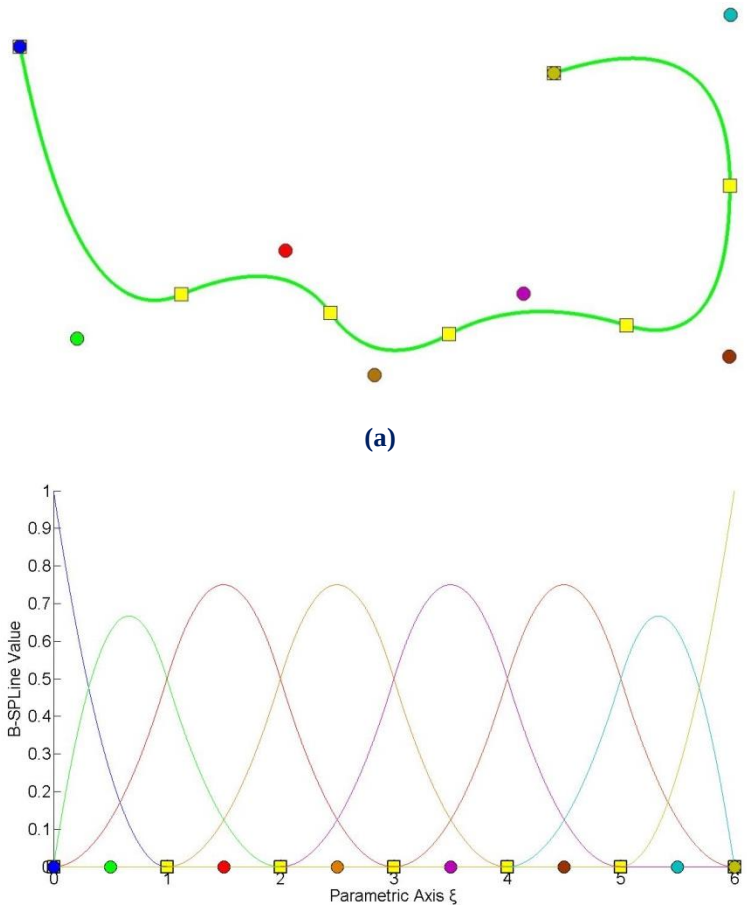
A B-SPLine curve is obtained through the curve function:

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

which is a linear combination of piecewise polynomial basis functions. This applies in multi-directional entities as well. Tensor product shape functions 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 certain control point. There are n basis functions and n control points in a B-SPLine curve.



(a)

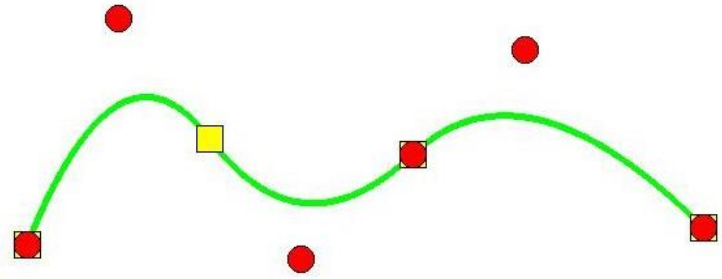
(b)

Figure 2.27. NURBS curve.
(a) Physical space, (b) Basis functions with the corresponding control points.

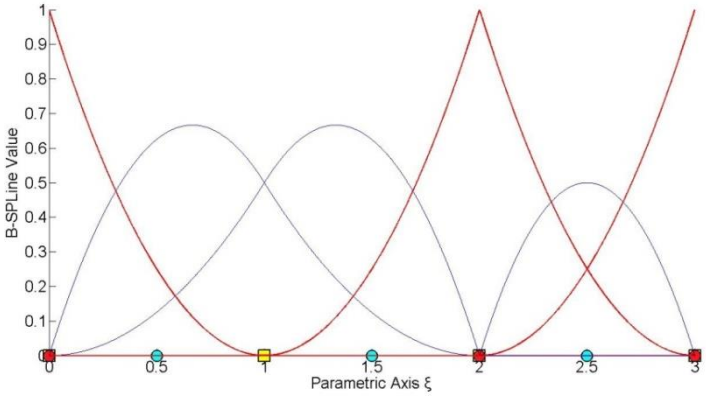
In **Figure 2.27**, control points are represented both in parameter and physical space. Each point controls a specific basis function. This property also applies for multiple directions. Every control point of the surface or the solid is tensor product of a control point in directions ξ, η and ζ. By extension, the corresponding B-SPLine is tensor product of the basis functions. In the most general case, there are $n \cdot m \cdot l$ control points and basis functions.

2.3.8.4 Interpolation to the Curve

The first and last control points are interpolatory to the curve. Any internal control point corresponding to C^0 continuous basis function is also interpolatory to the curve.



(a)



(b)

Figure 2.28. Control point interpolation.
(a) B-SPLine curve, (b) The reciprocal basis functions

In **Figure 2.28**, the first and the last control points, which have C^{-1} continuity, are interpolatory to the curve. This can be explained with the help of the equation of the curve:

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

For $\xi = 0$, it applies that:

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

where:

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

so,

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

And for $\xi = 3$:

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

$$N_{6,2}(3) = 1$$
$$N_{i,2}(3) = 0,\ i = 1,...,5$$

so,

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

Likewise, the internal control point, with C^0 continuity across $\xi = 2$ is interpolatory to the curve because:

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

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

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

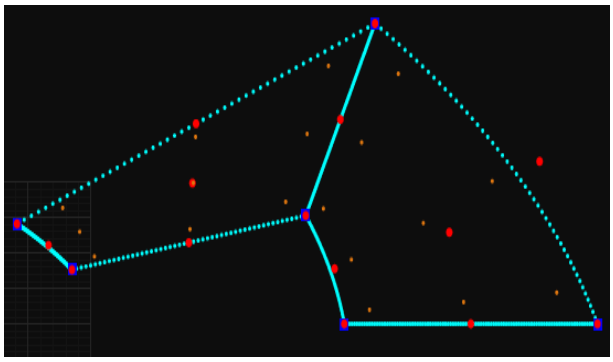
so,

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

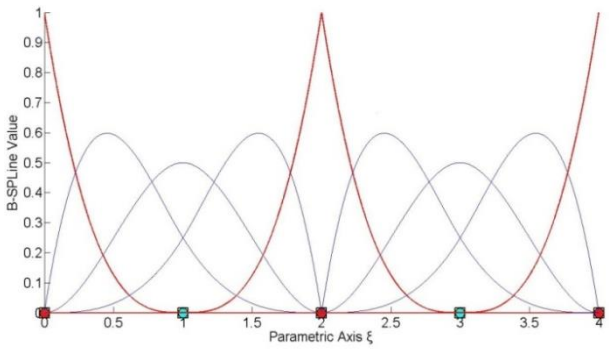
Observe that both the form of the curve and the form of the basis functions indicate that this geometry could be represented by two different sets of knot vectors and control points, with absolutely no deflections from the current representation. This will be examined thoroughly later. Interpolation also applies for surfaces and solids, when appropriately reduced continuity is used for all directions at a knot. C^{-1} continuity is required for external knots and C^0 for internal.

In **Figure 2.29**, $N_{5,2}(\xi)$ at axis ξ and $M_{6,2}(\eta)$ at axis η have C^0 continuity. Therefore, the control point corresponding to $R^{2,2}_{5,6}(\xi,\eta)$ is interpolatory to the surface. The external control points with C^{-1} continuity at both directions are also interpolatory to the surface.

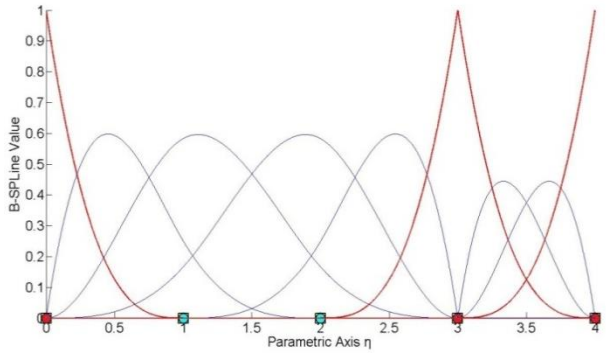
In **Figure 2.30**, $N_{4,2}(\xi)$ at parametric axis ξ and $M_{4,2}(\eta)$ at parametric axis η have C^0 continuity, thus control points corresponding to the shape functions $R^{2,2,1}_{4,4,1}(\xi,\eta,\zeta)$ and $R^{2,2,1}_{4,4,2}(\xi,\eta,\zeta)$, are interpolatory to the solid. As we can see, the external control points are also interpolatory to the solid, because continuity is reduced to C^{-1}.



(a) B-SPLine quadratic surface.

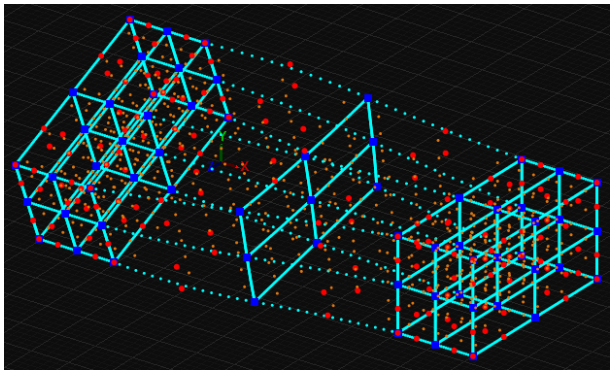


(b) Quadratic basis. Ξ={0,0,0,1,1,1}.

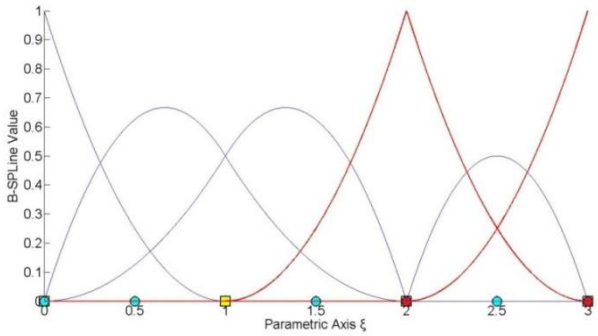


(b) Quadratic basis. H={0,0,0,0.5,0.5,1,1,1}.

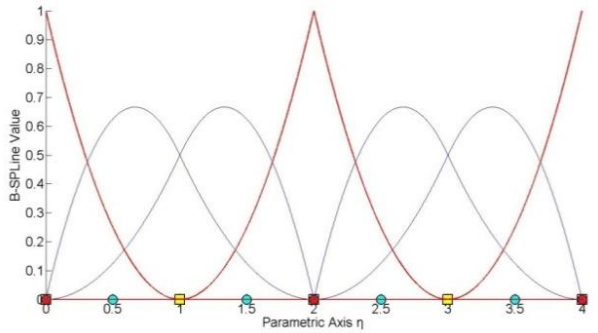
Figure 2.29. B-SPLine surface (n=3, m=5).



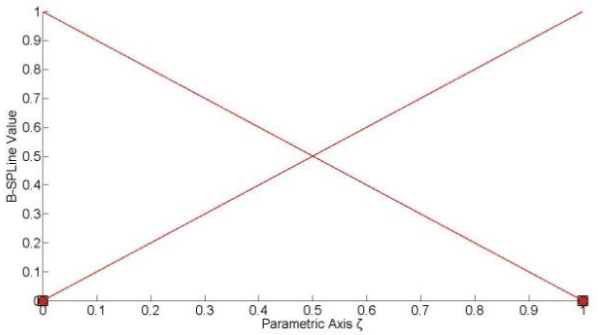
(a) B-SPLine quadratic volume.



(b) Quadratic basis. Ξ={0,0,0,0.16667,0.3333,0.33333,0.5,0.66667,0.66667,0.833333,1,1,1}.



(c) Quadratic basis. H={0,0,0,0.5,1,1,1).



(d) Quadratic basis. Z={0,0,0,0.5,1,1,1).

Figure 2.30. B-SPLine volume (n=10, m=4, l=4).

2.3.9 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.

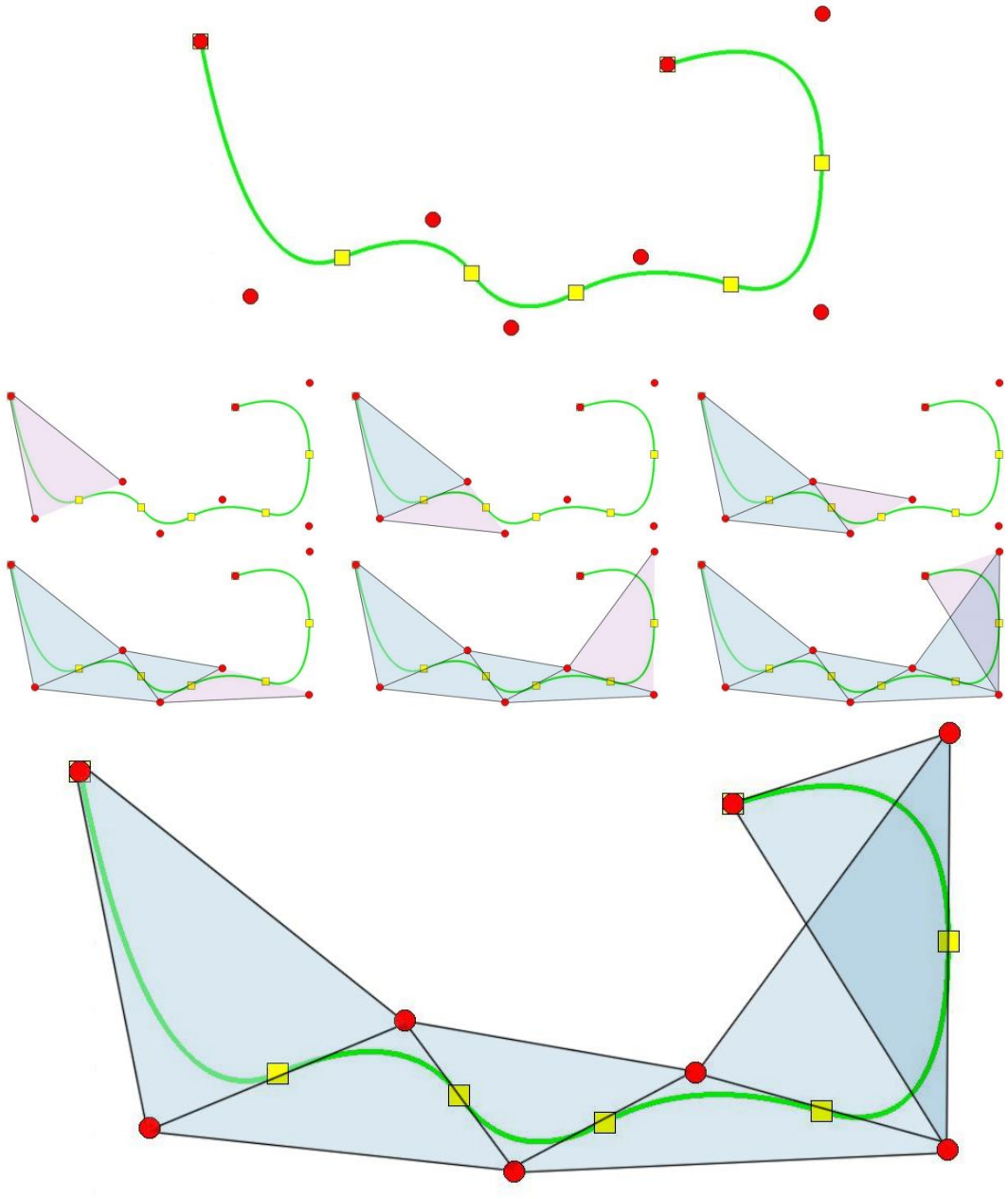


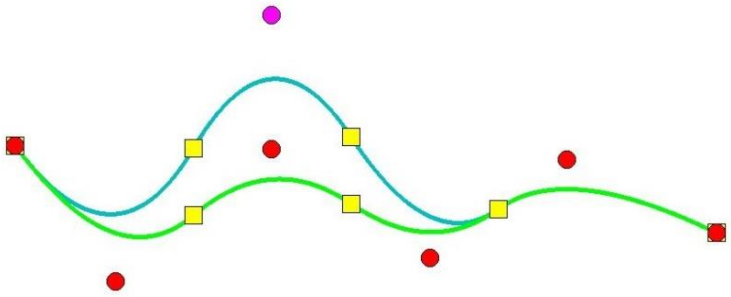
Figure 2.31. Step-by-step convex hull creation for a B-SPLine curve.

The curve in **Figure 2.31** has a degree of $p = 2$. The convex hull is formed by connecting each control point with the $p = 2$ successive ones. As we can easily see in the figure, 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.

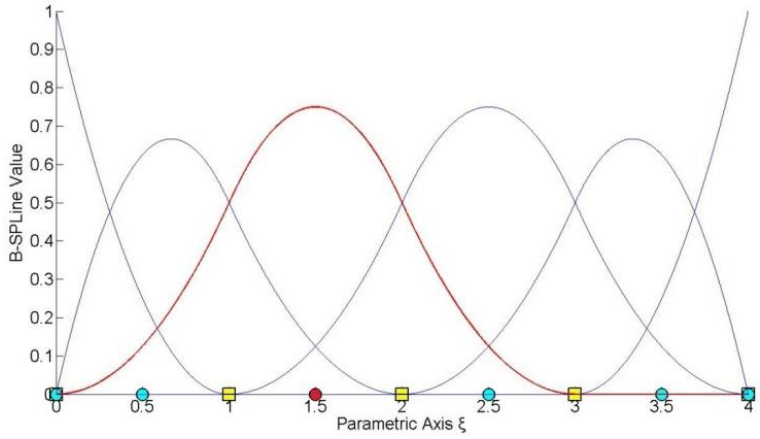
2.3.10 Control Point Local Support

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) Physical space.



(b) Parameter space.

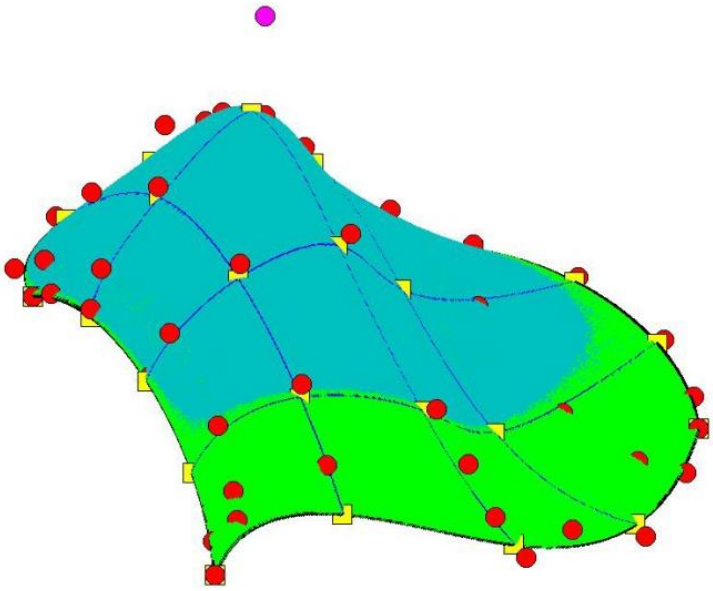
Figure 2.32. Control point local support.

Every 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.

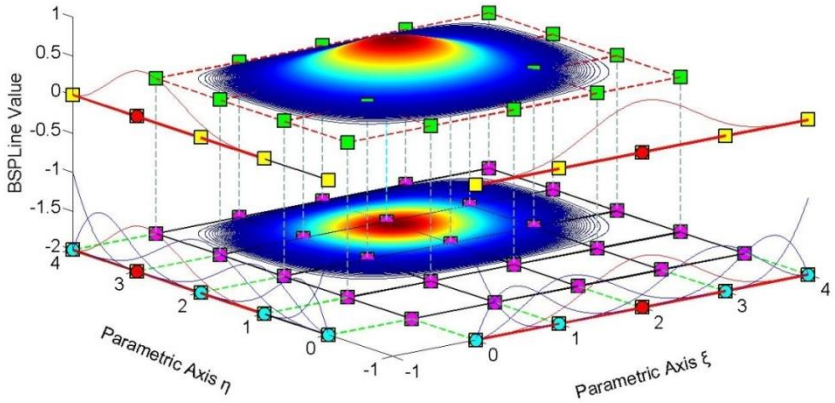
Figure 2.32 presents a curve with $p = 2$ and knot value vector $\{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 4 \quad 4\}$. The control point for $i = 3$ is moved and this affects partially the entity.

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.



(a) Physical space.

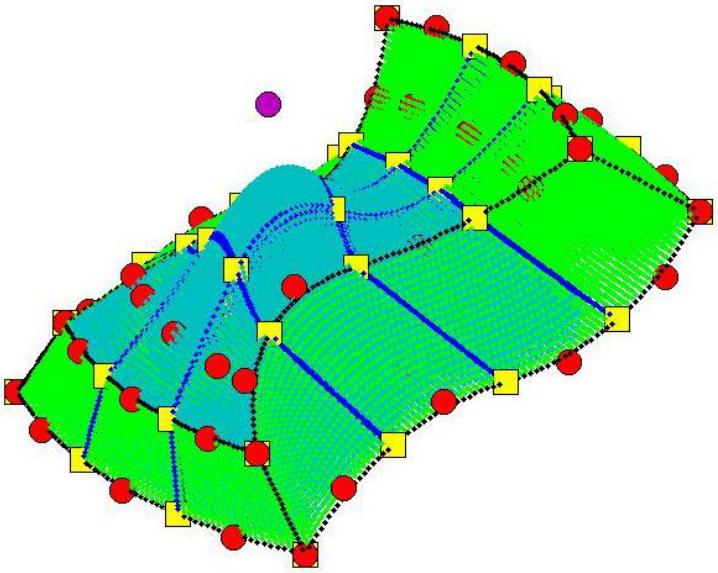


(b) Parameter space.

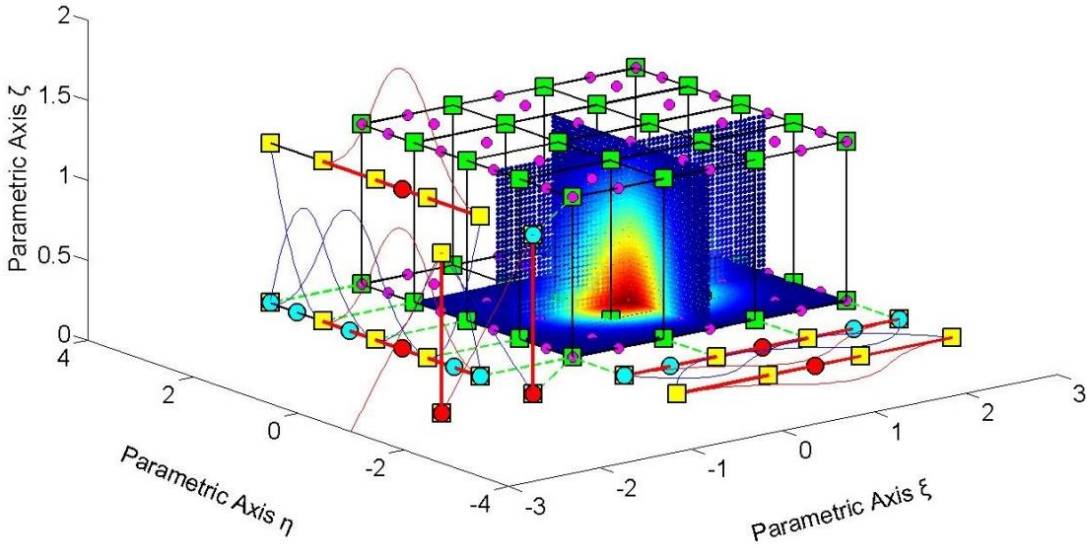
Figure 2.33. Local support of a 2D control point.

Figure 2.33 represents the local support in the parametric axis 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 2.33.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) Physical space.



(b) Parameter space.

Figure 2.34. Local support of a 3D control point.

Figure 2.34.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 2.34.a**, the local support is displayed in cyan and it does not reach all the knot spans in parametric axis η. A total of $3 \cdot 3 \cdot 1 = 9$ knot cuboids represent the support of the control point.

2.3.11 Control Polygon Approximation

The control polygon represents 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.

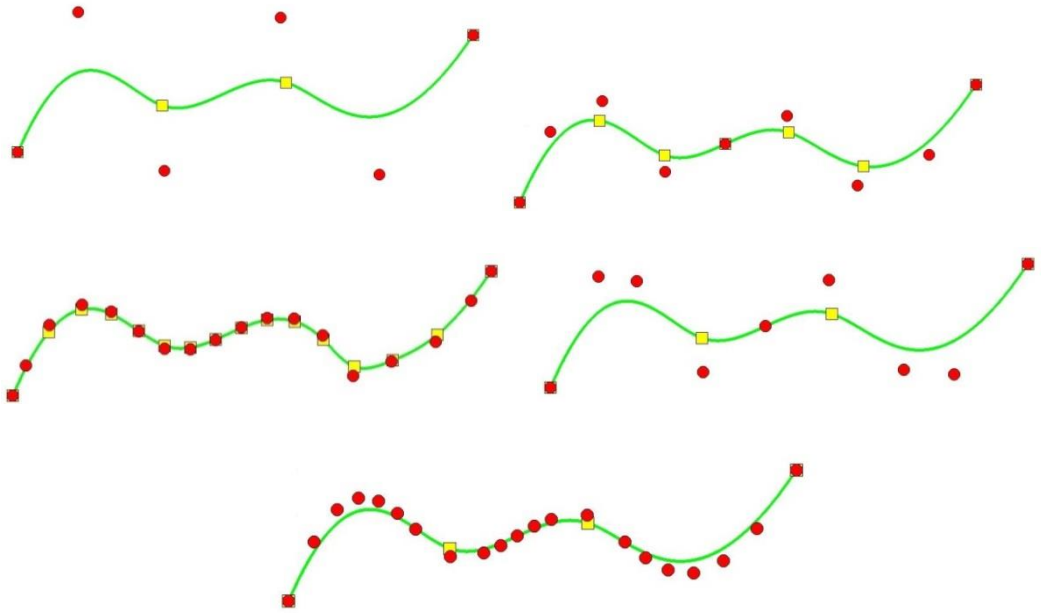


Figure 2.35. Control polygon approximation through refinement.

In **Figure 2.35** a curve of degree $p = 3$ is designed. The control polygon already represents 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.

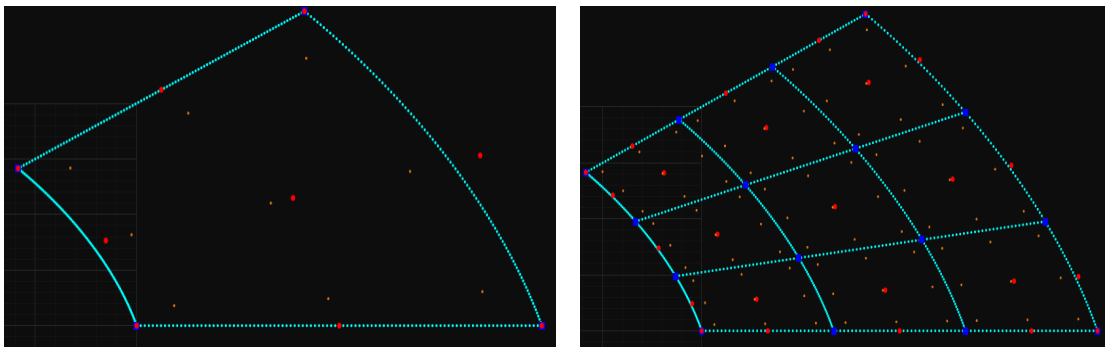
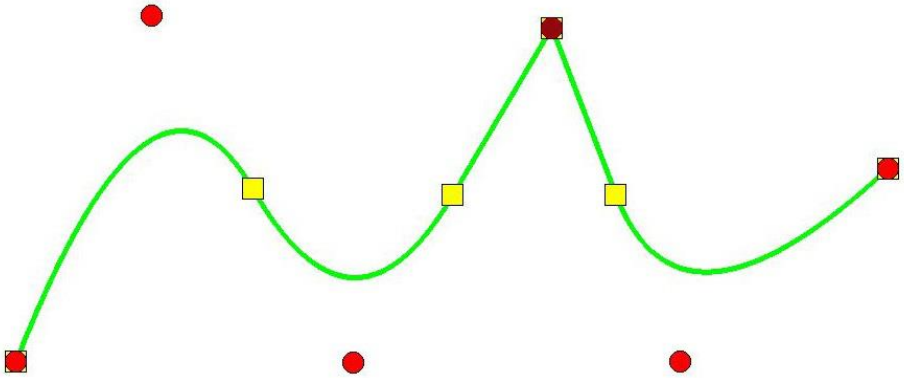


Figure 2.36. Control net approximation through surface h-refinement.

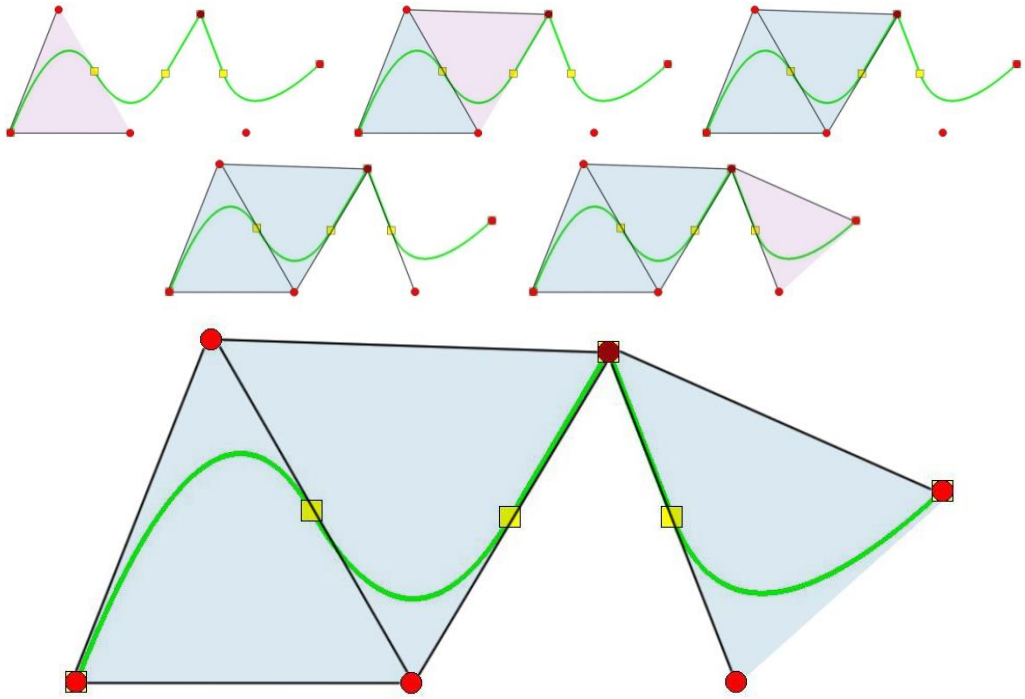
For example, in **Figure 2.36**, the refinements, that were made, brought the control net closer to the surface.

2.3.12 Multiple Control Points

It is possible to use multiple control points with the same coordinates. This can prove to be very useful.



(a) Physical space.



(b) **Convex hull creation for a curve with two coincident control points (drawn in deep red).**

Figure 2.37. B-SPLine curve and convex hull.

In **Figure 2.37.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 2.37.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 introduced Non-Uniform Rational B-Splines (NURBS). The basic idea is simple. A NURBS entity is produced from the actual section of a cone with a plane.

2.4.1 Basic Idea

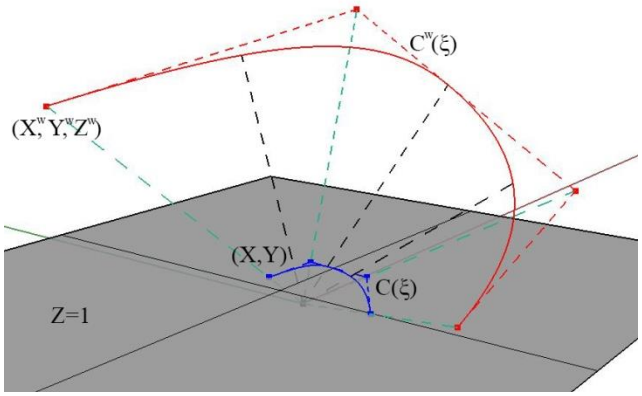


Figure 2.38. B-SPLine curve and projective transformation to NURBS curve.

As shown in **Figure 2.38**, 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:

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

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

2.4.2 NURBS Shape Functions

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

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

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 from:

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

$R_i^p(\xi)$ 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:

Shape functions for two directions:

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

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

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. Actually, NURBS entities are a generalization of B-SPLine entities. All the B-SPLine properties examined in this thesis apply for NURBS as well.

2.4.3 NURBS Shape Function 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:

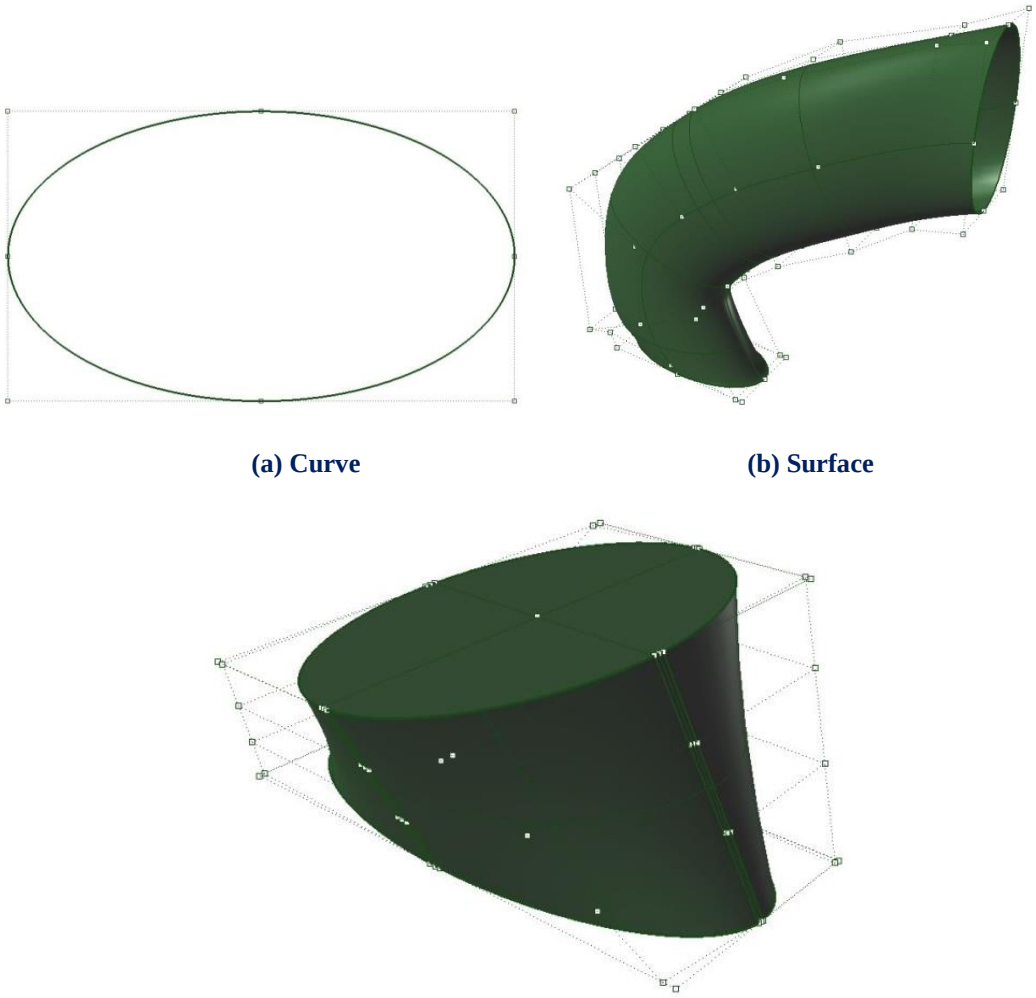
$$C(\xi) = \sum_{i=1}^{n} \left\{ R_i^p(\xi) \cdot X_i \right\}$$

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:

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

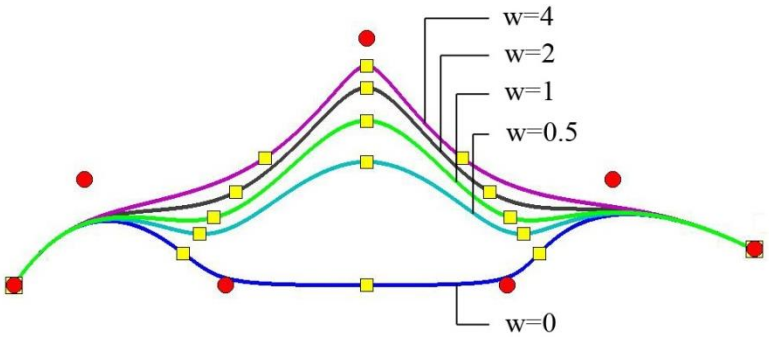


(a) Curve **(b) Surface**

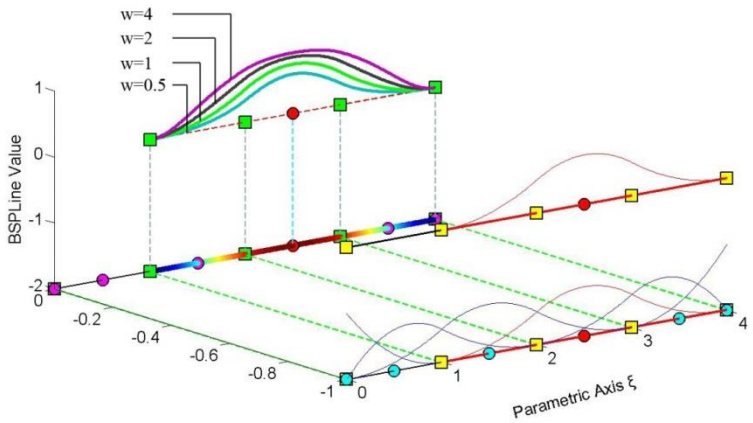
(c) Solid

Figure 2.39. NURBS elliptical entities.

2.4.5 NURBS Examples



(a)



(b)

Figure 2.40. (a) NURBS curves and (b) shape functions for different weight values.

In **Figure 2.40**, 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.

According to [1], 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 2.41**, 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.

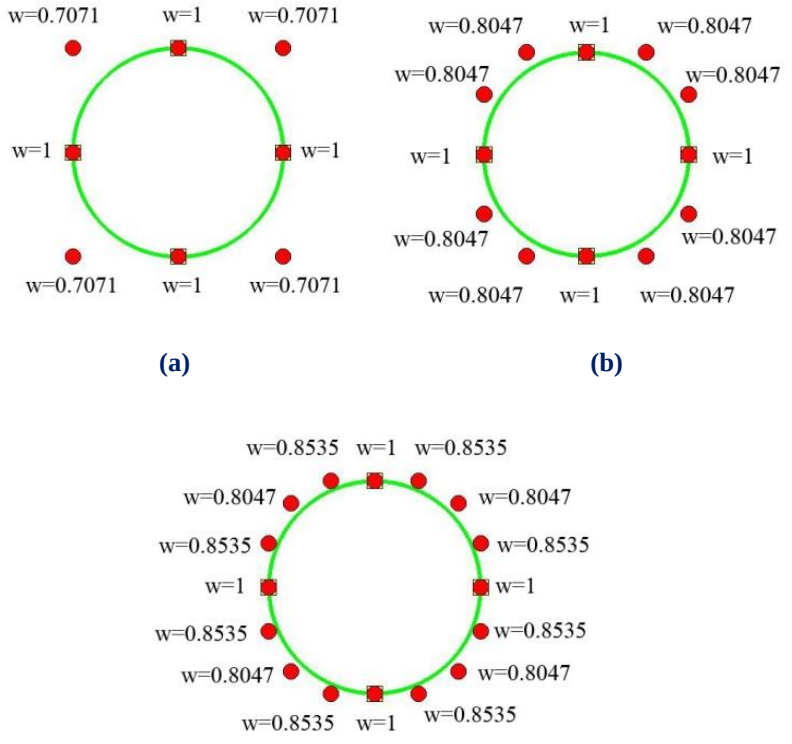


Figure 2.41. NURBS circle of different polynomial degree.
Weight values for each control point.
(a) Quadratic basis functions. $\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4\}$
(b) Cubic basis functions. $\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4 \quad 4\}$
(c) Quadric basis functions.
$\Xi = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4\}$

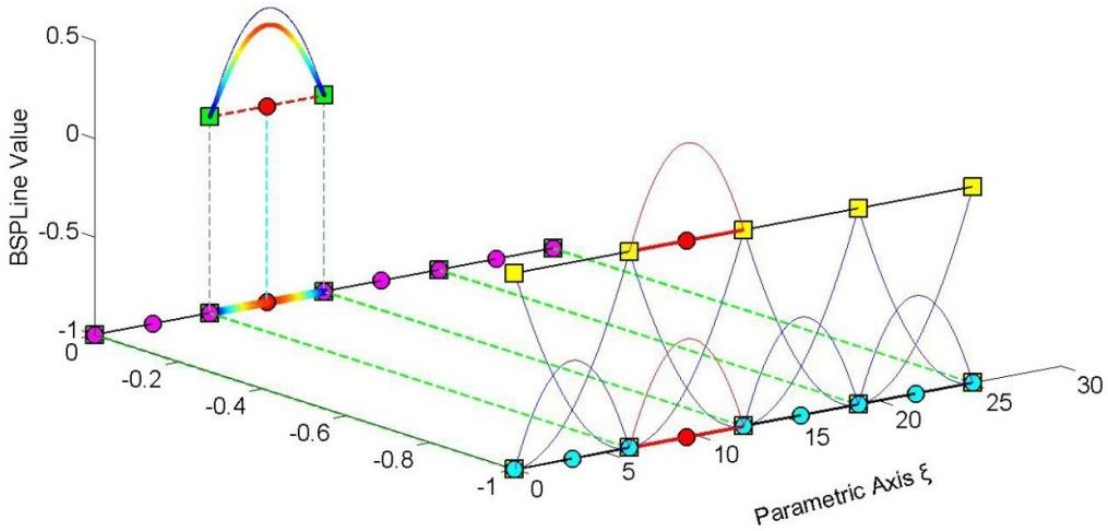


Figure 2.42. Basis functions for circle represented in Figure 2.41.
Contour distribution for shape function $R_5^2(\xi)$ **, with corresponding weight** $\frac{\sqrt{2}}{2} = 0.7071$**.**
Comparison with basis function $N_{5,2}(\xi)$ **shown in blue.**

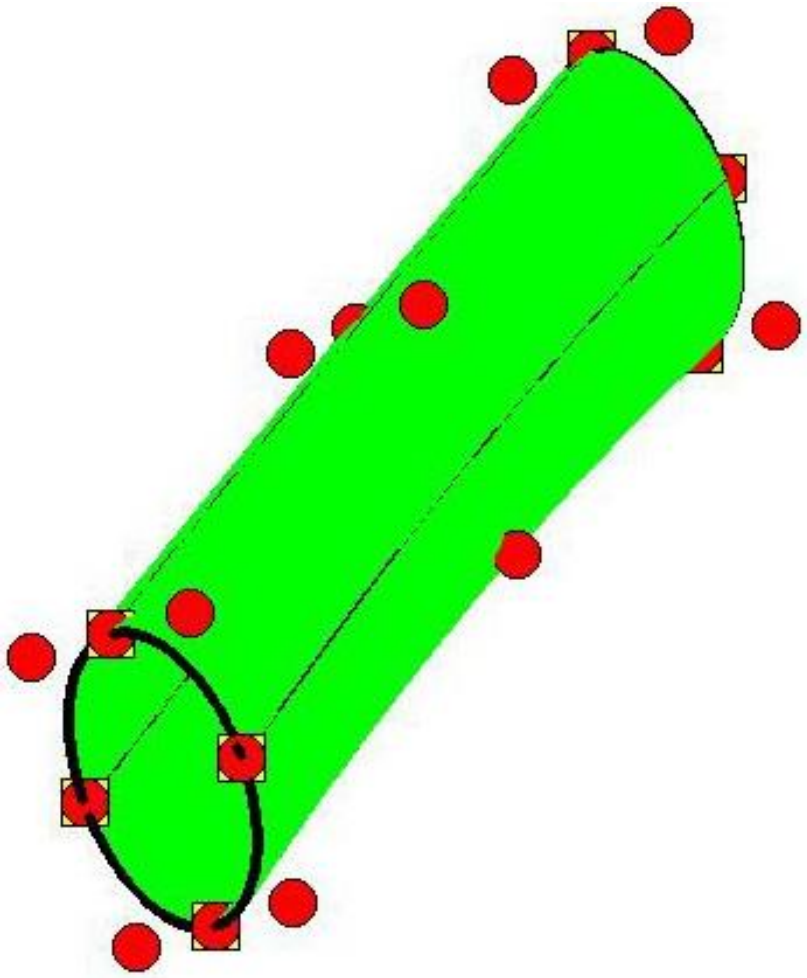


Figure 2.43. NURBS surface created from consecutive circle cross-sections.

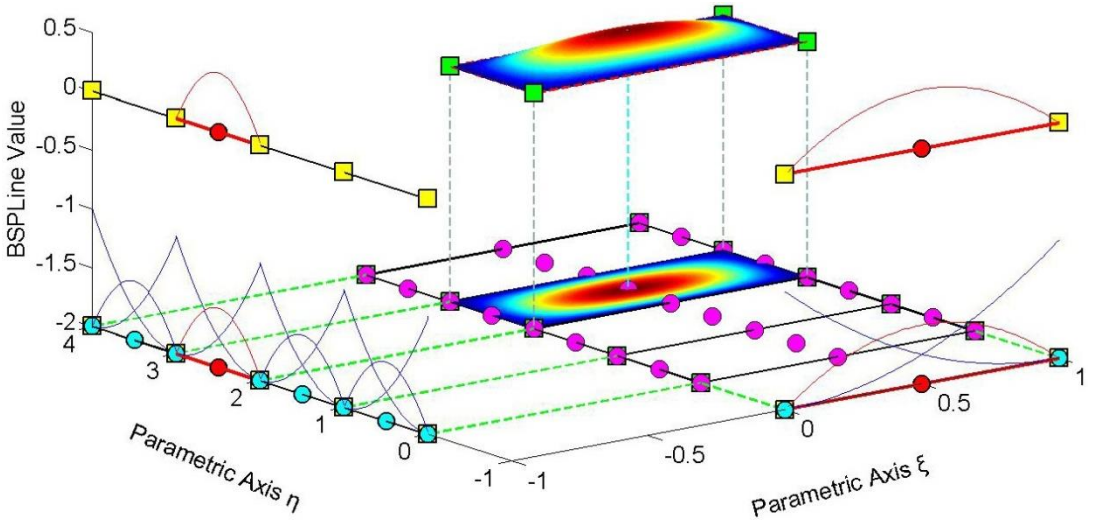
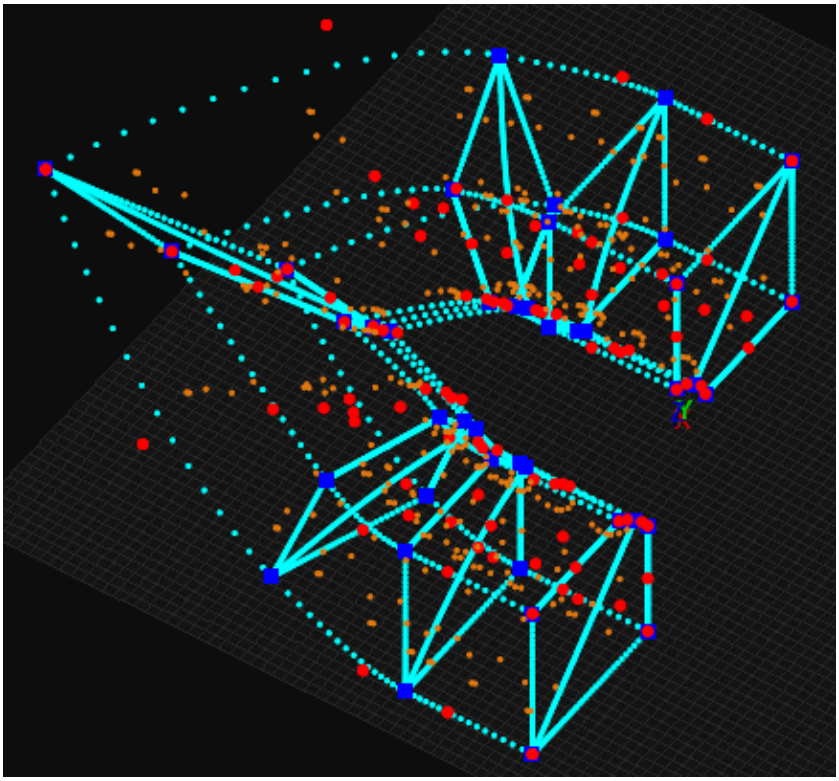


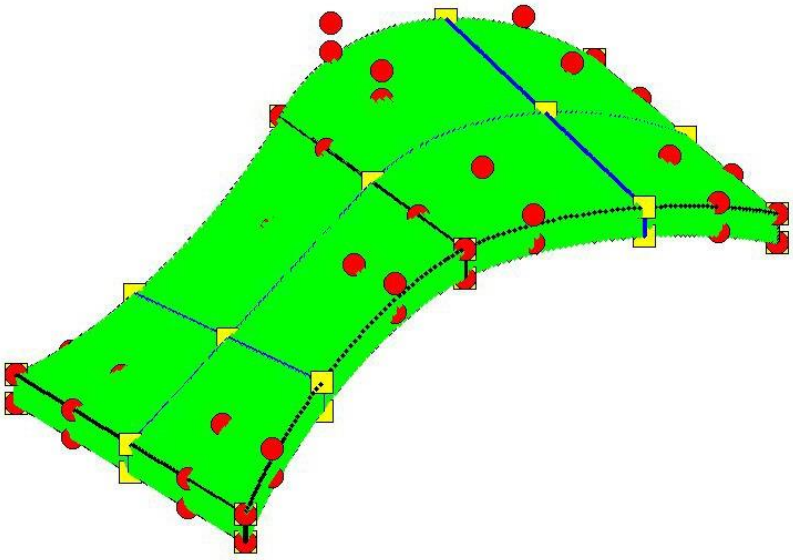
Figure 2.44. Shape function $R_{2,6}^{2,2}(\xi,\eta)$ for the surface of Figure 2.43.

The corresponding weight is $\frac{\sqrt{2}}{2}=0.7071$.

The representation and main attributes are exactly the same as those of B-SPLine shape functions.



(a) Wine glass.



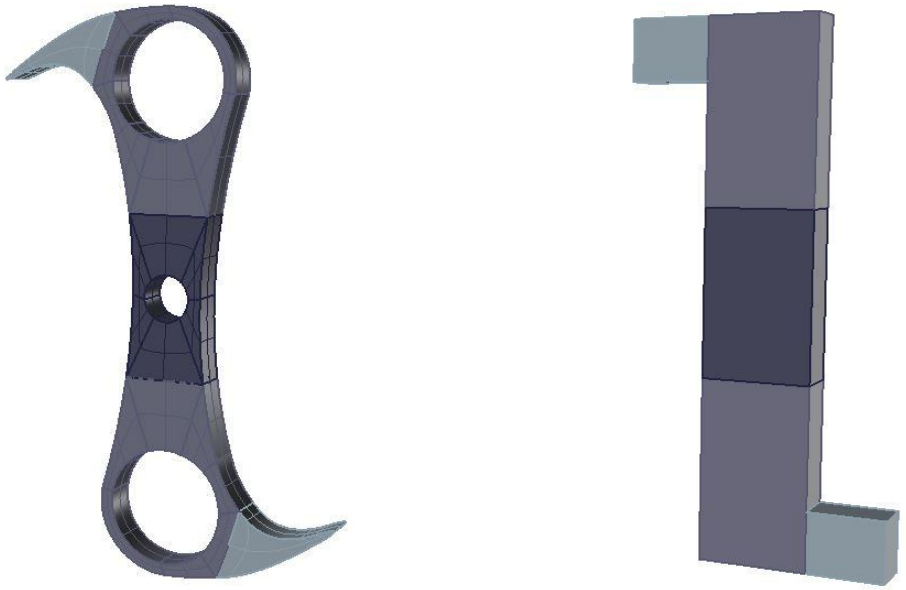
(b) Abstract NURBS solid.

Figure 2.45. NURBS solids.

The glass of wine displayed in **Figure 2.45.a** is a 3D NURBS solid. The potential of isogeometric analysis is clearly represented in this model. Observe the exact representation of conic sections and smooth surfaces, in combination with immediate mesh generation. The mesh, that is depicted in this figure, can be instantly used for analysis. An abstract form of another NURBS solid is represented in **Figure 2.45.b**.

2.4.6 Patches

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 optimal 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 2.46**, 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 2.46. The famous Falkirk Wheel "abutment".
(a) Geometrical representation with five separate patches.
(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, as displayed in **Figure 2.47**, 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. In the examples used in this book, the latter option is preferred, when possible.

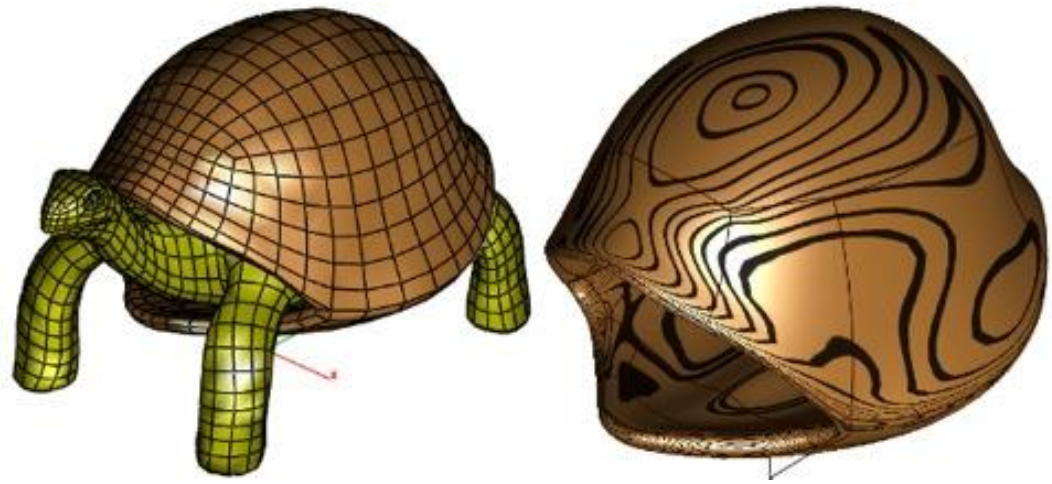


**Figure 2.47. NURBS patches enforced in order to distinct shell from turtle.
Each material requires separate patch stiffness matrix evaluation,
before global stiffness matrix formulation.
(**http://www.masterviacad.com/powerpackoverview/PowerPackPro/page36/page36.html**)**

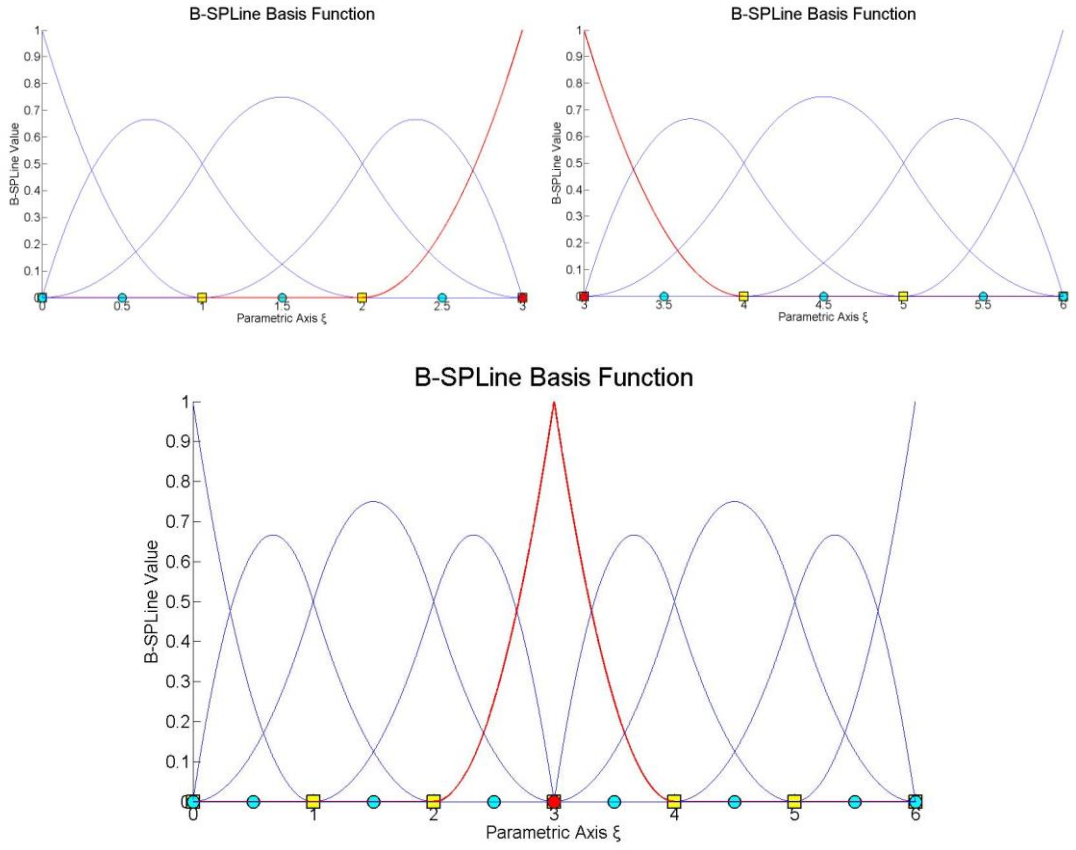


**Figure 2.48. Separate knot vectors united into one.
The control points at the boundary are merged. C^0 -continuity is applied.**

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. The importance of continuity in analysis is examined in the following chapters.

2.5 Refinement

2.5.1 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 mesh 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 refinement efficiently 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 small 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.

2.5.2 Knot Value 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.

According to Hughes [1], this can be achieved with the evaluation of a transformation matrix, so that:

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

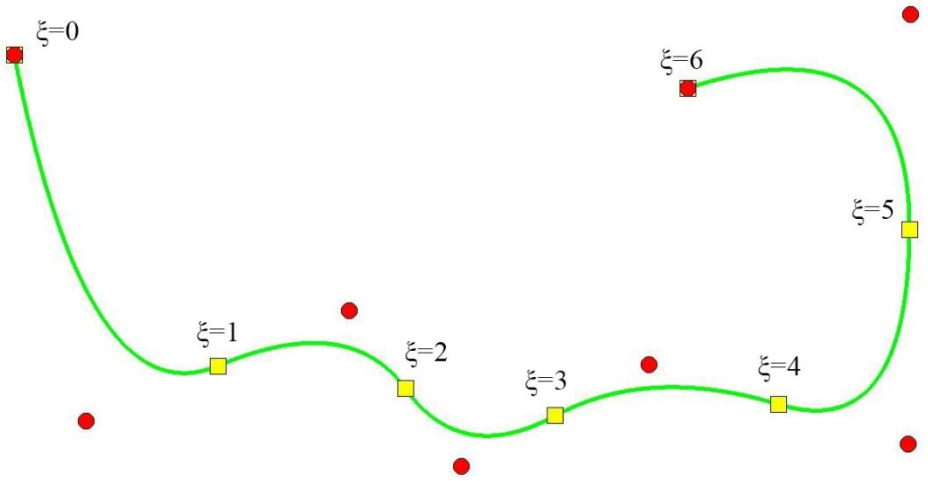
This matrix is formed recursively:

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

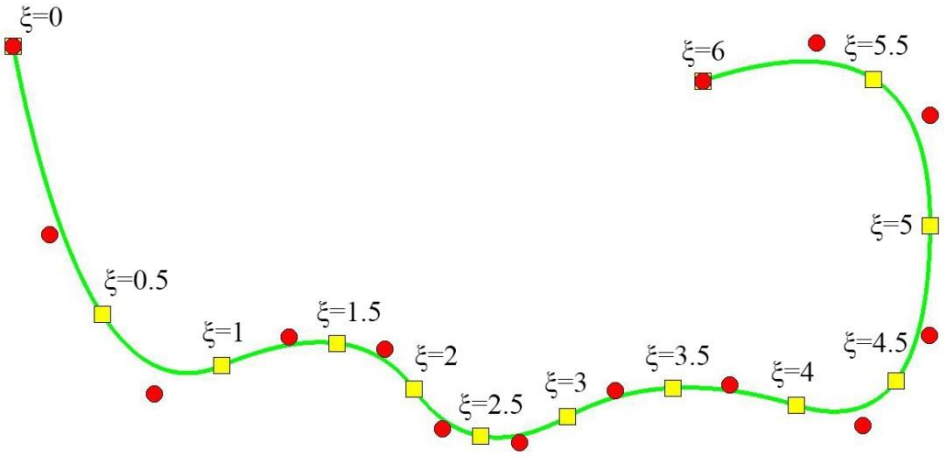
$$T_{i,j}^q = \frac{\overline{\xi}_{i+q} - \xi_j}{\xi_{j+q} - \xi} \cdot T_{i,j}^{q-1} + \frac{\xi_{j+q+1} - \overline{\xi}_{i+q}}{\xi_{j+q+1} - \xi_{j+1}} \cdot T_{i,j+1}^{q-1}, \text{ for } q = 1, 2, ..., p$$

This technique is called h-refinement.

The B-SPLine curve represented in **Figure 2.49.a** is refined by knot value insertion in **Figure 2.49.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. This can be seen in **Figure 2.50**. The limited area of effect for each control point leads to a more accurate approximation to the curve.



(a)



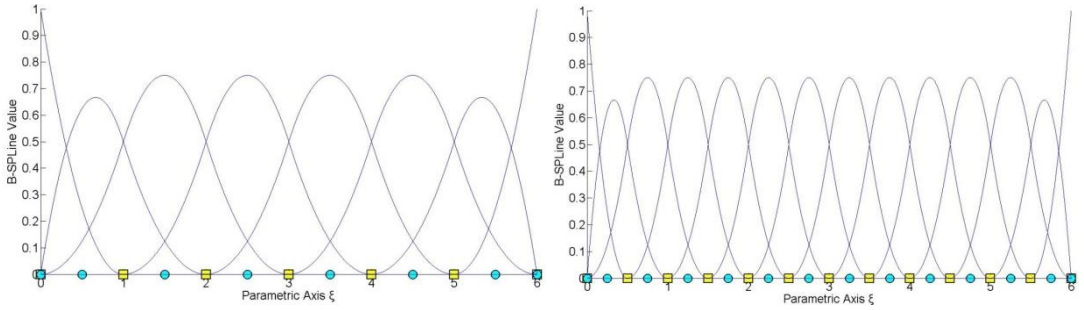
(b)

Figure 2.49. h-refinement applied on a B-SPLine curve.
(a) Coarse mesh.

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

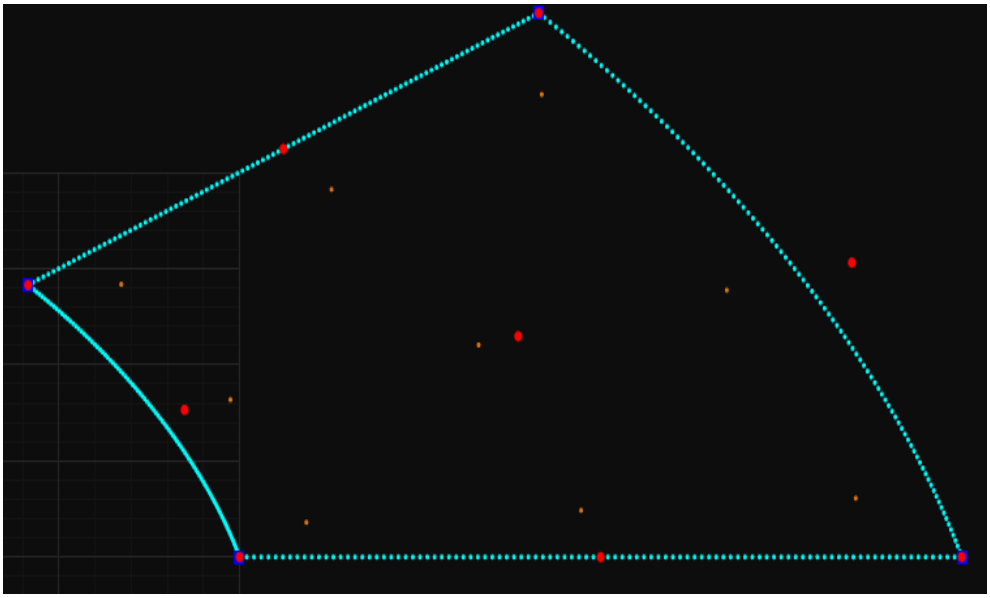
(b) Fine mesh.

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



(a) **(b)**

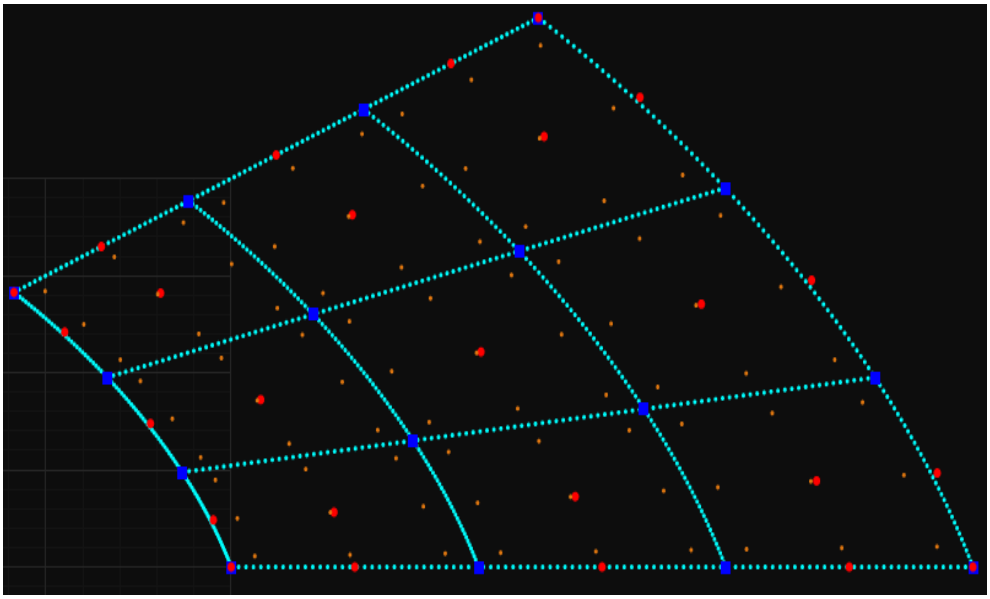
Figure 2.50. Basis functions (a) before and (b) after h-refinement.



(a) Coarse mesh.

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

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



(b) Fine mesh.

$$\Xi = \{0 \quad 0 \quad 0 \quad 0.333333 \quad 0.666667 \quad 1 \quad 1 \quad 1\}$$

$$H = \{0 \quad 0 \quad 0 \quad 0.333333 \quad 0.666667 \quad 1 \quad 1 \quad 1\}$$

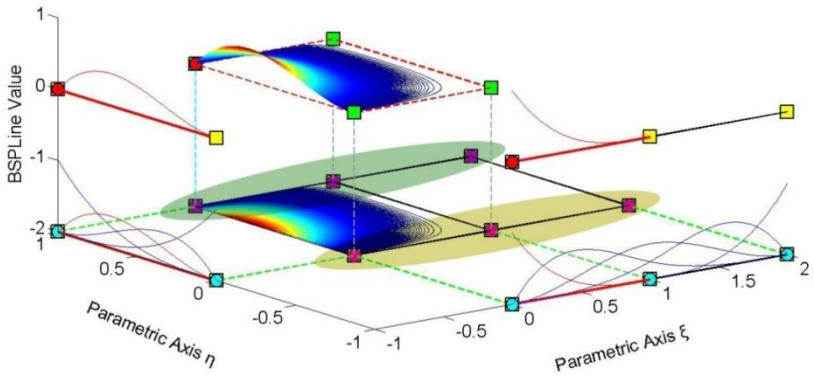
Figure 2.51. Knot insertion in multiple directions.

Refinement is applicable in multi-directional problems as well.

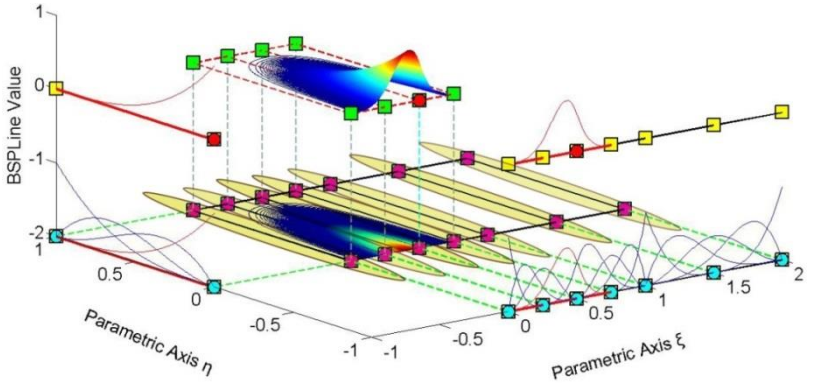
The surface in **Figure 2.51** is refined both per ξ and η. **Figure 2.52** represents basis functions before and after refinement.

Note that for every univariate (1D) control point per η, a whole set of control points per ξ is defined. Each set is refined individually, as shown in **Figures 2.52.a** and **2.52.b**. After the new control points per ξ are introduced, the same process is followed for refinement per η.

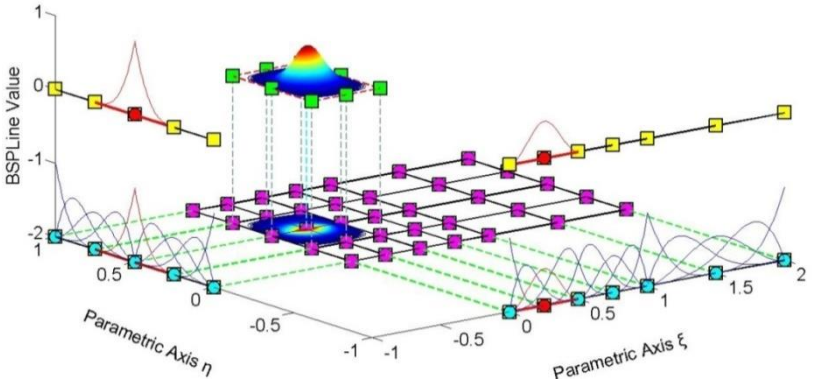
The order in which refinement is applied is of no importance; due to tensor product properties, control points per η could have been refined first, but the result would be the same.



(a) Coarse mesh.



(b) Refinement per ξ direction.



(c) Refinement per η direction. Fine mesh.

Figure 2.52. Basis and shape functions for surface (coarse, fine mesh).

2.5.3 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 $\bar{p}$, the new knot value vector has to be defined first. No new knots are added, but every existing knot's multiplicity is increased by $\bar{p}-p$ 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:

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

This leads to:

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

The new set of control points $\left\{\bar{X}\right\}$, 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, $\underset{(n\times m)}{[N]}$, which contains the values of the n basis functions for the m selected points.

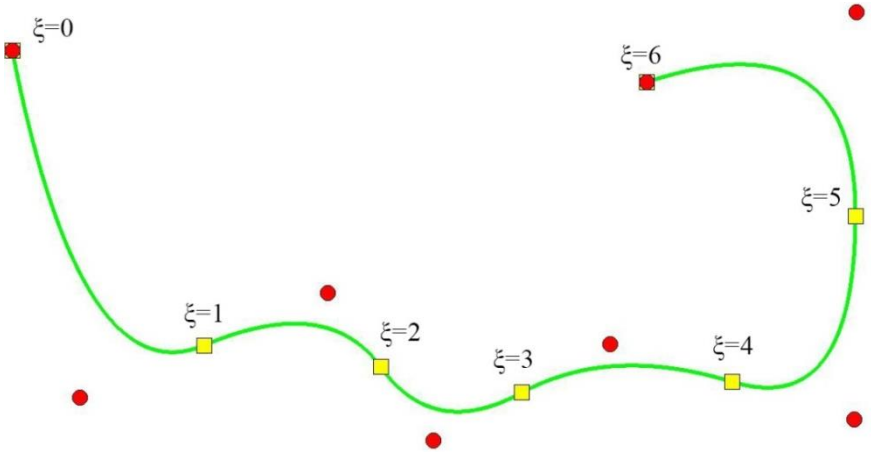
The same points are used for the evaluation of new basis functions, with the refined degree and the new knot value vector. Thus, $\underset{(m\times m)}{\left[\bar{N}\right]}$ is created. Since the same points are evaluated and the parametric mapping remains the same, it applies that:

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

Therefore,

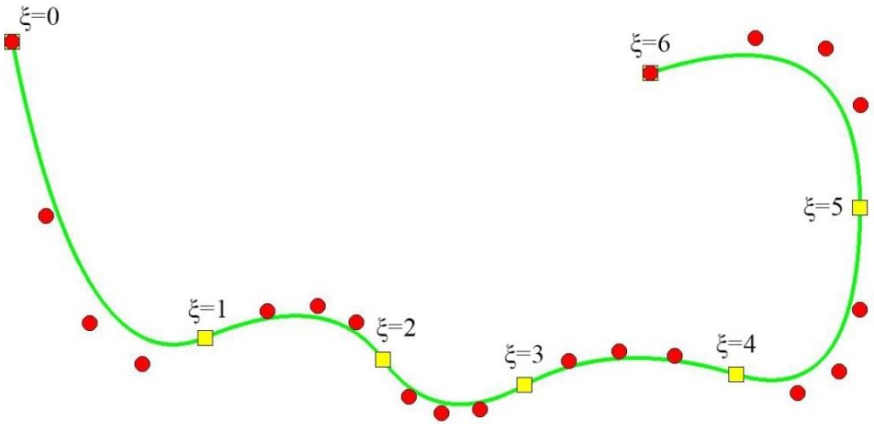
$$\underset{(m\times n)}{[T]} = \left(\underset{(m\times m)}{\left[\overline{N}\right]^T} \right)^{-1} \cdot \underset{(m\times n)}{[N]^T}$$

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.



(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.

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

Figure 2.53. p-refinement on a B-SPLine curve, from $p = 2$ to $\bar{p} = 4$.

The curve in **Figure 2.53** is subjected to p-refinement. Note that knot spans are unchanged. 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.

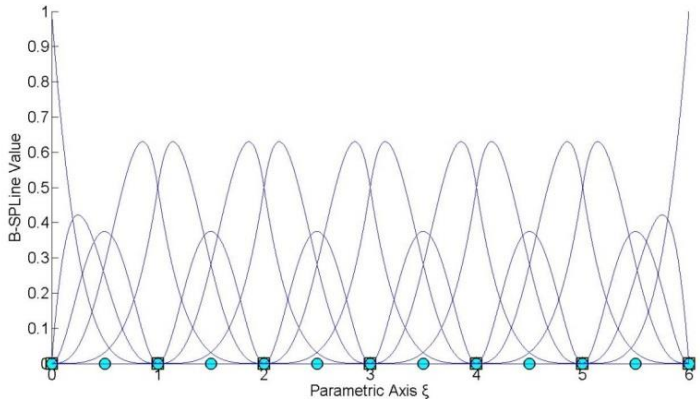
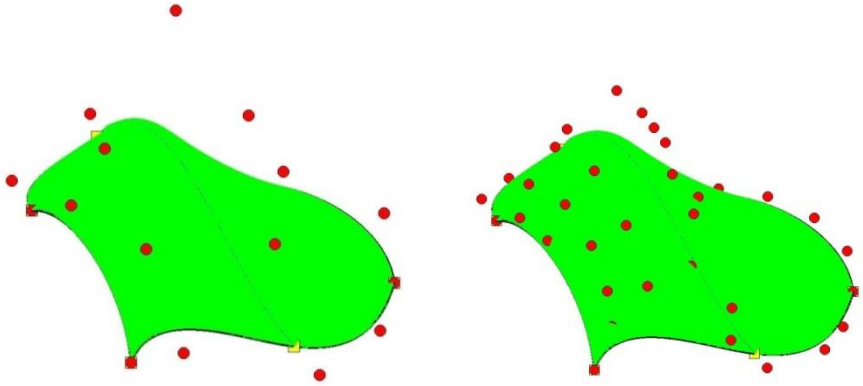


Figure 2.54. Basis functions for p-refinement of Figure 2.53. C^1 continuity across internal knots.

Tensor product properties prove that p-refinement can be utilized in multiple directions.



(a) **(b)**

Figure 2.55. Order elevation in multiple directions.
(a) Coarse mesh. Polynomial degree: $(p,q)=(3,3)$.
(b) Fine mesh. Polynomial degree: $(p,q)=(4,5)$.

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 remain the same. Basis function interconnectivity, however, is improved.

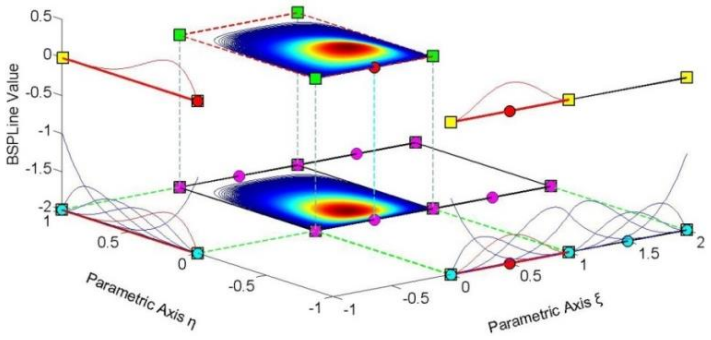
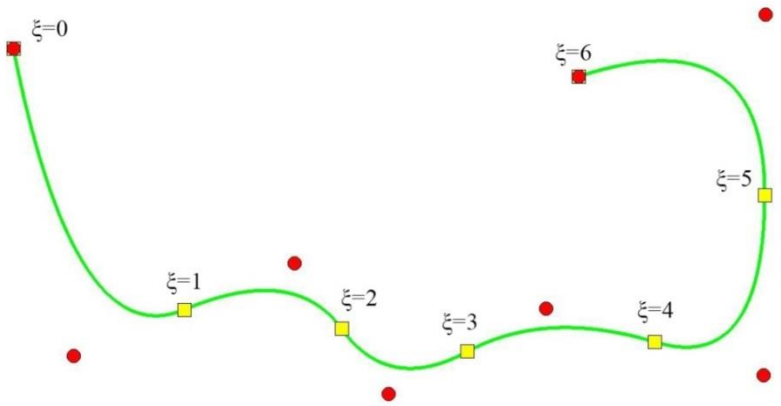


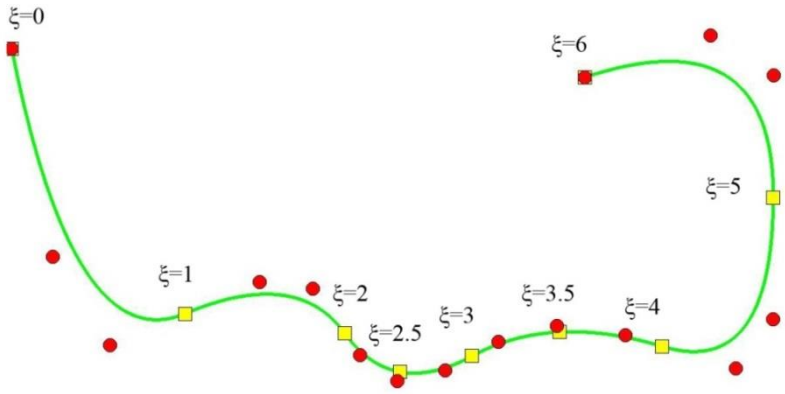
Figure 2.56. Basis and shape functions for p-refinement. Reduced continuity across internal knots.

2.5.4 Degree Elevation and Knot Insertion

Increasing polynomial degree by p-refinement is an improvement to the basis, but continuity remains the same 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 greater convergence rates for our models.



(a) Coarse mesh.

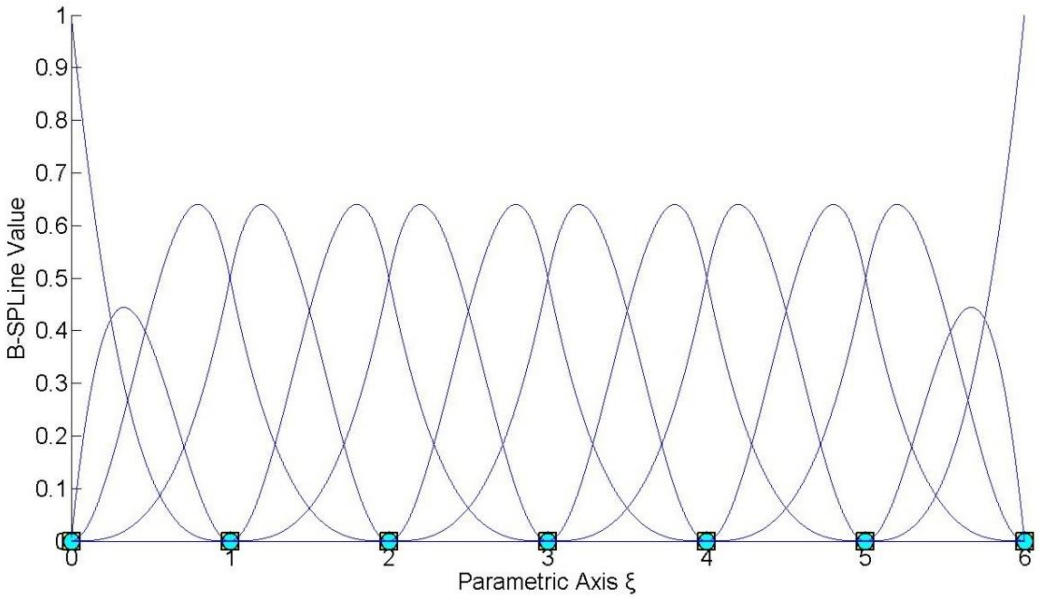


(b) Fine mesh. Polynomial order p=3.

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

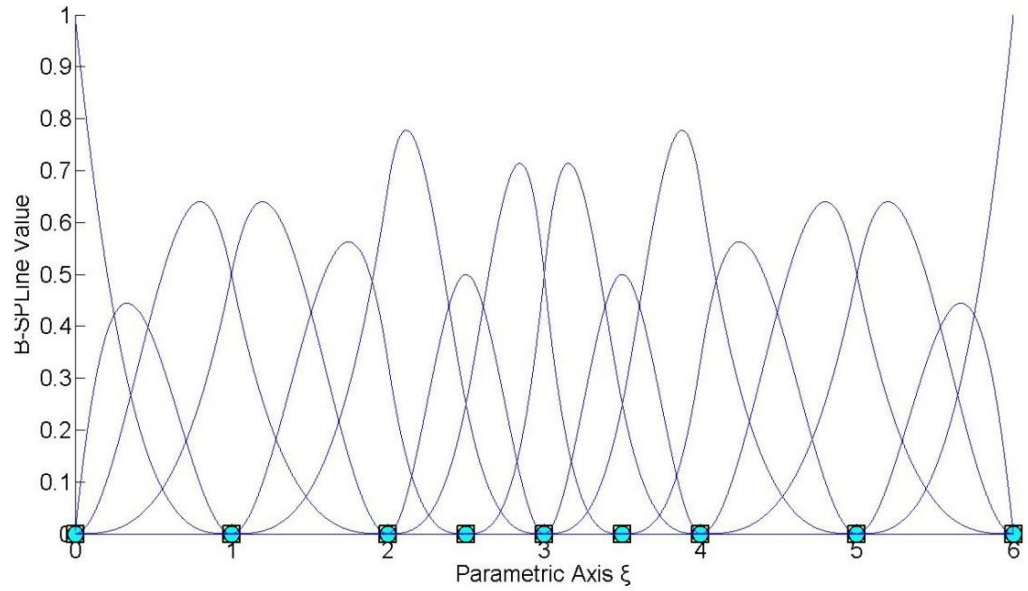
Figure 2.57. k-refinement on a B-SPLine curve.

In **Figure 2.57**, p-refinement is performed first, so the order is elevated to $\bar{p} = 3$. The C^1 continuity that existed in the coarse mesh is maintained. Afterwards, h-refinement is applied, with the insertion of two extra knot values. Continuity across this knot values is C^2, thus taking greater advantage of the new polynomial order. The combination of p- and h- refinement brings their best features together. This is why k-refinement is considered so effective.



(a) p-refinement to p=3

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



(b) h-refinement

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

Figure 2.58. B-SPLine basis functions for the two stages of k-refinement.

After p-refinement, internal knots still possess C^1 continuity. Note that only two basis functions are non-zero across every knot. After h-refinement, two new C^2 continuity knots are introduced. Three B-SPLine basis functions are non-zero across these knots, ξ=2.5 and ξ=3.5. This way, greater levels of continuity for the new polynomial order are utilized.

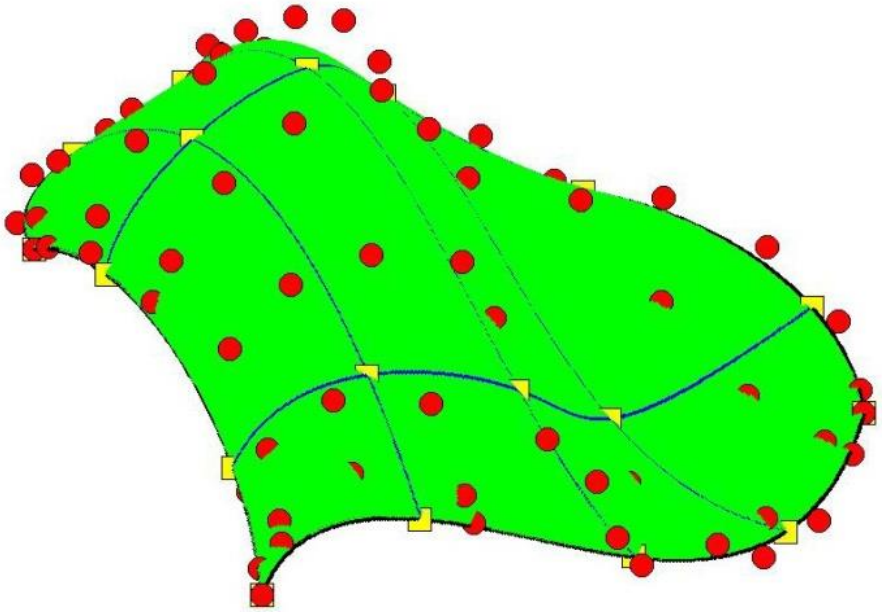


Figure 2.59. k-refinement for a NURBS surface.

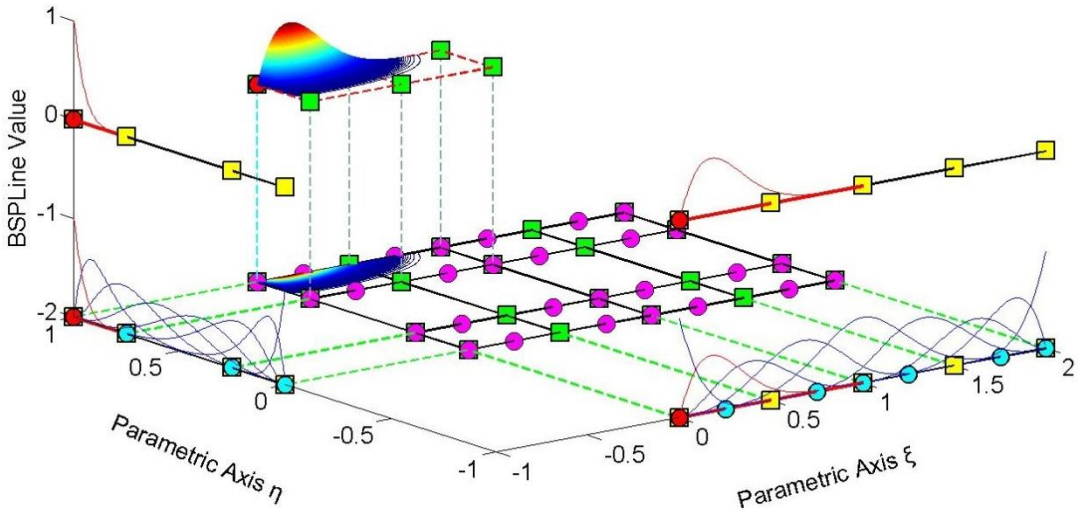


Figure 2.60. Basis functions for surface k-refinement.
Knot value vector for axis ξ.

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

Knot value vector for axis η.

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

In **Figure 2.59**, the application of k-refinement for a surface is represented. In fact, h-refinement is performed after the p-refinement of **Figure 2.58**. When p-refinement is applied, continuity of the internal knots is still C^2. After h-refinement, continuity of the additional control points in axes ξ, η is increased to C^3 and C^4 respectively. This augmentation of the continuity has proven to be very helpful for the analysis.

2.5.5 Reverse Refinement

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

Reverse refinement does not always provide an accurate solution. Removal of certain knots or downgrading of polynomial order may affect the shape of the entity. It is obvious, for example, that reverse p-refinement is not possible if there are knots of C^{p-1} continuity involved. In general, reverse refinement is an over determined problem and can only be solved in a certain number of cases. It is generally applicable when refinement has already been implemented and the engineer wants, for some reason, to return to the initial state of the model. The procedure we generally follow is similar to refinement.

In order to calculate the coarse mesh control points, a transformation matrix from coarse to fine mesh 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 from fine to coarse mesh, $\underset{(m\times n)}{[T_{FC}]}$.

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

Therefore,

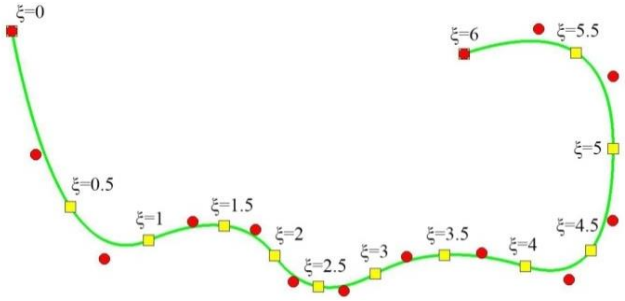
$$\underset{(n\times 1)}{\{X\}} = \left(\underset{(n\times m)}{[T_{FC}]^T} \cdot \underset{(m\times n)}{[T_{FC}]} \right)^{-1} \cdot \underset{(n\times m)}{[T_{FC}]^T} \cdot \underset{(mx1)}{\{\overline{X}\}}$$

and

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

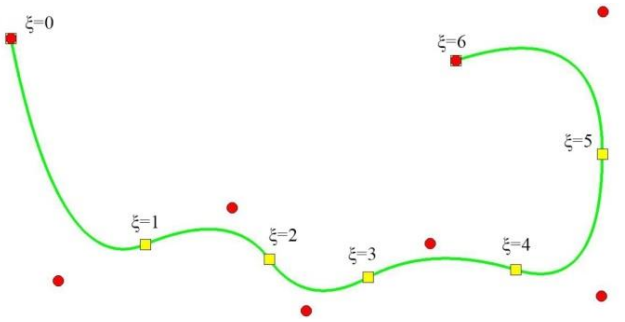
Reverse refinement works when going back-and-forth between coarse and fine mesh, but it is not always applicable. The curve in **Figure 2.61.c** has failed to accurately represent the required geometry.

Refinement reverse should be carefully used as a tool of intercommunication between initial and refined model.



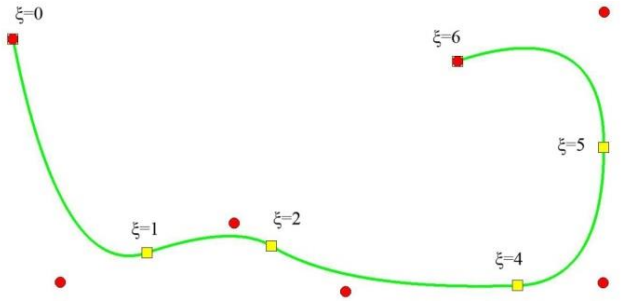
(a) Fine mesh.

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



(b) Reverse refined coarse mesh.

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



(c) Reverse-refined failed mesh.

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

Figure 2.61. Reverse h-refinement.
Success and failure of reverse refinement.

2.5.6 Refinement with NURBS

Everything mentioned about refinement is applicable to B-SPLine entities.

The question is what happens with NURBS. Different weight values and the complexity of NURBS shape functions prevent the refinement straight at the NURBS entity. However, every NURBS is created from the projection of a B-SPLine; therefore, NURBS refinement can be achieved by refining the corresponding projective B-SPLine curve.

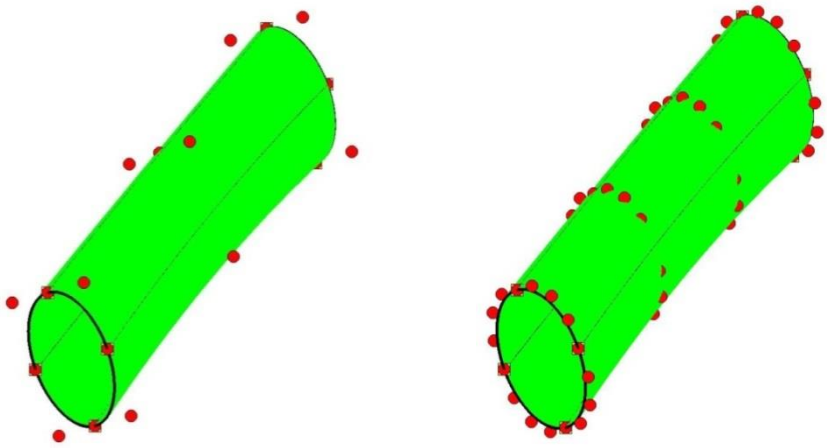
The first step is to evaluate the projective control points $\{B^w\}$, by multiplying every coordinate with the corresponding weight, as shown below:

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

As already mentioned, weights are the fourth-coordinate of the projective curve.

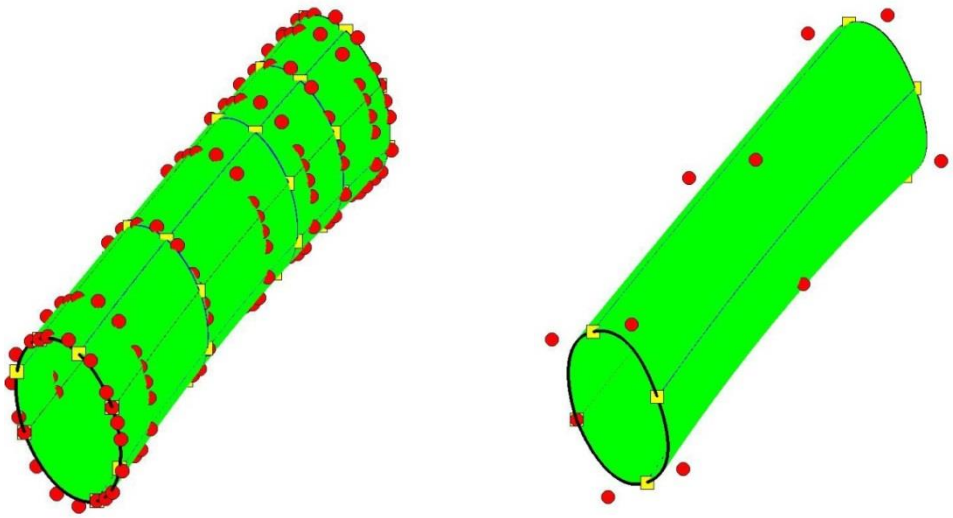
Refinement is applied for each set of the four coordinates, $\{X^w \quad Y^w \quad Z^w \quad W\}$. Afterwards, updated NURBS control point coordinates and weights are obtained by dividing the Cartesian coordinates with the weights:

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



(a) **(b)**

Figure 2.62. NURBS surface.
(a) Coarse mesh and (b) p-refinement as a part of k-refinement.



(a) **(b)**

Figure 2.63. NURBS surface.
(a) k-refinement (b) failed reverse refinement

3 Application Programming Interface

3.1 Object Oriented Programming

In computing OOP (object-oriented programming), we call a programming model which appeared in the late 1960s and introduced in the 1990s, largely by replacing the traditional model of structured programming. The most important feature of this program development methodology, is the fact that is backed by appropriate programming languages where the handling of related data and processes acting on them is shared via a data structure that surrounds them as an autonomous entity with its own identity and characteristics. This data structure is called "object" and it constitutes a real reaction in the definition of a complex, and probably designated by the user, data type that is called "class". The class specifies both data and procedures which act upon them and that has been the primary innovation of the OOP.

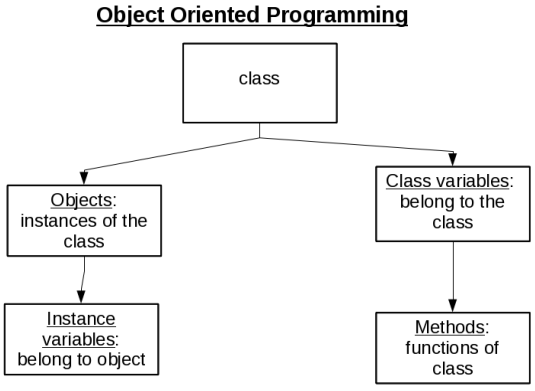


Figure 3.1. Object oriented programming.
(http://www.ping127001.com/python/pythonmain.html)

By its definition, OOP specifies a storage structure (e.g. a class "Door") that contains both properties (for example, a variable "Color of the Door") and acts or operations on these properties (e.g. a process of "Opening the Door"). In this example every physical Door (each object actually stored in memory) is represented as a separate "natural" snapshot of the model, the virtual class. Therefore only the objects occupy space in computer memory while classes are simply "molds". The main causes that led to the development of OOP were the same as those that led to the development of structured programming (ease of maintenance, organization, handling and reuse large and complex applications code), but ultimately the object orientation prevailed as it could easily afford much larger volume projects and complexity handling.

3.1.1 History

Most object-oriented concepts first appeared in the programming language Simula 67, which was oriented to perform simulations of the real world. The ideas of Simula 67 influenced by the 70s the development of Smalltalk, the language that first introduced the term object-oriented programming. The Smalltalk developed by Alan Kay of Xerox company in the purpose of creating a useful, and intuitive, personal computer. When the final version of Smalltalk became available in 1980, research for the replacement of structured programming with a more modern model was already underway. In this language, all data types were classes as objects and no longer as traditional data structures.

Around the same time, and also influenced by Simula, the development of C ++ as a strong expansion of the popular C programming language in which they were "transplanted" object-oriented features was finally completed. The influence of C ++ throughout the 80s was catalytic with as a result the gradual development of object-oriented versions of many well-known procedural programming languages. In the first half of the 90s the gradual introduction of graphic user interfaces (GUI), to microcomputers for the development of which the OOP seemed particularly appropriate, and the effect of C ++, led to the prevalence of object orientation as a key integrated programming model.

In 1995 the appearance of Java, a highly successful, fully object-oriented language that resembled structurally the C/ C ++ and offered pioneering opportunities, gave new impetus to the OOP. At the same time a variety of informal improvements in basic programming model, such as object-oriented software modeling languages, design patterns etc, appeared. In 2001, Microsoft focused on the .NET platform, a rival of Java development and execution platform software, which was entirely oriented in object orientation.

3.1.2 Features

Central idea in object-oriented programming is the "class", one independent and abstract representation of a class of objects or natural objects of the real world or imaginary, conceptual objects in a programming environment.

Practically, class is a data type, or else the draft of a data structure with its own contents, variables and procedures. The contents shall be declared either as "public" or as "private", and by the word "private" we mean that private contents are not accessible by code outside the class. The procedures in classes are usually called "methods" and their variables "attributes" or "fields". A class should ideally be conceptually separate, meaning that it should contain only fields that describe a category of objects and public methods, which act on them when invited by the external program, without being dependent on other data or code outside the class, and reusable. It must be capable of being operated without modifications as part of different programs.

Object is an instance of a class, that is itself data structure with exclusively reserved memory space based on the "mold" that offers the class. For example, in an object-oriented programming language we could define a class named Library, which represents a library of books and declare an object with name MyBook. This object will block memory space by the number of variables and methods described when we declared class. Thus, the object could present a Writer feature (the writer of the book) and a method GetWritersName (return the name of the writer of the book). Then we could even create one or more objects of the same class which are different data structures (different books in the example). The objects of a class can access private contents of other objects of the same class.

Data Encapsulation is called the ability that classes provide to "hide" their private data from the rest of the program and ensure that only through their public methods they can be accessed. This tactic has only benefits because it forces any external program to filter handling in the fields of a class through checks that can be contained in the public methods of the class.

Removing data is called the ability of classes to represent abstract entities in complex programming environment. A class is an abstract model of a category of objects. Also classes offer abstraction to the computer, as each one them could be considered a small and self-sufficient computer with his own situation, methods and variables.

Inheritance is called the ability that classes have to expand into new classes, explicitly declared as heirs (subclasses or 'subsidiaries classes'), which can reuse transferable methods and properties of the parent class and add their own. Snapshots of subsidiaries classes may be used where snapshots of parental are required (if the subsidiary is somehow a more specific version of the parent), but the reverse is not true. An example of inheritance is a parent class Vehicle and the two more specialized subclasses of Car and Bicycle, which "inherit" from it. Multiple inheritance is the ability that some programming languages offer, a class to inherit simultaneously from more than one parent. From a subclass can arise new subclasses inheriting from it, resulting the existence of a class hierarchy that linkes them together "by generation" with inheritance relationships.

Method Overloading is the condition in which they exist, in the same or different classes, methods with the same name and possibly different arguments. As far as methods of the same class concerned, they are differentiated only by their differences in the arguments and return type.

Method overriding is the condition under which a subsidiary class and its parent have a method with the same name and the same arguments. Thanks to the possibility of polymorphism compiler "knows" when to call the appropriate method, based on the type of the current object. So polymorphism is the ability of object-oriented compiler to dynamically decide which is the appropriate method to be called under supplant conditions.

Abstract class is a class defined only in order to be inherited in subsidiaries subclasses and it has not its own instances (objects).

The abstract class simply defines a "contract" which will be followed by the subclasses regarding to their methods signatures (where by the word signature we mean the name, the arguments or the return value of a procedure). An abstract class can have non-abstract methods that are implemented in the same class (although of course can be overridden in subclasses). Instead the abstract method is simply a definition of signature and it is up to subclasses to implement. An abstract class that has attributes and all its methods are abstract and public is called "interface". The classes that inherit from an interface are said to "implement".

Here's a simple example in programming language Java.

```
interface Logger
{
   public void log(String msg);
}

class ConsoleLogger implements Logger
{
   public void log(String msg)
   {
      System.err.println("\nConsole logging..." + msg + "\n");
   }
}

class FileLogger implements Logger
{
   public void log(String msg)
   {
       System.out.println("\nFile logging..." + msg + "\n");
   }
}

public class LogTest
{
    public static void main(String[] args)
    {
       if(args.length != 1)
       {
           System.out.println("\nError. Exiting...");
           return;
       }
       Logger logg;
       String choice = args[0];
       if(choice.equals("FileLogger"))
          logg = new FileLogger();
       else if(choice.equals("ConsoleLogger"))
          logg = new ConsoleLogger();
       else
       {
          System.err.println("\nError. Exiting...");
          return;
       }
       logg.log("Log This!");
    }
}
```

Figure 3.2. Source code in Java.
(https://el.wikipedia.org/wiki/Αντικειμενοστραφής_Προγραμματισμός)

3.1.3 Object Oriented Design Principles

Through time, some informal principles have been codified for the proper design of object-oriented software systems. These principles have been at times presented in books and recognized academic software engineer's articles. The most important principles are the following:

- **Open-closed principle** from the creator of the programming language Eiffel Bertrand Meier. This principle states that the components of a program should be "open" on the expansion of system capabilities, but "closed" regarding changes in implementation. Practically this means that it will not be necessary various classes and other software components to be modified in case new functionality will be added to the system (e.g. a new class) in order to utilize it. Of course, it is impossible not to have to modify anything, so what actually this principle requires is the minimization and the gathering of the lines that should be changed, preferably in a small section of code. This is usually accomplished by subtracting (with abstract classes or interfaces and real classes inheriting from them) and using polymorphism.

- **Liskov substitution principle**, from the computer scientist Barbara Liskof. This principle is condensed in the following rule for a proper hierarchy formation of classes: one class C1 may be implemented as a subclass of a class C2 if each program P which operates with objects K2 behaves the same way and with similar objects K1. So with Liskof substitution principle, it seems that in order to define a class as a subsidiary of another it is not enough to have intuitively a similar conceptual relationship (e.g. a class that represents a vehicle and another that represents a car) but in the context of this program, objects of the subclass must always have the same programming behavior as objects of the superclass in the same conditions have.

- **Dependency Inversion Principle**, from the famous software engineer Robert Cecil Martin. This principle is practically a sophistication of open-closed principle, assuming of course the use of Liskov substitution principle. It concerns class inheritance hierarchies and the use of object hierarchies from external programs. Within the inversion, principle dependencies a software section A (e.g. a class) that uses the services provided by another software component B, for example calling a method, is considered item "higher level" element in comparison with B. The principle says that high-level elements must not depend on lower-level elements' implementation, but both should be based on intermediate levels of abstraction. In practice this abstraction is an interface (or abstract class), which high level component A knows and low-level component B implements. Even if B is changed to a class C, which also implements the same interface, A will have to continue to operate without modification. The dependency inversion principle is nothing, but a tangible example of the use of hierarchical levels with the help of intermediate deductions, a practice applied extensively in computer science.

- **Interface segregation principle**, from the software engineer Robert Cecil Martin. This principle means that in cases, where different subsets of methods of a class concern different use cases of the class, it is appropriate to define individual interfaces that the class will implement. Any such interface only defines the corresponding subset of methods.

- **Single responsibility principle**, of Tom de Marco and Meir Paige Jones. According to this principle, every class should have only one, well defined and separated from the rest of the program responsibility, the existence of which serves a specific purpose. If we can identify in a class A two different competencies, then the best solution is to split into two classes B and C, each of which will receive a subset of the fields and the A methods. The subsets will be disjoint, so the reverse thinking means that if we can split a class A in two other classes (e.g. if some methods do not use some features, then these features can end up in class B and the fields in another class C), then probably class will undermine the principle of unique competence. So, some metrics have been proposed that attempt to identify the inconsistency (cohesion) in a class, that is whether the methods are not associated with the traits. Usually consistency contrasted with conjugation (coupling), i.e. the extent to which a class depends on one / any other / s, both of these quantities are inversely proportional.

3.2 C++ Object Oriented Language

C++ is a general-purpose programming language. It is based on the C language and it is object oriented. It includes facilities for low level memory manipulation. Its design is influenced by system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights. The main advantage of C++ is the fact that it is a software infrastructure and basis for creating resource-constrained applications, like desktop applications, servers performance-critical applications, and entertainment software. C++ is a compiled language, with implementations of it available on many platforms and provided by various organizations, including the FSF, LLVM, Microsoft, Intel and IBM.

C++ is standardized by the International Organization for Standardization (ISO), with ISO/IEC 14882:2014. It is also known as C++14. The C++ programming language was initially standardized in 1998 as ISO/IEC 14882:1998. The current C++14 standard supersedes all previous standards with new features and an enlarged standard library.

C++ was developed by Bjarne Stroustrup at Bell Labs in 1979, as an extension of the C language. His goal was an efficient and flexible language similar to C, which also provided high-level features for program organization. Many other programming languages have been influenced by C++, including C#, Java, and newer versions of C (after 1998).

3.2.1 History

Bjarne Stroustrup, a Danish computer scientist, began his work on C++ in 1979. While he was researching for his Ph.D, he found out that Simula had feautures appropriate for large software developments. It had though a major drawback; it was too slow for practical use. When he compared it with BCPL, BCPL was faster but also low level, thus not suitable for large software development. When Stroustrup started working in AT&T Bell Labs, he had the problem of analyzing the UNIX kernel with respect to distributed computing. Inspired by his PhD, he decided to enhance C language with features from Simula. He chose to work with C, because it is a general purpose language, fast and widely used. Initially, C++ was named "C with Classes". This first version added features like the class, derived class, strong typing, inlining and default argument features in the C Compiler. The name changed from "C with Classes" to C++ in 1983. "++" is the increment operator in C. Also a "+" is a common naming convention to indicate an enhanced computer program. The new version had new features were such as virtual functions, function name and operator overloading, references, constants, type-safe free-store memory allocation (new/ delete), improved type checking, and BCPL style single-line comments with two forward slashes (//), as well as the development of a proper compiler for C++, which it was named Cfront.

In 1985, the first edition of "The C++ Programming Language" was released in the market. At that point, there was not yet an official standard. The first commercial implementation of C++ was released later that year, in October. In 1989, C++ 2.0 was released, and it was updated in 1991. The new features of C++ 2.0 were the multiple inheritance, abstract classes, static member functions, const member functions, and protected members. In 1990, the Annotated C++ Reference Manual was published. This work became the basis for the future standard. Later feature additions included templates, exceptions, namespaces, new casts, and a boolean type. After the 2.0 update, C++ evolved relatively slowly. In 2011, the C++11 standard was released, adding numerous new features, enlarging the standard library further, and providing more facilities to C++ programmers. After a minor C++14 update, released in December 2014, various new additions are planned for 2017.

3.2.2 Philosophy

There is a set of guidelines and rules that govern every version of C++ and it is written below.

- It must be driven by actual problems and its features should be useful immediately in real world programs.
- Every feature should be implementable (with a reasonably obvious way to do so).
- Programmers should be free to pick their own programming style, and that style should be fully supported by C++.
- Allowing a useful feature is more important than preventing every possible misuse of C++.
- It should provide facilities for organising programs into well-defined separate parts, and provide facilities for combining separately developed parts.

- No implicit violations of the type system (but allow explicit violations; that is, those explicitly requested by the programmer).
- User-created types need to have the same support and performance as built-in types.
- Unused features should not negatively impact created executables (e.g. in lower performance).
- There should be no language beneath C++ (except assembly language).
- C++ should work alongside other existing programming languages, rather than fostering its own separate and incompatible programming environment.
- If the programmer's intent is unknown, allow the programmer to specify it by providing manual control.

3.2.3 Language

The C++ language has two main components, a direct mapping of hardware features provided primarily by the C subset and zero-overhead abstractions based on those mappings. Stroustrup describes C++ as "*a light-weight abstraction programming language [designed] for building and using efficient and elegant abstractions*" and "*offering both hardware access and abstraction is the basis of C++. Doing it efficiently is what distinguishes it from other languages*". C++ inherits most of C's syntax. “Hello world program” is the first program beginners write in C++. It simply prints “Hello world” on the computer screen. Although it is very simple, it contains all the fundamental components of C++. The following is Stroustrup's version of the “Hello world program” that uses the C++ Standard Library stream facility to write a message to standard output.

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
}
```

Figure 3.3. Source code in C++.
(https://en.wikipedia.org/wiki/C%2B%2B)

In the functions that define a non-void return, a failure to return a value before the end of running a program results in undefined behavior. The compiler will provide the diagnosis of the problem. There is only one exception in this rule and that is, the main function which implicitly returns a value of zero.

3.2.4 Object Storage

Similar to C, C++ supports four types of memory management:

- Static storage duration objects
- Thread storage duration objects
- Automatic storage duration objects
- Dynamic storage duration objects

Static Storage Duration Objects

Static storage duration objects are created before main is entered and destroyed in reverse order of creation after main exits. The exact order of creation is not specified by the standard to allow implementations some freedom in how to organize their implementation. Objects of this type have a lifespan that last for the duration of the program. Static storage duration objects are initialized in two phases.

- *Static Initialization*
  All objects are first initialized with zeros. After that, all objects that have a constant initialization phase are initialized with the constant expression (i.e. variables initialized with a literal or constexpr). Though it is not specified in the standard, this phase can be completed at compile time and saved in the data partition of the executable. It is always the first phase and it cannot be done after dynamic initialization.

- *Dynamic Initialization*
  All object initialization that is done via a constructor or function call. The dynamic initialization order is defined as the order of declaration within the compilation unit. No guarantees are provided about the order of initialization between compilation units.

Thread Storage Duration Objects

Variables of this type are very similar to static storage duration objects. The main difference is that the creation time is just prior to thread creation and destruction is done after the thread has been joined.

Automatic Storage Duration Objects

The most common variable types in C++ are local variables inside a function or block, and temporary variables. The common feature about automatic variables is that they have a lifetime that is limited to the scope of the variable. They are created and potentially initialized at the point of declaration and destroyed in the reverse order of creation when the scope is left.

There are three kinds of variables in the type of memory management:

- Local variables are created as the point of execution passes the declaration point. If the variable has a constructor or initializer, this is used to define the initial state of the object. Local variables are destroyed, when the local block or function that they are declared in is closed. C++ destructors for local variables are called at the end of the object lifetime, allowing a discipline for automatic resource management termed RAII, which is widely used in C++.

- Member variables are created, when the parent object is created. Array members are initialized from 0 to the last member of the array in order. Member variables are destroyed, when the parent object is destroyed in the reverse order of creation.

- Temporary variables are created as the result of expression evaluation and are destroyed when the statement containing the expression has been fully evaluated (usually at the ; at the end of a statement).

Dynamic Storage Duration Objects

These objects have a dynamic lifespan and are created with new call and destroyed with an explicit call to delete.

3.2.5 Templates

C++ templates enable generic programming. C++ supports function templates and class templates. Templates may be parameterized by types, compile-time constants, and other templates. Templates are implemented by instantiation at compile-time. In order to instantiate a template, some specific arguments are substituted by compilers, so that the parameters of a template can generate a concrete function or class instance. In some cases substitution is not possible; it is eliminated by an overload resolution policy known as SFINAE (Substitution Failure Is Not An Error). Templates are a powerful tool that can be used for generic programming, template metaprogramming, and code optimization, but this power implies a cost. Template use may increase code size, because each template instantiation produces a copy of the template code: one for each set of template arguments, however, this is the same or smaller amount of code that would be generated if the code was written by hand. Templates are used in static polymorphism and generic programming.

Templates have a huge difference from macros: While both are compile-time language features and can enable conditional compilation, templates are not restricted to lexical substitution.

Templates are aware of the semantics and type system of their companion language, as well as all compile-time type definitions, and can perform high-level operations including programmatic flow control based on evaluation of strictly type-checked parameters. Macros on the other hand are capable of conditional control over compilation based on predetermined criteria, but cannot instantiate new types, recurse, or perform type evaluation and in effect are limited to pre-compilation text-substitution and text-inclusion/exclusion. In other words, macros can control compilation flow based on pre-defined symbols but cannot, unlike templates, independently instantiate new symbols.

Furthermore, templates are a compile time mechanism in C++ that is ”Turing-complete”. That means that any computation expressible by a computer program can be computed, in some form, by a “template metaprogram” prior to runtime.

To sum it up, a template is a compile-time parameterized function or class written without knowledge of the specific arguments used to instantiate it. After instantiation, the resulting code is equivalent to code written specifically for the passed arguments. In this manner, templates provide a way to decouple generic, broadly applicable aspects of functions and classes (encoded in templates) from specific aspects (encoded in template parameters) without sacrificing performance due to abstraction.

3.2.5.1 Objects

C++ is an object-oriented programming language, unlike C. The main difference is the provision of classes, which provide the four features of object oriented programming. These features are:

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Another variation of C++ from other languages that work with classes is that C++ supports deterministic destructors. These destructors provide support for the RAII (Resource Acquisition Is Initialization) concept.

3.2.5.2 Encapsulation

Encapsulation is the feature of hiding information to ensure that data structures and operators are used as intended and to make the usage model more obvious and simple to the developer. With the primary encapsulation mechanisms of C++ classes and functions are defined.

Within a class, members can be declared as either public, protected, or private to explicitly enforce encapsulation. A public member of the class is accessible to any function. A private member is accessible only to functions that are members of that class and to functions and classes explicitly granted access permission by the class "friends". A protected member is accessible to members of classes that inherit from the class in addition to the class itself and any friends.

The object oriented principle is that all the functions that have access to an internal representation of a type should be encapsulated within the type definition. C++ supports this (through member functions and friend functions), but does not enforce it: the programmer can declare parts or all of the representation of a type to be public, and is allowed to make public entities that are not part of the representation of the type. Therefore, C++ supports not just object oriented programming, but other weaker decomposition paradigms, such as modular programming.

Usually it is suggested that all data should be private or protected and only the function that are part of a minimal interface for users of the class to be public. This way the details of data implementation are hidden and the designer can fundamentally change the implementation at a later time without any disturbance of the interface.

3.2.5.3 Inheritance

Inheritance is the feature of object oriented programming that allows one data type to acquire properties of other data types. Inheritance from a base class may be declared as public, protected, or private. By specifying the access, it is being determined whether unrelated and derived classes can access the inherited public and protected members of the base class. The most frequent use of inheritance is the public. The other two forms are much more rarely used. If the access specifier is omitted, a "class" inherits privately, while a "struct" inherits publicly. Base classes may be declared as virtual; this is called virtual inheritance. Virtual inheritance ensures that only one instance of a base class exists in the inheritance graph, avoiding some of the ambiguity problems of multiple inheritance.

Multiple inheritance is a unique feature of C++, which cannot be found in most other languages. It allows to a class to be derived from more than one base class; which leads to more elaborate inheritance relationships. There is a similar feature in languages such as C# and Java that allows inheritance of multiple interfaces while restricting the number of base classes to one. An interface can also be defined in C++ as a class containing only pure virtual functions, often known as an abstract base class or "ABC". Interfaces, unlike classes, provide only declarations of member functions, no implementation or member data. The member functions of such an abstract base class are normally explicitly defined in the derived class, not inherited implicitly.

C++ provides more than 35 operators, covering basic arithmetic, bit manipulation, indirection, comparisons, logical operations and others. Almost all operators can be overloaded for user-defined types, with a few notable exceptions such as member access (“and”) as well as the conditional operator. The set of overloadable operators is crucial for making user-defined types in C++ look like built-in types.

Overloadable operators are also an essential part of many advanced C++ programming techniques. Overloading an operator does not change the precedence of calculations involving the operator, nor does it change the number of operands that the operator uses. Overloaded "&&" and "||" operators lose their short-circuit evaluation property.

3.2.5.4 Polymorphism

Polymorphism enables one common interface for many implementations and for objects to act differently under different circumstances. There are two kinds of polymorphism in C++: static and dynamic.

C++ supports several kinds of static (compile-time) and dynamic (run-time) polymorphisms. Compile-time polymorphism does not allow for certain run-time decisions, while run-time polymorphism typically incurs a performance penalty.

Static Polymorphism

Static polymorphism is a compile-time polymorphism. Function overloading allows programs to declare multiple functions having the same name but with different arguments (i.e. ad hoc polymorphism). The functions are distinguished by the number or types of their formal parameters. Thus, the same function name can refer to different functions depending on the context in which it is used. The type returned by the function is not used to distinguish overloaded functions and would result in a compile-time error message.

When declaring a function, a programmer can specify for one or more parameters a default value. Doing so allows the parameters with defaults to optionally be omitted when the function is called, in which case the default arguments will be used. When a function is called with fewer arguments than the declared parameters, explicit arguments are matched to parameters in left-to-right order, with any unmatched parameters at the end of the parameter list being assigned their default arguments. In many cases, specifying default arguments in a single function declaration is preferable to providing overloaded function definitions with different numbers of parameters.

Templates in C++ provide a sophisticated mechanism for writing generic, polymorphic code (i.e. parametric polymorphism). In particular, through the CRTP (Curiously Recurring Template Pattern), it's possible to implement a form of static polymorphism that closely imitates the syntax for overriding virtual functions. Due to the fact that C++ templates are type-aware and Turing-complete, they can also be used to let the compiler resolve recursive conditionals and generate substantial programs through template metaprogramming.

Dynamic polymorphism

Dynamic polymorphism is a run-time polymorphism. Variable pointers to a base class type in C++ can refer to objects of any derived classes of that type in addition to objects exactly matching the variable type. This allows arrays and other kinds of containers to hold pointers to objects of differing types. Because assignment of values to variables usually occurs at run-time, this is part of dynamic polymorphism.

C++ also provides a dynamic cast operator, which allows the program to safely attempt conversion of an object into an object of a more specific object type. The inverse procedure, of converting an object to a more general type is always allowed. This feature relies on RTTI (Run Time Type Information). Objects known to be of a certain specific type can also be cast to that type with static cast, which is a purely compile-time construct that has no runtime overhead and does not require RTTI.

3.2.5.5 Virtual member functions

When a function in a derived class overrides a function in a base class, the function to call is being determined by the type of the object. A given function is overridden when there is no difference in the number or type of parameters between two or more definitions of that function. While the compiler runs, it may not be possible to determine the type of the object by a base class pointer and therefore the correct function to call. In this case the decision is put off until runtime. This procedure is called dynamic dispatch. Virtual member functions allow the most specific implementation of the function to be called, according to the actual run-time type of the object. In C++ implementations, this is commonly done using virtual function tables. If the object type is known, this may be bypassed by prepending a fully qualified class name before the function call, but in general calls to virtual functions are resolved at run time.

Not only standard member functions can be virtual, but also operator overloads and destructors. As a rule of thumb, if any function in the class is virtual, the destructor should be as well.

As the type of an object at its creation is known at compile time, constructors, and by extension copy constructors, cannot be virtual. Nonetheless a situation may arise where a copy of an object needs to be created when a pointer to a derived object is passed as a pointer to a base object. In such a case, a common solution is to create a clone virtual function that creates and returns a copy of the derived class when called.

A pure virtual function is a member function with a "=0" between the closing parenthesis and the semicolon. A class containing a pure virtual function is called abstract data type. Objects cannot be created from abstract data types; they can only be derived from. A program that attempts to create an object of a class with a pure virtual member function or inherited pure virtual member function is ill-formed.

3.2.6 Lambda Expressions

C++ provides support for anonymous functions, which are also known as lambda expressions. A lambda expression has the following form:

```
[capture](parameters) -> return_type { function_body }
```

Figure 3.4. Source code in C++.
(https://en.wikipedia.org/wiki/C%2B%2B)

The [capture] list supports the definition of closures. Such lambda expressions are defined in the standard as syntactic sugar for an unnamed function object. An example lambda function may be defined as follows:

```
[](int x, int y) -> int { return x + y; }
```

Figure 3.5. Source code in C++
(https://en.wikipedia.org/wiki/C%2B%2B)

3.2.7 Exception Handling

Exception handling is used for the purpose of communicating the existence of a runtime problem or error from, where it was detected to where the issue can be handled. It is done in a uniform manner by permission and separately from the main code, while detecting all errors. If an error occurs, an exception is thrown (raised), which is then caught by the nearest suitable exception handler.

The exception causes the current scope to exit, and also each outer scope (propagation) until a suitable handler is found, calling in turn the destructors of any objects in these exited scopes. At the same time, an exception is presented as an object carrying the data about the detected problem. The exception-causing code is placed inside a “try” block. The exceptions are handled in separate “catch” blocks (the handlers). Each “try” block can have multiple exception handlers, as it is shown in figure 4.6. There are some cases, where exception cannot be used, due to technical reasons. For example, it cannot be used in a critical component of an embedded system, where every operation must be guaranteed to complete within a specified amount of time. This cannot be determined with exceptions because there are no tools to determine the minimum time required for an exception to be handled.

3.3 OpenGL

3.3.1 Introduction

OpenGL (Open Graphics Library) is a cross-language, multi-platform Application Programming Interface (API) for rendering 2D and 3D vector graphics. The API interacts with a Graphics Processing Unit (GPU) in order to achieve hardware-accelerated rendering. It consists of more than 700 distinct commands. These commands are used for specification of the objects and operations needed to produce interactive three-dimensional applications. OpenGL was developed from Silicon Graphics Inc. (SGI) in 1991 and released in January 1992. OpenGL is managed by the non-profit technology consortium Khronos Group. It is used extensively by applications in the fields of Computer Aided Design (CAD), Virtual Reality, Scientific Visualization, Information Visualization, Flight Simulation and Video Games. OpenGL is designed as a streamlined, hardware-independent interface. Its purpose is to be implemented on many hardware platforms. The user cannot perform any windowing tasks or obtain user input. Because of this, commands for these operations do not exist. Instead the user must work through the windowing system controls that the hardware is using. In addition, OpenGL doesn’t provide high level commands for describing models of three dimensional objects. For that reason the user can’t specify relatively and design complicated shapes such as automobiles, parts of the body, airplanes or molecules. The OpenGL provides to the user only a set of primitive geometries (points, lines and polygons) for the design of his model. The OpenGL Utility Library (GLU) is a sophisticated library that provides all the features that OpenGL doesn’t. It can be built on top of OpenGL and can provide many of the modeling features such as quadric surfaces and Nurbs Curves and Surfaces.

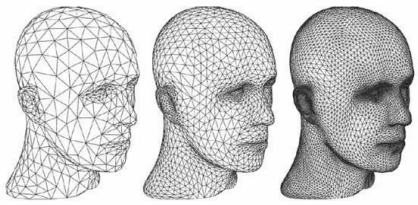


Figure 3.6. Mesh designed with OpenGL.
(http://www.xperiac.com/tus-juegos-al-maximo-como-aprovechar-al-100-el-hard-de-la-pc/)

3.3.2 An Introduction to OpenGL Code

A program in OpenGL can have a simple basic structure with tasks. The purpose of these tasks is to initialize the states that control how OpenGL renders and which objects should be rendered. Nevertheless, the procedure will lead to a complicate program if the code is not organized accordingly.

"Rendering" is the process where images are created from models. The models are created with primitive geometries, such as points, lines and polygons. The result of rendering, the image, will consist of pixels. Pixel is the smallest visible element that can be put on a screen.

The information of all the pixels of the image is organized in bitplanes. Bitplanes are areas of the memory that hold a bit of information for every pixel on the screen. This information can be about the color or the place. All bitplanes are further organized into a framebuffer, which holds the total amount of information about the image (e.g. the colors and intensity of all pictures)

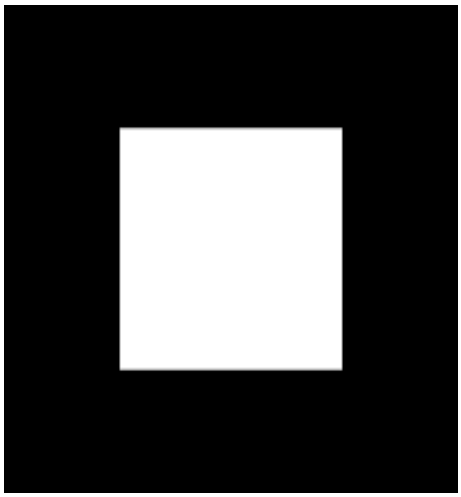


Figure 3.7. Rectangle made with OpenGL.

Example 1-1 Chunk of OpenGL Code

```
#include <whateverYouNeed.h>
main() {
InitializeAWindowPlease();
glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
glBegin(GL_POLYGON);
glVertex3f(0.25, 0.25, 0.0);
glVertex3f(0.75, 0.25, 0.0);
glVertex3f(0.75, 0.75, 0.0);
glVertex3f(0.25, 0.75, 0.0);
glEnd();
glFlush();
UpdateTheWindowAndCheckForEvents(); }
```

3.3.3 OpenGL Command Syntax

Usually command names in OpenGL use the prefix “gl” and every initial letter of a word is capital (e.g. glClearColor**()**). Similarly, the defined constants begin with “GL_”, use all capital letters, and use underscores to separate words (e.g. GL_COLOR_BUFFER_BIT).

Some OpenGL commands accept as many as eight different data types for their arguments. The letters used as suffixes to specify these data types for ISO C implementations of OpenGL are shown in Table 1-1, along with the corresponding OpenGL type definitions.

Open-GL Data Types

suffix	data type	C/C++ type	OpenGL type name
b	8-bit integer	signed char	GLbyte
s	16-bit integer	Short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit float	Float	GLfloat, GLclampf
d	64-bit float	Double	GLdouble,GLclampd
ub	8-bit unsigned number	unsigned char	GLubyte,GLboolean
us	16-bit unsigned number	unsigned short	GLushort
ui	32-bit unsigned number	unsigned int or unsigned long	GLuint,Glenum,GLbitfield

Figure 3.8. Open-GL data types.
(http://www.slideshare.net/kunalv508/hill-ch2ed3)

If an OpenGL command has a final letter “v”, that means that the command takes a pointer to a vector/ array of values and not to a series of individual arguments. Often commands have both vector and non-vector versions, but there are commands that strictly accept vector or individual arguments.

The following example shows the two versions of command that sets the color:

```
glColor3f(1.0, 0.0, 0.0);
GLfloat color_array[] = {1.0, 0.0, 0.0};
glColor3fv(color_array);
```

Finally, there is the type of GLvoid. This type is used for OpenGL commands that accept pointers to array of values.

3.3.4 OpenGL as a State Machine

OpenGL is a state machine. A state machine is a mathematical model of computation. It is used to design both computer programs and sequential logic circuits. Fixed function is an example of using OpenGL as a state machine. Fixed function consists of a set of function entry points that would map the logic of their named purpose in GPUs, that they are design to support them. Applying it to various model, they remain into effect until they are changed by the user. Such example of variables that control the color, the current viewing and projection transformations, line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn.

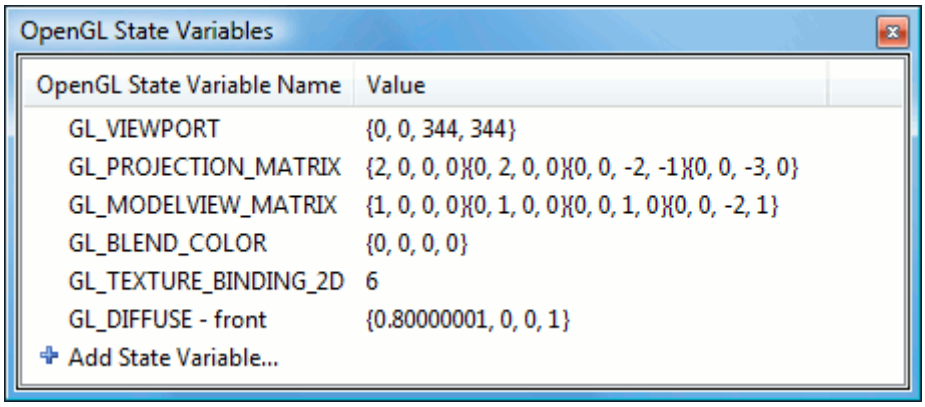


Figure 3.9. OpenGL commands.
(http://www.gremedy.com/screenshots.php)

Many state variables refer to modes that are enabled or disabled with the **commandglEnable**() or **glDisable**(). Each state variable or mode has a default value, and at any point the user can query the system for each variable's current value. In order to do that there are six commands to do this and each of them is used proprietary depending on what data type we want the answer to be given in:

1. **glGetBooleanv**()
2. **glGetDoublev**()
3. **glGetFloatv**()
4. **glGetIntegerv**()
5. **glGetPointerv**()
6. **glIsEnabled**()

Some state variables have a more specific query command (such as **glGetLight***(), **glGetError**(), or **glGetPolygonStipple**()). In addition, a collection of state variables can be saved on an attribute stack with **glPushAttrib**() or **glPushClientAttrib**(), temporarily be modified, and later restore its values with **glPopAttrib**() or **glPopClientAttrib**(). For temporary state changes, these commands should be used rather than any of the query commands, as they're likely to be more efficient.

3.3.5 OpenGl Rendering Pipeline

Most implementations of OpenGL have a similar order of operations. This order is a series of processing stages called the OpenGL rendering pipeline. As it is shown in **Figure 3.10**, the rendering pipeline is not a strict rule about how OpenGL is implemented, but it provides a reliable guide for predicting what OpenGL will do. The following diagram shows the Henry Ford assembly line approach, which OpenGL uses to processing data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps (rasterization and per-fragment operations) before the final pixel data is written into the framebuffer.

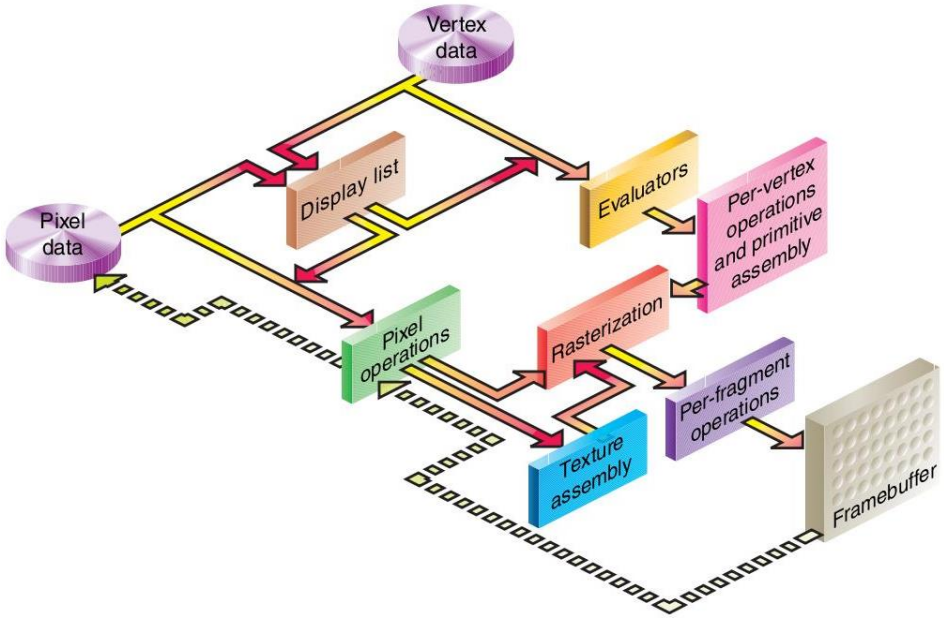


Figure 3.10. OpenGL rendering pipeline overview.
(http://amslaurea.unibo.it/6719/1/aguiari_davide_tesi.pdf)

Display Lists

All data, whether they describe geometry or pixels, can be saved in a display list for current or later use. The immediate mode is when the process of data is done immediately. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called *basis functions*. For more information see chapter 1 and 2. Evaluators provide a method for deriving the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal texture coordinates, colors, and spatial coordinate values from the control points.

Per-Vertex Operations

In the “per-vertex operations” the vertices are converted into primitives. Some types of vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D place to a position on your screen. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value. Since OpenGL Version 2.0, there is the option of using fixed-function vertex processing, as just previously described, or completely controlling the operation of the per-vertex operations by using vertex shaders. If shaders are employed, all of the operations in the per-vertex operations stage are replaced by the shader. In Version 3.1, all of the fixed-function vertex operations are removed and using a vertex shader is mandatory.

Primitive Assembly

Clipping, a major part of primitive assembly, is the elimination of portions of geometry that fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices. Line or polygon clipping can add additional vertices depending on how the line or polygon is clipped. In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then viewport and depth (*z*-coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending on the polygon mode, a polygon may be drawn as points or lines. The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

Pixel Operations

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the framebuffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixel copy operations for copying data in the framebuffer to other parts of the framebuffer or to the texture memory. A single pass is made through the pixel-transfer operations before the data is written to the texture memory or back to the framebuffer. Many of the pixel operations described are part of the fixed-function pixel pipeline and often move large amounts of data around the system. Modern graphics implementations tend to optimize performance by trying to localize graphics operations to the memory local to the graphics hardware.

OpenGL Version 3.0, which supports all of these operations, also introduces framebuffer objects that help optimize these data movements, in particular, these objects can eliminate some of these transfers entirely.

Framebuffer objects, combined with programmable fragment shaders replace many of these operations (most notably, those classified as pixel transfers) and provide significantly more flexibility.

Texture Assembly

OpenGL applications can apply texture images to geometric objects to make the objects look more realistic, which is one of the numerous techniques enabled by texture mapping. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them.

Almost all OpenGL implementations have special resources for accelerating texture performance (which may be allocated from a shared pool of resources in the graphics implementation). To help your OpenGL implementation manage these memory resources efficiently, texture objects may be prioritized to help control potential caching and locality issues of texture maps.

Rasterization

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the framebuffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are generated for each fragment square.

Fragment Operations

There is a series of operations that can alter or even throw out fragments, but can be enabled or disabled. All these operations take place before values are actually stored in the framebuffer. The first operation that a fragment might encounter is texturing, where a texel (texture element) is generated from texture memory for each fragment and applied to the fragment. Next, primary and secondary colors are combined, and a fog calculation may be applied. If the application is employing fragment shaders, the preceding three operations may be done in a shader. After the final color and depth generation of the previous operations, then the scissor test, the alpha test, the stencil test, and the depth-buffer test are evaluated.

Failing an enabled test may end the continued processing of a fragment's square. Then, blending, dithering, logical operation, and masking by a bitmask may be performed. Finally, the thoroughly processed fragment is drawn into the appropriate buffer, where it has finally become a pixel and achieved its final resting place.

3.3.6 OpenGL-Related Libraries

OpenGL provides a primitive set of rendering commands that are used for all higher-level drawing. In addition, OpenGL programs have to use the underlying mechanisms of the windowing system. There are several libraries that help the user to simplify the programming tasks, including the following:

- OpenGL Utility Library (GLU): it contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation.

- For every window system, there is a library that extends the functionality of that window system to support OpenGL rendering. For machines that use the X Window System, the OpenGL Extension to the X Window System (GLX) is provided as an adjunct to OpenGL. GLX routines use the prefix glX. For Microsoft Windows, the WGL routines provide the Windows to OpenGL interface. All WGL routines use the prefix wgl. For Mac OS, three interfaces are available: AGL (with prefix agl), CGL (cgl), and Cocoa (NSOpenGL classes).

- OpenGL Utility Toolkit (GLUT): it is a window-system-independent toolkit, that hides the complexities of differing window system APIs.

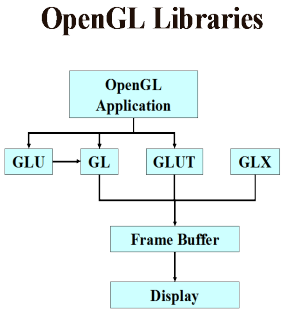


Figure 3.11. OpenGL libraries.
(http://ranger.uta.edu/~kamangar/CSE-4303-SP15/OpenGLlibraries.html)

For all OpenGL applications, the OpenGL header files should be included in every file. Many OpenGL applications may use GLU (OpenGL Utility Library), which requires inclusion of the glu.h header file. So in that case the OpenGL source file begins like this:

```
#include <GL/gl.h>
#include <GL/glu.h>
```

The OpenGL library changes all the time. The various vendors that make graphics hardware add new features that may be too new to have been incorporated in gl.h. In order for the users to take advantage of these new extensions to OpenGL, an additional header file is available, named glext.h. This header contains all of the latest version and extension functions and tokens.

GLUT, the OpenGL Utility Toolkit

As you know, OpenGL contains rendering commands but it is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse.

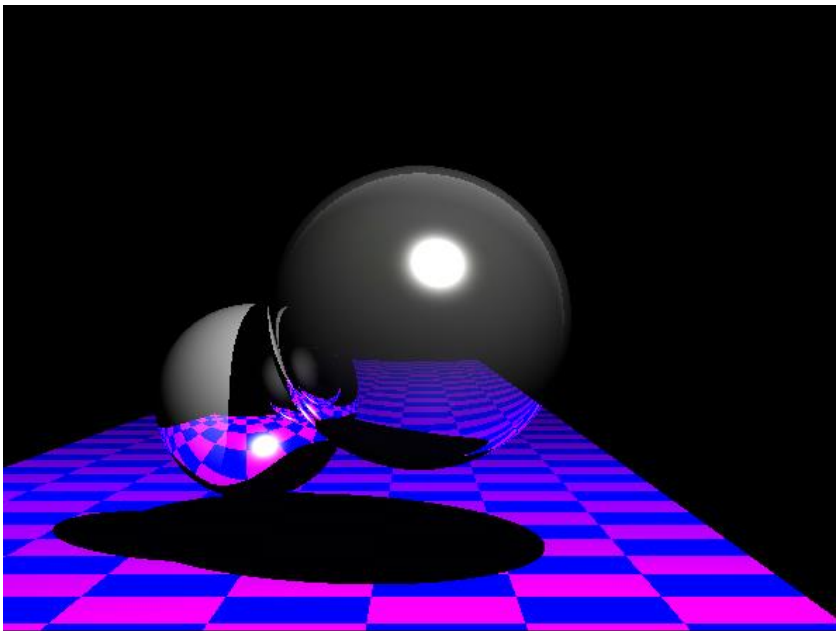


Figure 3.12. Mesh with Glut library.
(http://halogenica.net/graphics/ray-tracer/)

Furthermore, since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects, such as a sphere, a torus, and a teapot.

GLUT may not be satisfactory for full-featured OpenGL applications, but it is a useful starting point for learning OpenGL.

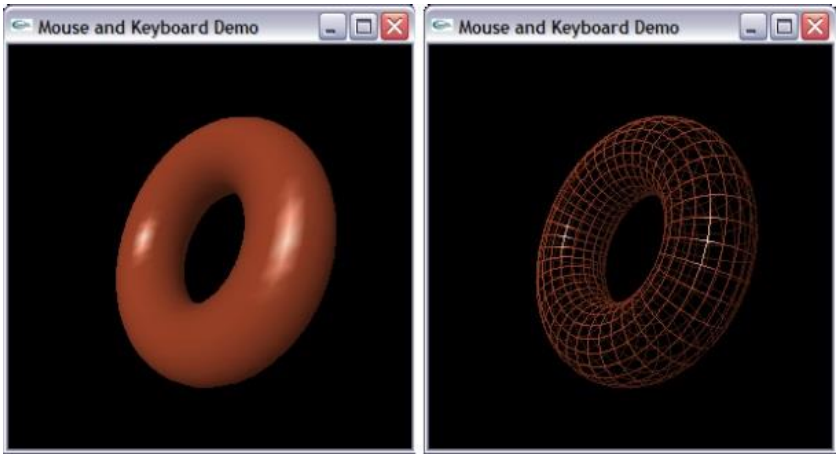


Figure 3.13. Mesh with Glut Library.
(http://csclab.murraystate.edu/bob.pilgrim/515/lectures_03.html)

OpenGL and Its Deprecation Mechanism

Evolution of OpenGL

As mentioned before, OpenGL is constantly undergoing improvement and refinement. New ways of doing graphics operations are developed, and entire new fields, arise that lead to evolution in graphics hardware capabilities. New extensions to OpenGL are suggested by vendors, and eventually some of those extensions are incorporated as part of a new core revision of OpenGL. Over the years, this development process has allowed numerous redundant methods for accomplishing the same activity to appear in the API. In many cases, while the functionality was similar, the methods' application performance generally was not, giving the impression that aspects of the OpenGL API were slow and didn't work well on modern hardware. With OpenGL Version 3.0, the Khronos OpenGL ARB Working Group specified a *depreciation model* that indicated how features could be removed from the API. However, this change required more than just changes to the core OpenGL API—it also affected how OpenGL contexts were created, and the types of contexts available.

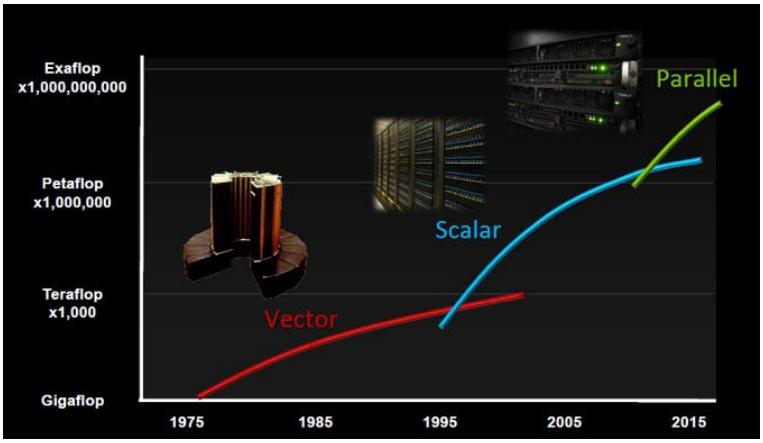


Figure 3.14. Top Preis GPGPU.
(http://digitalreviewproduct.tk/gpgpu)

OpenGL Contexts

An OpenGL context is the data structure where OpenGL stores state information to be used when you're rendering images. In this structure are included:

- Textures
- Server-side buffer objects
- Compiled shader objects
- Other stuff

After Version 3.0, there is a new type of context, the forward-compatible context, which hides the features marked for future removal from the OpenGL API. With this new type of context, the application developers can modify their applications to accommodate future versions of OpenGL.

Access on the OpenGL Functions

The operating system on which the applications are developed, has a major influence on the amount of work that is required. Some operating systems need some additional work in order to access some OpenGL functions. When this is required, the compiler will give a report that "Various functions are undefined". In case this happens, the function's address must be retrieved. There are various ways to accomplish this:

- If the application uses the native windowing system for opening windows and event processing, then the appropriate***GetProcAddress()** function for the operating system should be used. Examples of these functions include **wglGetProcAddress()** and **glXGetProcAddress()**.

- If GLUT is used, then GLUT's function pointer retrieval routine, **glutGetProcAddress(),** should be used**.**

- The open-source project GLEW (short for "OpenGL Extension Wrangler") can be used. GLEW defines every OpenGL function, by retrieving function pointers and verifying extensions automatically for the user.

4 Partial Differential Equations

4.1 Stiffness Matrix Formulation

4.1.1 Preliminary Steps for Analysis

The main difference between Isogeometric Analysis and Finite Element Analysis is the type of the shape functions used for the approximation of the solution field. The substitution of Lagrange polynomials with NURBS (widely utilized in the CAD community) leads to several special characteristics of Isogeometric Analysis.

4.1.2 Shape Functions

Finite Element Method shape functions are usually polynomials (e.g. Lagrange polynomials). However, they hold certain disadvantages. FEM shape functions are interpolatory at all nodes, internal and external, and have C^{-1} Continuity at the edge. This results in the incompetence of FEM to define stresses and strains at any boundary of any element, leading in the introduction of other corrective methods such as extrapolation, in order to achieve that. NURBS as shape functions hold an overlapping that is useful, as nearby elements are strongly connected, thus the simulation provides an improved approach to the natural problem. Continuous shape function derivatives lead to a continuous stress and strain field, minimizing the need for application of corrective methods.

4.1.3 Control Points

Classical FEM downsizes the natural problem of infinite unknowns to a finite number of unknowns, which are the degrees of freedom of the nodes. The position of the nodes depends on the element type. As a general rule, the nodes can be usually found in the corners and middle of the sides of the elements. They are part of the element and therefore part of the physical model. Displacements in other areas of the model can be approximated by a linear combination of displacements on the degrees of freedom. Distribution in the model is evaluated via the corresponding shape functions. In isoparametric elements, shape functions and their respective nodes are also used to approximate the geometry, thus enabling relatively complex shapes to be approximated with Lagrange polynomials. In Isogeometric Analysis, NURBS are chosen as shape functions. The isoparametric concept is reversed, as geometrical mapping now defines the solution approximation. The geometrical representation is achieved through a combination of control points and their corresponding shape functions. Degrees of freedom at the control points are now the unknowns.

4.1.4 Data Definition

4.1.4.1 Tab ControlPoint

Tab ControlPoint is the first tab of many that the software's graphic user interface consists of and its completion the first of logic operations sequence that must be done in order a correct analysis procedure to be accomplished.

4.1.4.2 Problem Type

There are three kinds of problems.

- **1D Problems:**
  Problems whose degrees of freedom expand into one parametric axis.
- **2D Problems:**
  Problems whose degrees of freedom expand into two parametric axes.
- **3D Problems:**
  Problems whose degrees of freedom expand into three parametric axes.

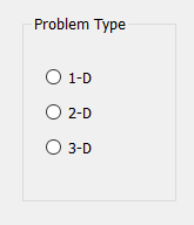


Figure 4.1. Tab Control Point. Problem type definition.

The user will have to choose the type of problem that he is going to solve in order next options that concern the geometry definition to be activated.

4.1.4.3 Patch

After problem type has been chosen, the user has to describe and design the model that he is going to analyze by defining for each patch the relative geometry, and giving to each patch a desired ID so that it will later be able to be discretized and identified by the user in the graphic environment.

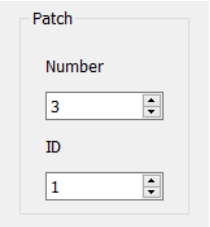


Figure 4.2. Tab Control Point. Patch ID definition.

4.1.4.4 Geometry Data

The software provides four ways of defining the geometry of each patch and it is allowed to the user to choose the desired procedure.

- **Import Data**

If the user chooses to define his geometry by importing the appropriate data through a relative archive that is compatible with the software then there are the following options:

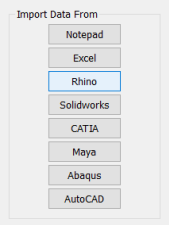


Figure 4.3. Tab Control Point. Import data.

The user has the ability to import data not only through existing widespread computer aided design software applications but also through acquainted computer aided engineering software applications.

- **Draw Control Points**

If the user chooses to define his geometry by freely drawing the desired control points in the graphic environment that the software provides then a button which is responsible for the designing is activated and after it is pressed down the user is automatically transferred to the graphic environment where he can draw the geometry.

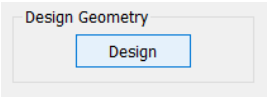


Figure 4.4. Tab Control Point. Design geometry.

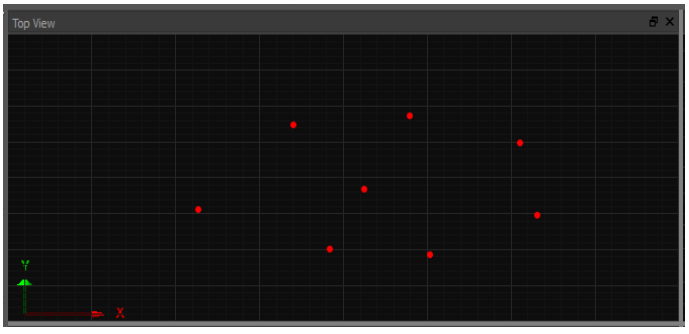


Figure 4.5. Control points on graphic user interface (Top View).

- **Give Control Point Variables**

The third way of defining the desired geometry is by writing the user himself in the given TableWidget that is provided, the relative X, Y, Z control point coordinates as also their weights that describe the geometry. Before that action the user must first determine the number of control points that exist in each parametric axis ξ, η, ζ.

After control point number in each parametric axis is defined according to the problem type, the control point variables table widget is activated and there have been created as many rows and columns responsible for the data completion as necessary according to the control point number definition.

For example, if we give the following options:
«Problem Type»: 2
«Control Point Number Axis ξ»: 4
«Control Point Number Axis η»: 3

There will be created twelve (12) rows, so as the user can assign his desired data.

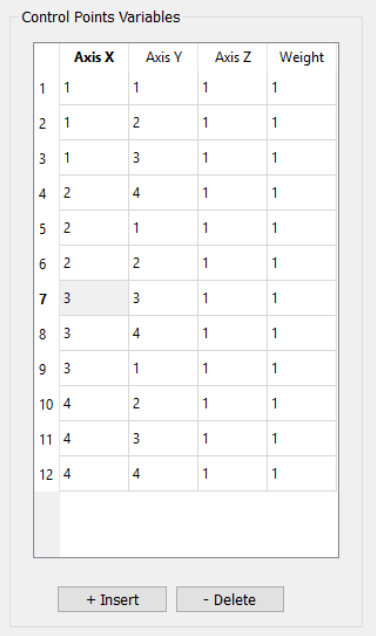


Figure 4.6. Tab Control Point. Control point variables definition.

But since we have chosen «Problem Type» : 2 as option, the user must also assign the thickness of his patch, as our software has the ability to solve as far as 2-Dimensional problems concerned only «Stress problems» and «Strain Problems».

Figure 4.7. Tab Control Point. Dimensions definition.

As we can see, «Width» and «Height» as options have been deactivated because the width and height of the model will be determined by the given data we will provide.

- **Select Predefined Geometry**

The code provides the ability of analyzing specific, known and standard geometry models in which he is allowed to determine specific characterizations such as the number of control points in each parametric axis as also the model's dimensions.

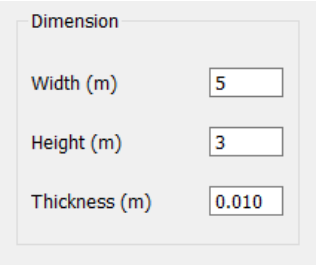


Figure 4.8. Tab Control Point. Dimensions of predefined geometry.

After the dimensions have been determined, the control point coordinates are alculated according to refinement principles.

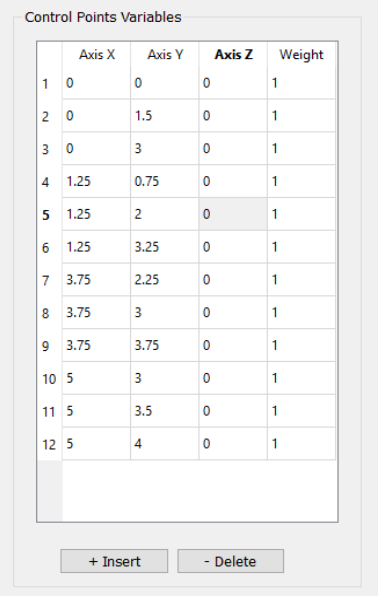


Figure 4.9. Tab Control Point. Control Point variables of predefined geometry.

The final step for the user to complete the first step of the analysis procedure after he has completely defined his geometry is to simply confirm his actions so that they can be stored from the software by simply pressing the button «Apply» that now has been activated.

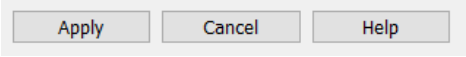


Figure 4.10. Tab Control Point. Confirmation or cancel of procedure.

4.1.5 Elements

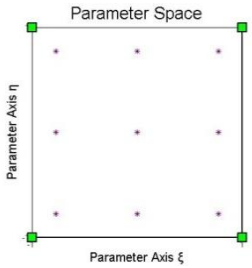
Two ingredients of the isogeometric universe correspond to the essence of FEM "element". The isogeometric element could be either the patch or the knot span of the patch. In order to perform exact numerical integration, a certain number of gauss points are chosen for the domain of every piece of the polynomial basis functions. The domain of this piece 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 isogeometric elements. In IGA, shape functions are not restricted to the interior of each element (knot span). Instead, 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 Finite Element Matrix with the same degrees of freedom. Apart from that, the fact that B-SPLine functions are defined in the whole domain allows for integration throughout the patch without building local element matrices separately. This would be time-consuming and it is not advised for advanced software technologies, but serves well for research purposes, where a flexible quadrature code is needed in order to test and discover new methods and ideas.

4.1.6 Gauss Points

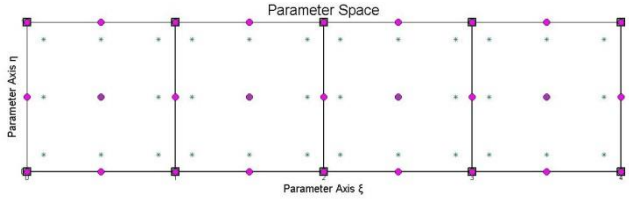
4.1.6.1 Parametric Coordinates

As mentioned above, gauss points are chosen for each knot span. Their coordinates are obtained on a reference element spanning [-1,1] as the roots of the Legendre Polynomial. The next step is to transform the coordinates and weights from the reference knot span ξ^R to the desired knot span $\left[\xi_i, \xi_{i+1}\right)$.

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



(a)



(b)

Figure 4.11. Gauss points
(a) on Reference knot span and (b) transferred to parameter space.

Full tensor product properties apply here as well, leading in similar equations for the other two parametric directions:

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

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

4.1.6.2 Gauss Point Number

Gauss point Coordinates and Weights are evaluated for every patch. According to [4], 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:

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

Conclusively per knot span:

- For 1D problems, $\mathbf{p}$ gauss points per knot span are required.
- For 2D and 3D problems, $\mathbf{p+1}$ gauss points per knot span are required.

4.1.7 Patches

As mentioned before, patches are used when a change in geometry type or material occurs. patches 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 produced 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 a single, global Stiffness Matrix.

4.1.7.1 Tab Mesh

As far as geometry definition of a model concerned it is not enough the definition of the control point Coordinates or the control point number but also the degree and knot value vector determination in order to have a complete and unique geometry that can be described in univocal way. Tab Mesh is the second tab that the user must complete in order to have a successful analysis procedure. The Tab concludes the Degree, knot value vector, gauss point Per knot span and Illustration point determination for each Parametric Axis.

Polynomial Degree

The order of a NURBS curve defines the number of nearby control points that influence any given point on the curve. The curve is represented mathematically by a polynomial of degree one less than the order of the curve. Hence, second-order curves (which are represented by linear polynomials) are called linear curves, third-order curves are called quadratic curves, and fourth-order curves are called cubic curves. The number of control points must be greater than or equal to the order of the curve.

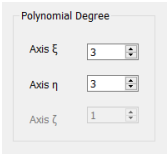


Figure 4.12. Tab Mesh. Polynomial degree definition for each parametric axis.

In practice, cubic curves are the ones most commonly used. Fifth- and sixth-order curves are sometimes useful, especially for obtaining continuous higher order derivatives, but curves of higher orders are practically never used because they lead to internal numerical problems and tend to require disproportionately large calculation times.

Knot Value Vector

The knot vector is a sequence of parameter values that determines where and how the control points affect the NURBS curve. The number of knots is always equal to the number of control points plus curve degree plus one (i.e. number of control points plus curve order). The knot vector divides the parametric space in the intervals mentioned before, usually referred to as knot spans. Each time the parameter value enters a new knot span, a new control point becomes active, while an old control point is discarded. It follows that the values in the knot vector should be in nondecreasing order, so (0, 0, 1, 2, 3, 3) is valid while (0, 0, 2, 1, 3, 3) is not.

Consecutive knots can have the same value. This then defines a knot span of zero length, which implies that two control points are activated at the same time (and of course two control points become deactivated). This has impact on continuity of the resulting curve or its higher derivatives; for instance, it allows the creation of corners in an otherwise smooth NURBS curve. A number of coinciding knots is sometimes referred to as a knot with a certain multiplicity. knots with multiplicity two or three are known as double or triple knots. The multiplicity of a knot is limited to the degree of the curve; since a higher multiplicity would split the curve into disjoint parts and it would leave control points unused. For first-degree NURBS, each knot is paired with a control point. The knot vector usually starts with a knot that has multiplicity equal to the order. This makes sense, since this activates the control points that have influence on the first knot span. Similarly, the knot vector usually ends with a knot of that multiplicity. Curves with such knot vectors start and end in a control point.

Our code differs from other software applications as it gives the user the ability not only to interfere in polynomial degree definition but also in knot value vector determination. Since control point Number definition has been accomplished in Tab Control point and after the polynomial degree has been set up, rows are automatically calculated so as the user gives the right number of knot values and for the purpose of minimizing the possibility of error appearing. The values of the knots control the mapping between the input parameter and the corresponding NURBS value. For example, if a NURBS describes a path through space over time, the knots control the time that the function proceeds past the control points. For the purposes of representing shapes, however, only the ratios of the difference between the knot values matter; in that case, the knot vectors (0, 0, 1, 2, 3, 3) and (0, 0, 2, 4, 6, 6) produce the same curve. The positions of the knot values influences the mapping of parameter space to curve space. Rendering a NURBS curve is usually done by stepping with a fixed stride through the parameter range. By changing the knot span lengths, more sample points can be used in regions where the curvature is high.

Another use is in situations where the parameter value has some physical significance, for instance if the parameter is time and the curve describes the motion of a robot arm. The knot span lengths then translate into velocity and acceleration, which are essential to get right to prevent damage to the robot arm or its environment. This flexibility in the mapping is what the phrase non uniform in NURBS refers to. Necessary only for internal calculations, knots are usually not helpful to the users of modeling software. Therefore, many modeling applications do not make the knots editable or even visible. It's usually possible to establish reasonable knot vectors by looking at the variation in the control points. More recent versions of NURBS software (e.g., Autodesk Maya and Rhinoceros 3D) allow for interactive editing of knot positions, but this is significantly less intuitive than the editing of control points.

Gauss Point Per Knot Span

For the purpose of the Analysis procedure the user has the ability to define the number of gauss points for each Parametric Axis that will correspond to each knot span and will determine the accuracy of the analysis results. Of course the software provides the option those characteristics to be filled automatically by the minimum number of gauss points that are necessary for a dignified accuracy which is the polynomial degree plus one.

4.1.8 Elasticity Matrix

Elasticity Matrix types vary depending on the stress and strain field for each case. Data is obtained straight from classical FEM applications. Elasticity matrices for 1D elasticity, plane strain, plane stress and 3D elasticity are presented as follows, where E is the Young's modulus and 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:

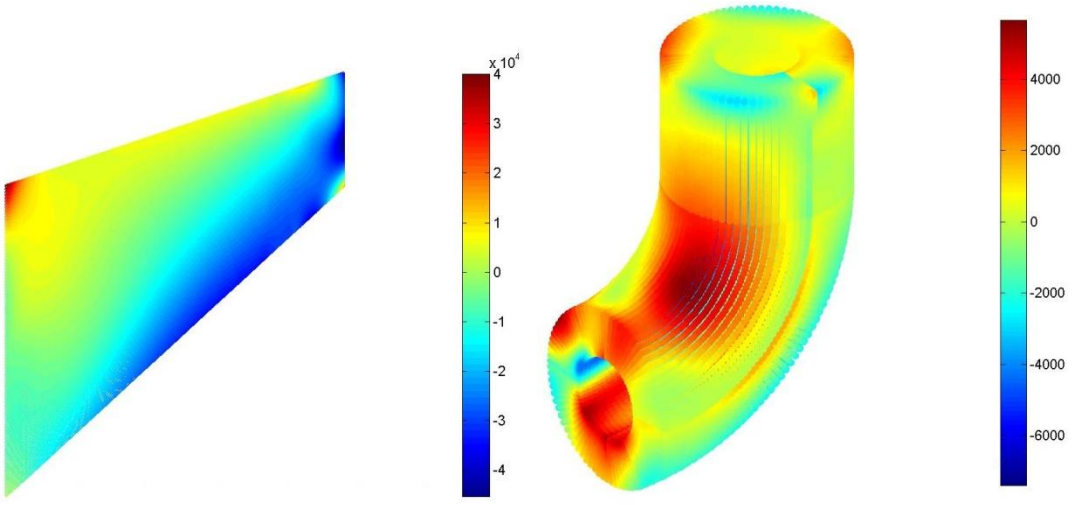
$$\underset{(1\times1)}{\{\sigma\}} = \{\sigma_X\}$$
$$\underset{(1\times1)}{\{\varepsilon\}} = \{\varepsilon_X\}$$

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:

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



(a) **(b)**

Figure 4.13. Stress contour distribution.
(a) σ_{xx} for Plane stress, (b) τ_{yz} 3D elasticity

4.1.8.1 Tab Material

After geometry has completely defined the user must now determine not only the material but also the basic characteristics that govern its nature.

The first constitutive equation (constitutive law) was developed by Robert Hooke and is known as Hooke's law. It deals with the case of linear elastic materials. Following this discovery, this type of equation, often called a "stress-strain relation" in this example, but also called a "constitutive assumption" or an "equation of state" was commonly used.

Quantity (common name/s)	(Common) symbol/s	Defining equation	SI units	Dimension
General stress, Pressure	*P*, σ	$\sigma = F/A$ *F* may be any force applied to area *A*	Pa = N m^{-2}	[M] $[T]^{-2}[L]^{-1}$
General strain	ε	$\epsilon = \Delta D/D$ *D* = dimension (length, area, volume) Δ*K* = change in material	dimensionless	dimensionless
General elastic modulus	E_{mod}	$E_{mod} = \sigma/\epsilon$	Pa = N m^{-2}	[M] $[T]^{-2}$ $[L]^{-1}$
Young's modulus	*E*, *Y*	$Y = \sigma/(\Delta L/L)$	Pa = N m^{-2}	[M] $[T]^{-2}[L]^{-1}$
Shear modulus	*G*	$G = \Delta x/L$	Pa = N m^{-2}	[M] $[T]^{-2}[L]^{-1}$
Bulk modulus	*K*, *B*	$B = P/(\Delta V/V)$	Pa = N m^{-2}	[M] $[T]^{-2}[L]^{-1}$
Compressibility	*C*	$C = 1/B$	Pa^{-1} = m^2 N^{-1}	[L] $[T]^2[M]^{-1}$

Figure 4.14. Constitutive Law
(https://en.wikipedia.org/wiki/Constitutive_equation)

The software provides two ways of defining the material of each patch. One option is to use a predefined by the software material and the other is to create or use a new kind of material simply by determining the constitutive law and constitutive law type.

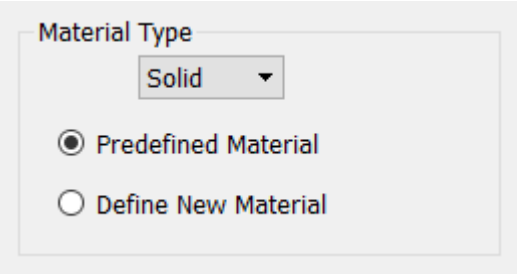


Figure 4.15. Tab Material. Material type definition.

Predefined Material

If the user chooses the option of predefined geometry the software provides at the moment three kind of materials :

- **Steel**

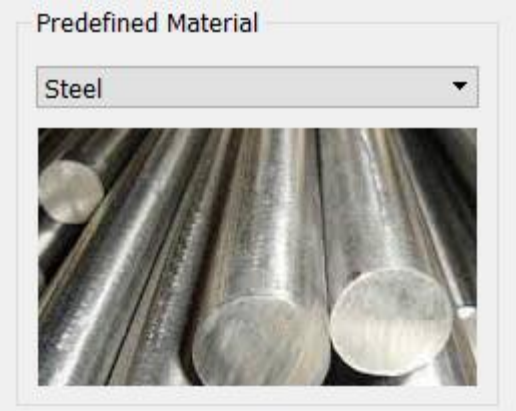


Figure 4.16. Tab Material. Predefined material steel.

- **Aluminum**

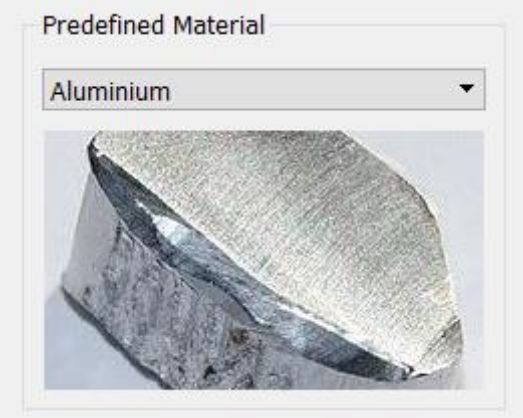


Figure 4.17. Tab Material. Predefined material aluminum

- **Copper**

Figure 4.18. Tab Material. Predefined material copper

Define New Material

If the user chooses to define a new material then he has to determine the following characteristics:

First of all he has to define the constitutive law type; the elasticity type and the behavior type.

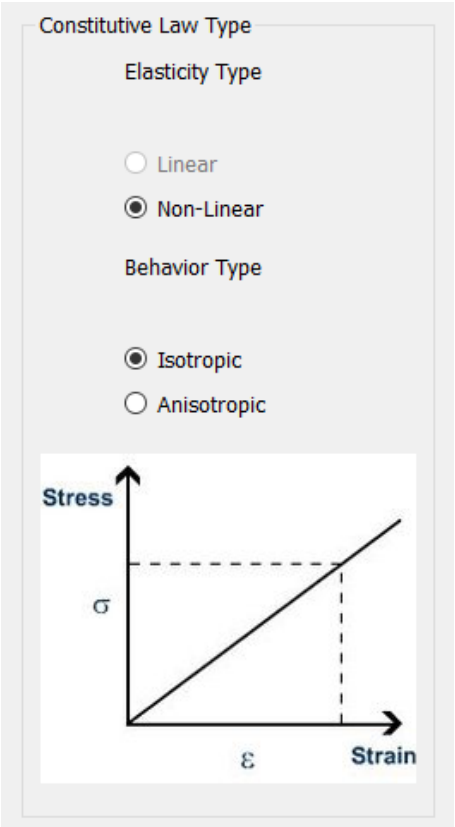


Figure 4.19. Tab Material. Constitutive Law type.

Secondly he has to fulfill the necessary information that describe the constitutive law.

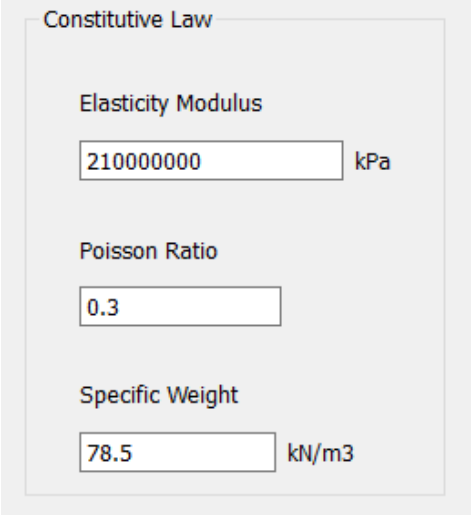


Figure 4.20. Tab Material. Constitutive Law.

Plane

It is very important to add at this point that the software in the case of deciding to analyze 2 Dimensional problems the user must determine if the problem concerns stress or strain condition.

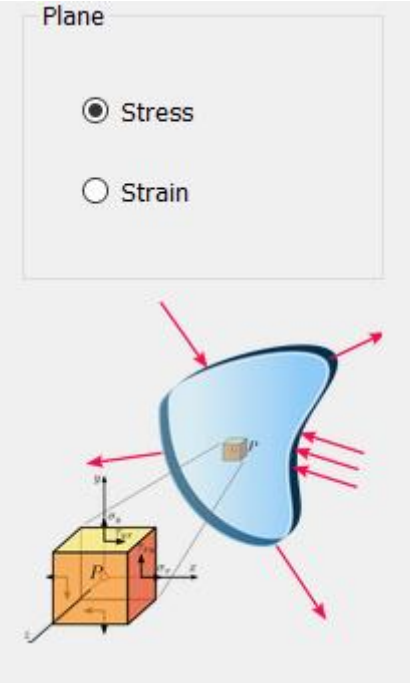


Figure 4.21. Tab Material. Plane stress.

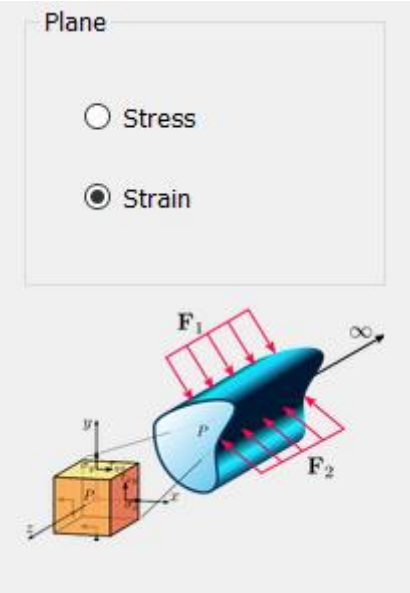


Figure 4.22. Tab Material. Plane strain.

4.1.9 Stiffness Matrix Assembly.

4.1.9.1 General Procedure

The general process for the Global Stiffness Matrix assembly, as obtained from Finite Element Method, is shown in the following flow chart:

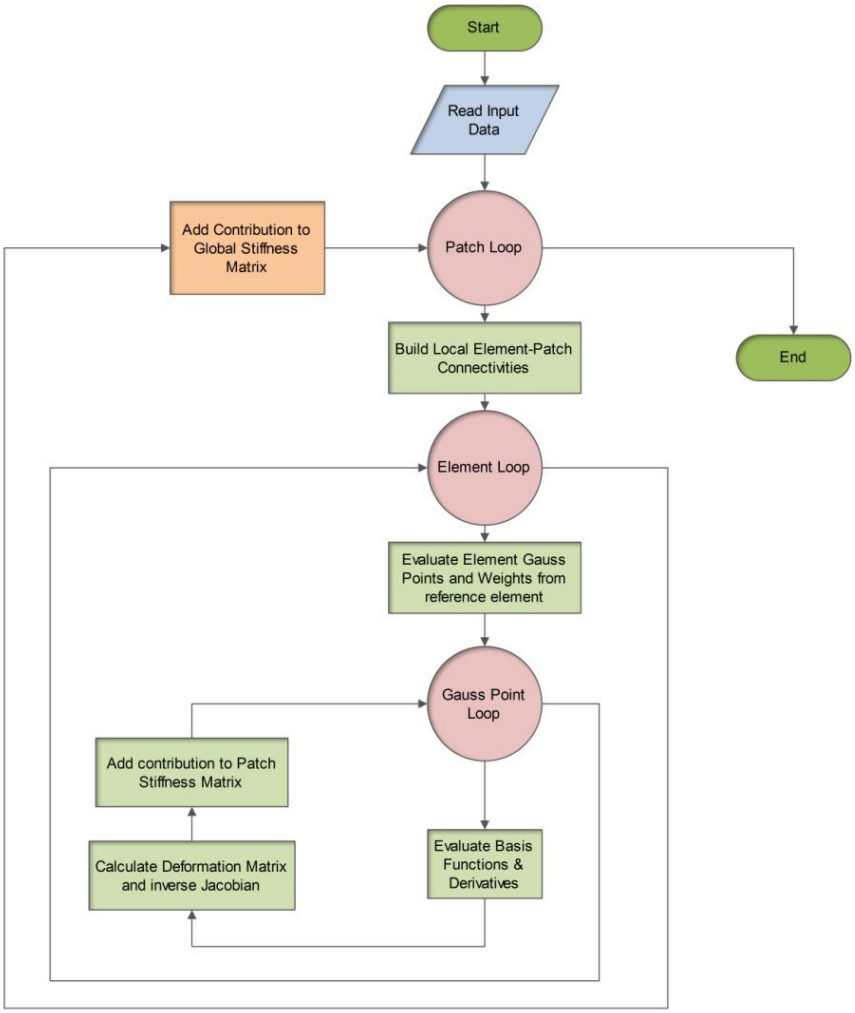


Figure 4.23. Stiffness Matrix Assembly in Finite Element Method.

There are three loops:

- Patch loop
- Element loop
- Gauss point loop

It is worth mentioning that the element loop in IGA can be avoided. In this case, the stiffness contribution of each control point pair is added directly to the Stiffness Matrix of the patch. The reason for this is that parameter space is local to patch rather than elements.

Therefore, an engineer accustomed to the methods of Isogeometric Analysis knows that the element loop can be completely avoided, leading to the following flow chart:

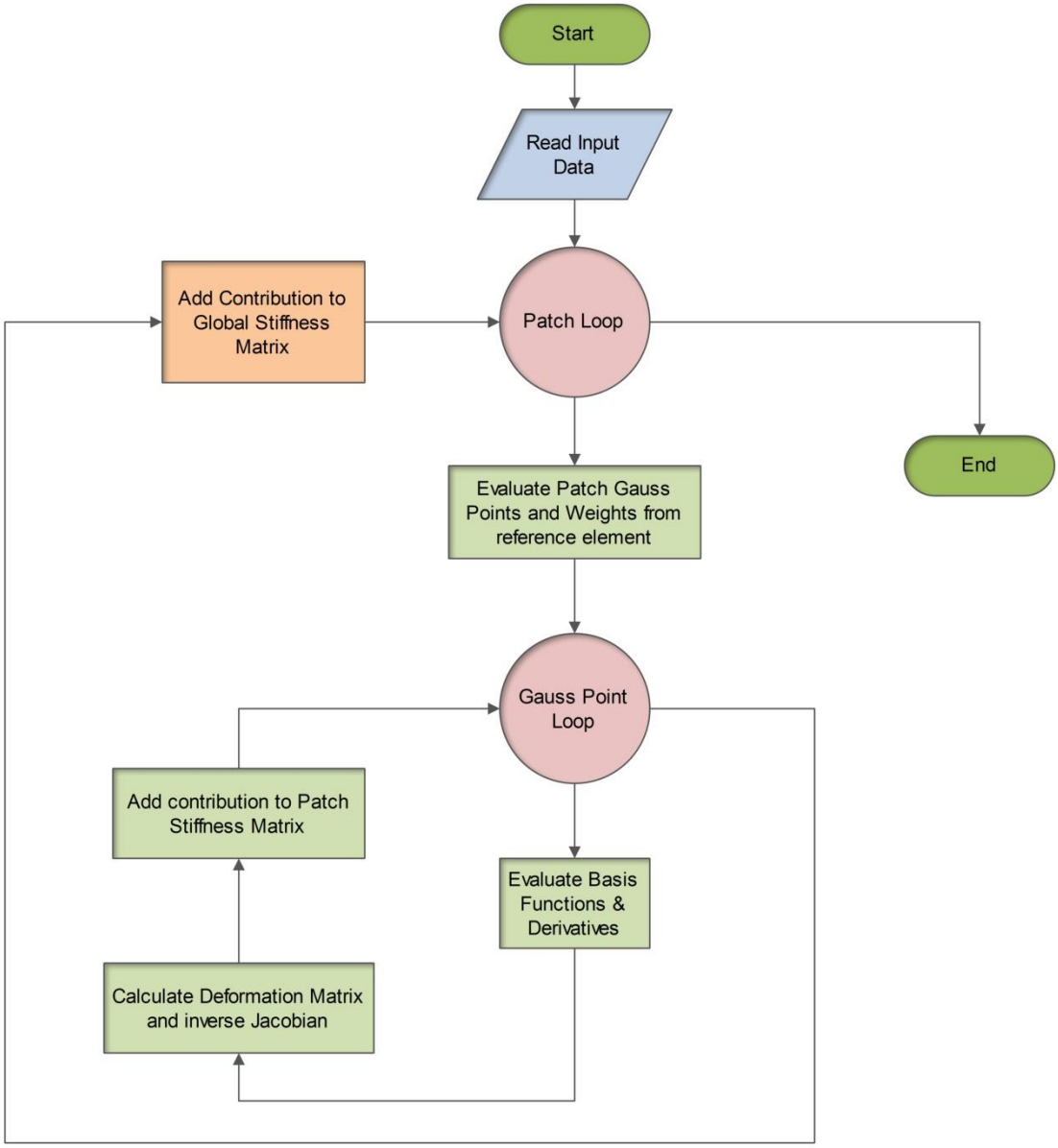


Figure 4.24. Stiffness Matrix Assembly in Isogeometric Analysis

4.1.9.2 Input Data

Information necessary for the whole process of analysis is given as input at this point. The information essential for the formulation of the Stiffness Matrix is divided into two categories:

- Structural Analysis and Material
- Computational Geometry

4.2 Stiffness Matrix

4.2.1 Stiffness Matrix 1D

Let us 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:

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

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

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:

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

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 $\left[J\right]^{-1}$ is needed.

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

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 $\left[J\right]$ 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 $\left[B_1\right]$ and $\left[B_2\right]$. The matrix $\left[B_1\right]$ transfers the strains of the element from parameter to physical space and the matrix $\left[B_2\right]$ transfers the nodal displacements of the elements to the strains at the parameter space. Therefore, the matrices $\left[B_1\right]$and $\left[B_2\right]$ can be calculated from the equations below:

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

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

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

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

$$\underset{(1\times n)}{\left[B_2(\xi)\right]} = \left[R_{1,\xi}(\xi) \quad R_{2,\xi}(\xi) \quad \ldots \quad \ldots \quad \ldots \quad R_{n,\xi}(\xi)\right]$$

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

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.

4.2.2 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:

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

In order to calculate the Deformation Matrix for 2D problems, $\left[B_1\right]$ and $\left[B_2\right]$ have to be evaluated as usual.

To obtain matrix $\left[B_1\right]$:

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

Hence,

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

To calculate matrix $\left[B_2\right]$:

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

Hence,

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

Having determined $[B_1]$ and $[B_2]$, the Deformation Matrix is calculated as:

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

In order to evaluate the Stiffness Matrix, integration is required.

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

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

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

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.

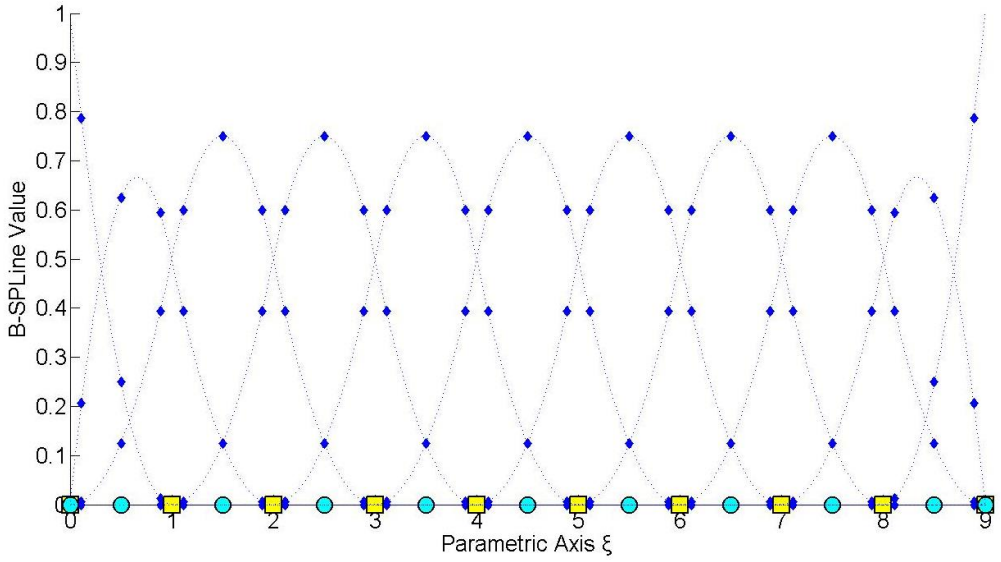


Figure 4.25. Basis functions evaluated as gauss points.

$\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 9 \quad 9\}$

4.2.3 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

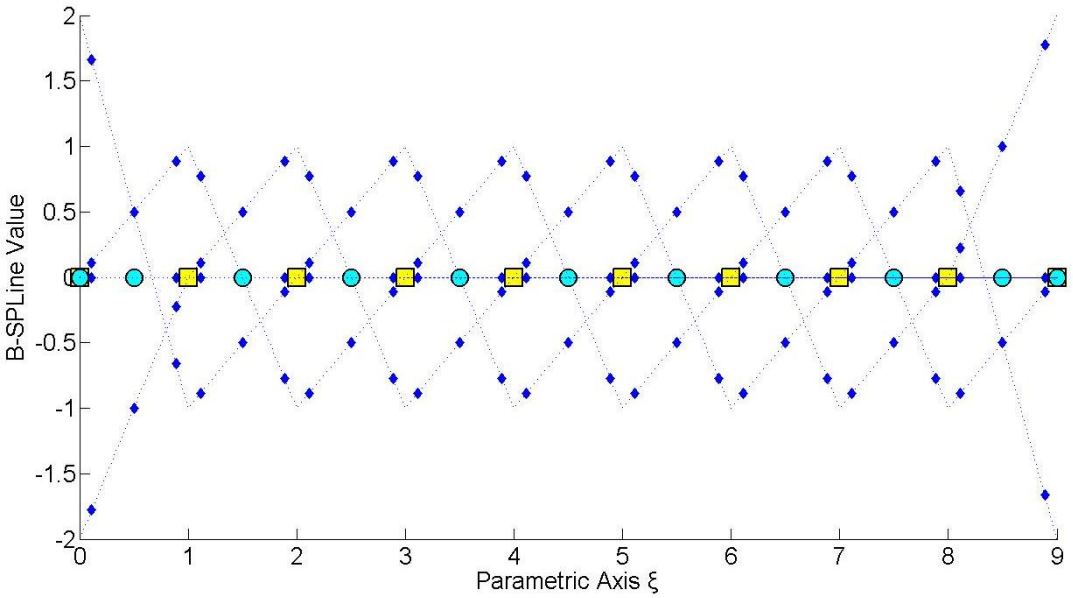
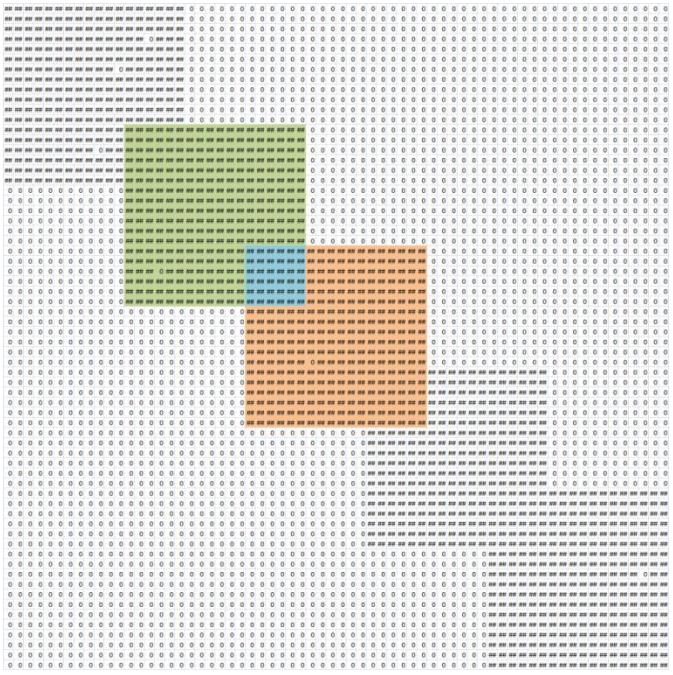


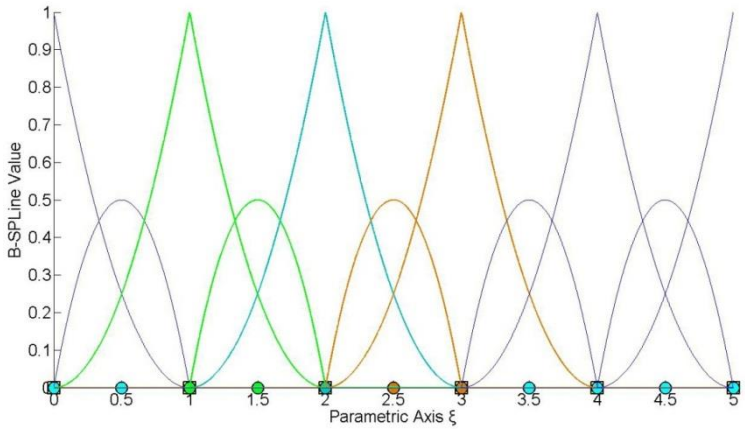
Figure 4.26. Basis function derivatives evaluated as gauss points.
knot value vector

$$\Xi=\{0\quad 0\quad 0\quad 1\quad 2\quad 3\quad 4\quad 5\quad 6\quad 7\quad 8\quad 9\quad 9\quad 9\}$$

4.2.4 Stiffness Matrix Examples



(a)

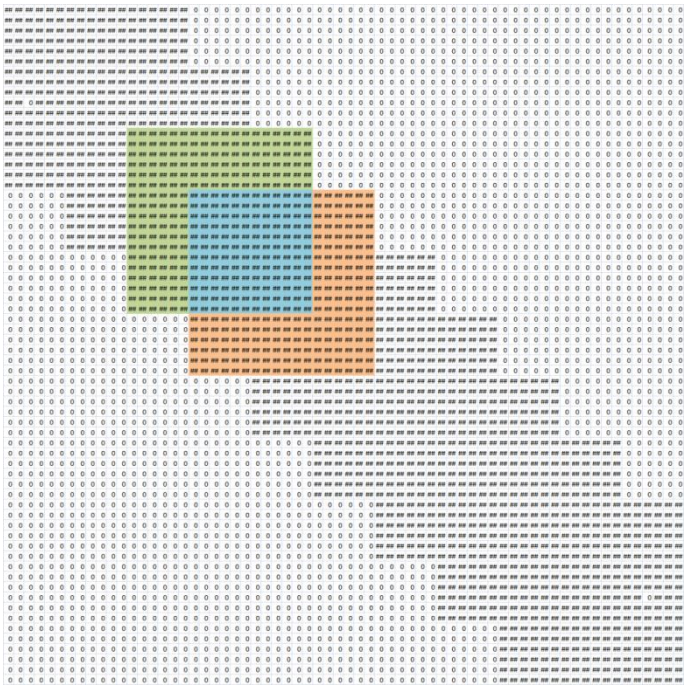


(b)

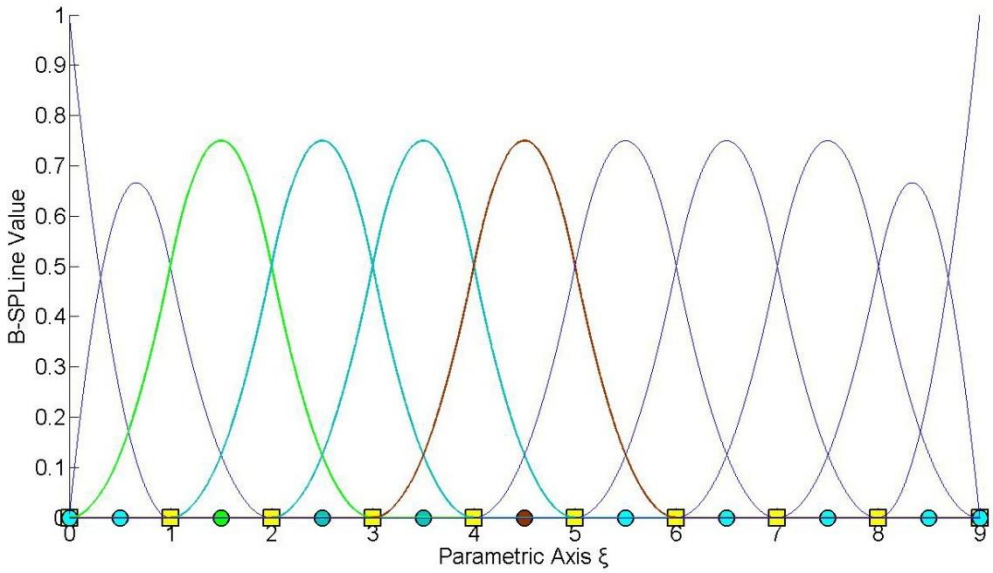
**Figure 4.27. Stiffness Matrix and basis functions of the model.
(a) 1470 non-zero elements are produced
and (b) its corresponding basis functions.**

In Figure 4.7, 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 1470. The interconnectivity between the elements applies in a small part of the matrix, as it is shown in Figure 4.7.a. This occurs because the overlapping between the B-SPLines exists only for $N_{5,2}(\xi)$.

The non-zero elements at the matrix are a 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 $N_{3,2}(\xi)$, $N_{4,2}(\xi)$, $N_{6,2}(\xi)$ and $N_{7,2}(\xi)$. For these B-SPLines, the elements of the matrix are non-zero. This process requires less composition time, because of the limited number of interconnections.



(a)



(b)

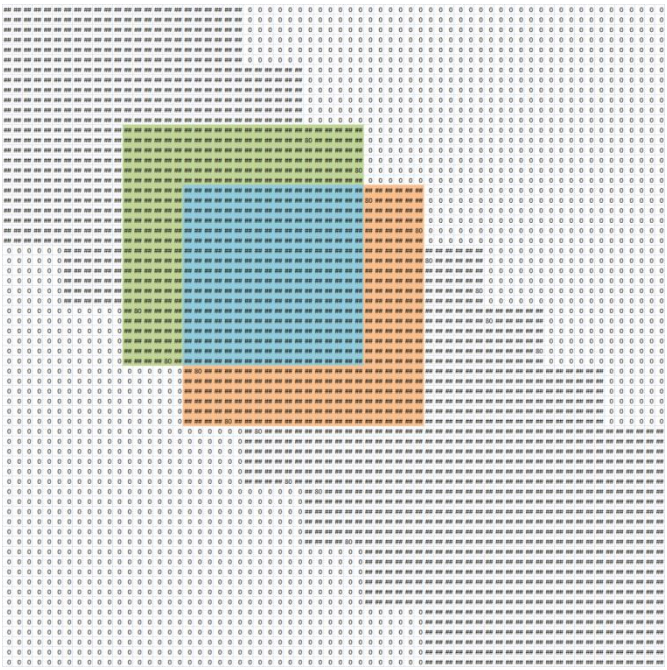
**Figure 4.28. Stiffness Matrix and basis functions of the model.
(a) 1762 non-zero elements are produced
and (b) its corresponding basis functions.**

In Figure 4.28, 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 4.27. In this case, the configuration of the matrix requires more time, but the overlapping between the B-SPLines leads to better results.

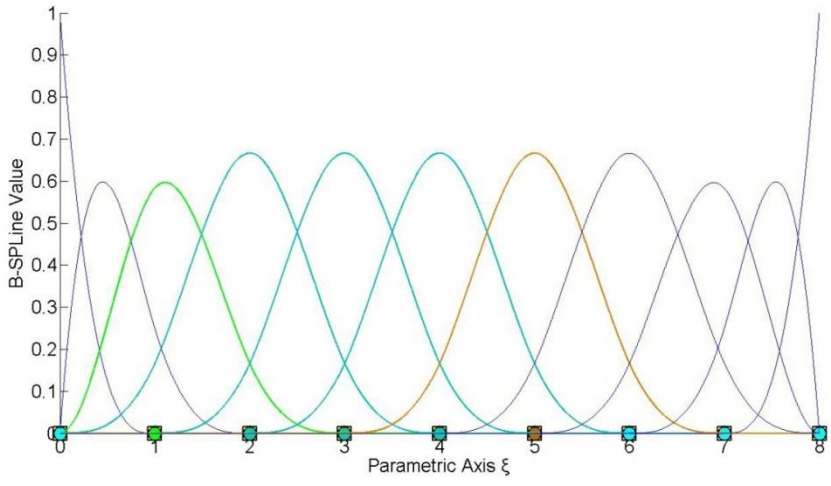
A Stiffness Matrix coefficient is non-zero when shared support applies for the corresponding B-SPLine and extensively between control points.

In particular, every B-SPLine shares support with $2p$ B-SPLines. In knot span $[2,3)$ of Figure 4.28, the B-SPLines $N_{3,2}(\xi)$, $N_{4,2}(\xi)$ and $N_{5,2}(\xi)$ are non-zero and in knot span $[3,4)$, $N_{4,2}(\xi)$, $N_{5,2}(\xi)$ and $N_{6,2}(\xi)$. These two knot spans are connected to each other with the B-SPLines $N_{4,2}(\xi)$ and $N_{5,2}(\xi)$.

The B-SPLine $N_{4,2}(\xi)$ shares support with p B-SPLines on each side, in particular $N_{2,2}(\xi)$, $N_{3,2}(\xi)$, $N_{5,2}(\xi)$ and $N_{6,2}(\xi)$. The same applies for the B-SPLine $N_{5,2}(\xi)$, as it shares support with $N_{3,2}(\xi)$, $N_{4,2}(\xi)$, $N_{6,2}(\xi)$ and $N_{7,2}(\xi)$.



(a)



(b)

Figure 4.29. Stiffness Matrix and basis functions of the model.
(a) 2340 non-zero elements are produced and
(b) its corresponding basis functions.

In Figure 4.29, 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 4.29. 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.

Figure 4.11 represents the influence of the continuity in the analysis of the model in Figure 4.10. 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 $2\cdot p$ 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.

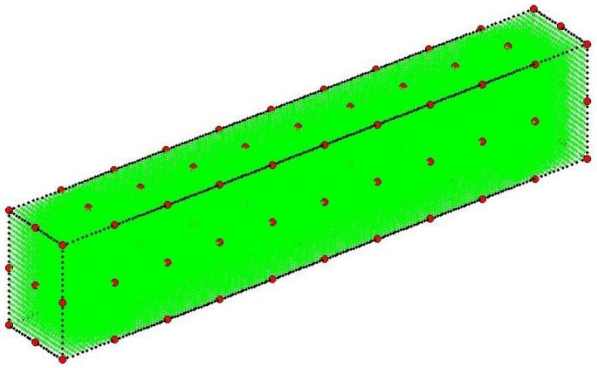
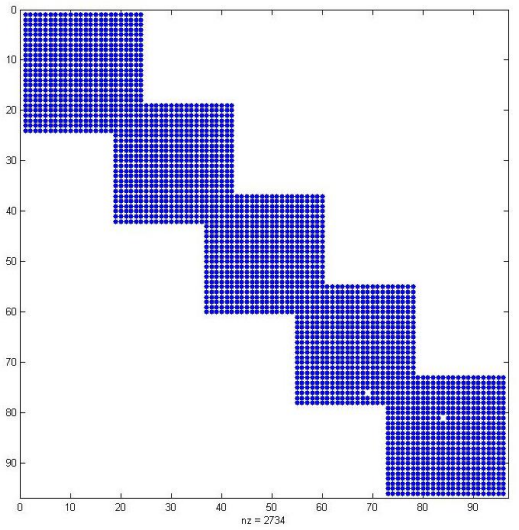
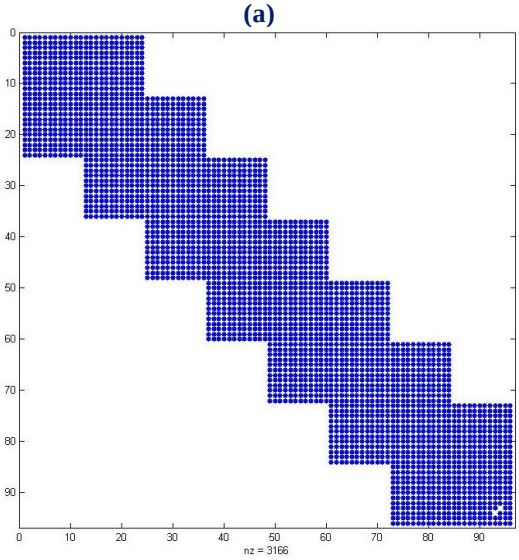


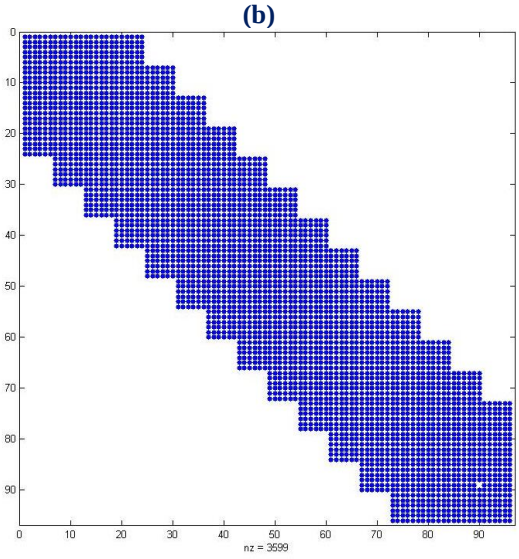
Figure 4.30. 3D structure for continuity investigation.



(a)



(b)



(c)

Figure 4.31. Stiffness Matrix for $p=3$ and 99 degrees of freedom.

(a) C^0 Continuity

(b) C^1 Continuity

(c) C^2 Continuity

4.3 External Loads, Boundary Conditions and Stress Field

4.3.1 External Load

After the Stiffness Matrix Assembly, the external Load vector has to be calculated. Concentrated Loads can be assigned directly at the Degrees of Freedom. Distributed loads $f(\xi,\eta,\zeta)$ have to be transformed into equivalent concentrated loads by integration [2]:

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

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

This way, the Load vector $\{F\}$ is assembled.

4.3.2 Refined Load

If a load vector has already been evaluated for a coarse mesh, there is no need to apply numerical integration for the new load vector of the fine mesh. New load values can be evaluated directly from the coarse mesh.

Let $\{L^C\}$, $\{L^F\}$ be the load vectors, $\{N^C(\xi)\}$, $\{N^F(\xi)\}$ the basis functions and $\{X^C\}$, $\{X^F\}$ the control point Cartesian Coordinates for the coarse and fine mesh respectively. Let $f(\xi)$ be the load distribution in the physical space. It applies that:

$$\underset{(n\times 1)}{\{L^C\}} = \int_{\xi_0}^{\xi_{max}} \underset{(n\times 1)}{\{N^C(\xi)\}} \cdot \underset{(1\times 1)}{f(\xi)}\, d\xi$$

Both meshes provide the same geometrical representation, resulting in:

$$C(\xi) = \underset{(1\times n)}{\{N^C(\xi)\}^T} \cdot \underset{(n\times 1)}{\{X^C\}}$$

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

Thus,

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

It has been established, in Chapter 3, that:

$$\underset{(n\times 1)}{\left\{X^C\right\}} = \underset{(n\times m)}{\left[T^{CF}\right]} \cdot \underset{(m\times 1)}{\left\{X^F\right\}}$$

Therefore,

$$\underset{(1\times m)}{\left\{N^F(\xi)\right\}^T} \cdot \underset{(m\times 1)}{\left\{X^F\right\}} = \underset{(1\times n)}{\left\{N^C(\xi)\right\}^T} \cdot \underset{(n\times 1)}{\left\{X^C\right\}} = \underset{(1\times n)}{\left\{N^C(\xi)\right\}^T} \cdot \underset{(n\times m)}{\left[T^{CF}\right]} \cdot \underset{(m\times 1)}{\left\{X^F\right\}} \Rightarrow$$

$$\underset{(1\times m)}{\left\{N^F(\xi)\right\}^T} = \underset{(1\times n)}{\left\{N^C(\xi)\right\}^T} \cdot \underset{(n\times m)}{\left[T^{CF}\right]}$$

This leads to

$$\underset{m\times 1}{\left\{N^F(\xi)\right\}} = \underset{m\times n}{\left[T^{CF}\right]^T} \cdot \underset{n\times 1}{\left\{N^C(\xi)\right\}}$$

The fine mesh load vector is evaluated by:

$$\underset{(m\times 1)}{\left\{L^F\right\}} = \int_{\xi_0}^{\xi_{max}} \underset{(m\times 1)}{\left\{N^F(\xi)\right\}} \cdot \underset{(1\times 1)}{f(\xi)}\, d\xi \Rightarrow \underset{(m\times 1)}{\left\{L^F\right\}} = \int_{\xi_0}^{\xi_{max}} \underset{(m\times n)}{\left[T^{CF}\right]^T} \cdot \underset{(n\times 1)}{\left\{N^C(\xi)\right\}} \cdot \underset{(1\times 1)}{f(\xi)}\, d\xi \Rightarrow$$

$$\underset{(m\times 1)}{\left\{L^F\right\}} = \underset{(m\times n)}{\left[T^{CF}\right]^T} \int_{\xi_0}^{\xi_{max}} \underset{(n\times 1)}{\left\{N^C(\xi)\right\}} \cdot \underset{(1\times 1)}{f(\xi)}\, d\xi$$

In conclusion,

$$\underset{(m\times 1)}{\left\{L^F\right\}} = \underset{(m\times n)}{\left[T^{CF}\right]^T} \cdot \underset{(n\times 1)}{\left\{L^C\right\}}$$

In the same manner, equivalent loads for Reverse-Refinement can be defined as:

$$\underset{(n\times 1)}{\left\{L^C\right\}} = \underset{(n\times m)}{\left[T^{FC}\right]^T} \cdot \underset{(m\times 1)}{\left\{L^F\right\}}$$

An engineer must always bear in mind the restrictions set for Reverse Refinement. Both in geometrical representation and in equivalent load evaluation, this procedure only works under certain circumstances. More specifically, loads refined from a coarse mesh can be transformed back to that coarse mesh. On the other hand, loads created initially from a fine mesh will probably be transformed incorrectly to the coarse mesh.

4.3.2.1 Tab External Load

The most important thing in an analysis procedure is to finally the user determine the external loads that he wants to enforce to his model. The software again provides two ways of external load defining. The first way is to enforce the external load directly to the material points of the model and the second way is by putting loads to the control points of the patch. Additionally the user is capable of deciding whether or not to consider gravity or not.

Material Points

If the user decides to apply the load directly to material points, then he will have to choose between four categories.

Control Points

If the user decides to apply the external load to the control points then a Table Widget that is responsible for the control points ID definition is automatically activated and through that he can decide not only the control points that will accept the load but also the value of the load as a vector.

4.3.3 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. This leaves a Stiffness Matrix and a Load vector having only free Degrees of Freedom, $[K_{ff}]$ and $\{F_f\}$ respectively. The solution of the equation is the final step in analysis:

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

The (zero) displacements for the stationary Degrees of Freedom are added back to the result, thus creating the Displacement vector $\{D\}$.

4.3.3.1 Tab Constraint

The final step for a successful analysis procedure is to finally determine the boundary conditions of the model. The software provides again two options to the user. He can tether either the material points or the control points that describe its geometry.

Material Points

If the user decides to tether material points then two options are activated so as the user to be able to decide to apply Dirichlet or Neumann conditions.

Control Points

If the user decides to apply the boundary conditions to the control points then a Table Widget that is responsible for the control points ID definition is automatically activated and through that he can decide not only the control points that will accept the enchain but also to determine the kind of boundary conditions.

4.4 Displacement, Strain and Stress Field

4.4.1 Displacement

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:

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

2D:

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

3D:

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

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:

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

Displacements of interpolatory control points are physical model's displacements as well.

4.4.2 Stress and Strain

The Strain vector can be evaluated at any point in the field with the help of control point Displacements and the Deformation Matrix $\left[\mathrm{B}\right]$ [2]:

1D

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

2D

$$\underset{(3\times1)}{\left\{\varepsilon(\xi,\eta)\right\}} = \underset{(3\times2N)}{\left[\mathrm{B}(\xi,\eta)\right]} \cdot \underset{(2N\times1)}{\left\{\mathrm{D}\right\}}$$

3D

$$\underset{(6\times1)}{\left\{\varepsilon(\xi,\eta,\zeta)\right\}} = \underset{(6\times3N)}{\left[\mathrm{B}(\xi,\eta,\zeta)\right]} \cdot \underset{(3N\times1)}{\left\{\mathrm{D}\right\}}$$

Applying Hooke's Constitutive Law leads to:

1D

$$\underset{(1\times1)}{\left\{\sigma(\xi)\right\}} = \underset{(1\times1)}{\left[\mathrm{E}\right]} \cdot \underset{(1\times1)}{\left\{\varepsilon(\xi)\right\}} = \underset{(1\times1)}{\left[\mathrm{E}\right]} \cdot \underset{(1\times N)}{\left[\mathrm{B}(\xi)\right]} \cdot \underset{(N\times1)}{\left\{\mathrm{D}\right\}}$$

2D

$$\underset{(3\times1)}{\left\{\sigma(\xi,\eta)\right\}} = \underset{(3\times3)}{\left[\mathrm{E}\right]} \cdot \underset{(3\times1)}{\left\{\varepsilon(\xi,\eta)\right\}} = \underset{(3\times3)}{\left[\mathrm{E}\right]} \cdot \underset{(3\times2N)}{\left[\mathrm{B}(\xi,\eta)\right]} \cdot \underset{(2N\times1)}{\left\{\mathrm{D}\right\}}$$

3D

$$\underset{(6\times1)}{\left\{\sigma(\xi,\eta,\zeta)\right\}} = \underset{(6\times6)}{\left[\mathrm{E}\right]} \cdot \underset{(6\times1)}{\left\{\varepsilon(\xi,\eta,\zeta)\right\}} = \underset{(6\times6)}{\left[\mathrm{E}\right]} \cdot \underset{(6\times3N)}{\left[\mathrm{B}(\xi,\eta,\zeta)\right]} \cdot \underset{(3N\times1)}{\left\{\mathrm{D}\right\}}$$

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.

4.4.3 Analysis Results

After the data definition has been successfully completed, the user will receive the analysis results after the procedure itself has been successfully accomplished.

For that purpose the software provides to the user a tab that will contain the analysis results so as the user can obtain them and make the appropriate changes or take the final results.

- **Stiffness Matrix**

One of the analysis results that the user can obtain is the stiffness matrix of the analyzed model.

	1	2	3	4	5	6	7
1	6.62075e+07	837719	1.01149e+07	-6.37799e+07	426647	-2.04004e+06	3.27463e+07
2	837719	6.62075e+07	1.01149e+07	426647	-6.37799e+07	-2.04004e+06	111696
3	1.01149e+07	1.01149e+07	2.26812e+08	1.9974e+06	1.9974e+06	-2.25733e+08	5.05746e+06
4	-6.37799e+07	426647	1.9974e+06	6.61973e+07	832390	-1.00723e+07	-3.2072e+07
5	426647	-6.37799e+07	1.9974e+06	832390	6.61973e+07	-1.00723e+07	56886.2
6	-2.04004e+06	-2.04004e+06	-2.25733e+08	-1.00723e+07	-1.00723e+07	2.26807e+08	-1.02002e+0
7	3.27463e+07	111696	5.05746e+06	-3.2072e+07	56886.2	-1.02002e+06	4.42575e+07
8	-111696	3.18528e+07	-1.34866e+06	-56886.2	-3.25271e+07	6.73192e+06	-8.1803e-11
9	5.05746e+06	1.34866e+06	1.13048e+08	998701	6.72623e+06	-1.13048e+08	6.74329e+06
10	-3.2072e+07	56886.2	998701	3.27435e+07	110985	-5.03615e+06	-4.24593e+0
11	-56886.2	-3.25271e+07	-6.72623e+06	-110985	3.18556e+07	1.34297e+06	1.53975e-11
12	-1.02002e+06	-6.73192e+06	-1.13048e+08	-5.03615e+06	-1.34297e+06	1.13048e+08	-1.36003e+0
13	1.07963e+07	55848	1.68582e+06	-1.07513e+07	28443.1	-340007	3.27463e+07
14	-55848	1.02006e+07	-674329	-28443.1	-1.10547e+07	3.36596e+06	-111696
15	1.68582e+06	674329	3.75637e+07	332900	3.36312e+06	-3.77435e+07	5.05746e+06

Figure 4.32. Stiffness matrix.

- **Pseudodisplacement**

	PseudoDisplacement X	PseudoDisplacement Y	PseudoDisplacement Z
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	-0.00548988	-0.00481651	-0.00114057
8	-0.003182	-0.00482496	-0.000941824
9	-0.00279156	-0.00286441	0.000573995
10	0.000419401	-0.00337351	0.000594966
11	0.00595316	-0.00128641	0.00129206
12	0.00324044	-0.00106567	0.000860276
13	-0.00616325	-0.0105391	-0.043965
14	-0.00186598	-0.00848999	-0.0440369
15	-0.005547	-0.0074203	-0.0665915

Figure 4.33. Pseudodisplacement.

- **Gauss Point Displacement**

	Displacement X	Displacement Y	Displacement Z
1	-0.00100268	-0.00100288	-0.000751972
2	-0.000853856	-0.00100662	-0.000743553
3	-0.000601438	-0.00101297	-0.000729274
4	-0.000260052	-0.000709159	-0.000333544
5	-0.000166798	-0.000736528	-0.000335742
6	-8.62615e-06	-0.000782949	-0.00033947
7	0.000873003	-0.000454605	0.00056141
8	0.000784206	-0.000486836	0.000543899
9	0.000633594	-0.000541505	0.000514197
10	-0.00372079	-0.00466139	-0.0121975
11	-0.00307159	-0.00463567	-0.0121778
12	-0.00197046	-0.00459203	-0.0121442
13	-0.00102859	-0.00375173	-0.00767158
14	-0.00060756	-0.00410097	-0.00766992
15	0.000106563	-0.00469332	-0.00766711

Figure 4.34. Gauss point displacement.

- **Gauss Point Strain**

	Strain X	Strain Y	Strain Z	Strain XY	Strain YZ	Strain ZX
1	-0.00847328	0.00080896	0.000615368	-0.00760861	0.000191708	-0.000951581
2	-0.00715924	0.000723729	0.000615368	-0.00759735	0.000170359	-0.000887102
3	-0.00493043	0.000579165	0.000615368	-0.00757826	0.000134147	-0.000777737
4	-0.00223461	0.000707824	-0.000160668	-0.00422329	-0.000304941	0.000253277
5	-0.00140292	0.000671044	-0.000160668	-0.00490548	-0.000338417	0.000240414
6	7.75122e-06	0.000608659	-0.000160668	-0.00606257	-0.000395196	0.000218597
7	0.00754941	0.000606688	-0.00127999	-0.00135271	0.000570018	0.00127688
8	0.00677489	0.000618359	-0.00127999	-0.00259735	0.000524416	0.00114848
9	0.0054612	0.000638153	-0.00127999	-0.00470844	0.00044707	0.00093069
10	-0.00556302	0.00294392	0.00144644	-0.00533519	0.000109729	0.000177816
11	-0.00429312	0.00194707	0.00144644	-0.00503571	8.21099e-05	0.000172051
12	-0.00213919	0.000256278	0.00144644	-0.00452775	3.52647e-05	0.000162271
13	-0.00173409	0.00175354	0.000121231	0.000300224	-0.000384524	-0.000555441
14	-0.000873165	0.000814101	0.000121231	-0.00230347	-0.000450528	-0.000522663
15	0.000587081	-0.000779313	0.000121231	-0.00671971	-0.000562481	-0.000467066

Figure 4.35. Gauss point strain.

- **Gauss Point Stress**

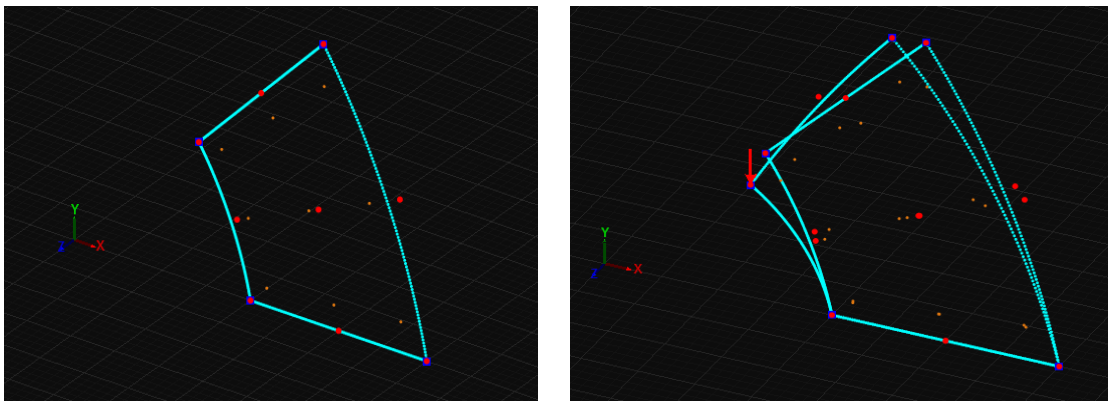
	Stress X	Stress Y	Stress Z	Stress XY	Stress YZ	Stress ZX	Stress von M
1	-2.22277e+06	-723330	-754603	-614542	15484.1	-76858.5	1.83135e+06
2	-1.86162e+06	-588222	-605727	-613632	13759.7	-71650.6	1.65686e+06
3	-1.24907e+06	-359061	-353212	-612090	10834.9	-62817.2	1.39051e+06
4	-565418	-90101	-230396	-341112	-24629.9	20457	728747
5	-334760	264.39	-134089	-396212	-27333.7	19418.1	748067
6	56467	153537	29260.8	-489669	-31919.7	17655.9	857976
7	2.05259e+06	931070	626300	-109257	46039.9	103133	1.32912e+06
8	1.83505e+06	840534	533878	-209786	42356.7	92761.7	1.24551e+06
9	1.46608e+06	686971	377118	-380297	36109.5	75171.1	1.18286e+06
10	-1.04071e+06	333483	91583.3	-430919	8862.7	14362.1	1.47392e+06
11	-802495	205536	124665	-406730	6631.96	13896.4	1.19923e+06
12	-398441	-11481.1	180776	-365703	2848.3	13106.5	814157
13	-263078	300307	36627.4	24248.9	-31057.7	-44862.5	499072
14	-133518	139040	27115.3	-186050	-36388.8	-42215.1	411669
15	86234.2	-134491	10981.5	-542746	-45431.1	-37724.6	965377

Figure 4.36. Gauss point stress.

5 Applications

5.1 Annulus 2D

The annulus presented here is subjected to plane stress. The third dimension (thickness) is significantly smaller than the other two. The degrees of freedom on the bottom are fixed. Load of 50000 *kN* is applied on the left side of the annulus.



(a) **(b)**

Figure 5.1. Annulus 1x2x0.01m.
(a) Physical space and (b) Physical space deformed.
(Inner radius 1.00 m, Outer radius 2.00 m , Thickness 0.01 m)
Knot value vector: $\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$
Knot value vector $\mathrm{H} = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$

The first representation consists of quadratic basis functions on axes ξ, η. There is formed 1 knot Rectangle (Isogeometric elements). Knot boundaries are displayed in blue. Steel has been chosen as material.

(a) (b)

Figure 5.2. (a) Index space and (b) Parameter space

Gauss point coordinates are evaluated for every knot span. In this 2D problem, $p+1=3$ gauss points per knot span are required. Thus, $3\cdot 3=9$ gauss points per Isogeometric Element.

There are $n=3$ control points on ξ and $m=3$ control points on η, for a total of 9 points and 18 degrees of freedom.

In Figure 5.3, contours for Displacement Undeformed, Displacement X and Y Undeformed are presented.

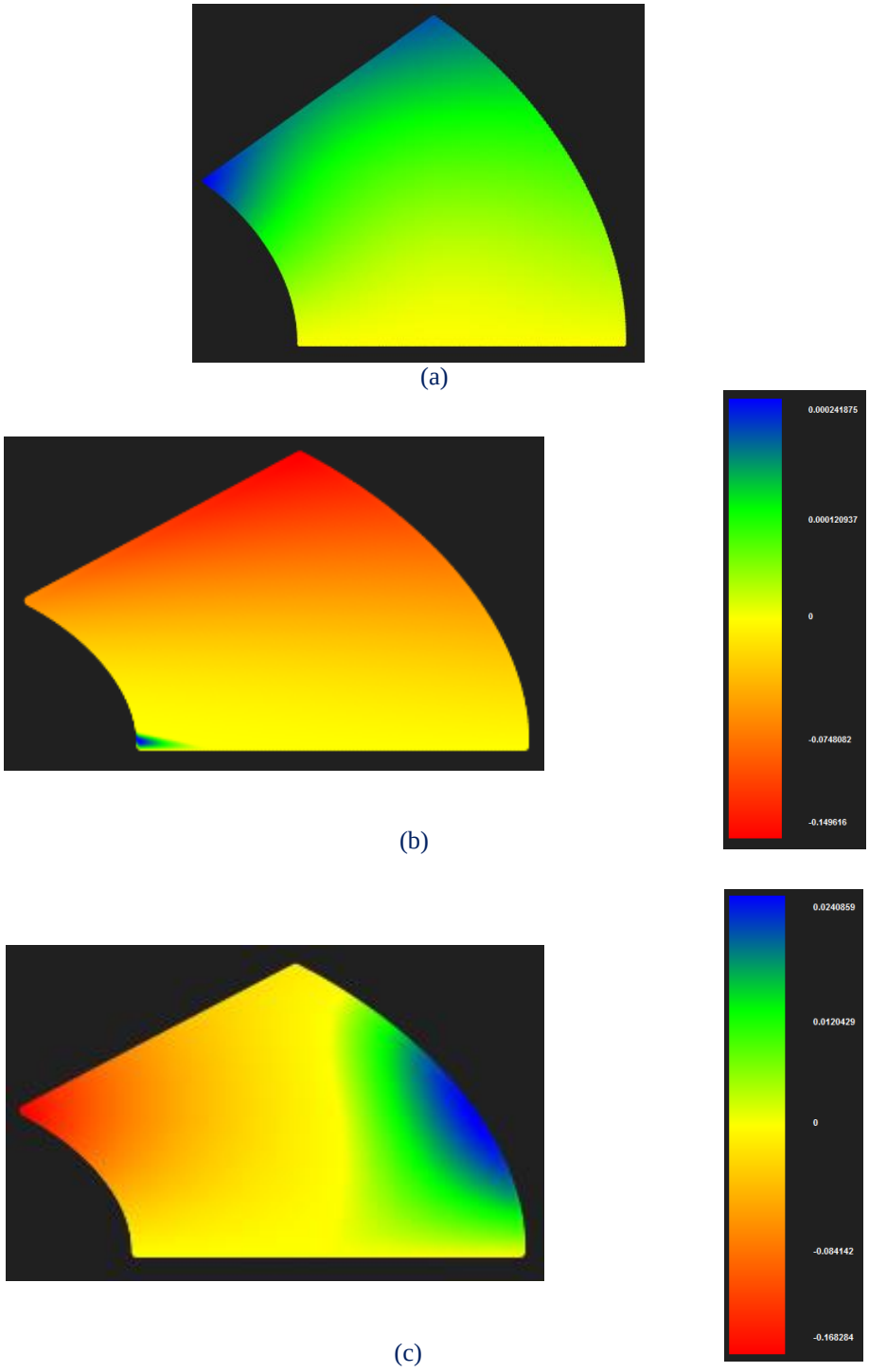


Figure 5.3. (a) Displacement(b), (c) Displacement X, Y undeformed for annulus.

As we can see from the apparent contours that have occurred for the specific load, in Displacement X the red area declares that the top side of the annulus has negative (left) displacement while at the left bottom we can discern a slight local positive displacement with blue color.

In Displacement Y we can see the negative (always according to the Cartesian system of the graphic environment-physical system) displacement as it was expected for the load we applied.

In Figure 5.4, contours for Strain X , Y and XY Undeformed are presented.

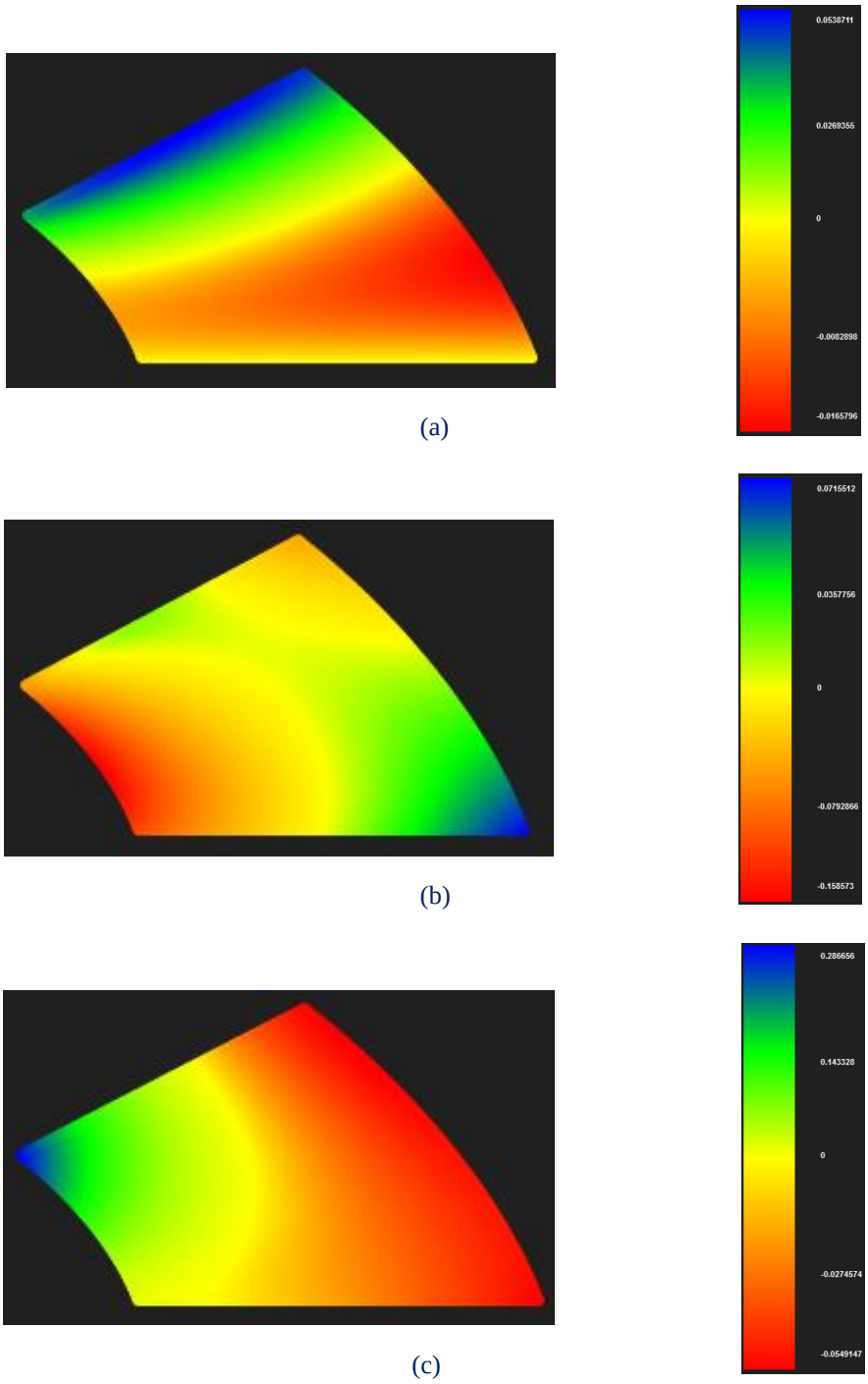


Figure 5.4. (a) Strain X (b) Strain Y, (c) Strain XY undeformed for annulus

In Figures 5.5, 5.6 contours for Stress X , Y ,XY and von Mises Undeformed are presented.

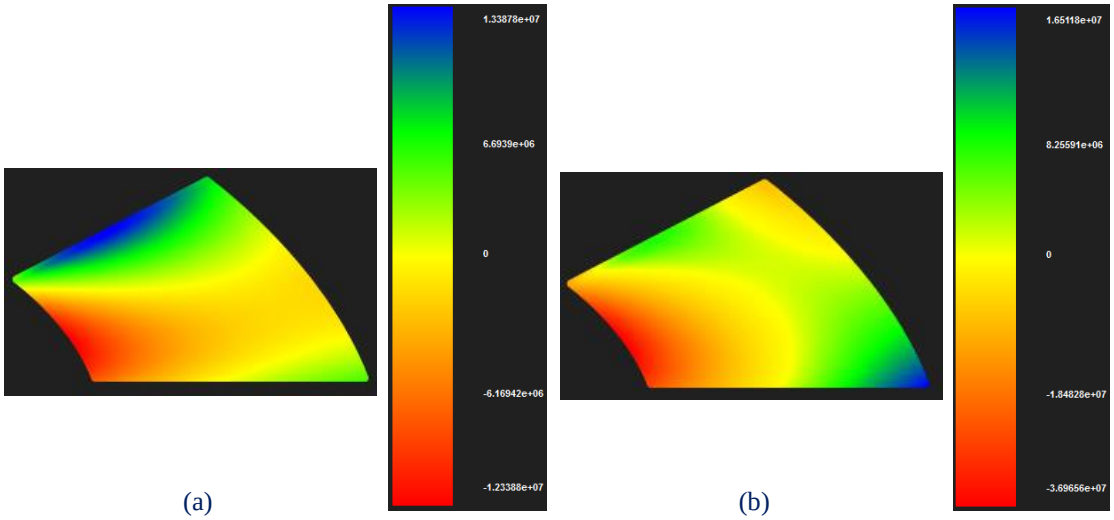


Figure 5.5. (a) Stress X undeformed(b) Stress Y undeformed for annulus

In contour (a) where Stress X is presented, we can obviously see the tensile stresses (positive values) with blue color that vertical load causes and the relative compressive stresses that harass the middle left side of the model not only at X but also at Y direction too as contour (b) indicates with the red color.

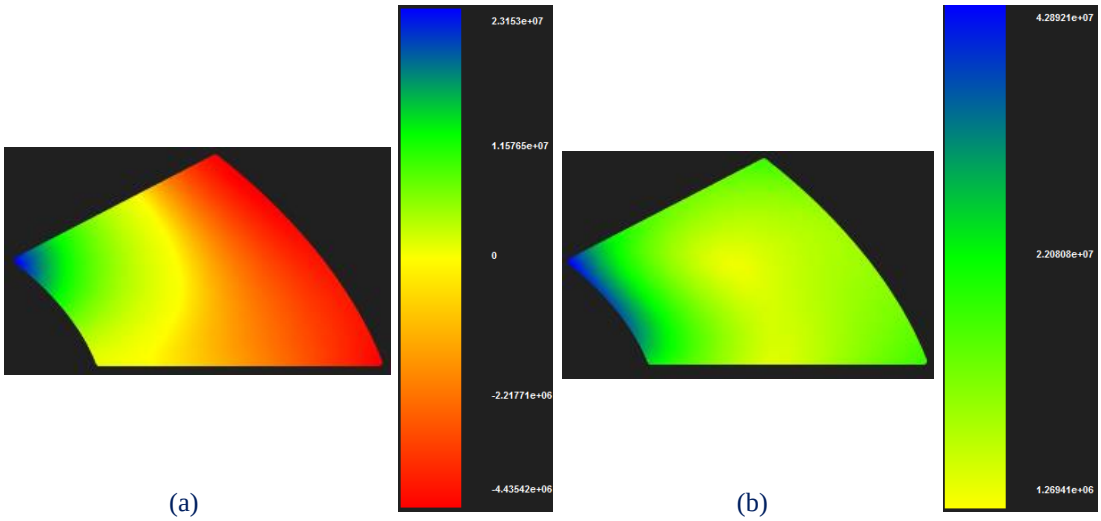
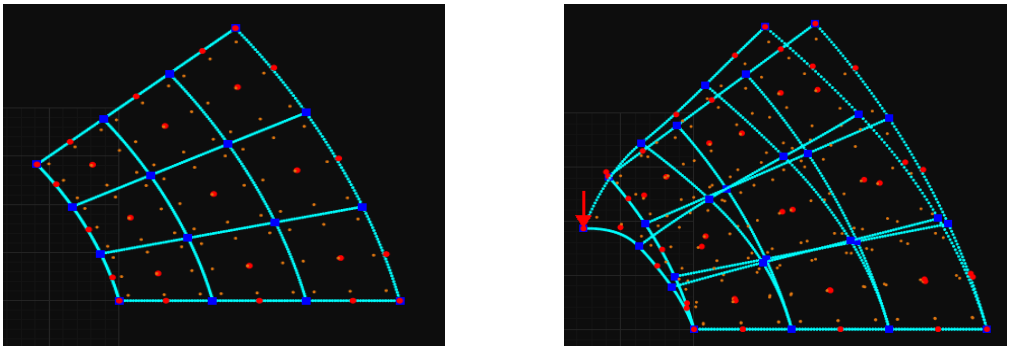


Figure 5.6. (a) Stress XY undeformed(b) Stress von Mises undeformed for annulus.

Contour (b) in Figure 5.6 shows the main harass of the model as the blue area is mainly restricted to the left corner where the concentrated load has been applied.

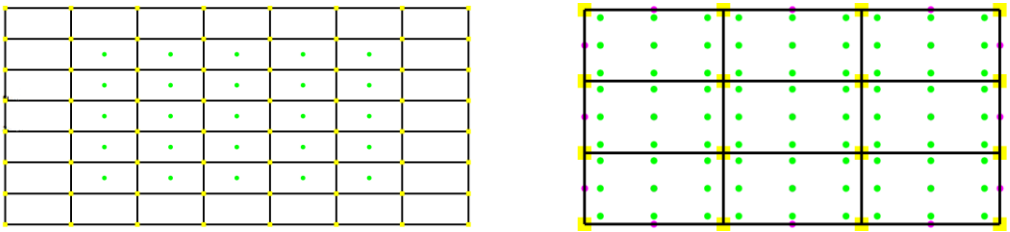
.In order to diagnose better and more accurate results and avoid higher deviation of the real solution, a parametric investigation was conducted to the Annulus Model.

Two h-Refinements were applied. Physical space, control Mesh and knot Mesh are presented in Figures 5.7 and 5.8 below.



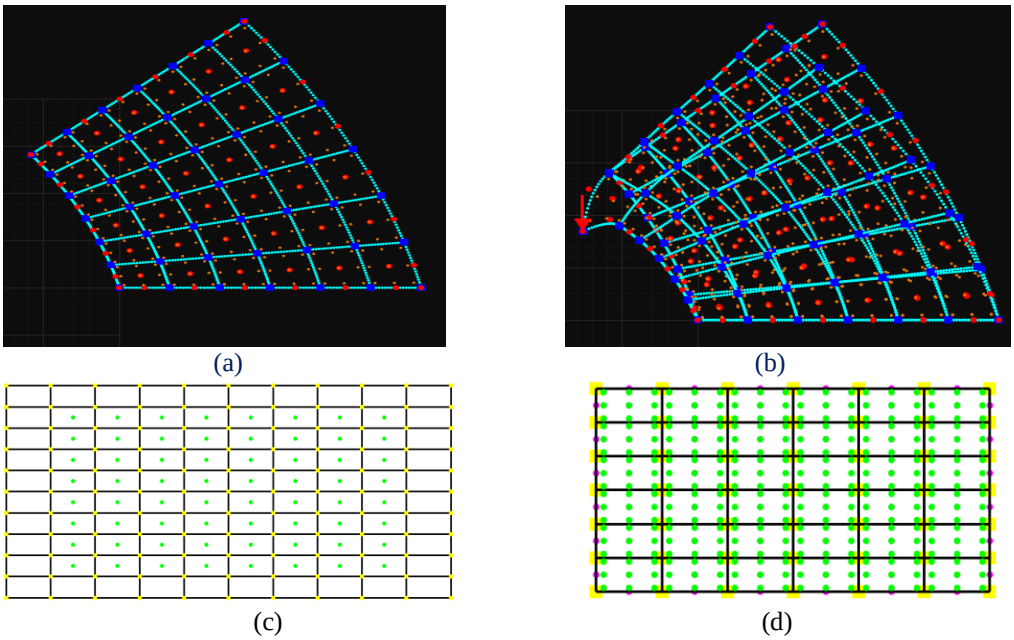
(a) (b)

In the first case the new mesh consists of $n=5$ on ξ and $m=5$ control points on η , for a total of 25 points and 50 degrees of freedom. There are formed 9 knot Rectangles this time as we can see.



(c) (d)

Figure 5.7. Annulus 5x5 control points ,
(a),(b) Physical space undeformed & deformed (c) Index space , (d) Parameter space



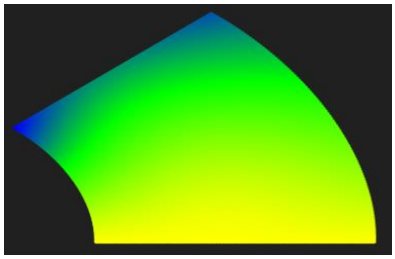
(a) (b)

(c) (d)

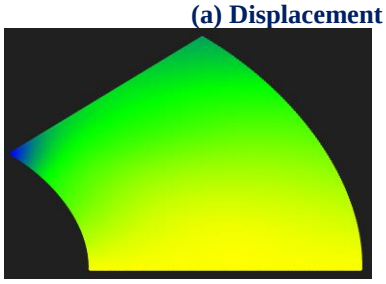
Figure 5.8. Annulus 8x8 control points ,
(a),(b) Physical space undeformed & deformed (c) Index space , (d) Parameter space.

In the second case the new mesh consists of $n=8$ on ξ and $m=8$ control points on η, for a total of 64 points and 128 degrees of freedom. There are formed 36 knot Rectangles.

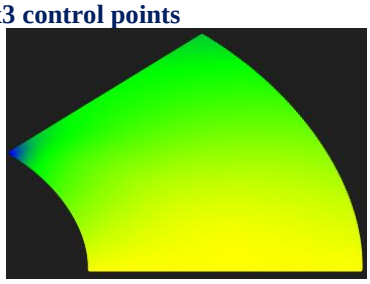
In Figure 5.9, contours for Displacement, Displacement X and Y Uneformed are presented.



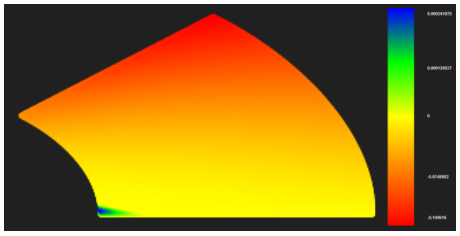
(a) Displacement 3x3 control points



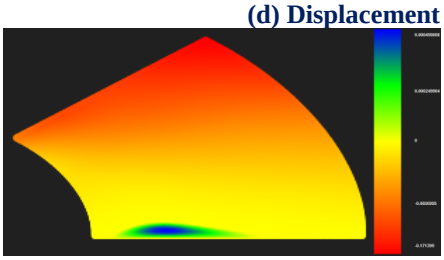
(b) Displacement 5x5 control points



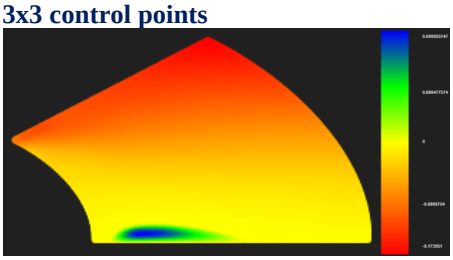
(c) Displacement 8x8 control points



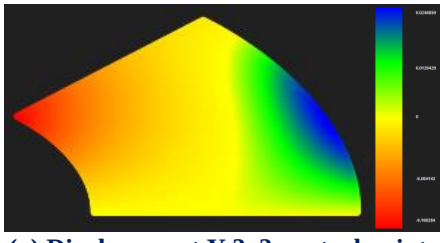
(d) Displacement X 3x3 control points



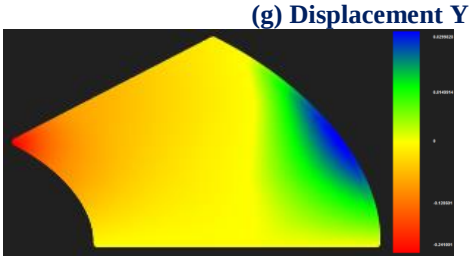
(e) Displacement X 5x5 control points



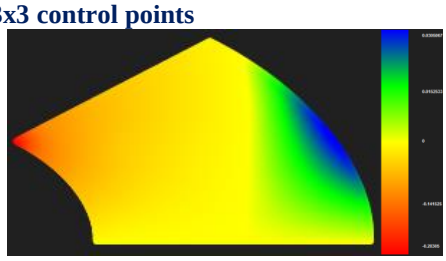
(f) Displacement X 8x8 control points



(g) Displacement Y 3x3 control points



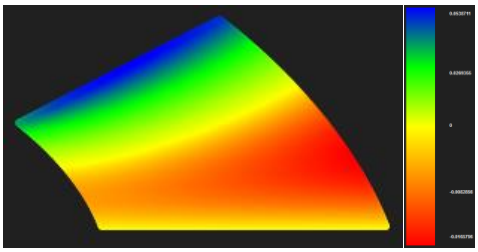
(h) Displacement Y 5x5 control points



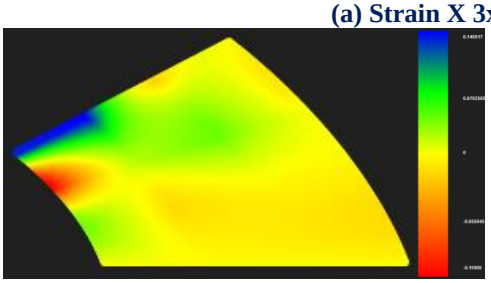
(i) Displacement Z 8x8 control points

Figure 5.9 Contours for displacement, displacement X, and Y undeformed for annulus.

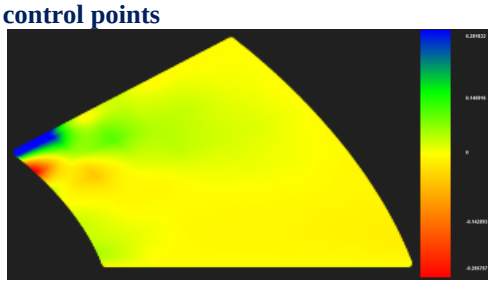
In Figure 5.10, contours for Strain X ,Y and XY Undeformed are presented.



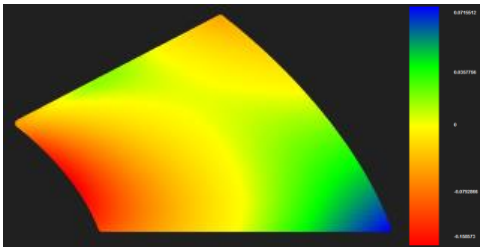
(a) Strain X 3x3 control points



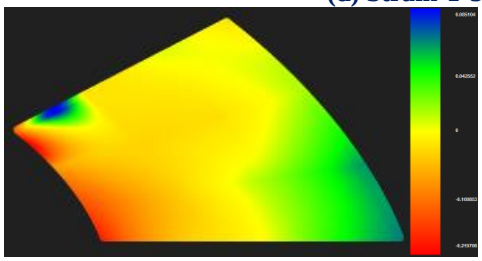
(b) Strain X 5x5 control points



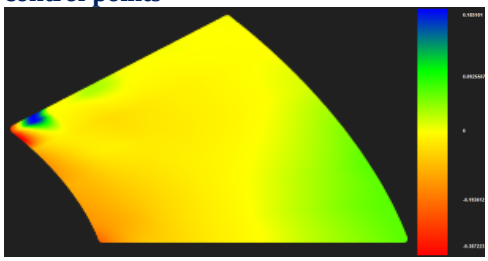
(c) Strain X 8x8 control points



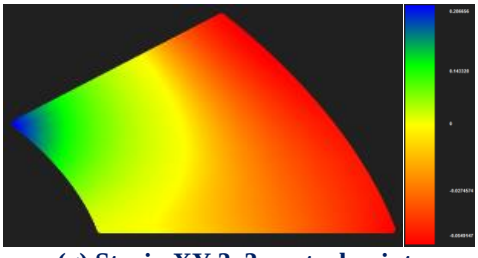
(d) Strain Y 3x3 control points



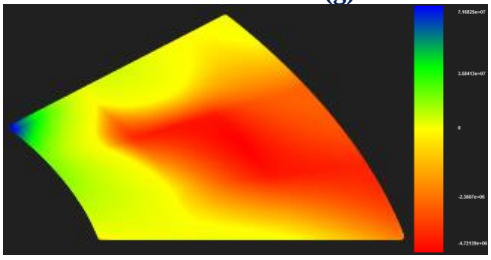
(e) Strain Y 5x5 control points



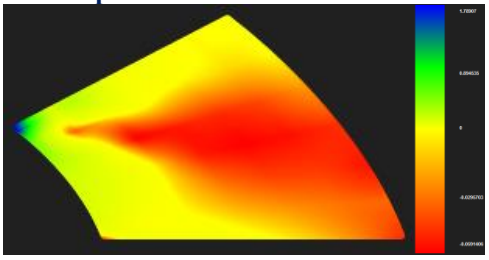
(f) Strain Y 8x8 control points



(g) Strain XY 3x3 control points



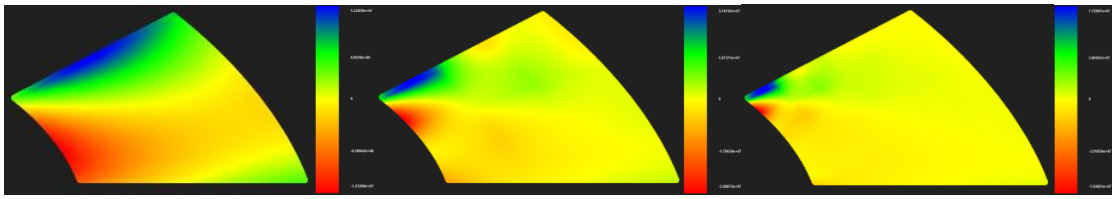
(h) Strain XY 5x5 control points



(i) Strain XY 8x8 control points

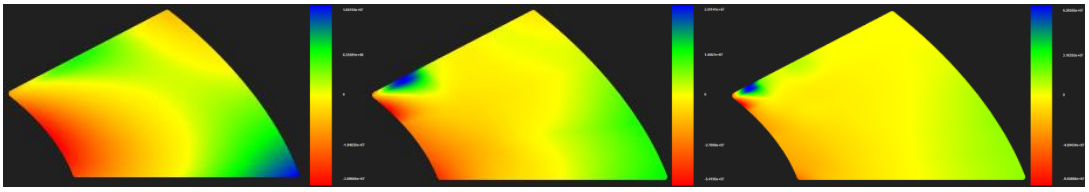
Figure 5.10 Contours for strain, X, Y and XY undeformed for annulus.

In Figure 5.11, contours for Stress X ,Y, XY and von Mises Undeformed are presented.



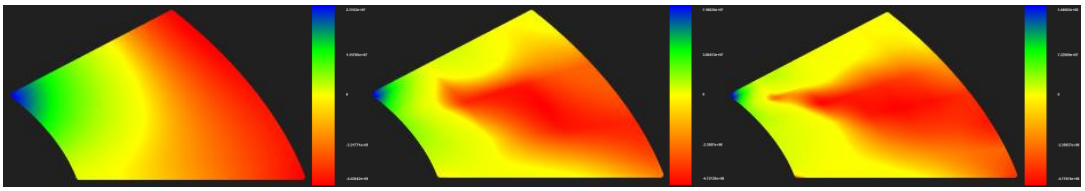
(a) Stress X 3x3 control points **(b) Stress X 5x5 control points** **(c) Stress X 8x8 control points**

In Stress X contours (Strain contours also) we can notice the fact that the greater the discretization becomes, the smaller the influenced area from the apllied load becomes.

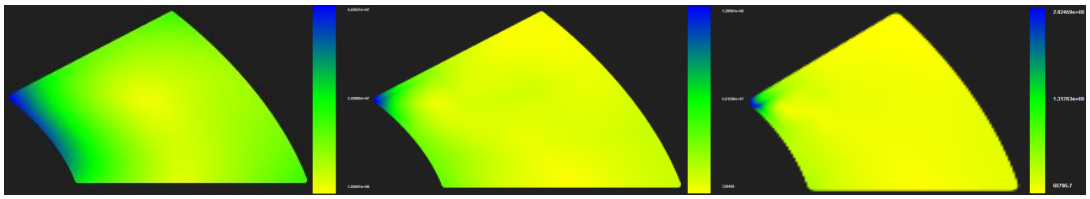


(d) Stress Y 3x3 control points **(e) Stress Y 5x5 control points** **(f) Stress Y 8x8 control points**

In Stress Y contours we can also observe that if we use only one Isogeometric element the harass of the right bottom of the model is much bigger than reality while tensile stresses on top left of the model are not even discerned. We can see how these errors are reduced by using more Isogeometric elements.



(g) Stress XY 3x3 control points **(h) Stress XY 5x5 control points** **(i) Stress XY 8x8 control points**



(j) v.Mises 3x3 control points **(k) v.Mises 5x5 control points** **(l) v.Mises 8x8 control points**

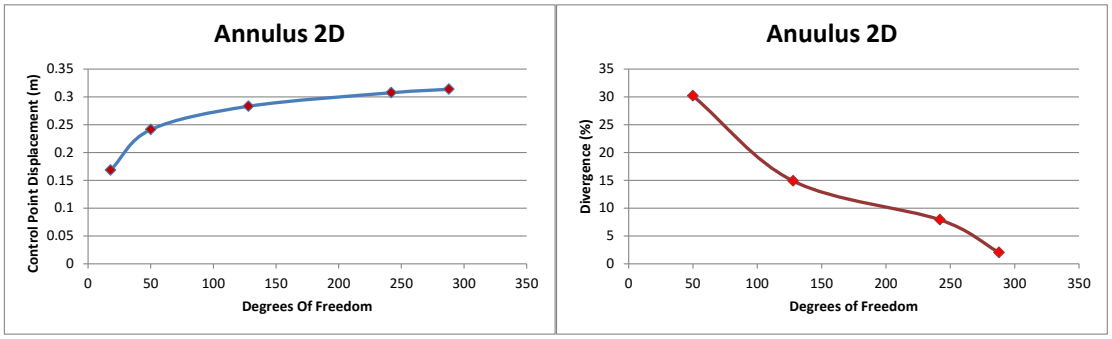
Figure 5.11 Contours for stress X, Y, XY and von Mises undeformed for annulus.

Finally at the last two contours that are presented we can verify the fact that the greater the parameterization becomes, the smoother the stress field becomes across the model.

Greater discretization results greater convergence as we can verify from the relative diagrams below.

Control point Displacement (m):

- Degrees of Freedom (18) → 0.168284 m
- Degrees of Freedom (50) → 0.241001 m → Deviation (30.17%)
- Degrees of Freedom (128) → 0.28305 m → Deviation (14.86%)
- Degrees of Freedom (242) → 0.30731 m → Deviation (7.89%)
- Degrees of Freedom (288) → 0.313609 m → Deviation (2.09%)

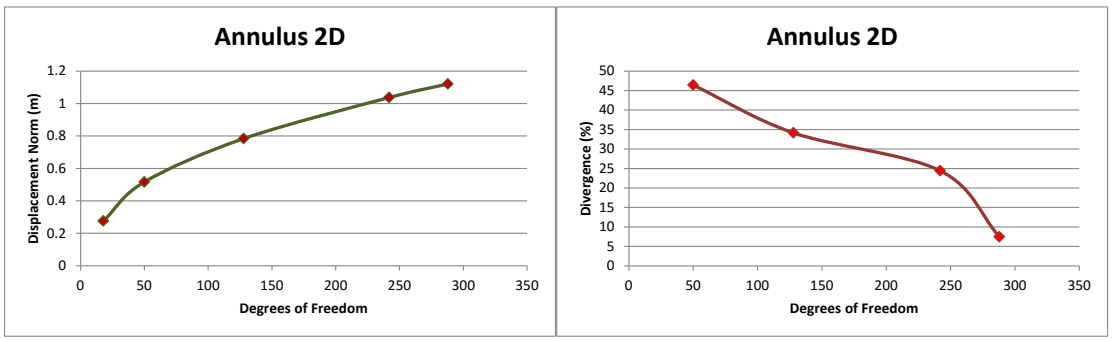


(a) **(b)**

Figure 5.12 Control point displacement results.
(a) Control point displacement in comparison with total degrees of freedom.
(b) Displacement deviation from previous solution in comparison with total degrees of freedom.

Displacement Norm (m):

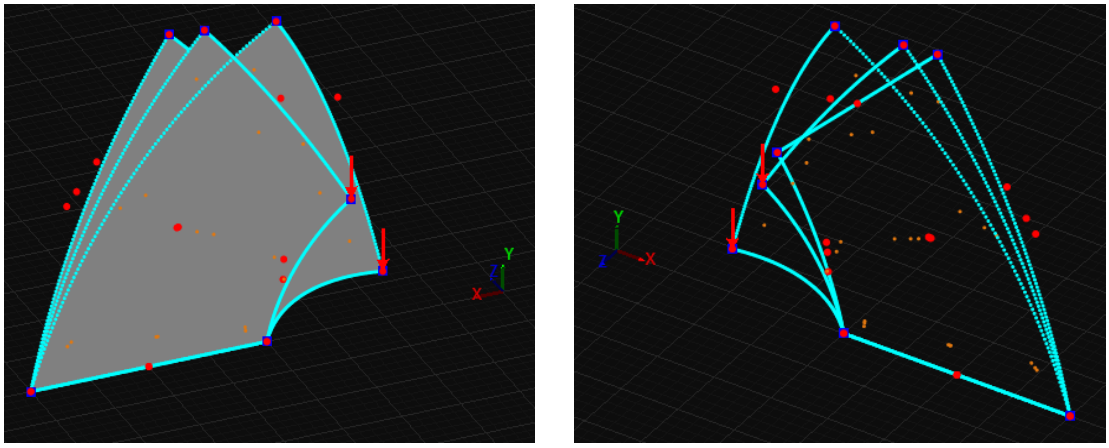
- Degrees of Freedom (18) → 0.276807 m
- Degrees of Freedom (50) → 0.5165 m → Deviation (46.41%)
- Degrees of Freedom (128) → 0.784235 m → Deviation (34.14%)
- Degrees of Freedom (242) → 1.03742 m → Deviation (24.41%)
- Degrees of Freedom (288) → 1.12069 m → Deviation (7.43%)



(a) **(b)**

Figure 5.13 Displacement norm results.
(a) Displacement norm in comparison with total degrees of freedom.
(b) Displacement norm deviation from previous solution in comparison with total degrees of freedom

We are now going to conduct an investigation on our model's material and we are going to compare their results. We will use aluminum and copper.

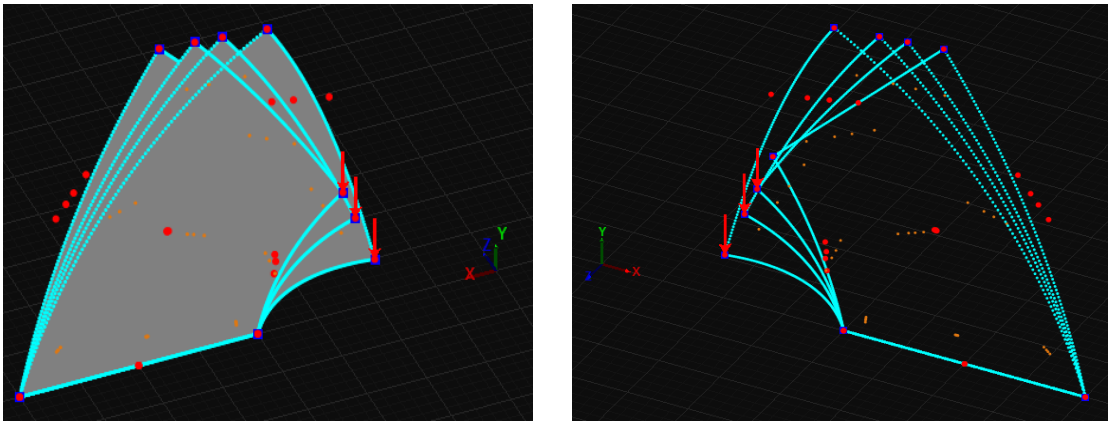


(a) Steel & Aluminum 3x3 control points **(b) Steel & Aluminum 3x3 control points**

Figure 5.14. Annulus with aluminum and steel.
(a), (b) Physical space undeformed and deformed.

Low density of aluminum is the main driver for using it in many structural applications. The high strength to weight ratio is the number one reason for the development of the aircraft industry.

The Young modulus, E is important for the structural behavior. Its value is about 1/3 that of steel, but contrary to density, this is a disadvantage compared to steel. The low value of the Young modulus, E has a big influence on the deformations of an aluminum structure. A well-known example is the bending of beams, where the stiffness EI is the governing factor and $I_{\text{Al}} = 3\ I_{\text{Steel}}$ to arrive at the same stiffness as a steel beam.



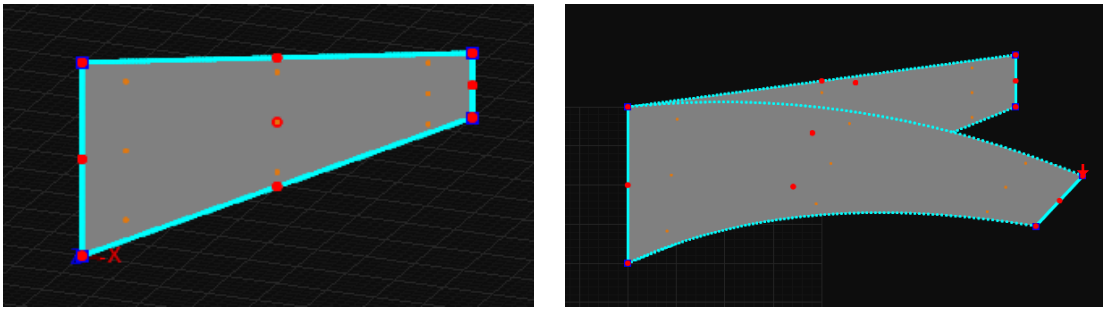
(a) Steel, Aluminum, Copper 3x3 control points **(b) Steel, Aluminum, Copper 3x3 control points**

Figure 5.15. Annulus with steel, aluminum and copper.
(a), (b) Physical space undeformed and deformed.

In Figure 5.14 we can observe the original shape of the model as also the fact that aluminum is relatively weaker from steel with as a result to engender greater deformation. In figure 5.15 we can see that copper reacts better from aluminum and worse from steel under the same load.

5.2 Cook's Cantilever

Cook's Cantilever is a characteristic plane stress problem. The third dimension (thickness) is significantly smaller than the other two. Both Cook's Cantilever and the Annulus were represented using the exact same parameterization. Only the Cartesian coordinates of the control points differentiate.



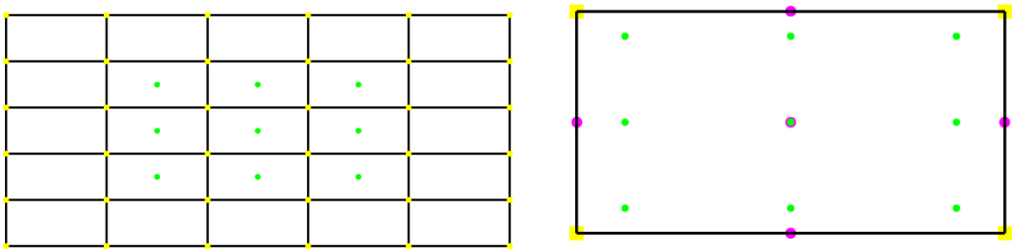
(a) **(b)**

Figure 5.16. Rectangle 1x2x0.01 m.
(a) Physical space and (b) Physical space Deformed.

Knot value vector: $\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$
Knot value vector $H = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$

The first representation consists of quadratic basis functions on axes ξ, η. There is formed 1 knot Rectangle. Knot boundaries are displayed in blue.

The degrees of freedom on the left are fixed. Load of 20000 *kN* is applied on the right side of the cantilever.



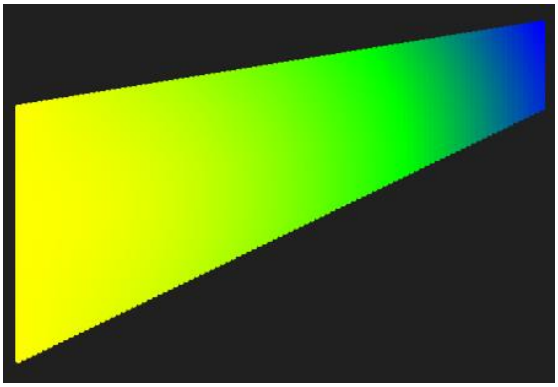
(a) **(b)**

Figure 5.17. (a) Index space and (b) Parameter space.

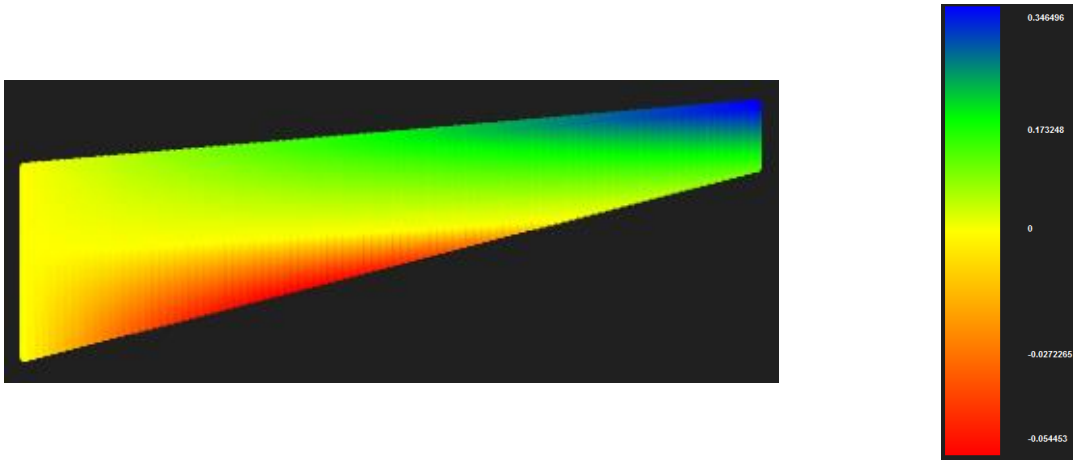
Gauss point coordinates are evaluated for every knot span. In this 2D problem, $p+1=3$ gauss points per knot span are required. Thus, $3 \cdot 3 = 9$ gauss points per Isogeometric Element.

There are $n=3$ control points on ξ and $m=3$ control points on η, for a total of 9 points and 18 degrees of freedom.

In Figure 5.18, contours for Displacement, Displacement X and Y Undeformed are presented.



(a) Displacement



(b) Displacement X

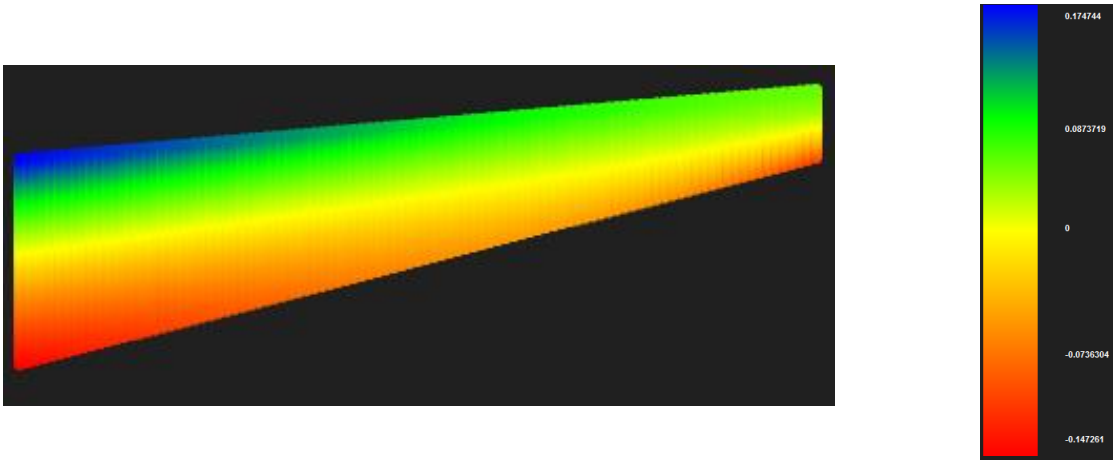
We can see in Displacement X contour the positive displacement that the applied load causes. Such a behavior was expected according to Figure 5.16 (b).

(c) Displacement Y

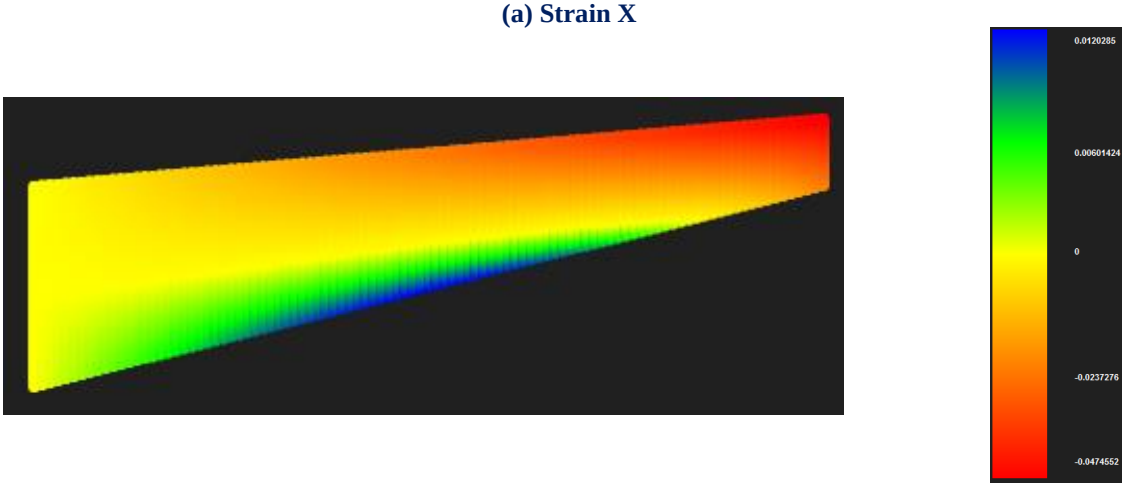
The top right red area in Displacement Y contour declares the negative (according to the physical system axes) displacement which was expected too according to Figure 5.16 (b).

Figure 5.18. Contours for displacement. undeformed

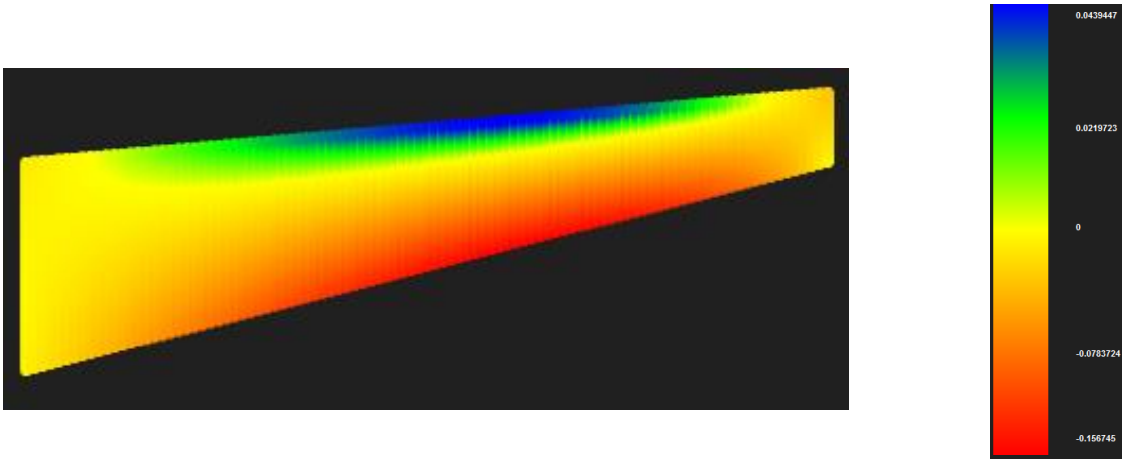
In Figure 5.19, contours for Strain X, Y and XY Undeformed are presented.



(a) Strain X



(b) Strain Y

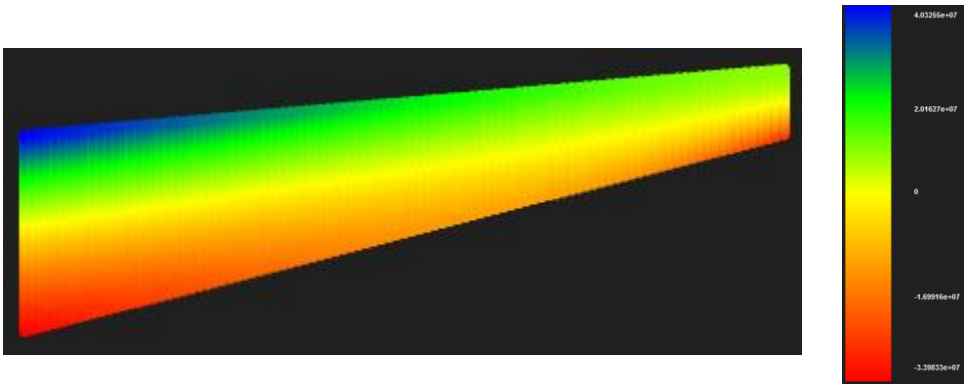


(c) Strain XY

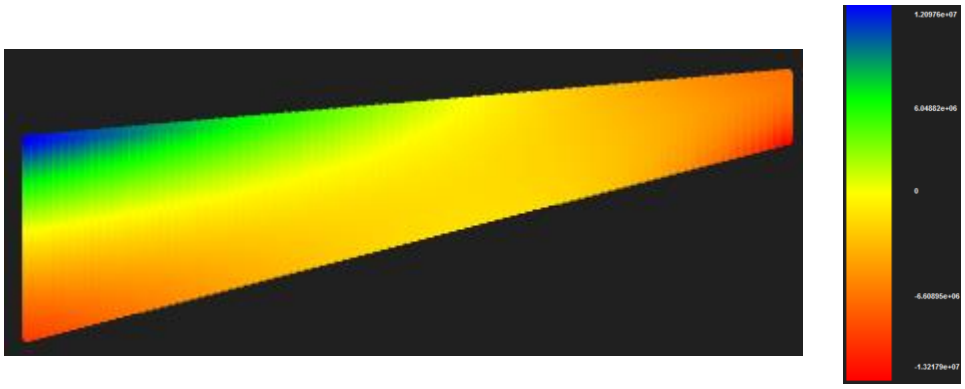
Figure 5.19. Contours for strain.X, Y and XY undeformed

In Figure 5.19 (a) the blue area at the top left and the red area at the bottom side of the model indicate the positive and negative strains that tensile and compressive stresses respectively cause according to Figure 5.20 (a).

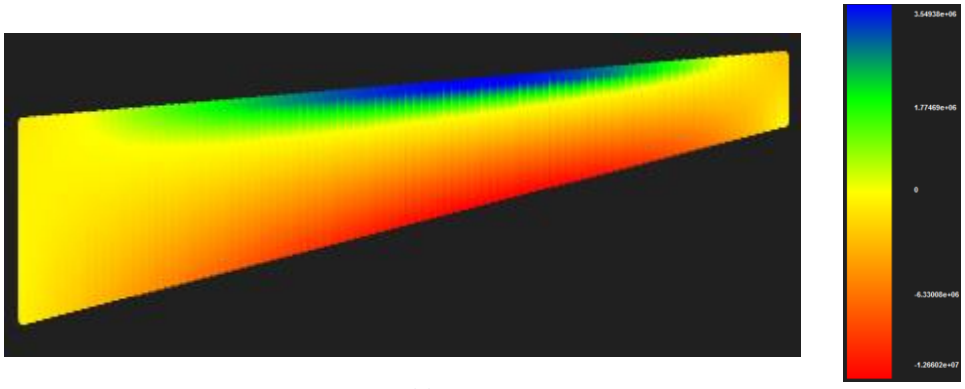
In Figure 5.20, contours for Stress X, Y, XY and von Mises Undeformed are presented.



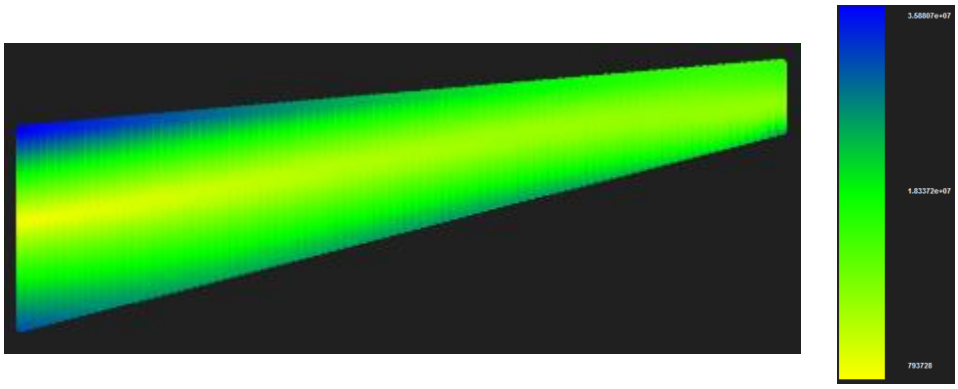
(a) Stress X



(b) Stress Y



(c) Stress XY



(d) Stress von Mises

Figure 5.20. Contours for stress X, Y, XY and von Mises.

A parametric investigation was conducted to Cook's Membrane model. Two h-Refinements, were applied. Physical space, control Net and knot Mesh are presented in Figure 5.21, 5.22 below.

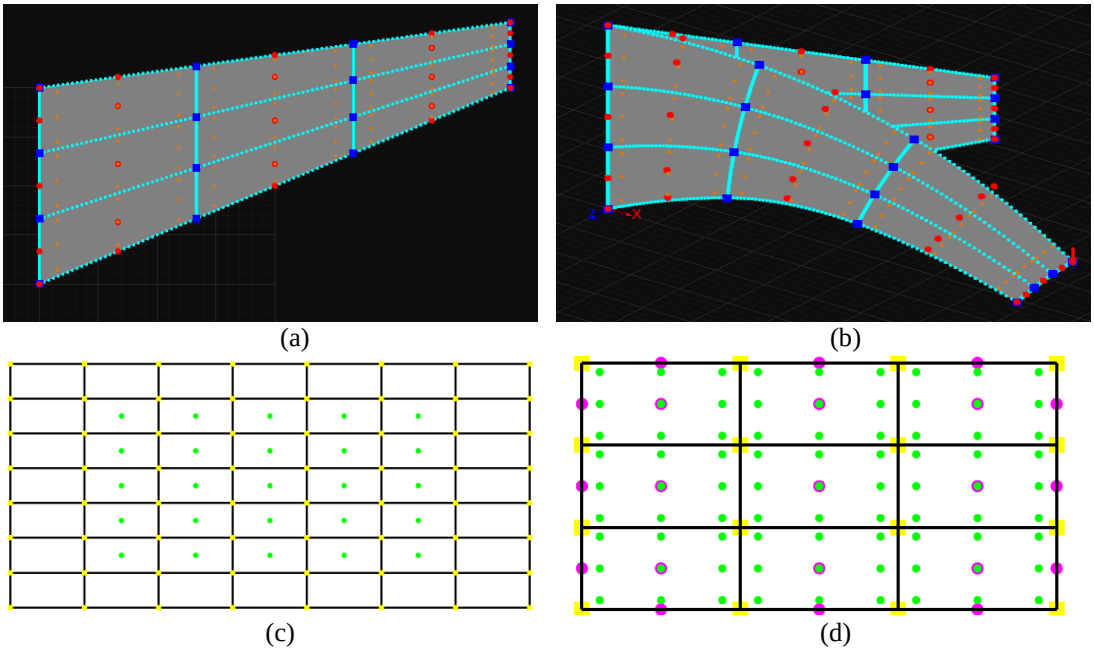


Figure 5.21. Cook's Membrane 5x5 control points.
(a) Physical space , (b) Physical space deformed(c) Index space , (d) Parameter space

In first h-Refinement the new mesh consists of $n=5$ on ξ and $m=5$ control points on η, for a total of 25 points and 50 degrees of freedom. There are formed 9 knot Rectangles this time as we can see.

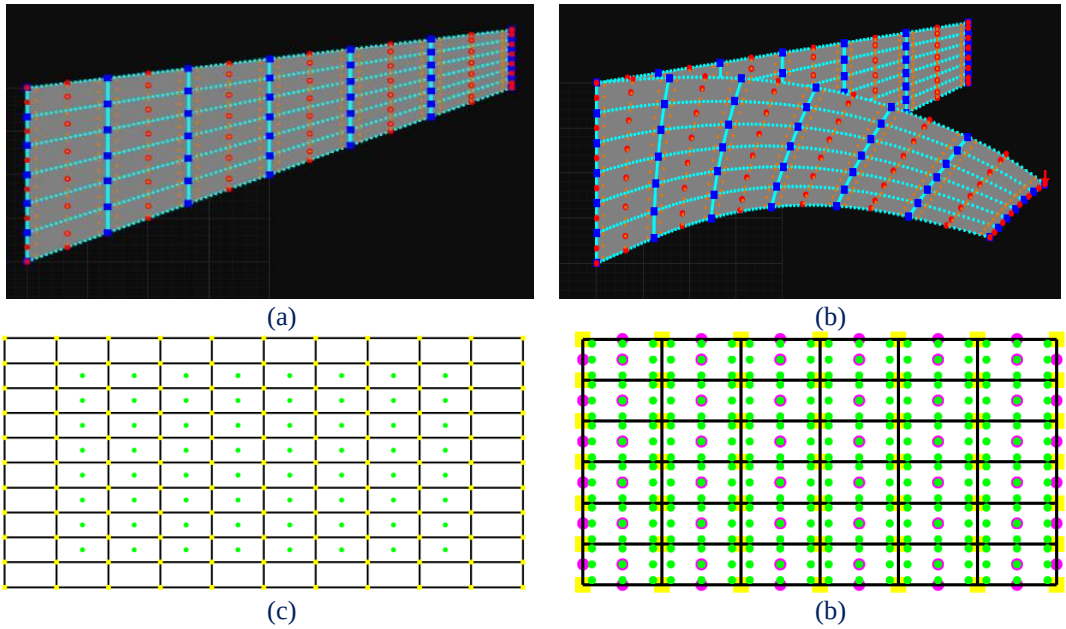
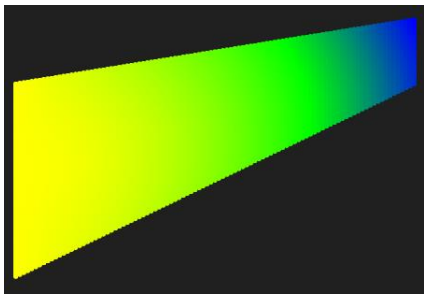


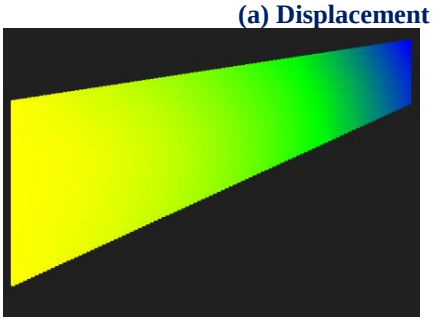
Figure 5.22. Cook's Membrane 8x8 control points.
(a) Physical space, (b) Physical space deformed(c) Index space , (d) Parameter space

In second h-Refinement the new mesh consists of $n=8$ on ξ and $m=8$ control points on η, for a total of 64 points and 128 degrees of freedom. There are formed 36 knot Rectangles

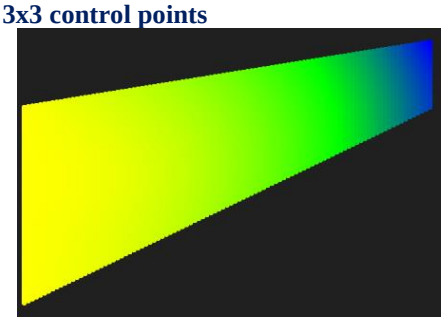
In Figure 5.23, contours for Displacement, Displacement X and Y Undeformed are presented.



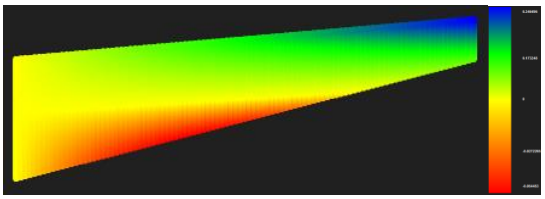
(a) Displacement 3x3 control points



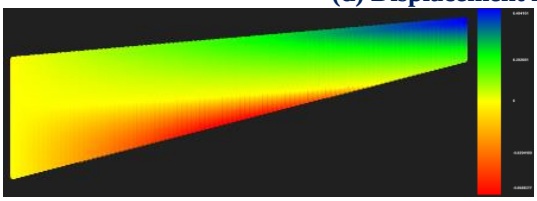
(b) Displacement 5x5 control points



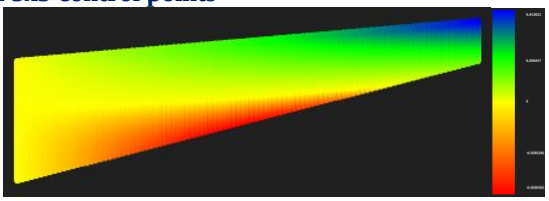
(c) Displacement 8x8 control points



(d) Displacement X 3x3 control points

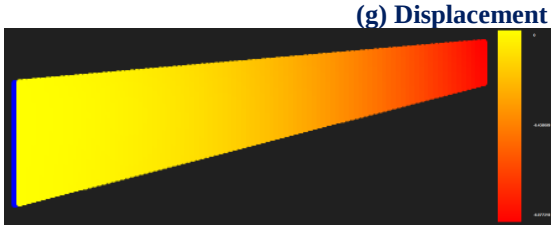


(e) Displacement 5x5 control points

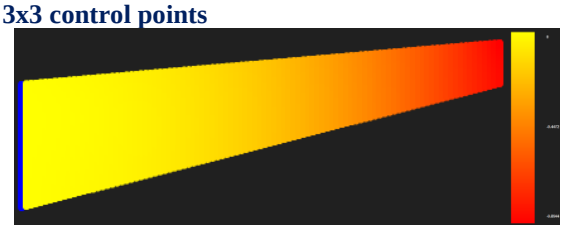


(f) Displacement 8x8 control points

(g) Displacement 3x3 control points



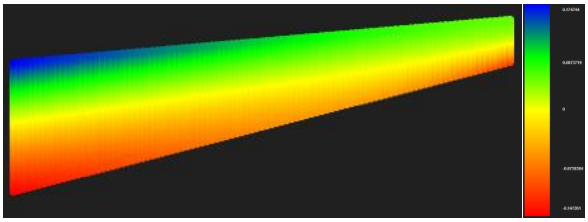
(h) Displacement 5x5 control points



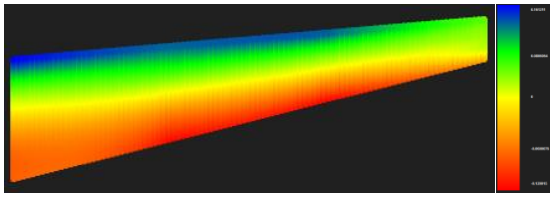
(i) Displacement 8x8 control points

Figure 5.23. Contours for displacement, displacement X, Y undeformed

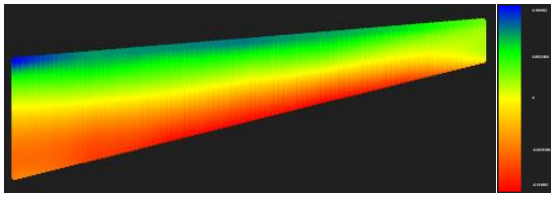
In Figure 5.24, contours for Strain X, Y and XY Undeformed are presented.



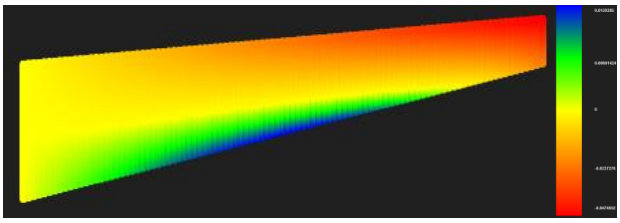
(a) Strain X 3x3 control points



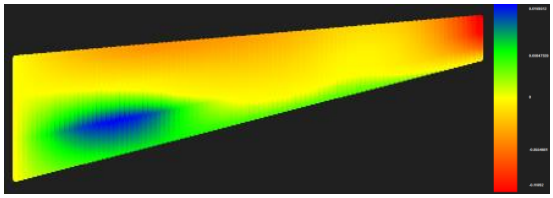
(b) Strain X 5x5 control points



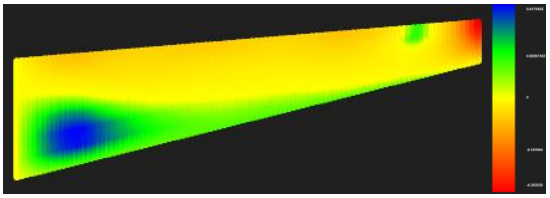
(c) Strain X 8x8 control points



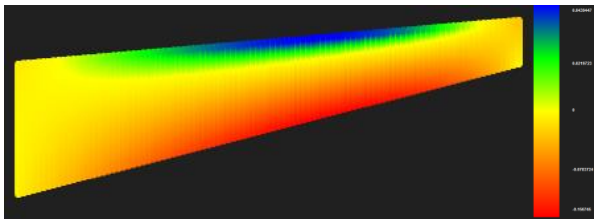
(d) Strain Y 3x3 control points



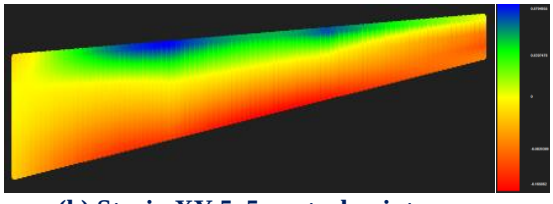
(e) Strain Y 5x5 control points



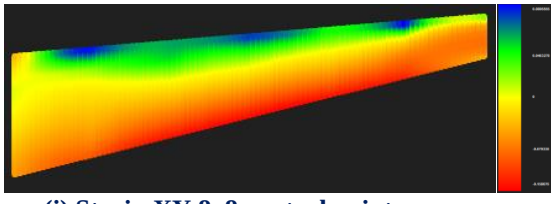
(f) Strain Y 8x8 control points



(g) Strain XY 3x3 control points



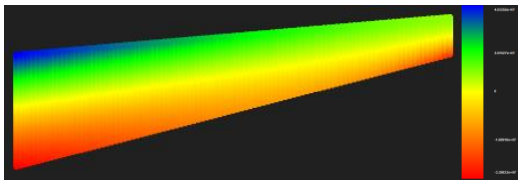
(h) Strain XY 5x5 control points



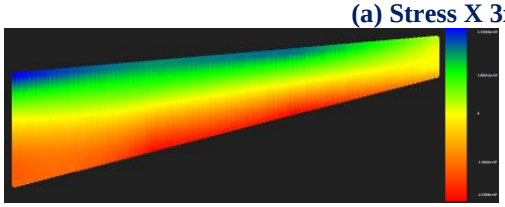
(i) Strain XY 8x8 control points

Figure 5.24. Contours for strain X, Y and XY undeformed.

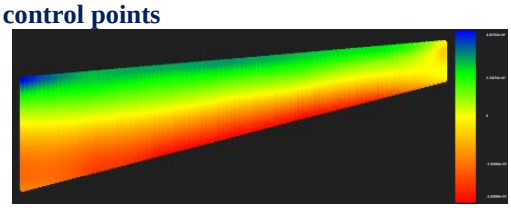
In Figure 5.25, contours for stress X, Y, XY and von Mises (undeformed) are presented.



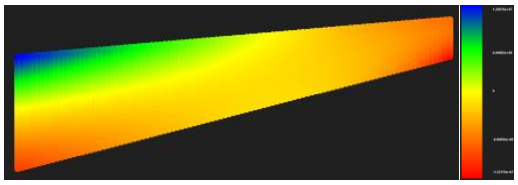
(a) Stress X 3x3 control points



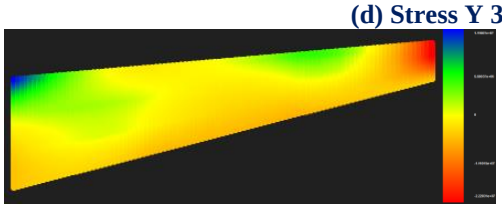
(b) Stress X 5x5 control points



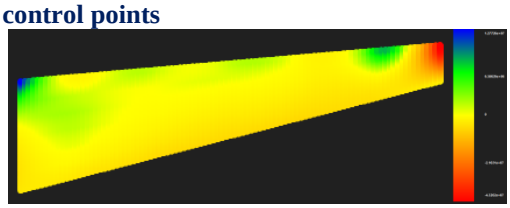
(c) Stress X 8x8 control points



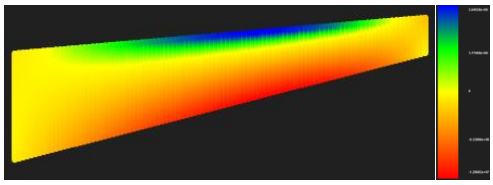
(d) Stress Y 3x3 control points



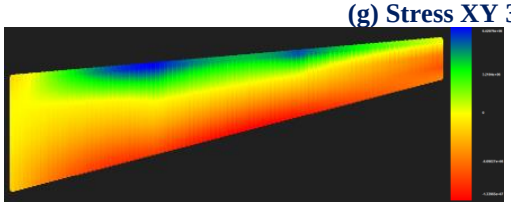
(e) Stress Y 5x5 control points



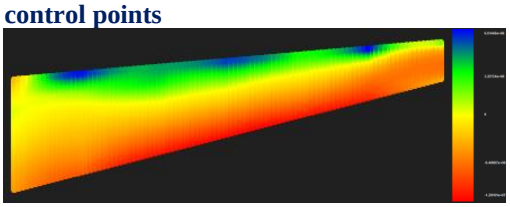
(f) Stress Y 8x8 control points



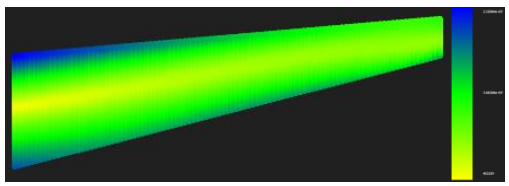
(g) Stress XY 3x3 control points



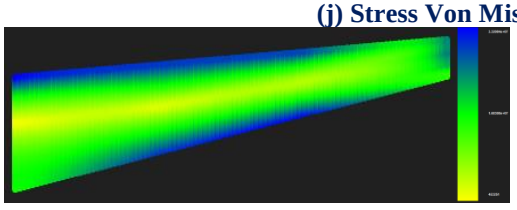
(h) Stress XY 5x5 control points



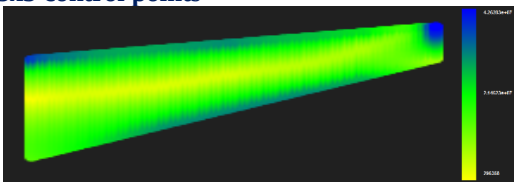
(i) Stress XY 8x8 control points



(j) Stress Von Mises 3x3 control points



(k) Stress Von Mises 5x5 control points



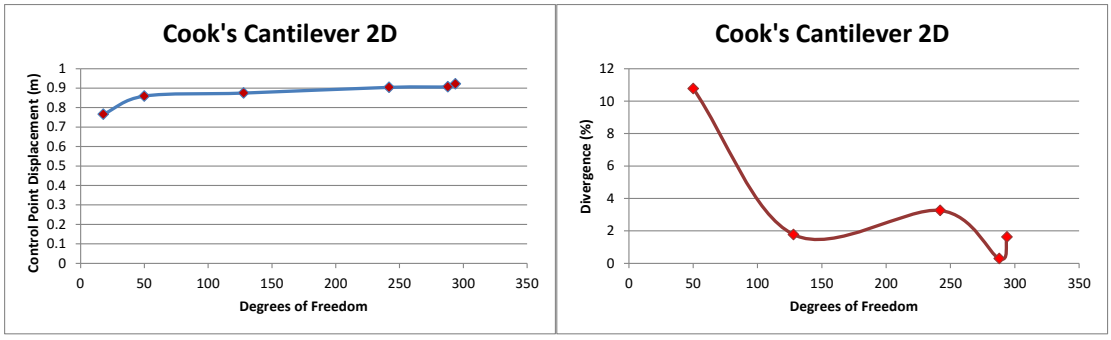
(l) Stress Von Mises 8x8 control points

Figure 5.25. Contour for stress X, Y, XY and von Mises (undeformed).

Applying several h-Refinements we get the following results.

Control Point Displacement (m):

- (3x3 CP) → Degrees of Freedom (18) → 0.766542 m
- (5x5 CP) → Degrees of Freedom (50) → 0.859137 m → Deviation (10.78%)
- (8x8 CP) → Degrees of Freedom (128) → 0.874672 m → Deviation (1.78%)
- (11x11 CP)→Degrees of Freedom (242) → 0.904115 m → Deviation (3.26%)
- (12x12 CP)→Degrees of Freedom (288) → 0.906736 m → Deviation (0.29%)
- (21x7 CP) → Degrees of Freedom (294) → 0.921668 m → Deviation (1.62%)



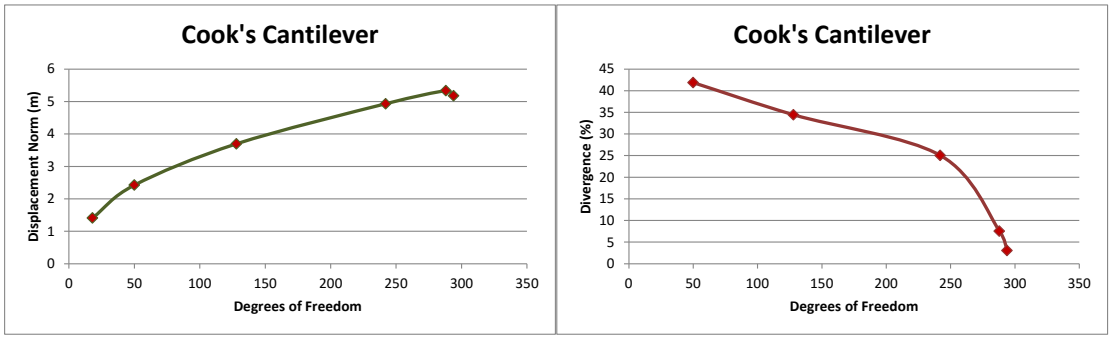
(a) (b)

Figure 5.26 Control point displacement results.
(a) Control point displacement in comparison with total degrees of freedom.
(b) Displacement deviation from previous solution in comparison with total degrees of freedom.

We can understand that the more detailed the discretization becomes the greater the convergence to the final solution becomes. What we must pay attention on, is the fact that the inappropriate increase of the DOF results greater deviation on displacement.

Displacement Norm (m):

- (3x3 CP) → Degrees of Freedom (18) → 1.40842 m
- (5x5 CP) → Degrees of Freedom (50) → 2.42302 m → Deviation (41.87%)
- (8x8 CP) → Degrees of Freedom (128) → 3.69584 m → Deviation (34.44%)
- (11x11 CP)→Degrees of Freedom (242) → 4.93103 m → Deviation (25.05%)
- (12x12 CP)→Degrees of Freedom (288) → 5.33305 m → Deviation (7.54%)
- (21x7 CP) → Degrees of Freedom (294) → 5.17476 m → Deviation (3.06%)

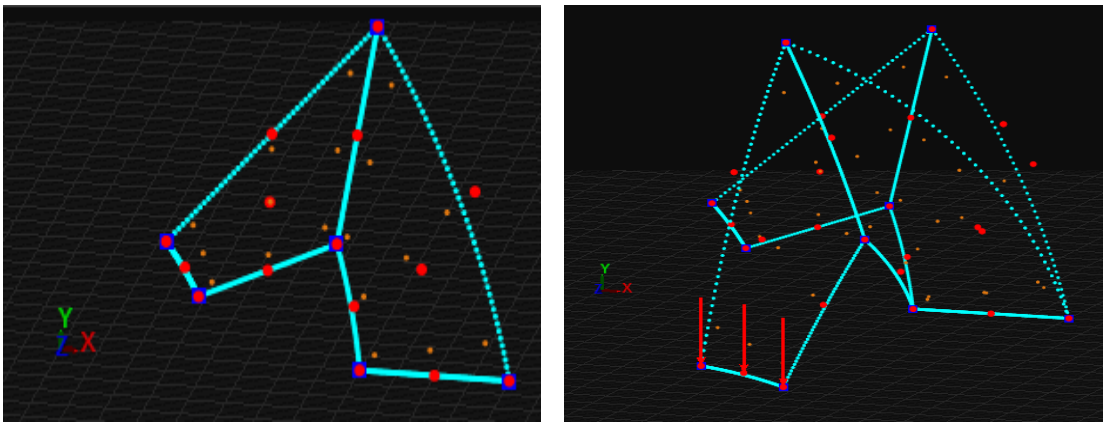


(a) (b)

Figure 5.27 Displacement norm results.
(a) Displacement norm in comparison with total degrees of freedom.
(b) Displacement norm deviation from previous solution in comparison with total degrees of freedom.

5.3 L-Shaped Curved

The next model is an L-Shaped Curved of quadrant shape. The curved shape is represented by NURBS entities of different weights. Coarse mesh representation and the corresponding weight values are displayed in Figure 5.28.



(a) (b)

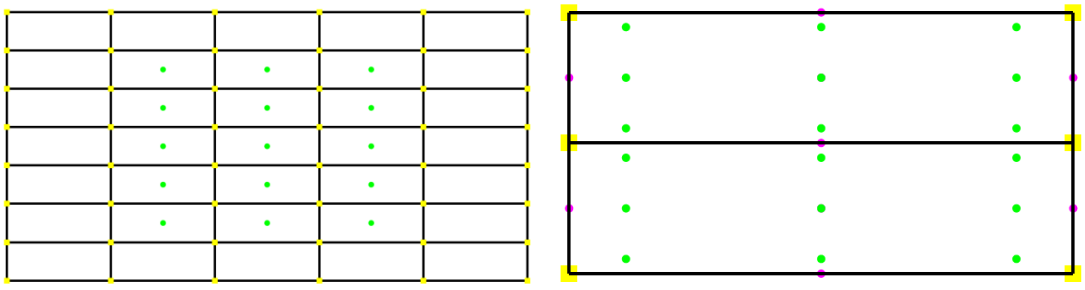
Figure 5.28. L-Shaped Curved.
(a) Physical space and (b) Physical space deformed.

Knot value vector: $\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$

Knot value vector $H = \{0 \quad 0 \quad 0 \quad 0.5 \quad 0.5 \quad 1 \quad 1 \quad 1\}$

The first representation consists of quadratic basis functions on axes ξ, η. There are formed 2 knot Rectangles. Knot boundaries are displayed in blue.

The degrees of freedom of the bottom are fixed. Load of 20000 *kN* in each of the lower three control points that the left knot Rectangle consists of are applied.



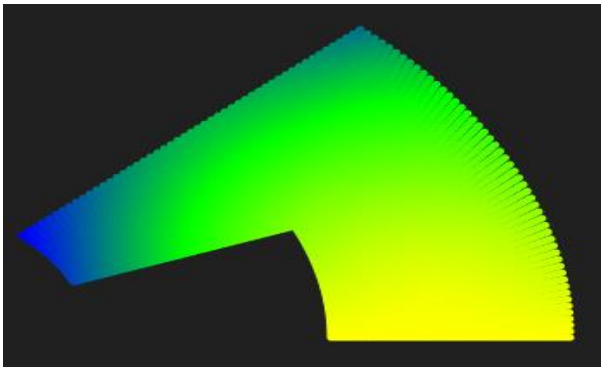
(a) (b)

Figure 5.29. (a) Index space and (b) Parameter space

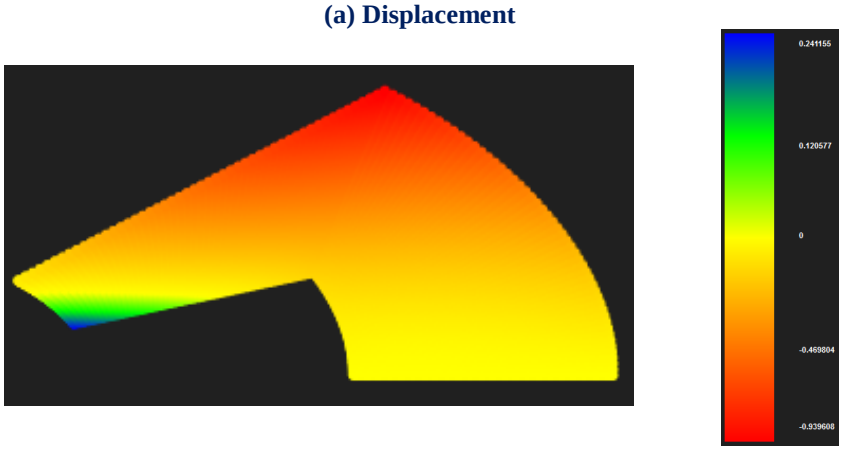
Gauss point coordinates are evaluated for every knot span. In this 2D problem, $p+1=3$ gauss points per knot span are required. Thus, $3 \cdot 3 = 9$ gauss points per Isogeometric Element.

There are $n=3$ control points on ξ and $m=5$ control points on η, for a total of 15 points and 30 degrees of freedom.

In Figure 5.30, contours for Displacement, Displacement X and Y Undeformed are presented.

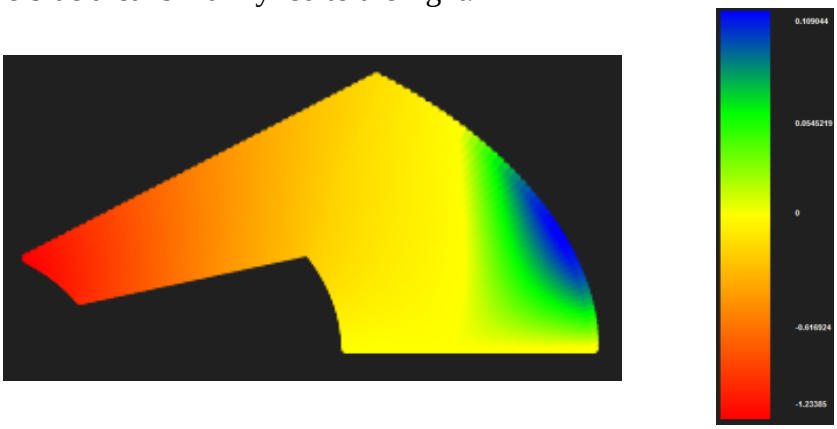


(a) Displacement



(b) Displacement X

In contour (b) where Displacement X is presented we can see that the applied load leads to a negative displacement (to the left direction, red area) the top side of the model while blue area is mainly led to the right.

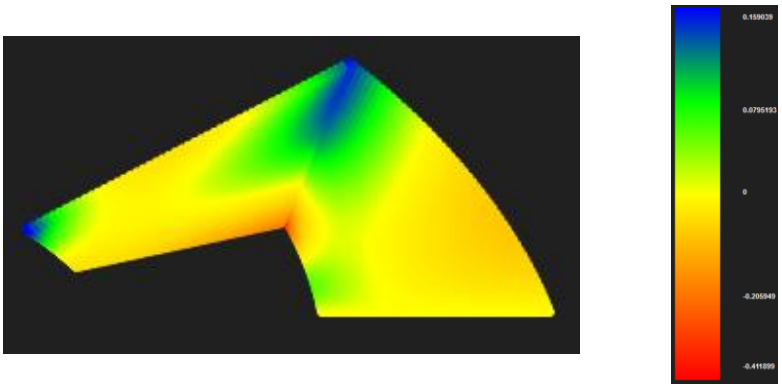


(c) Displacement Y

Figure 5.30. Contours for Displacement

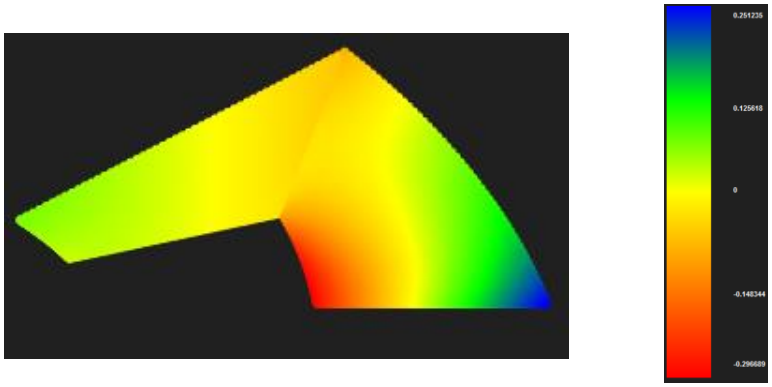
In contour (c) where Displacement Y is represented we can respectively see the red area of the model that is getting moved to the bottom of the picture something totally logical and expected if we consider the direction of the applied loads.

In Figure 5.31, contours for Strain X, Y and XY Undeformed are presented.



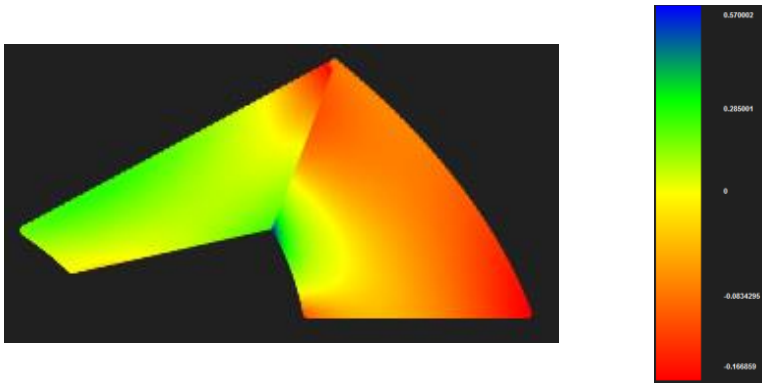
(a) Strain X

The blue area at the top of the model declares that there are positive strains at X direction which predispose us that positive stresses will be developed, something logical if we consider the load we applied. At the same view we can understand why negative strains are developed at the bottom corner of the model.



(b) Strain Y

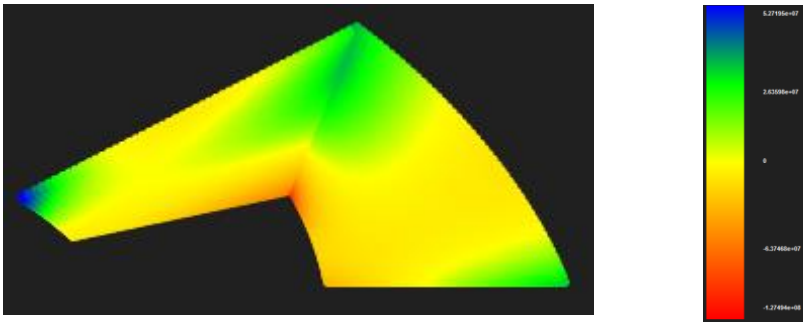
The fact that the bottom of the model has been fixed combined with the direction of the load explains the reason why the right side of the model deploys positive strains at Y direction and respectively why the left side deploys negative strains.



(c) Strain XY

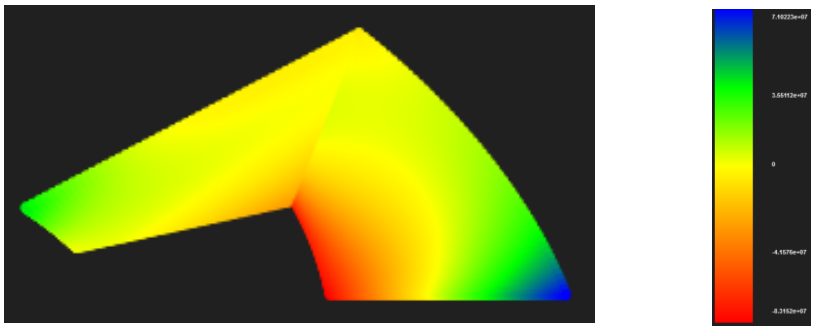
Figure 5.31. Contours for strain X, Y and XY

In Figure 5.32, contours for Stress X, Y, XY and von Mises Undeformed are presented.



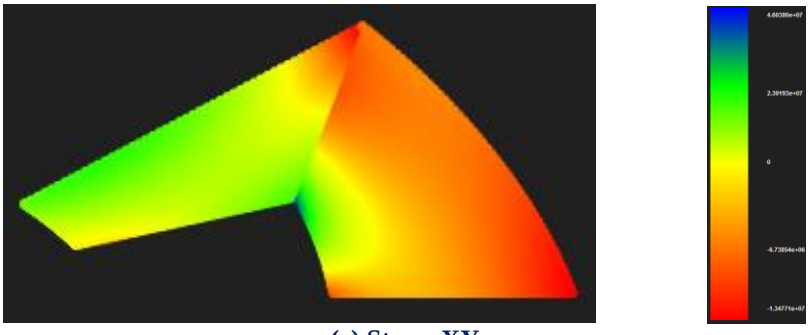
(a) Stress X

We can see in Stress X contour the green color (tensile stresses) concentrated to the area where the positive strains in Figure 5.31 (a) were respectively concentrated.

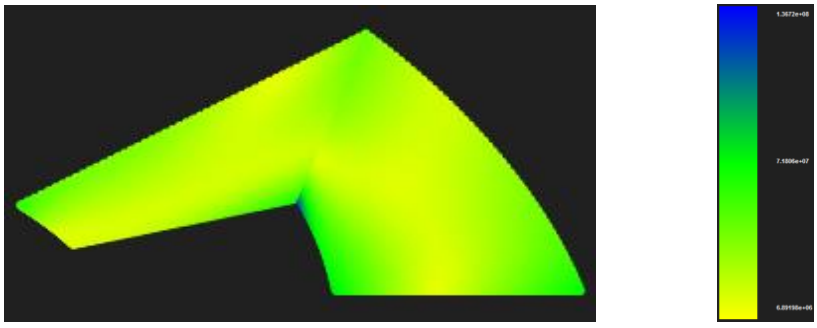


(b) Stress Y

At the same scope of view as we were predisposed tensile and compressive stresses are gathered at the same location where the respective strains showed up.



(c) Stress XY

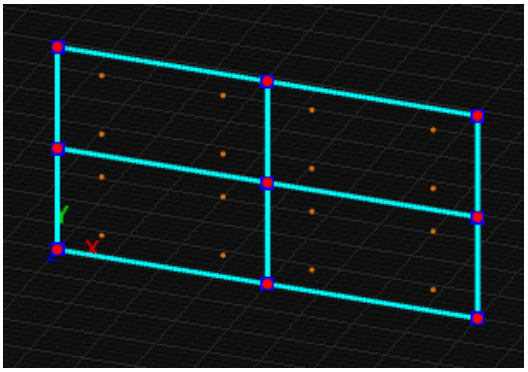


(d) Stress von Mises

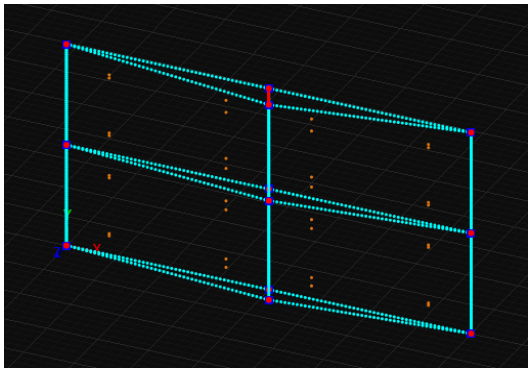
Figure 5.32. Contours for stress X, Y, XY and von Mises undeformed.

5.4 Rectangle

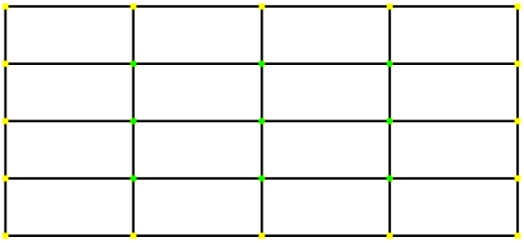
The next model is a rectangle of quadrant shape 2x1x0.01m and subjected to plane stress. The first representation consists of linear basis functions on axes ξ, η, while the second of quadratic.



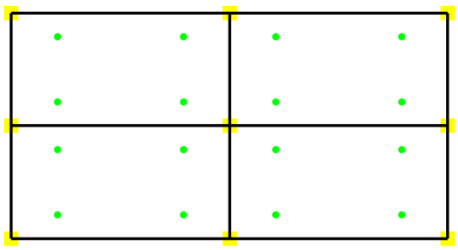
(a) Physical space 3x3 control points, p=1



(b) Physical space deformed, p=1

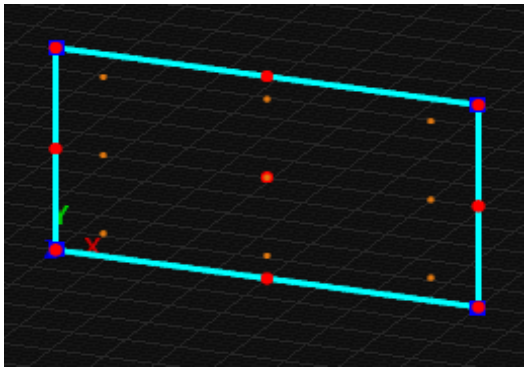


(c) Index space, p=2

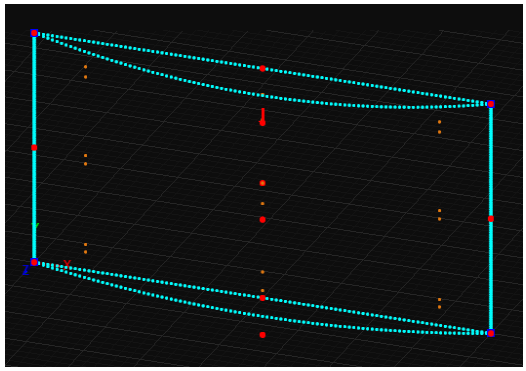


(d) Parameter space, p=2

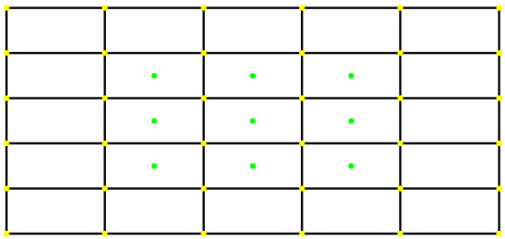
The degrees of freedom on right and left side are fixed. Load of 100000 *kN* at the middle of the model is applied. Material is steel.



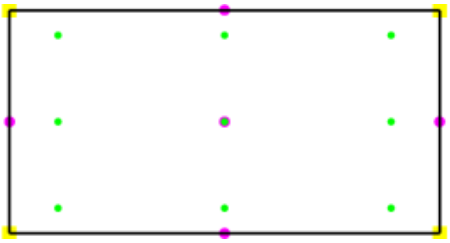
(e) Physical space 3x3 control points, p=2



(f) Physical space deformed, p=2



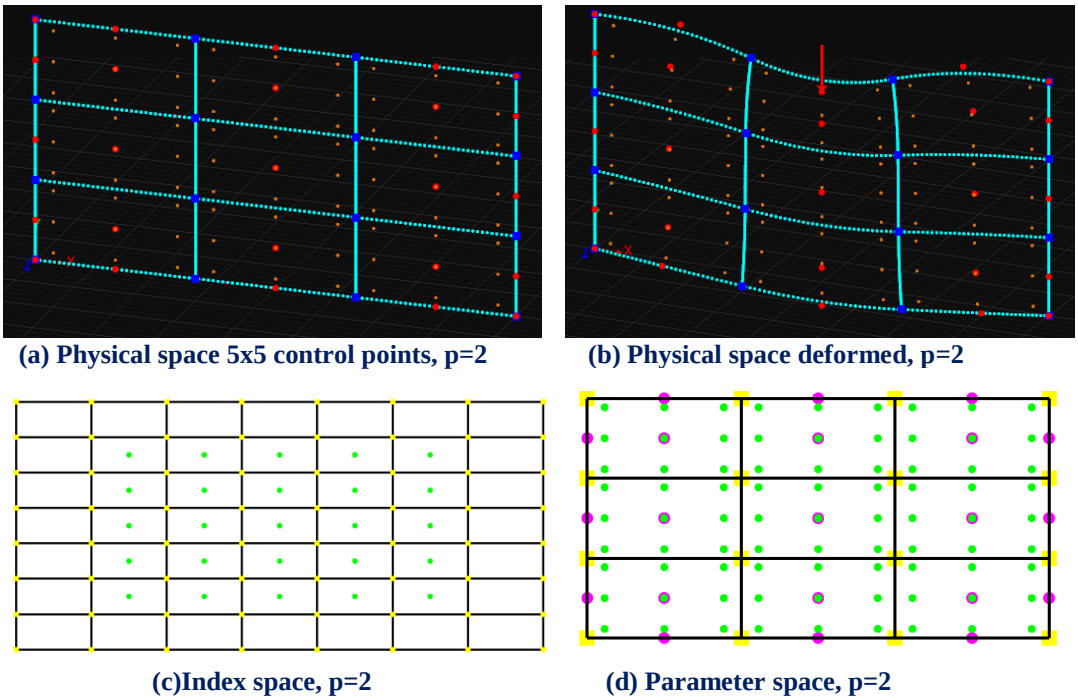
(g) Index space, p=2



(h) Parameter space, p=2

Figure 5.33 Mesh representation of 3x3 control points rectangle, p=1,2.

The third representation consists of quadratic basis functions on axes ξ,η. There are $n=5$ control points on ξ and $m=5$ control points on η, for a total of 25 points and 50 degrees of freedom.



(a) Physical space 5x5 control points, p=2

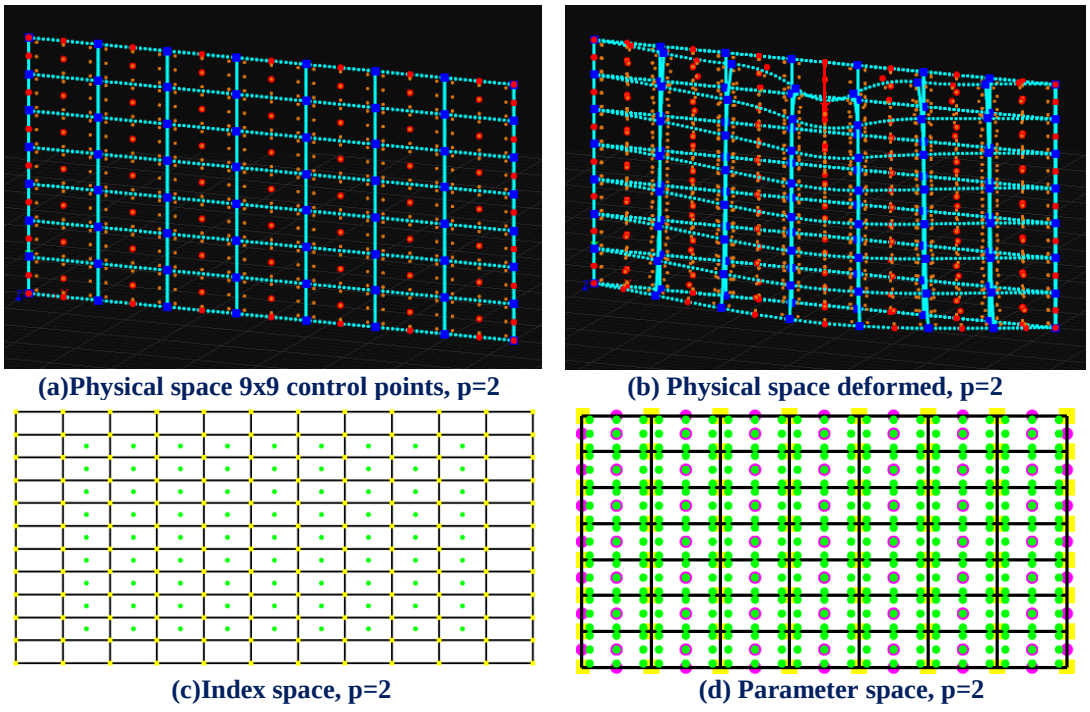
(b) Physical space deformed, p=2

(c)Index space, p=2

(d) Parameter space, p=2

Figure 5.34. Mesh representation of 5x5 control points rectangle.

The forth representation consists of quadratic basis functions on axes ξ,η. There are $n=9$ control points on ξ and $m=9$ control points on η, for a total of 81 points and 128 degrees of freedom.



(a)Physical space 9x9 control points, p=2

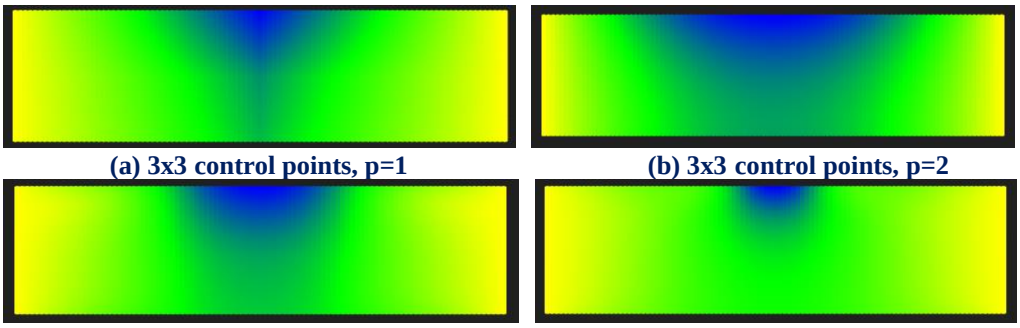
(b) Physical space deformed, p=2

(c)Index space, p=2

(d) Parameter space, p=2

Figure 5.35. Mesh representation of 9x9 control points rectangle.

In Figures 5.36, 5.37, 5.38, contours for Displacement, Displacement X and Y Undeformed are presented

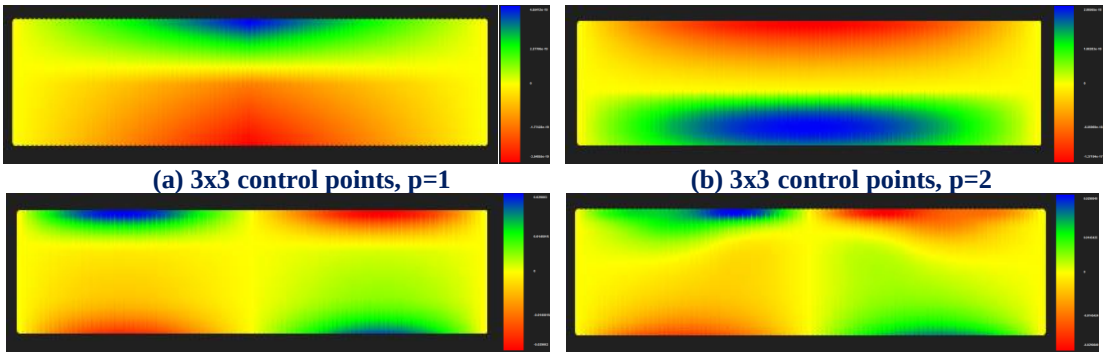


(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

Figure 5.36. Displacement

In Displacement X contours what we mainly observe is the fact that the greater the parameterization becomes, the better the accuracy becomes as the blue area is concentrated at the applied load. Another thing that may puzzle as is that by using linear basis fanctions with the same number of control points we get better results than using quadratic as far as displacement concerned and the reason is that in (a) four Isogeometric elements are created while in (b) we have only one.



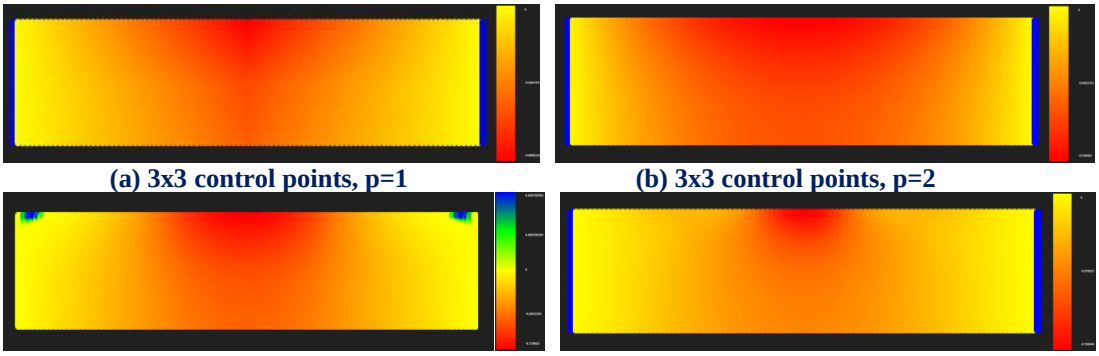
(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

Figure 5.37. Displacement X

As far as Displacement X contours concerned we can see that if we use 3x3 control points, whatever degree, we get completely wrong results. On the contrary on (c) and (d) the results become much better while at the top of the model we have concentration of the material points to the applied load and at the bottom decentralization.

The blue area is removed from X axis physical system while the red comes closer.

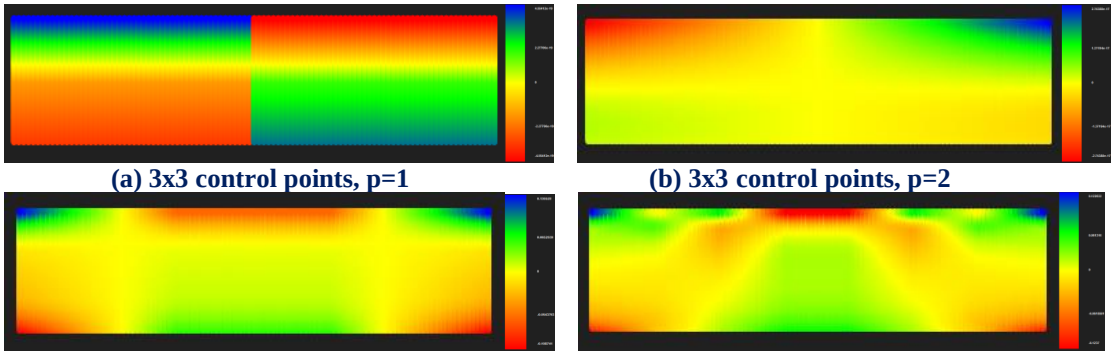


(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

Figure 5.38. Displacement Y

In Figure 5.39, 5.40, 5.41, contours for Strain X, Y and XY Undeformed are presented.

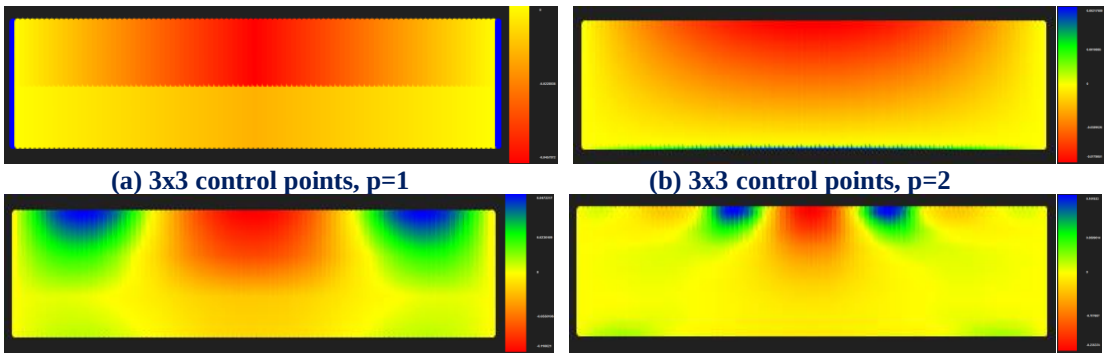


(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

Figure 5.39. Strain X

In Strain X contours we can mainly see the fact that the greater the parameterization becomes, the smoother the flow of Strain contours become across the model.

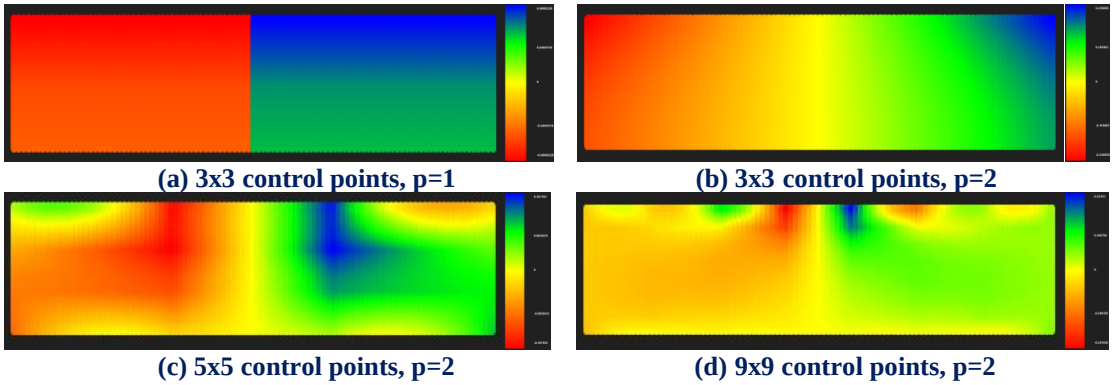


(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

Figure 5.40. Strain Y

In Strain Y contours we can verify again the increasing accuracy. The red area is concentrated above the applied load (which was expected according to figure 5.35 (b) and that predispose us for the compressive stresses that we are going to see in Figure 5.43 (d).

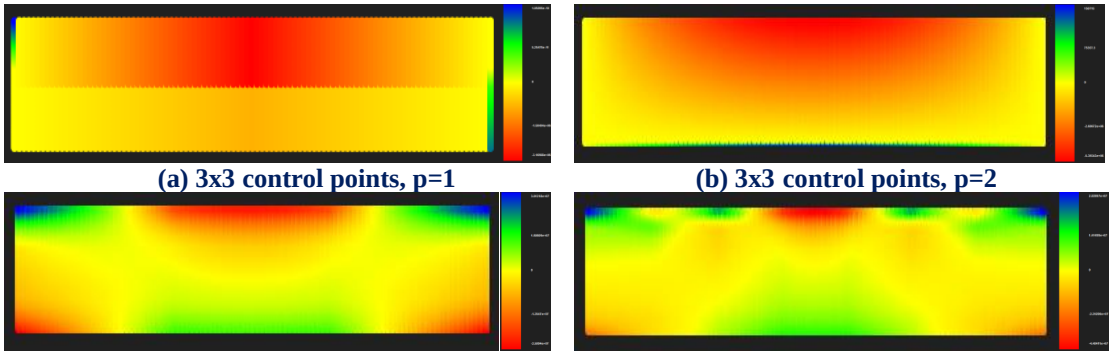


(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

Figure 5.41. Strain XY

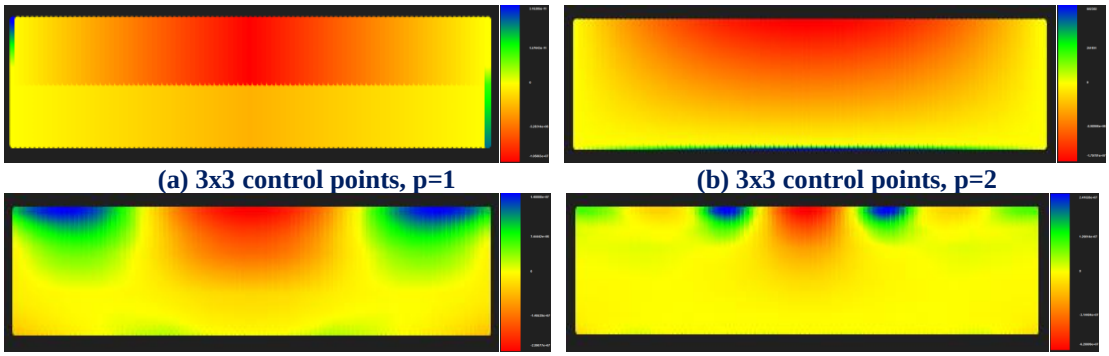
In Figure 5.42, 5.43, 5.44, 5.45 contours for Stress X, Y, XY and von Mises Undeformed are presented.



(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=1 **(d) 9x9 control points, p=2**

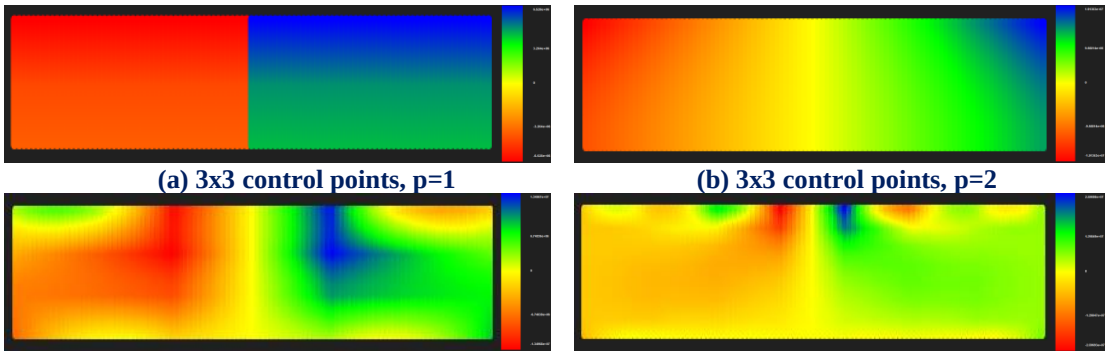
Figure 5.42. Stress X



(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

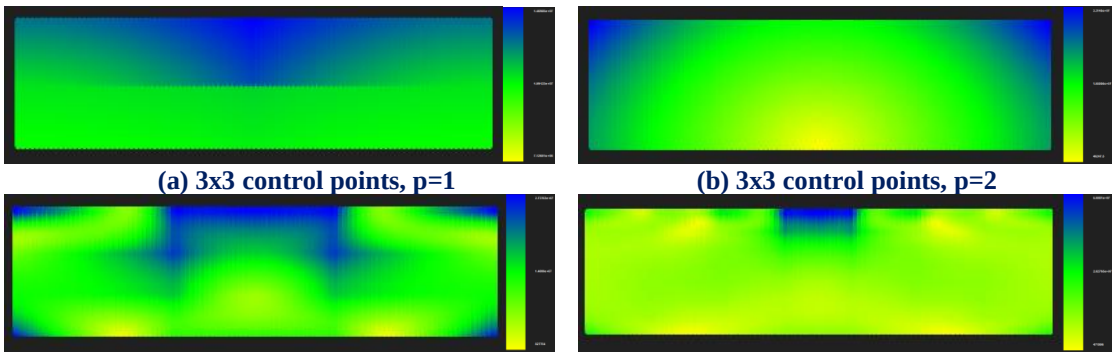
Figure 5.43. Stress Y



(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

Figure 5.44. Stress XY



(a) 3x3 control points, p=1 **(b) 3x3 control points, p=2**

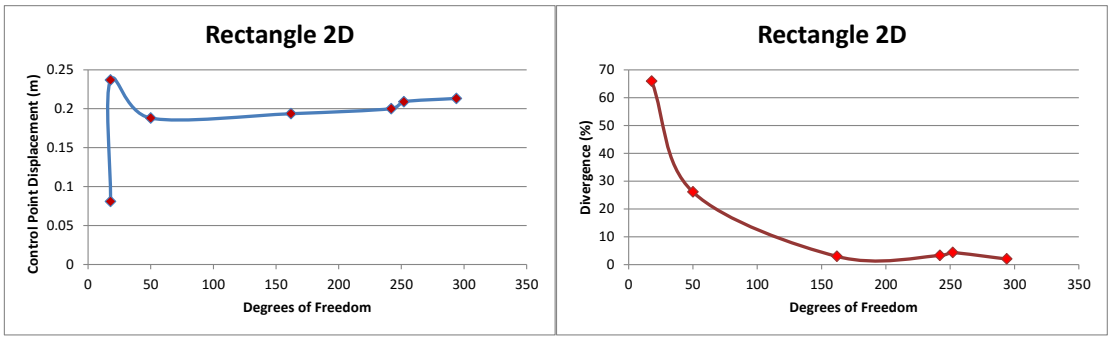
(c) 5x5 control points, p=2 **(d) 9x9 control points, p=2**

Figure 5.45. Stress von Mises

Applying several h-Refinements we get the following results.

Control Point Displacement (m):

- (3x3 CP) → Degrees of Freedom (18) → 0.080823 m
- (3x3 CP) → Degrees of Freedom (18) → 0.236925 m → Deviation (65.89%)
- (5x5 CP) → Degrees of Freedom (50) → 0.187931 m → Deviation (26.07%)
- (9x9 CP) → Degrees of Freedom (162) → 0.19359 m → Deviation (2.92%)
- (11x11 CP)→Degrees of Freedom (242) → 0.200053 m → Deviation (3.23%)
- (6x21 CP) → Degrees of Freedom (252) → 0.20902 m → Deviation (4.29%)
- (7x21 CP) → Degrees of Freedom (294) → 0.213239 m → Deviation (1.98%)

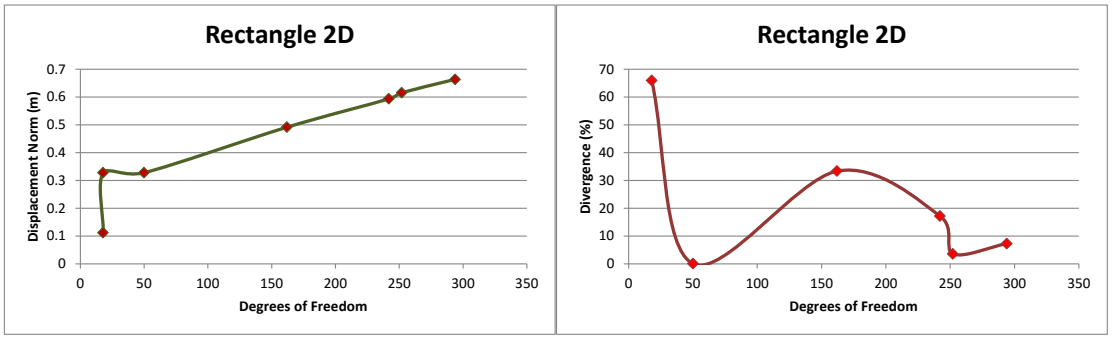


(a) **(b)**

Figure 5.46 Control point displacement results.
(a) Control point displacement in comparison with total degrees of freedom.
(b) Displacement deviation from previous solution in comparison with total degrees of freedom.

Displacement Norm (m):

- (3x3 CP) → Degrees of Freedom (18) → 0.111726 m
- (3x3 CP) → Degrees of Freedom (18) → 0.327728 m → Deviation (65.91%)
- (5x5 CP) → Degrees of Freedom (50) → 0.327558 m → Deviation (0.052%)
- (9x9 CP) → Degrees of Freedom (162) → 0.491299 m → Deviation (33.33%)
- (11x11 CP)→Degrees of Freedom (242)→ 0.593351 m→ Deviation (17.20%)
- (6x21 CP) → Degrees of Freedom (252) → 0.61535 m → Deviation (3.58%)
- (7x21 CP) → Degrees of Freedom (294) → 0.663504 m → Deviation (7.26%)

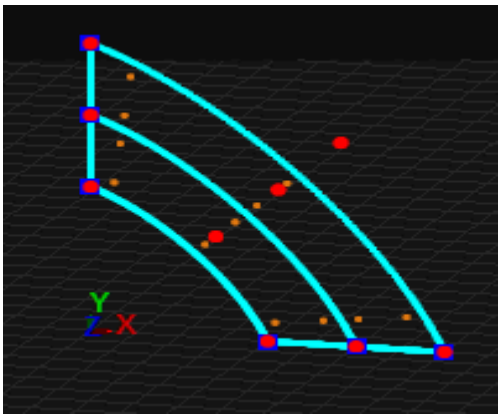


(a) **(b)**

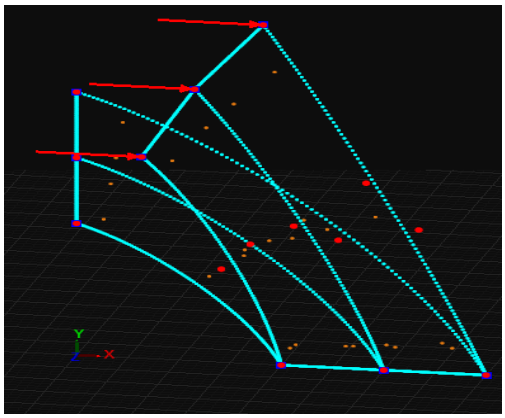
Figure 5.47 Displacement norm results.
(a) Displacement norm in comparison with total degrees of freedom.
(b) Displacement norm deviation from previous solution in comparison with total degrees of freedom.

5.5 Ring

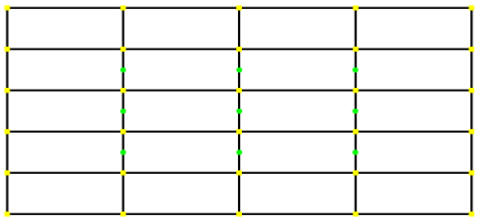
The next model is a quarter of a ring shape subjected under plane strain. The degrees of freedom of the bottom are fixed. Load of 20000 *kN* on each of the figure's points is applied. The three first representations consist of linear and quadratic basis functions.



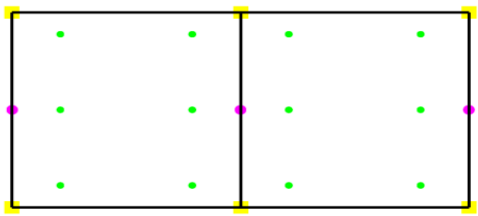
(a) Physical space, p=1 on ξ, p=2 on η



(b) Physical space deformed

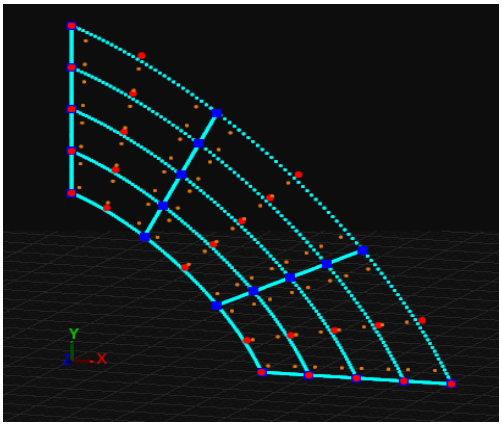


(c) Index space

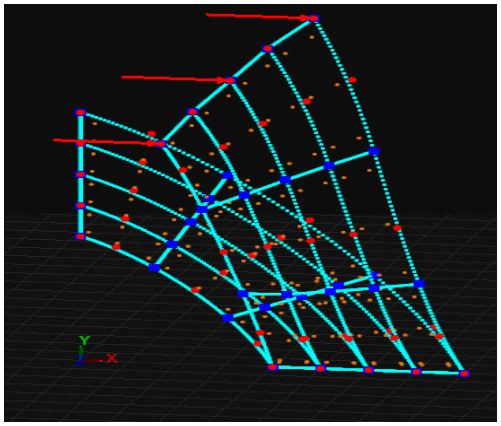


(d) Parameter space

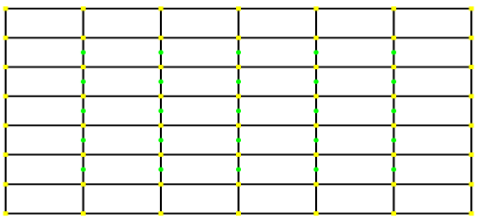
Figure 5.48. Mesh representation 3x3 control points



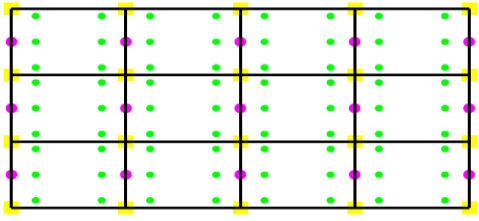
(a) Physical space, p=1 on ξ, p=2 on η



(b) Physical space deformed



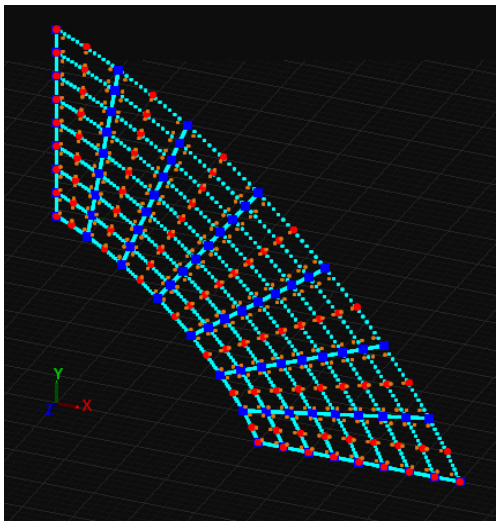
(c) Index space



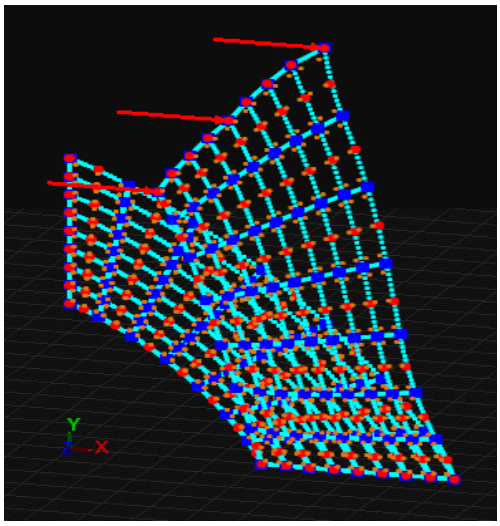
(d) Parameter space

Figure 5.49. Mesh representation 5x5 control points

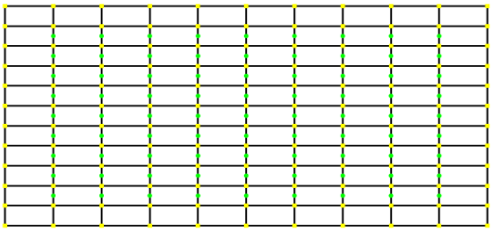
The forth representation consists of quadratic basis functions on axes ξ,η.



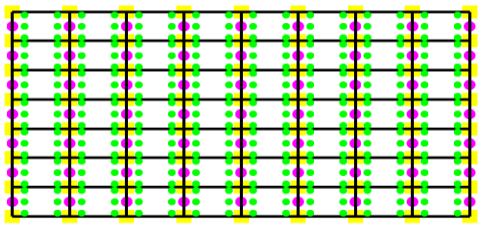
(a) Physical space, p=1 on ξ, p=2 on η



(b) Physical space deformed

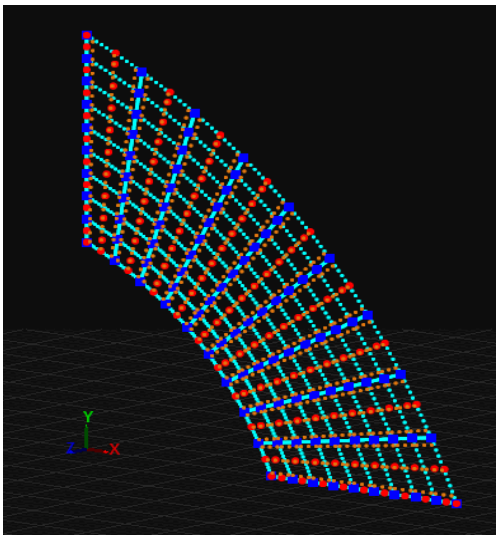


(c) Index space

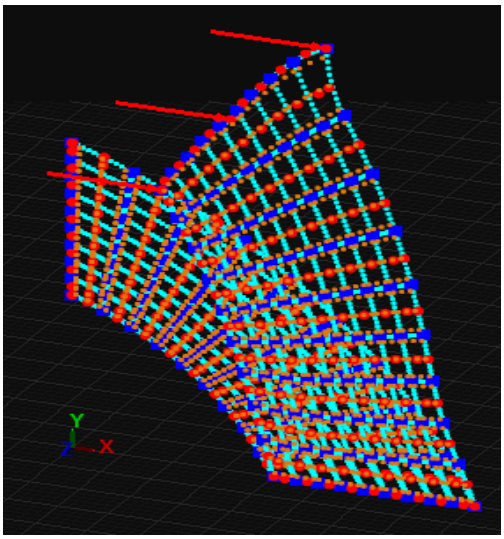


(d) Parameter space

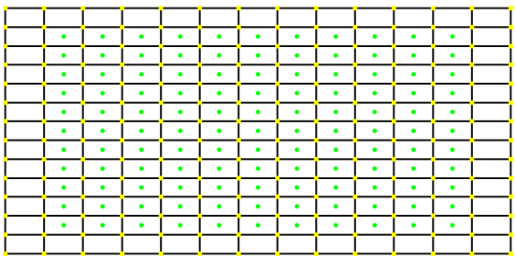
Figure 5.50. Mesh representation 9x9 control points



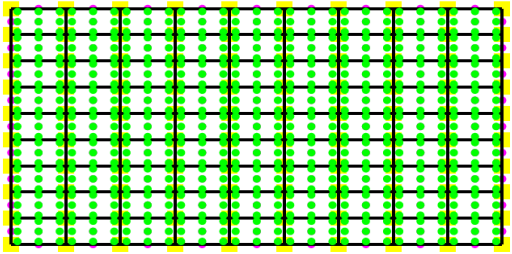
(a) Physical space, p=2 on ξ, p=2 on η



(b) Physical space deformed



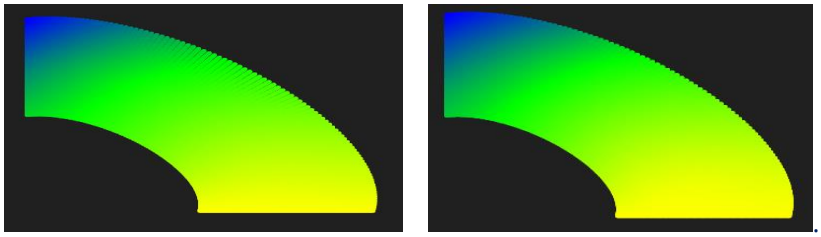
(c) Index space



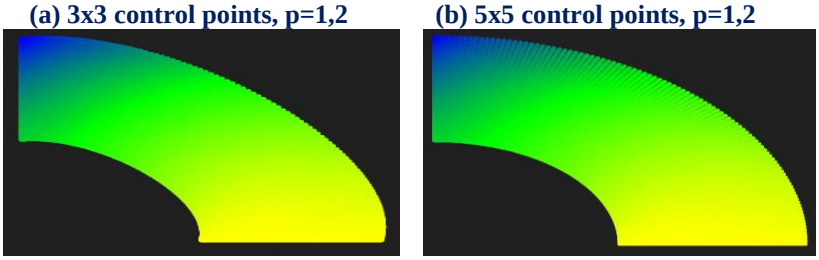
(d) Parameter space

Figure 5.51. Mesh representation 11x11 control points

In Figure 5.52, 5.53, 5.54 contours for Displacement, Displacement X and Y Undeformed are presented.

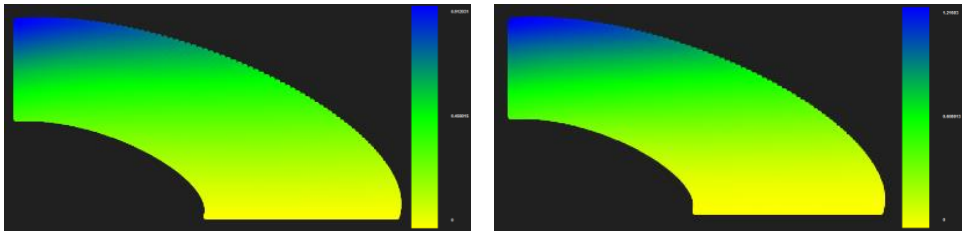


(a) 3x3 control points, p=1,2 **(b) 5x5 control points, p=1,2**

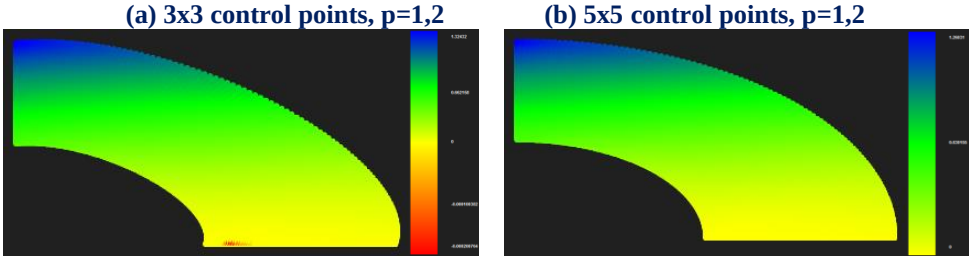


(c) 9x9 control points, p=1,2 **(d) 11x11 control points, p=2**

Figure 5.52. Displacement

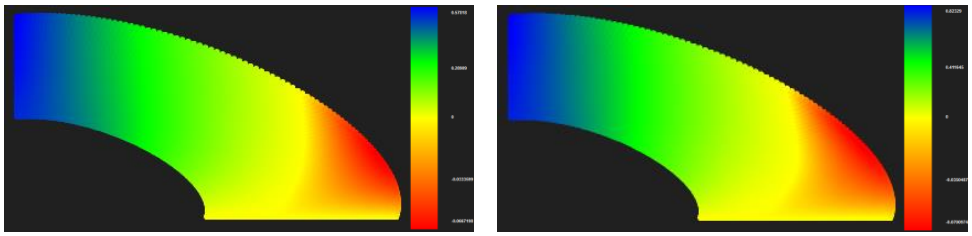


(a) 3x3 control points, p=1,2 **(b) 5x5 control points, p=1,2**

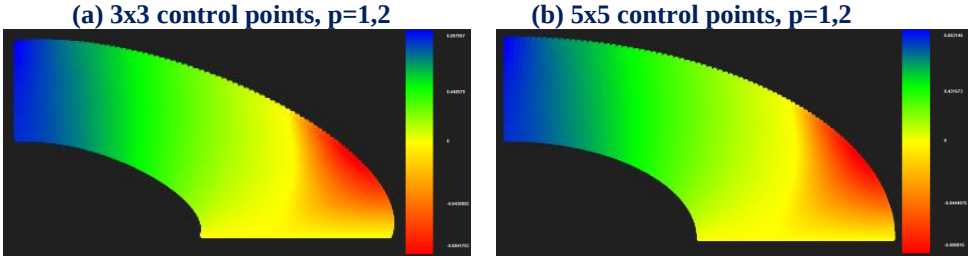


(c) 9x9 control points, p=1,2 **(d) 11x11 control points, p=2**

Figure 5.53. Displacement X.



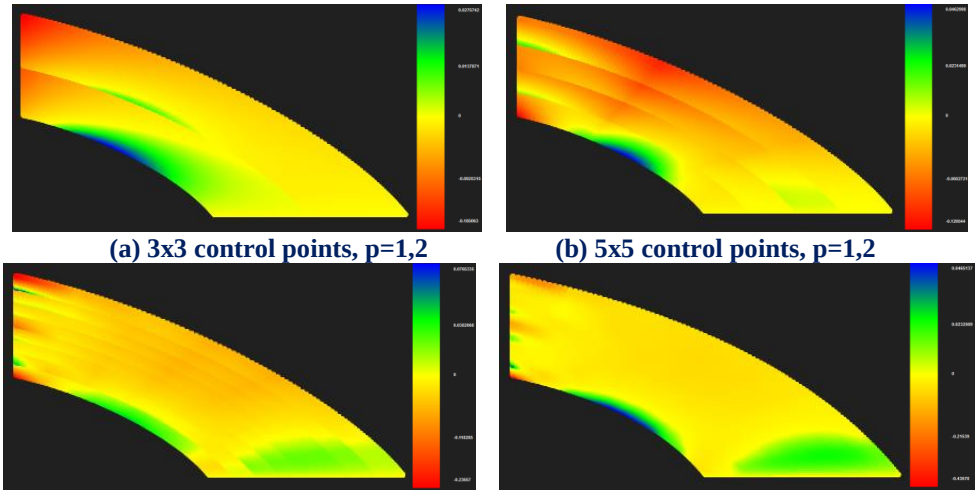
(a) 3x3 control points, p=1,2 **(b) 5x5 control points, p=1,2**



(c) 9x9 control points, p=1,2 **(d) 11x11 control points, p=2**

Figure 5.54. Displacement Y.

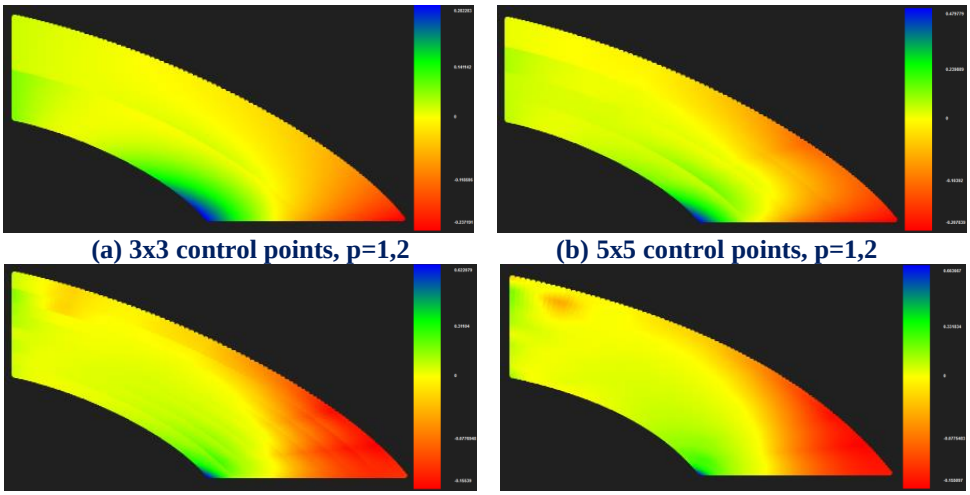
In Figures 5.55, 5.56, 5.57, contours for Strain X, Y and XY Undeformed are presented



(a) 3x3 control points, p=1,2 (b) 5x5 control points, p=1,2

(c) 9x9 control points, p=1,2 (d) 11x11 control points, p=2

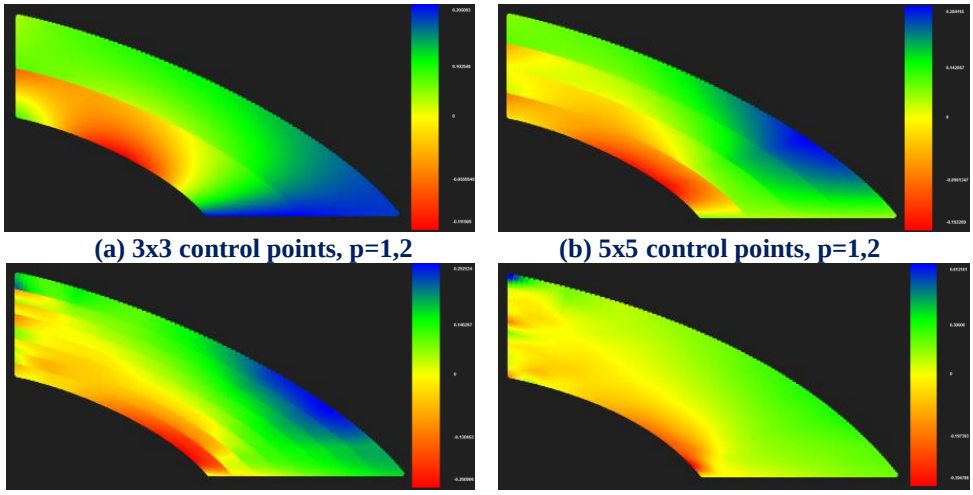
Figure 5.55. Strain X



(a) 3x3 control points, p=1,2 (b) 5x5 control points, p=1,2

(c) 9x9 control points, p=1,2 (d) 11x11 control points, p=2

Figure 5.56. Strain Y

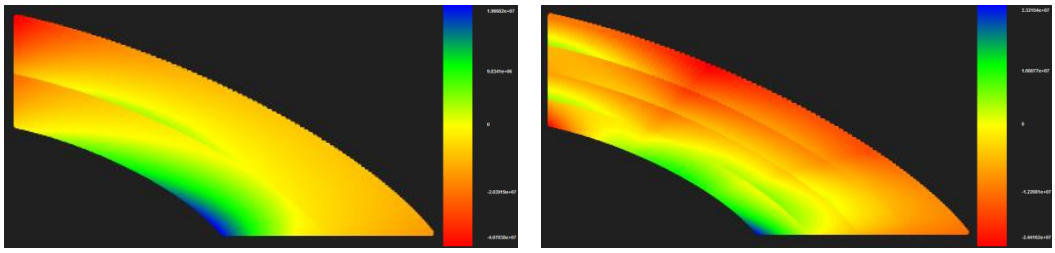


(a) 3x3 control points, p=1,2 (b) 5x5 control points, p=1,2

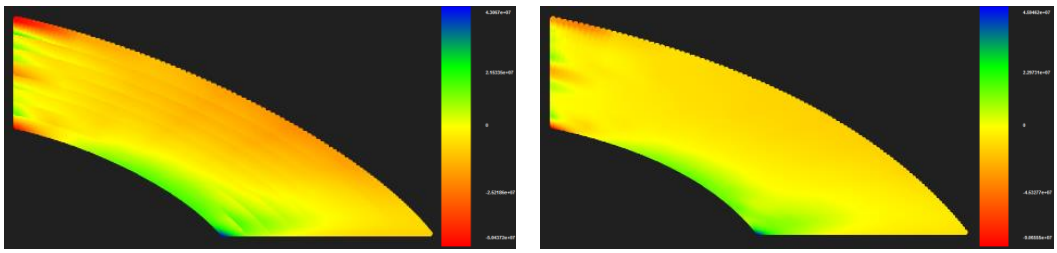
(c) 9x9 control points, p=1,2 (d) 11x11 control points, p=2

Figure 5.57.Strain XY

In Figures 5.58, 5.59 contours for Stress X and Y Undeformed are presented.



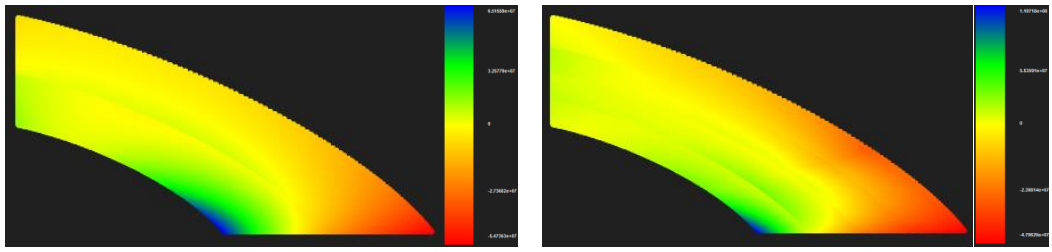
(a) 3x3 control points, p=1,2 **(b) 5x5 control points, p=1,2**



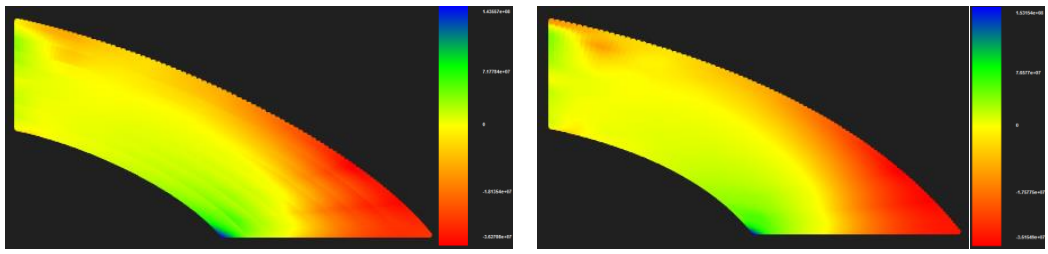
(c) 9x9 control points, p=1,2 **(d) 11x11 control points, p=2**

Figure 5.58. Stress X

Analysis has been conducted with shape functions of C^1 and C^0 continuity. What we can mainly observe in the results not only on Strain but also on Stress contours is the fact that the discontinuity of the strain or the stress field is remarkably obvious for the C^0 representation.



(a) 3x3 control points, p=1,2 **(b) 5x5 control points, p=1,2**

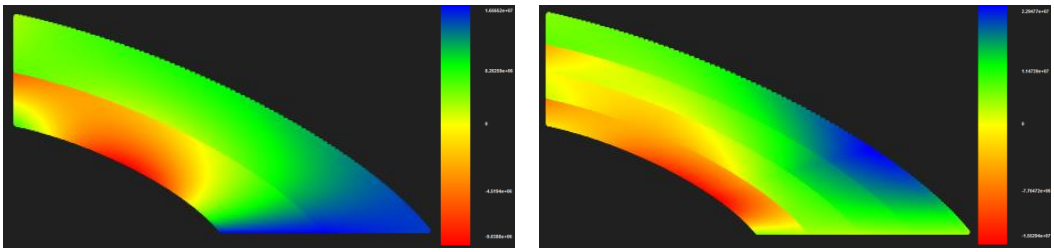


(c) 9x9 control points, p=1,2 **(d) 11x11 control points, p=2**

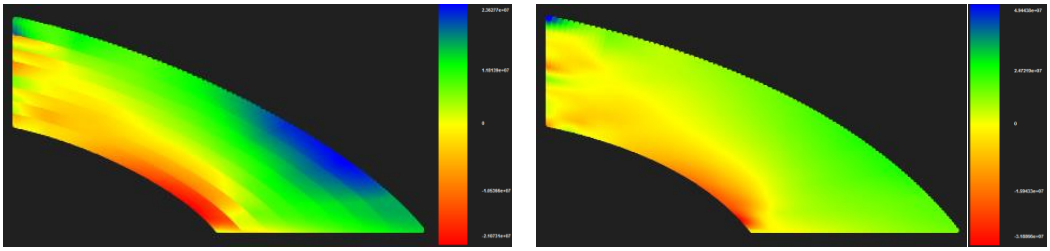
Figure 5.59. Stress Y

Another thing we can verify is that despite the discontinuity that is observed in the results, the greater the parameterization becomes, the more the contour approaches the final solution. We can see the similarities in figure 5.59 (c) and (d) even though the discontinuity is steel obvious.

In Figures 5.60, 5.61 contours for Stress XY and von Mises Undeformed are presented.

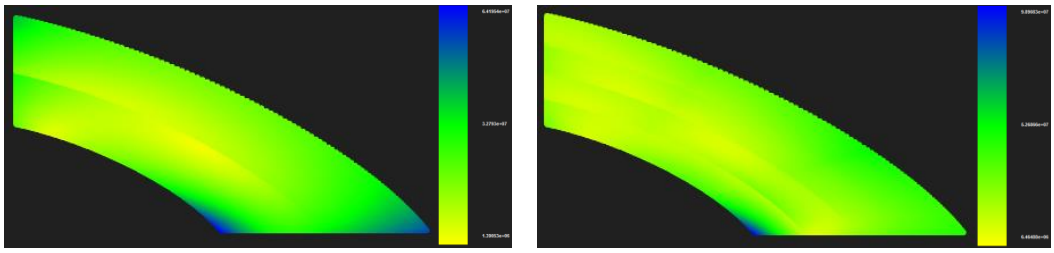


(a) 3x3 control points, p=1,2 **(b) 5x5 control points, p=1,2**

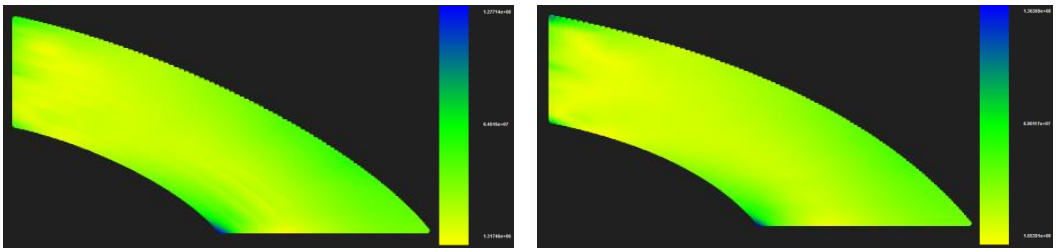


(c) 9x9 control points, p=1,2 **(d) 11x11 control points, p=2**

Figure 5.60 Stress XY.



(a) 3x3 control points, p=1,2 **(b) 5x5 control points, p=1,2**



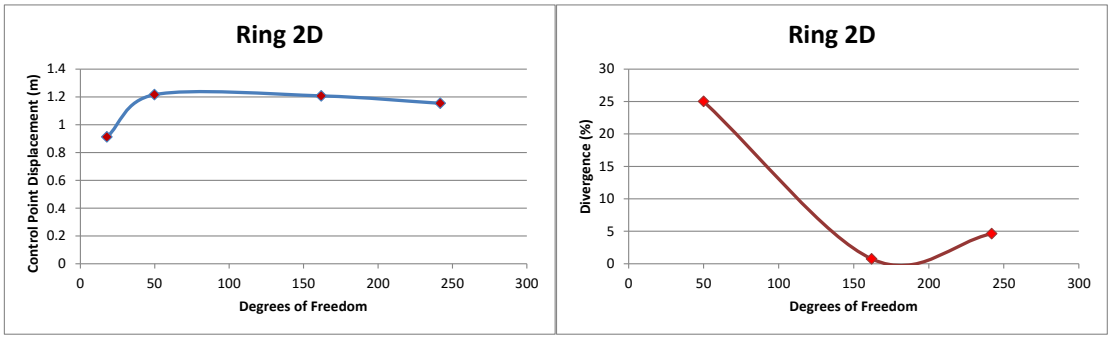
(c) 9x9 control points, p=1,2 **(d) 11x11 control points, p=2**

Figure 5.61. Stress von Mises.

Applying several h-Refinements we get the following results.

Control Point Displacement (m):

- (3x3 CP) → Degrees of Freedom (18) → 0.912031 m
- (5x5 CP) → Degrees of Freedom (50) → 1.21603 m → Deviation (25.00%)
- (9x9 CP) → Degrees of Freedom (162) → 1.20708 m → Deviation (0.74%)
- (11x11 CP) → Degrees of Freedom (242) → 1.15369 m → Deviation (4.63%)

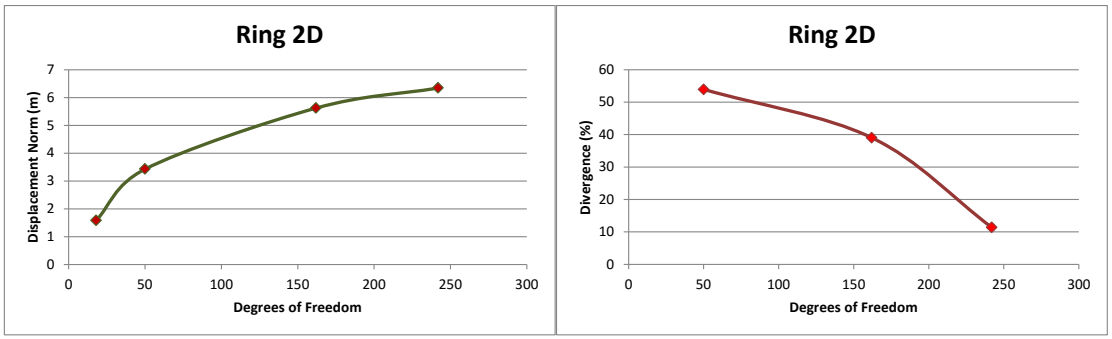


(a) **(b)**

Figure 5.62 Control point displacement results.
(a) Control point displacement in comparison with total degrees of freedom.
(b) Displacement deviation from previous solution in comparison with total degrees of freedom.

Displacement Norm (m):

- (3x3 CP) → Degrees of Freedom (18) → 1.57843 m
- (5x5 CP) → Degrees of Freedom (50) → 3.42699 m → Deviation (53.94%)
- (9x9 CP) → Degrees of Freedom (162) → 5.62157 m → Deviation (39.04%)
- (11x11 CP)→ Degrees of Freedom (242)→ 6.34256 m → Deviation (11.37%)



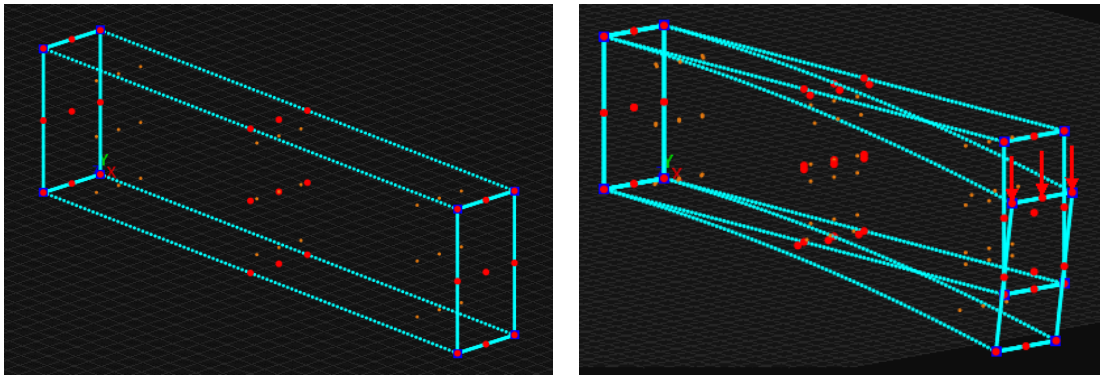
(a) **(b)**

Figure 5.63 Displacement norm results.
(a) Control point displacement in comparison with total degrees of freedom.
(b) Displacement norm deviation from previous solution in comparison with total degrees of freedom.

5.6 Cantilever 3D

The first 3D representation examined is a simple cantilever, with a rectangle cross-section. The initial mesh is similar to the 2D Cantilever. A parametric direction ζ with one knot span for the first case is added. The Cantilever is subjected to bending and torsion loads. Suitable Refinements are applied for each case.

The degrees of freedom on the left are fixed. As far as bending concerned three concentrated loads of 250000 *kN* are applied on the right side of the cantilever.



(a) **(b)**

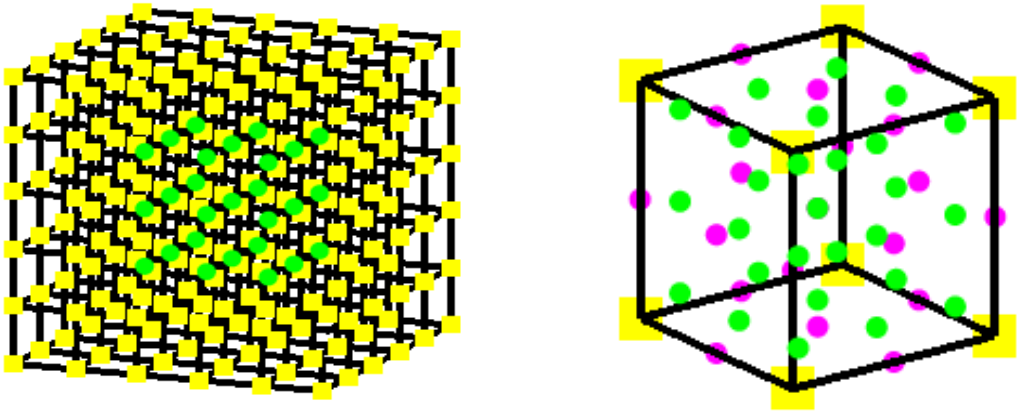
Figure 5.64. Rectangle 3D 3x3x3 control points.
(a) Physical space and (b) Physical space deformed for rectangle 8x2x1 m

Knot value vector: $\Xi = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$

Knot value vector $\mathrm{H} = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$

Knot value vector $Z = \{0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1\}$

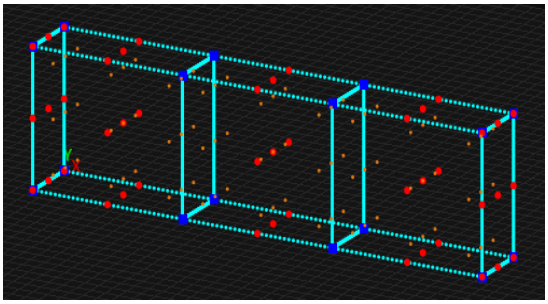
The first representation consists of quadratic basis functions on axes ξ, η, ζ. There is formed 1 knot Rectangle (Isogeometric elements). Knot boundaries are displayed in blue.



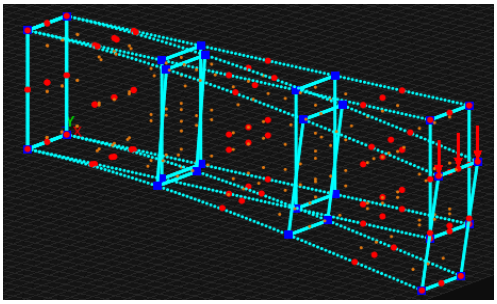
(a) **(b)**

Figure 5.65. (a) Index space and (b) Parameter space

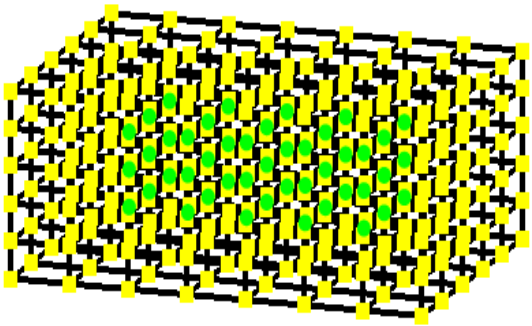
The second representation consists of quadratic basis functions on axes ξ,η,ζ. There are $n=5$ control points on ξ, $m=3$ control points on η and $r=3$ control points on ζ for a total of 45 points and 135 degrees of freedom.



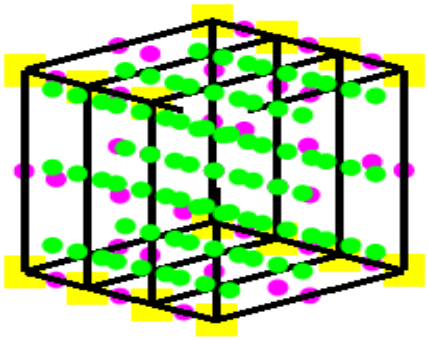
(a) Physical space 5x3x3 control points, p=2



(b) Physical space deformed, p=2



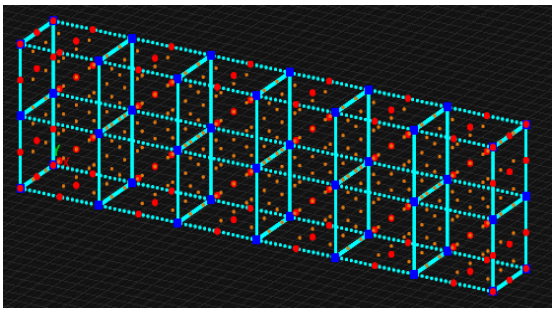
(c) Index space, p=2



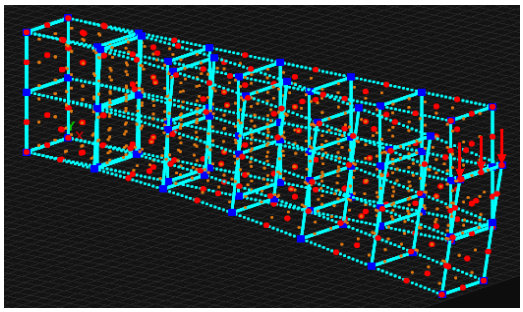
(d) Parameter space, p=2

Figure 5.66. Mesh representation of 5x3x3 control points rectangle.

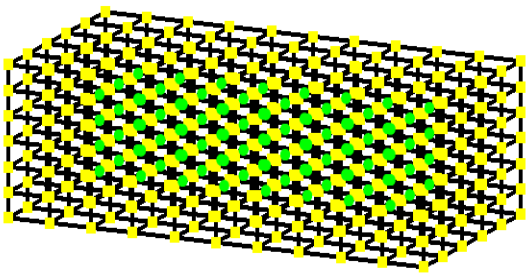
The third representation consists of quadratic basis functions on axes ξ,η,ζ. There are $n=8$ control points on ξ, $m=4$ control points on η and $r=3$ control points on ζ for a total of 96 points and 288 degrees of freedom.



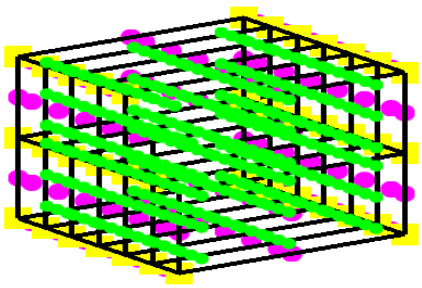
(a) Physical space 8x4x3 control points, p=2



(b) Physical space deformed, p=2



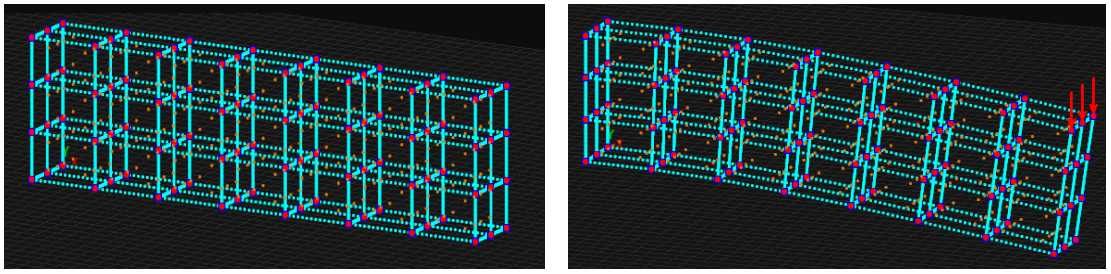
(c) Index space



(d) Parameter space

Figure 5.67. Mesh representation of 8x4x3 control points rectangle.

The forth representation consists of linear basis functions on axes ξ,η,ζ. There are $n=8$ control points on ξ, $m=4$ control points on η and $r=3$ control points on ζ for a total of 96 points and 288 degrees of freedom.



(a) Physical space 8x4x3 control points, p=1 **(b) Physical space deformed, p=1**

Figure 5.68. Mesh representation of 8x4x3 control points rectangle.

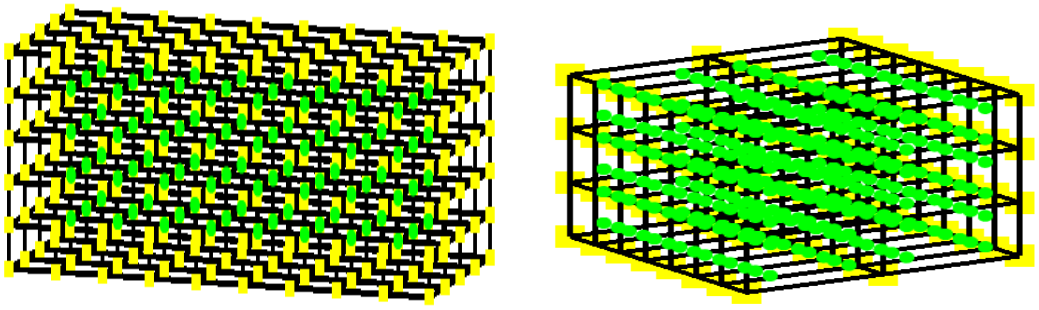
Knot value vector:

$$\Xi=\{0\quad 0\quad 0.142857\quad 0.285714\quad 0.428571\quad 0.571429\quad 0.714286\quad 0.857143\quad 1\quad 1\}$$

Knot value vector $H=\{0\quad 0\quad 0.33333\quad 0.66667\quad 1\quad 1\}$

Knot value vector $Z=\{0\quad 0\quad 0.5\quad 1\quad 1\}$

Gauss point coordinates are evaluated for every knot span. In this 3D problem, $p+1=3$ gauss points for quadratic and p+1=2 gauss points for linear basis functions per knot span in each direction are required. Thus, $3\cdot3\cdot3=27$ gauss points per Isogeometric Element for the first three cases and $2\cdot2\cdot2=8$ for the forth.

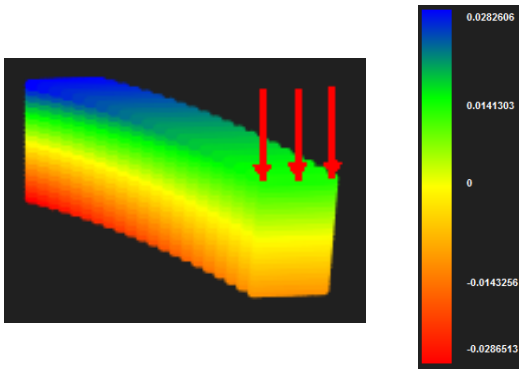


(c) Index space, p=1 **(d) Parameter space, p=1**

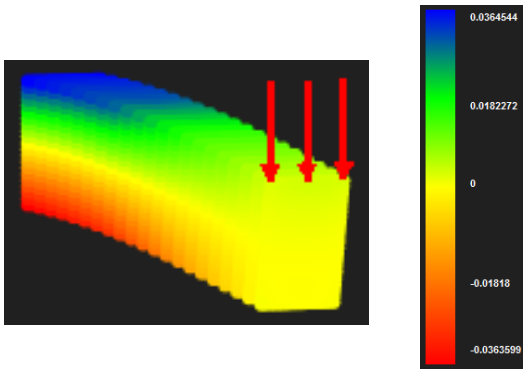
Figure 5.69. Mesh representation of 8x4x3 control points rectangle.

The second case concludes 3 knot Rectangles (Figure 5.56), the third 12 (Figure 5.57)and in the last 42 Isogeometric elements are formed (Figure 5.58). Despite the fact that we use 42 elements in the last case the influence of the linear basis functions give much worse results as far as the flow of strain and stress field across the model concerned. We can see the difference between quadratic and linear basis functions on the following figures where the discontinuity is quite obvious.

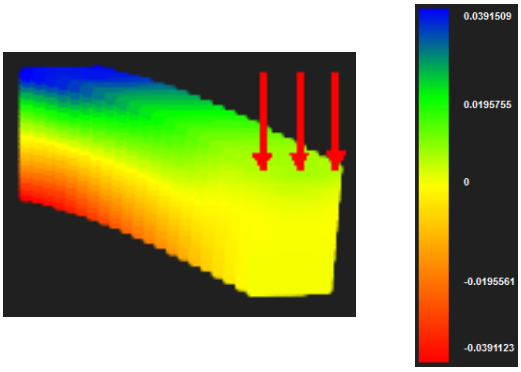
In Figures 5.70, 5.71, contours for Strain X and XY Deformed are presented.



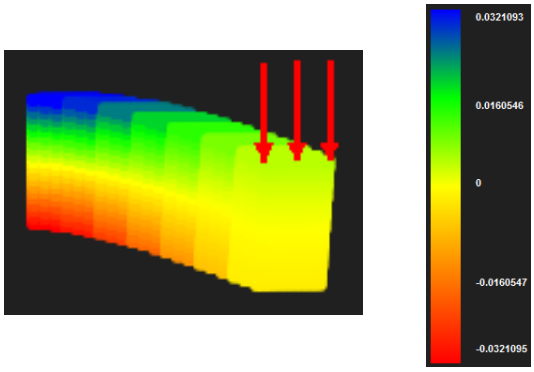
(a) 3x3x3 control points, p=2



(b) 5x3x3 control points, p=2

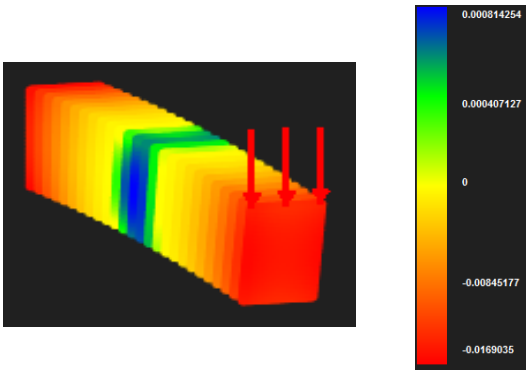


(c) 8x4x3 control points, p=2

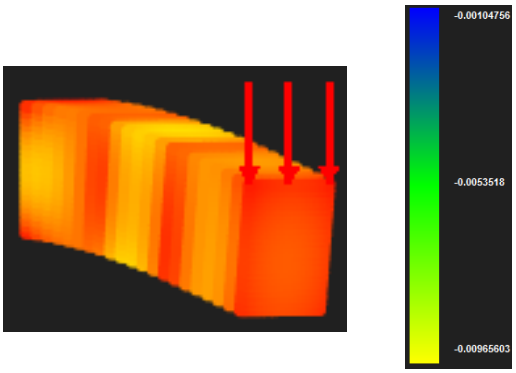


(d) 8x4x3 control points, p=1

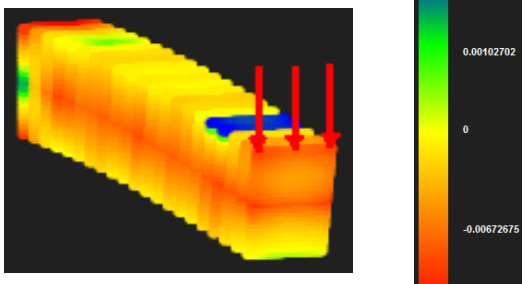
Figure 5.70. Strain X



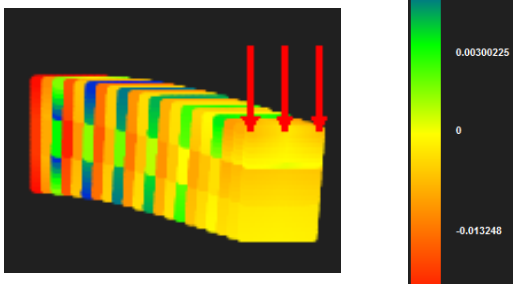
(a) 3x3x3 control points, p=2



(b) 5x3x3 control points, p=2



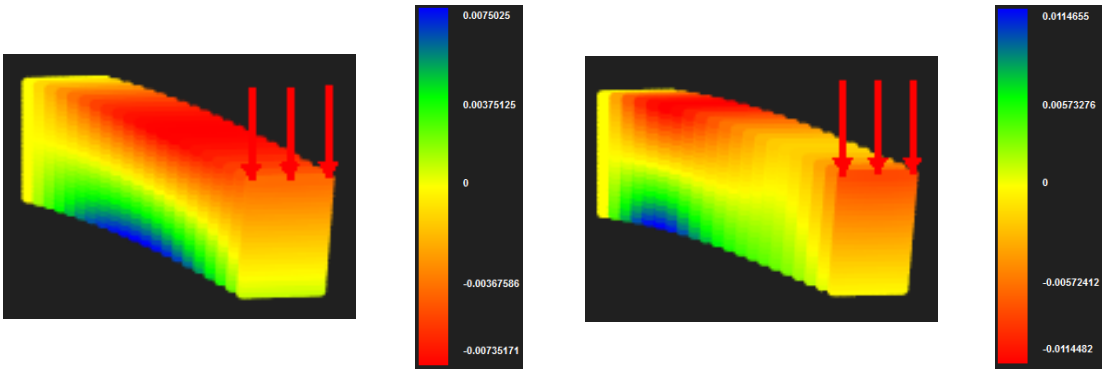
(c) 8x4x3 control points, p=2



(d) 8x4x3 control points, p=1

Figure 5.71. Strain XY

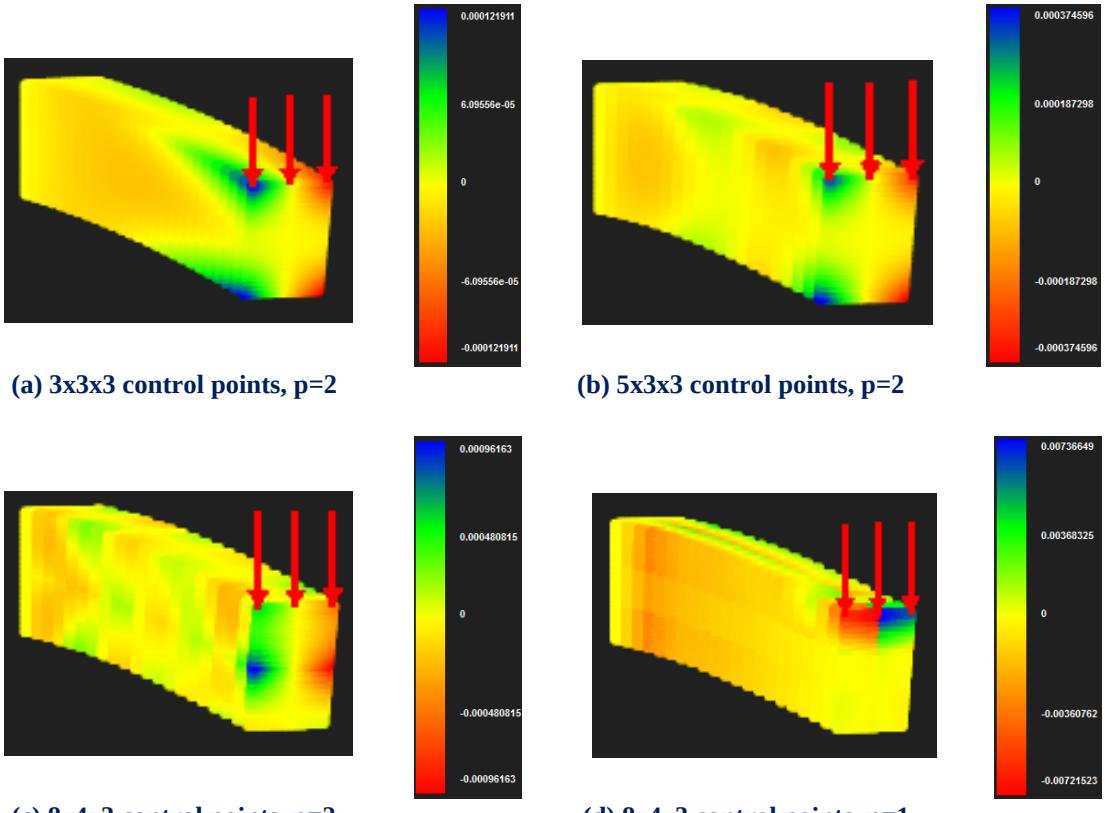
In Figures 5.72, 5.73, contours for Strain Y and YZ Deformed are presented.



(a) 3x3x3 control points, p=2 **(b) 5x3x3 control points, p=2**

(c) 8x4x3 control points, p=2 **(d) 8x4x3 control points, p=1**

Figure 5.72. Strain Y

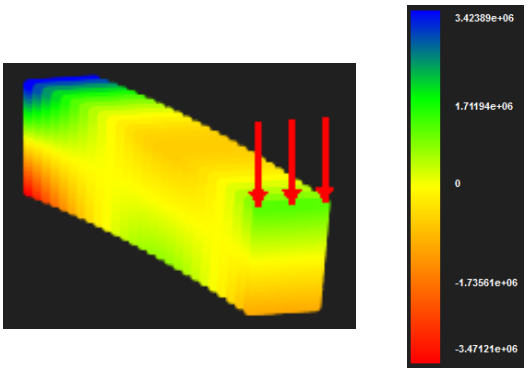


(a) 3x3x3 control points, p=2 **(b) 5x3x3 control points, p=2**

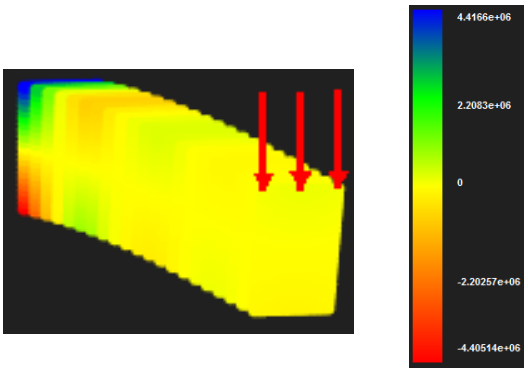
(c) 8x4x3 control points, p=2 **(d) 8x4x3 control points, p=1**

Figure 5.73. Strain YZ

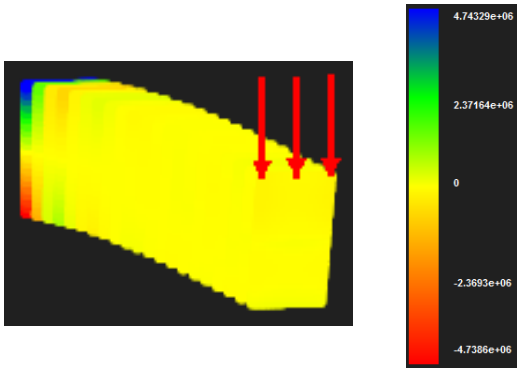
In Figure 5.74, 5.75, contours for Strain Z and ZX Deformed are presented.



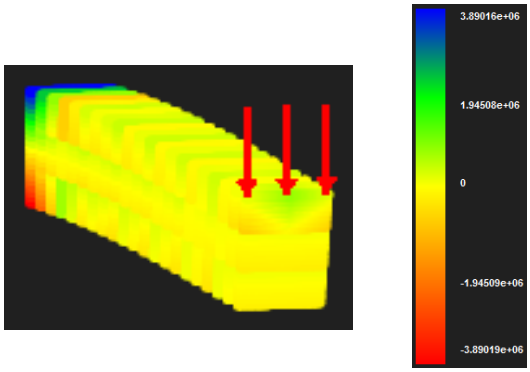
(a) 3x3x3 control points, p=2



(b) 5x3x3 control points, p=2

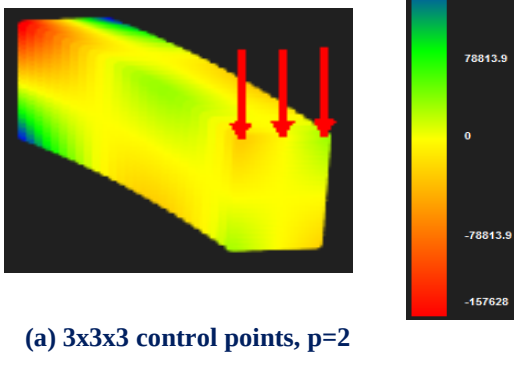


(c) 8x4x3 control points, p=2

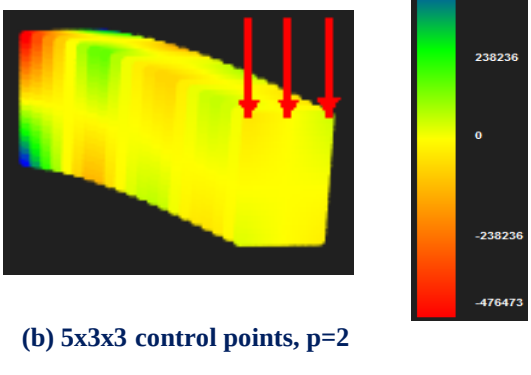


(d) 8x4x3 control points, p=1

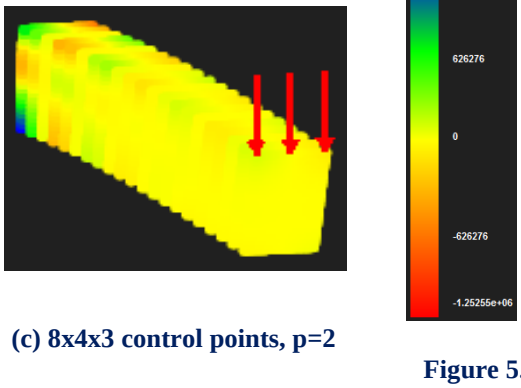
Figure 5.74. Strain Z



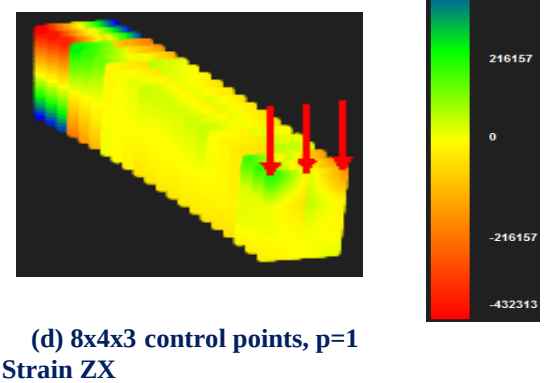
(a) 3x3x3 control points, p=2



(b) 5x3x3 control points, p=2



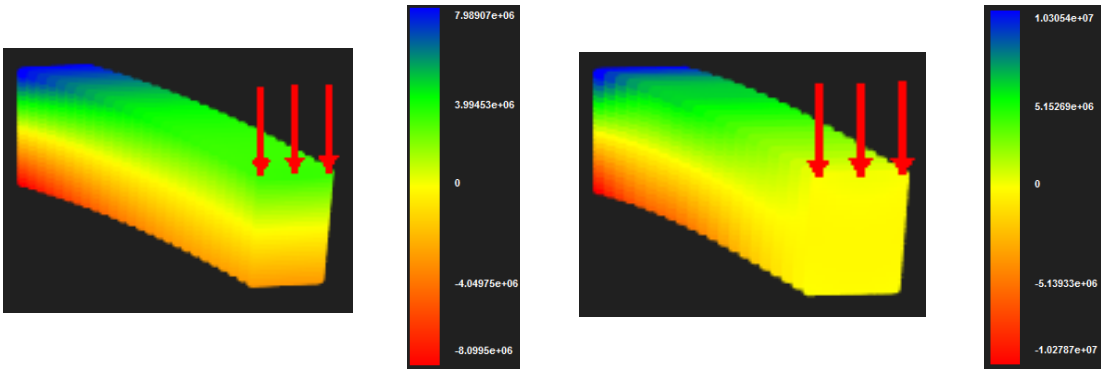
(c) 8x4x3 control points, p=2



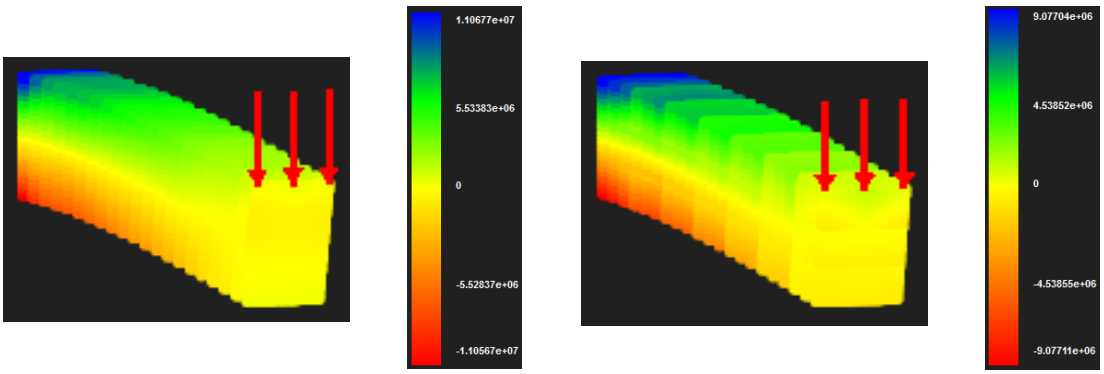
(d) 8x4x3 control points, p=1

Figure 5.75. Strain ZX

In Figure 5.76, 5.77, contours for Stress X and XY Deformed are presented.

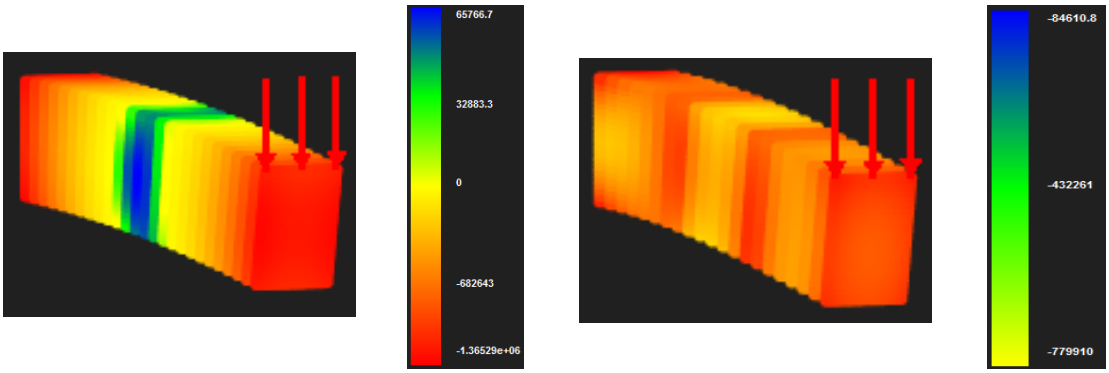


(a) 3x3x3 control points, p=2 **(b) 5x3x3 control points, p=2**

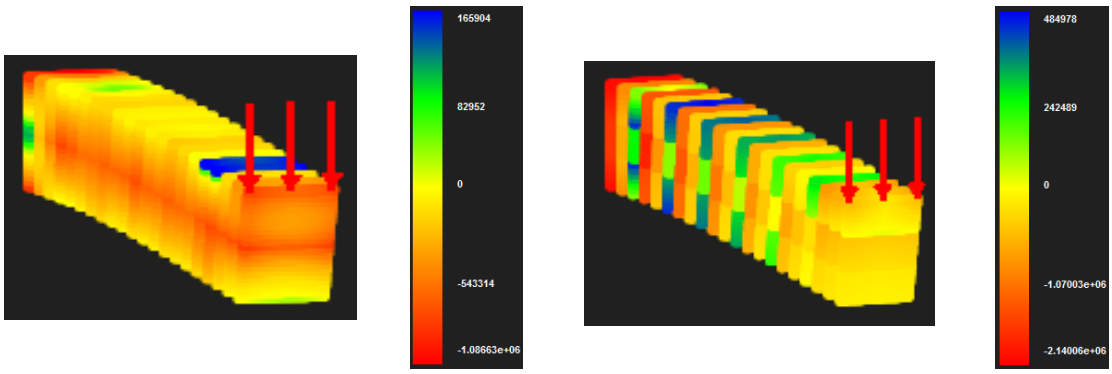


(c) 8x4x3 control points, p=2 **(d) 8x4x3 control points, p=1**

Figure 5.76. Stress X



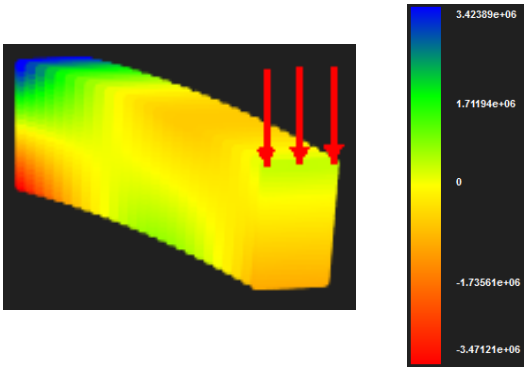
(a) 3x3x3 control points, p=2 **(b) 5x3x3 control points, p=2**



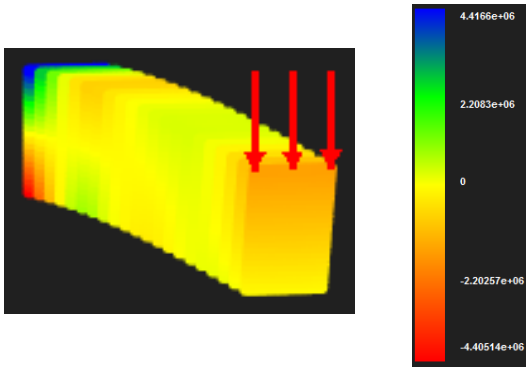
(c) 8x4x3 control points, p=2 **(d) 8x4x3 control points, p=1**

Figure 5.77. Stress XY

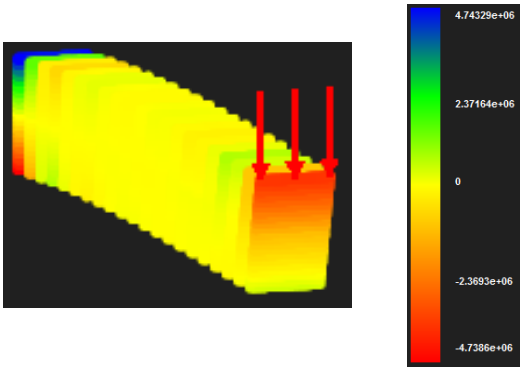
In Figure 5.78, 5.79, contours for Stress Y and YZ Deformed are presented.



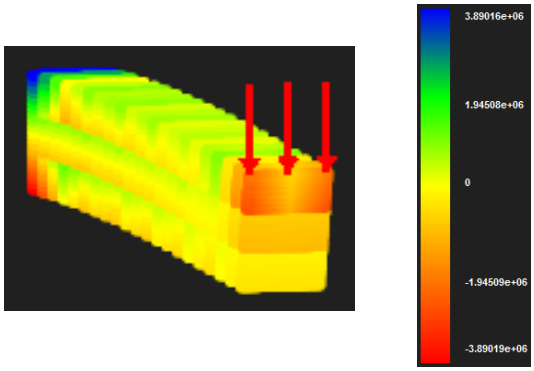
(a) 3x3x3 control points, p=2



(b) 5x3x3 control points, p=2

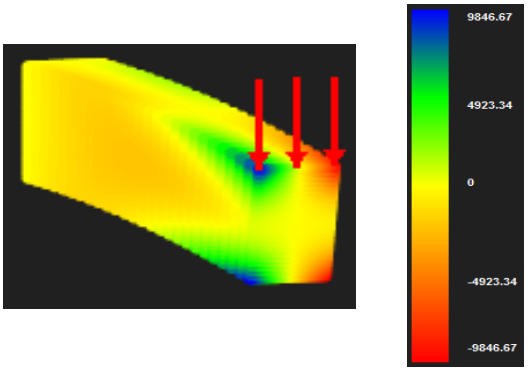


(c) 8x4x3 control points, p=2

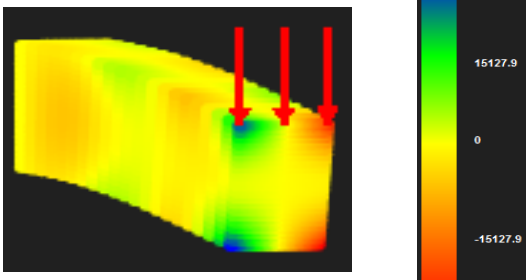


(d) 8x4x3 control points, p=1

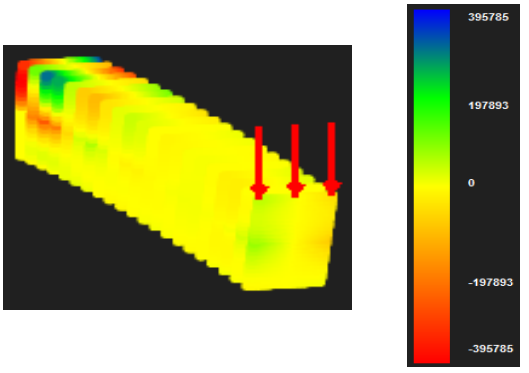
Figure 5.78. Stress Y



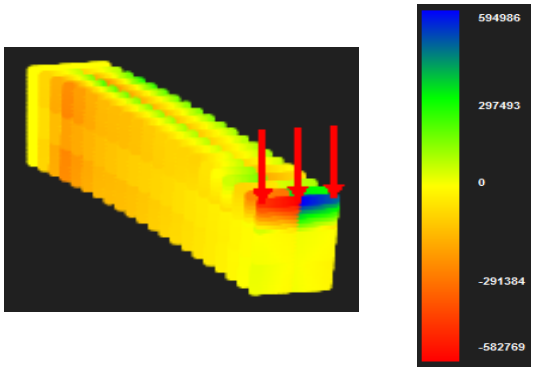
(a) 3x3x3 control points, p=2



(b) 5x3x3 control points, p=2



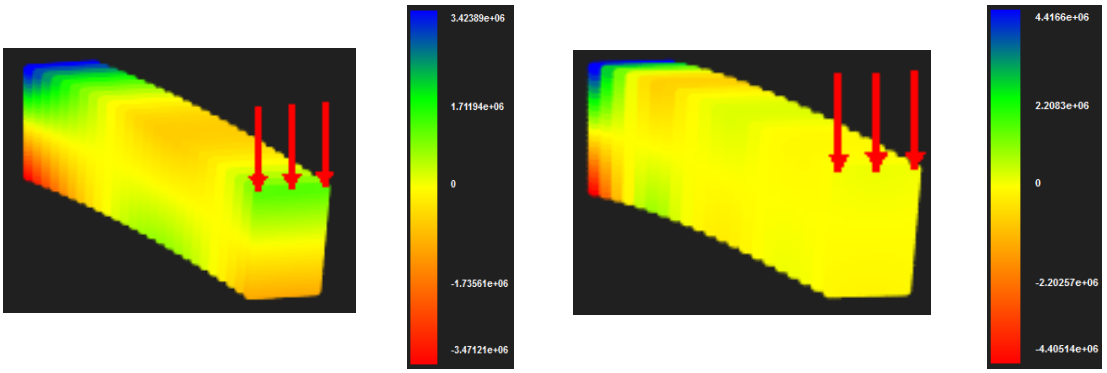
(c) 8x4x3 control points, p=2



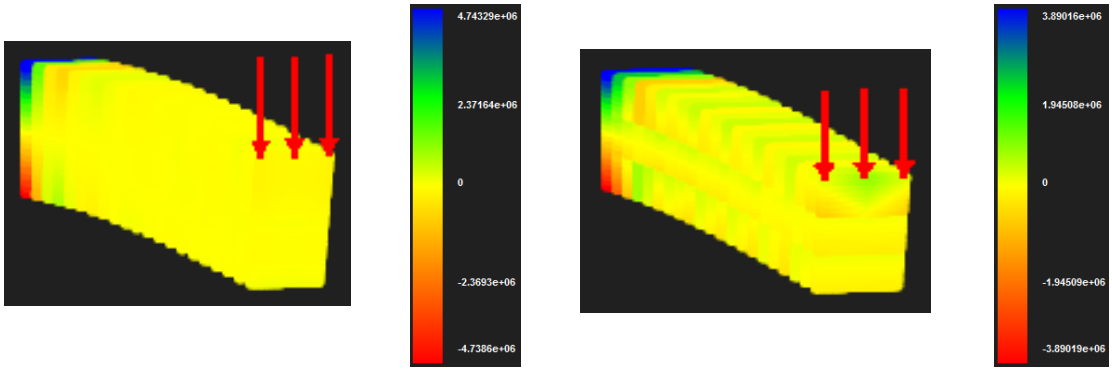
(d) 8x4x3 control points, p=1

Figure 5.79. Stress YZ

In Figure 5.80, 5.81, contour for Stress Z, ZX Deformed are represented.

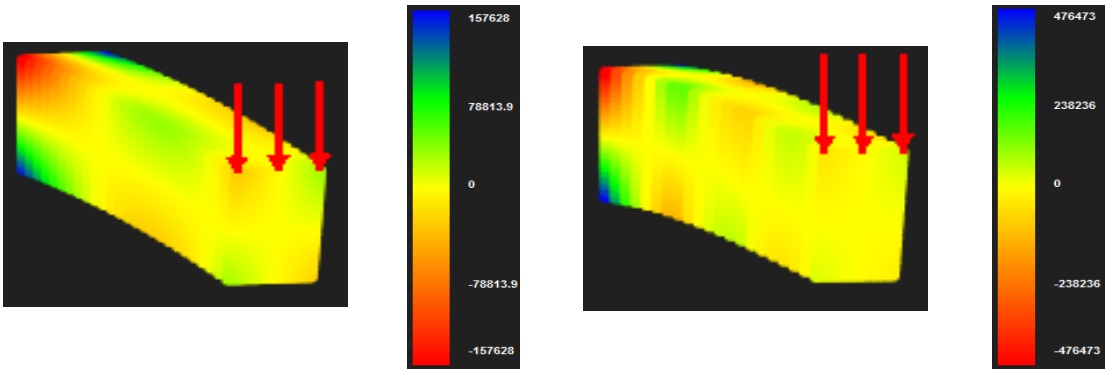


(a) 3x3x3 control points, p=2 **(b) 5x3x3 control points, p=2**

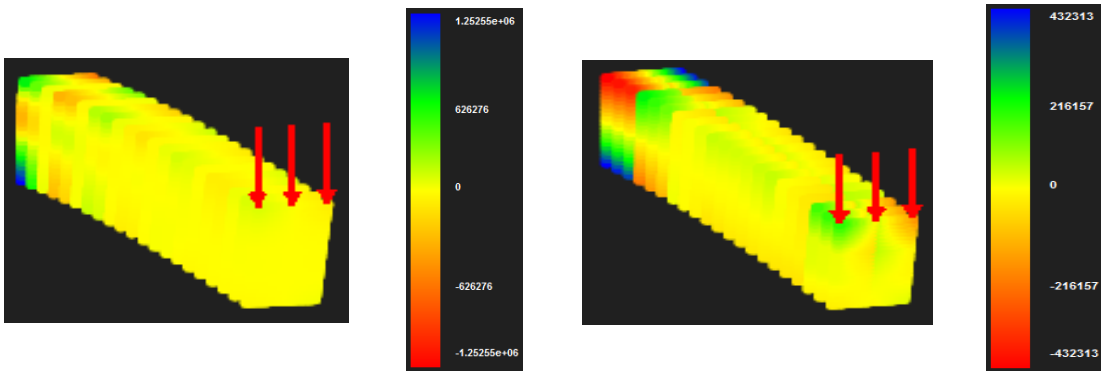


(c) 8x4x3 control points, p=2 **(d) 8x4x3 control points, p=1**

Figure 5.80. Stress Z



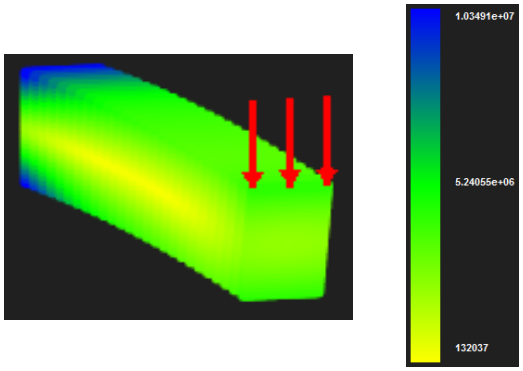
(a) 3x3x3 control points, p=2 **(b) 5x3x3 control points, p=2**



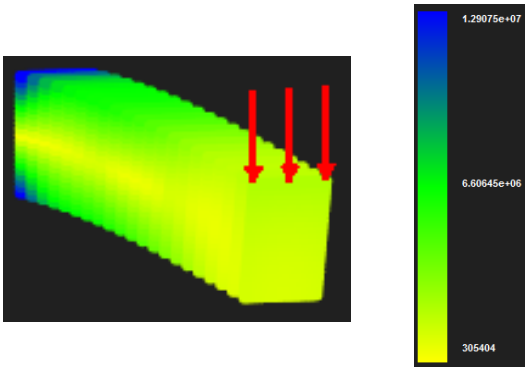
(c) 8x4x3 control points, p=2 **(d) 8x4x3 control points, p=1**

Figure 5.81. Stress ZX.

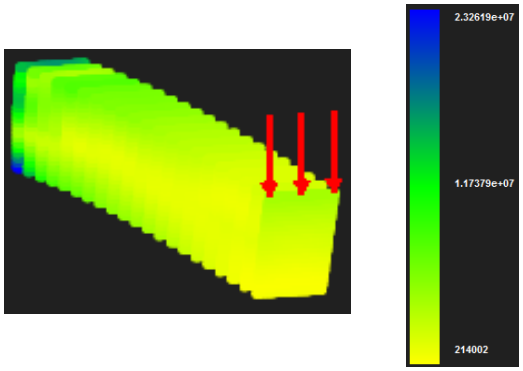
In Figure 5.82, contour for Stress von Mises Deformed are represented.



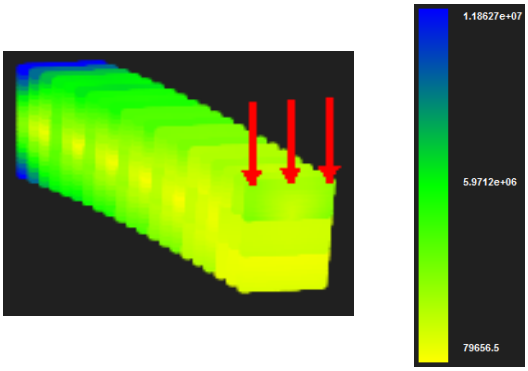
(a) 3x3x3 control points, p=2



(b) 5x3x3 control points, p=2



(c) 8x4x3 control points, p=2



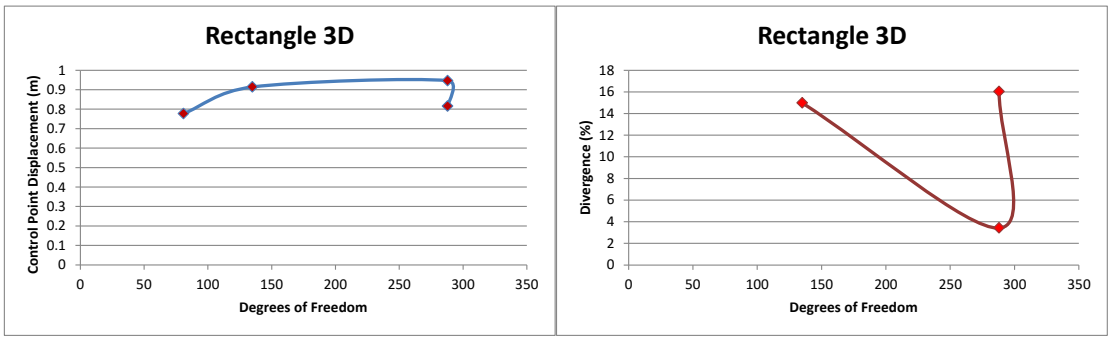
(d) 8x4x3 control points, p=1

Figure 5.82. Stress von Mises

Applying several h, k-Refinements we get the following results.

Control point Displacement (m):

- (3x3x3 CP) → Degrees of Freedom (81) → 0.777258 m
- (5x3x3 CP)→Degrees of Freedom (135)→ 0.914191 m→ Deviation (14.98%)
- (8x4x3 CP)→ Degrees of Freedom (288)→ 0.946528 m → Deviation (3.42%)
- (8x4x3 CP)→Degrees of Freedom (288)→ 0.815764 m→ Deviation (16.03%)



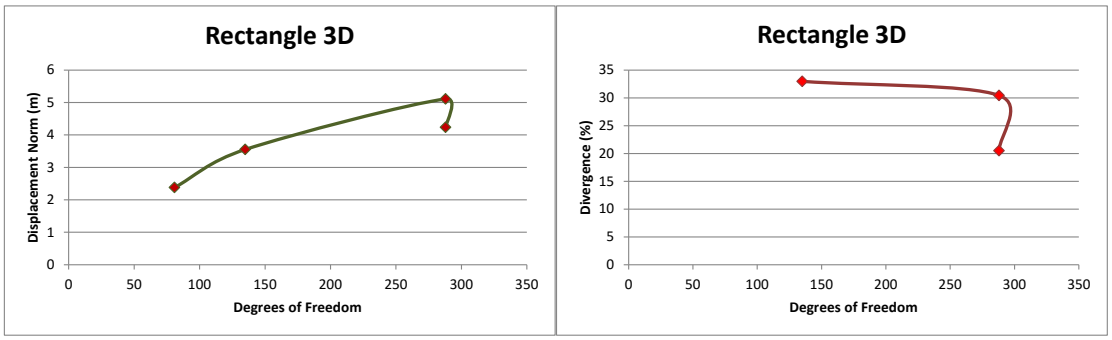
(a) **(b)**

Figure 5.83 Control point displacement results.
(a) Control point displacement in comparison with total degrees of freedom,
for bending. (b) Divergence from previous solution in comparison with total degrees of freedom.

In Figure 6.83 we can observe that for the same parameterization of the model (8x4x3 control points) with only difference the degree of the polynomial basis functions the deviation of the displacement is ejected to 16,03% from 3.4% that we had achieved with quadratic basis functions.

Displacement Norm (m):

- (3x3x3 CP) → Degrees of Freedom (81) → 2.38146 m
- (5x3x3 CP)→ Degrees of Freedom (135)→ 3.55288 m → Deviation (32.97%)
- (8x4x3 CP)→ Degrees of Freedom (288)→ 5.10711 m → Deviation (30.43%)
- (8x4x3 CP)→ Degrees of Freedom (288)→ 4.23834 m → Deviation (20.49%)



(a) **(b)**

Figure 5.84 Displacement results.
(a) Displacement norm in comparison with total degrees of freedom,
for bending. (b) Divergence from previous solution in comparison with total degrees of freedom.

It would be of interest to examine the case of bending where the load would be concentrated in the middle area of the model. Six loads of 800000 *kN* are applied while both sides are fixed.

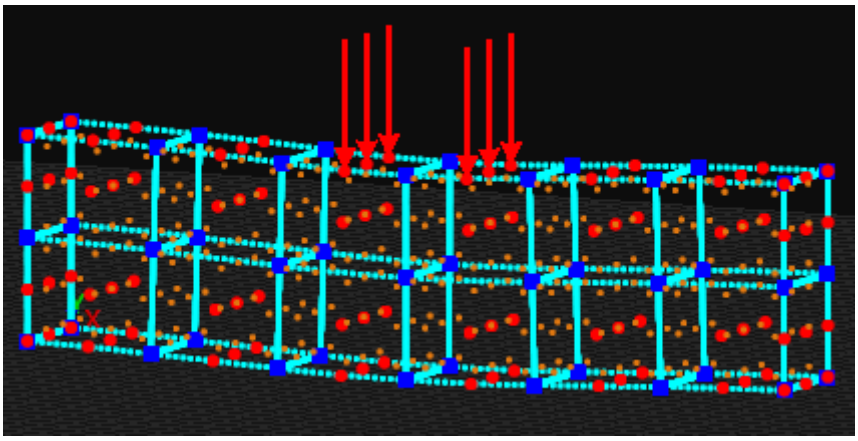
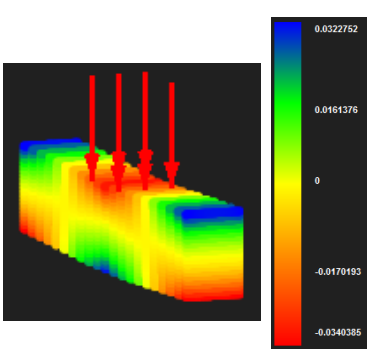
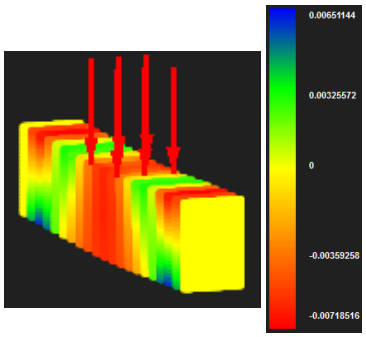


Figure 5.85. Stress ZX

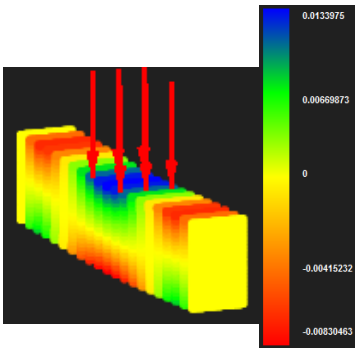
The mesh consists of quadratic basis functions on axes ξ,η,ζ. There are $n=8$ control points on ξ, $m=4$ control points on η and $r=3$ control points on ζ for a total of 96 points and 288 degrees of freedom.



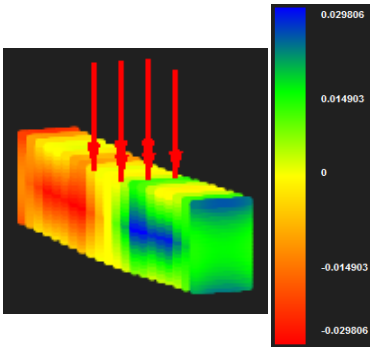
(a) Strain X



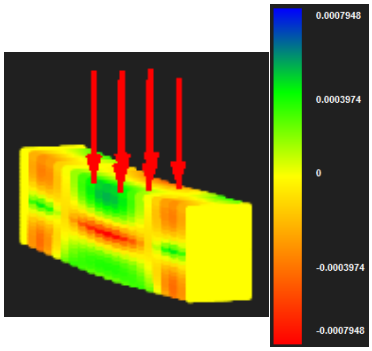
(b) Strain Y



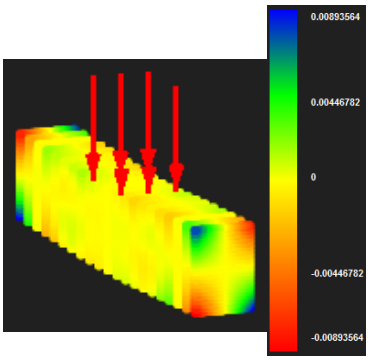
(b) Strain Z



(c) Strain XY

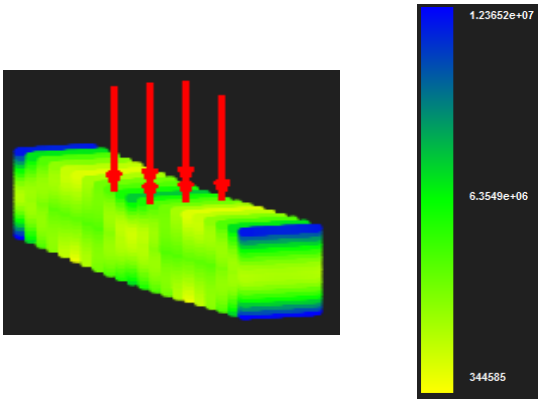


(d) Strain YZ

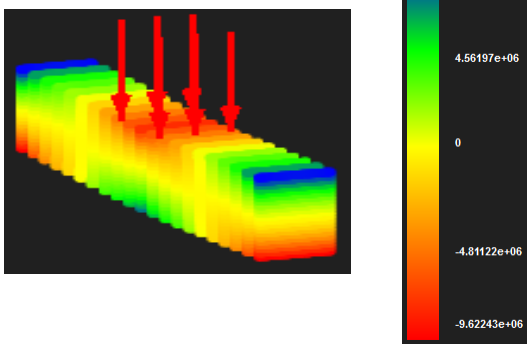


(e) Strain ZX

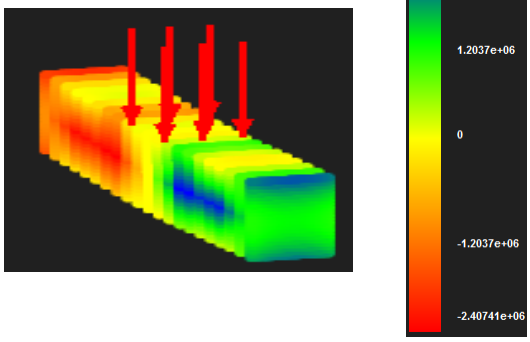
Figure 5.86. Contours for strains



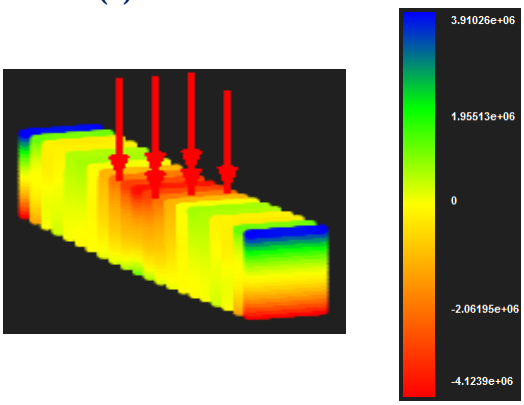
(a) Stress von Mises



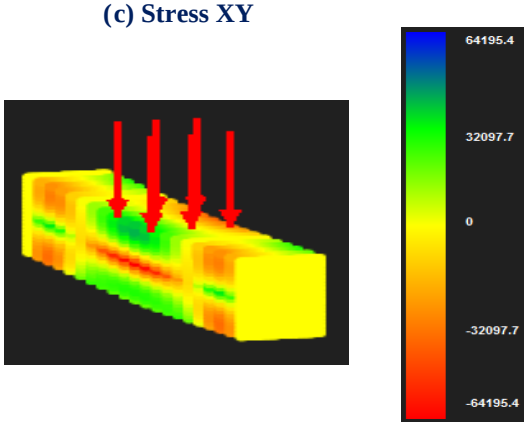
(b) Stress X



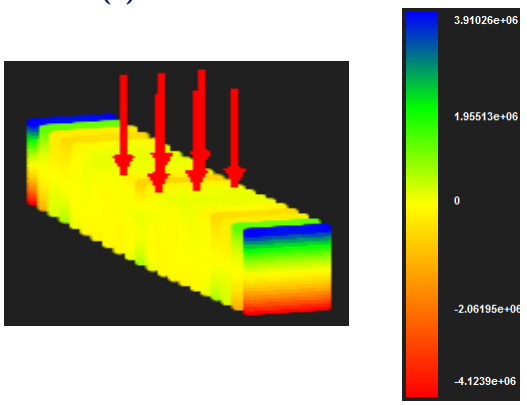
(c) Stress XY



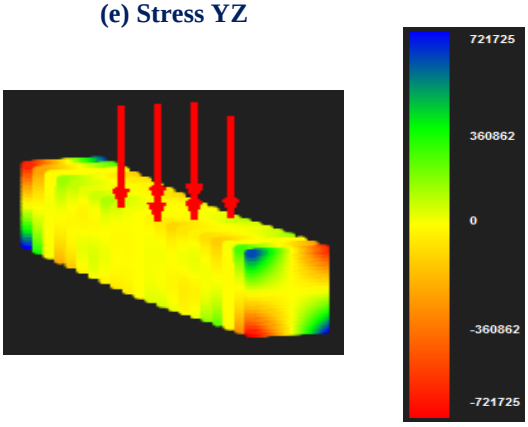
(d) Stress Y



(e) Stress YZ



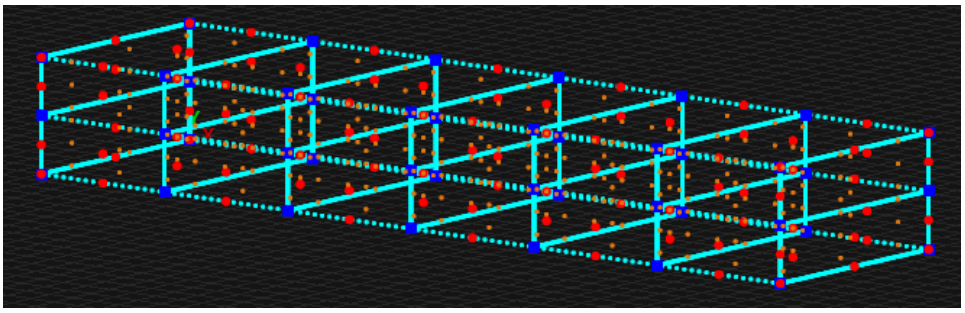
(f) Stress Z



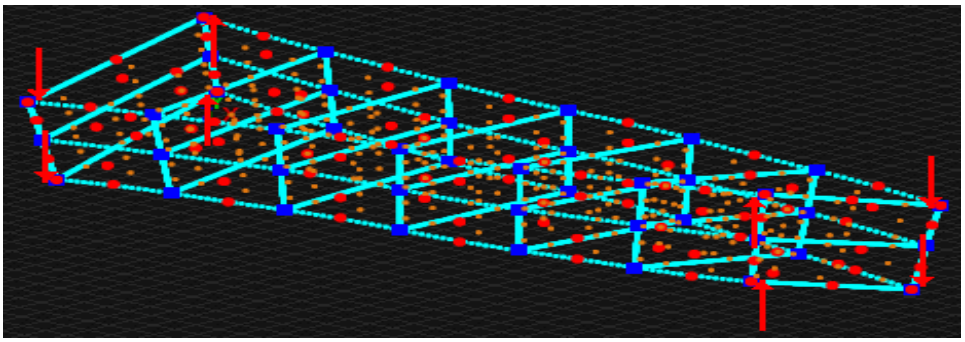
(g) Stress ZX

Figure 5.87. Contours for stresses

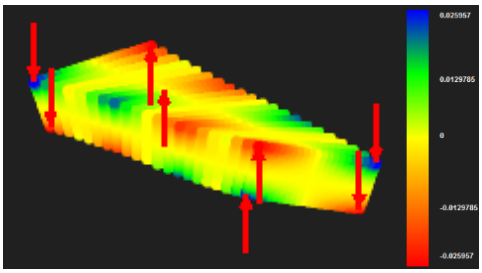
Another load case that presents quite interest is torsion. We apllied 8 concentrated loades of 800000 *kN* in each corner of a similar model. The parameterization is again the same with the previous one. This time the middle area is fixed.



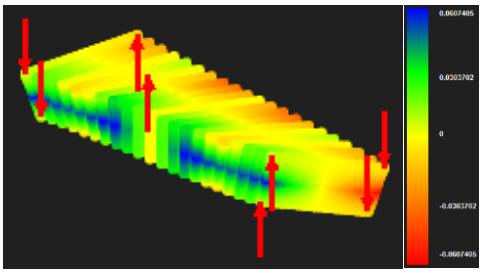
(a) Physical space 8x4x3 control points, 8x1x2 m for rectangle 3D



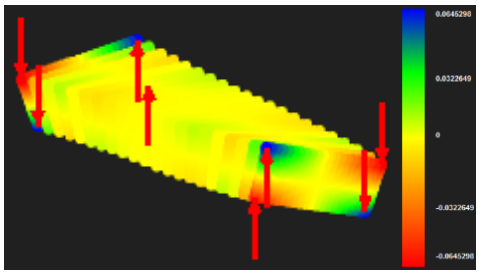
(b) Physical space deformed



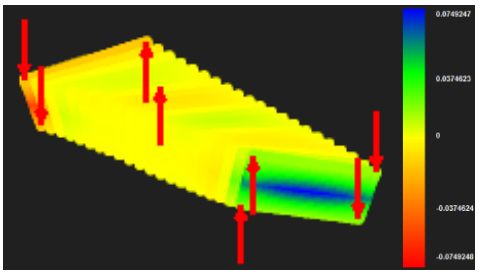
(c) Strain X



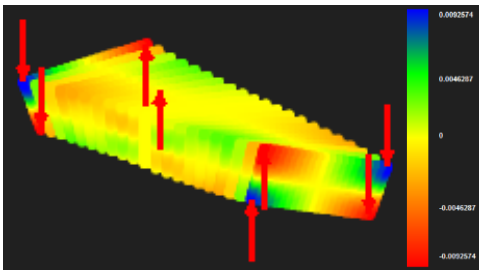
(d) Strain XY



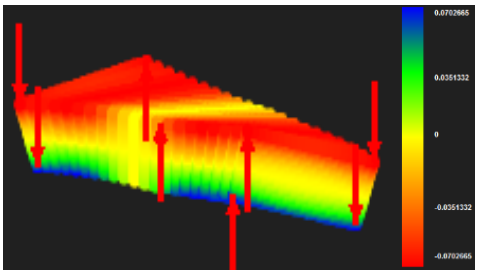
(e) Strain Y



(f) Strain YZ

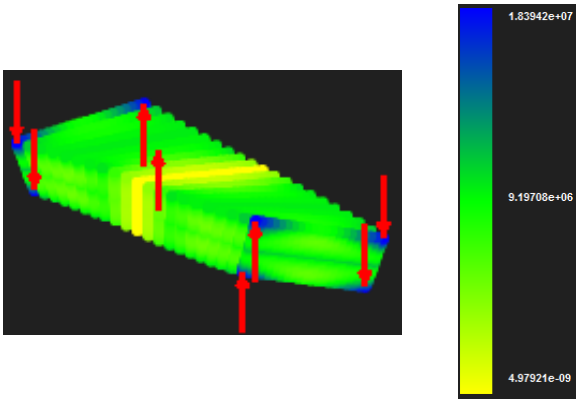


(g) Strain Z

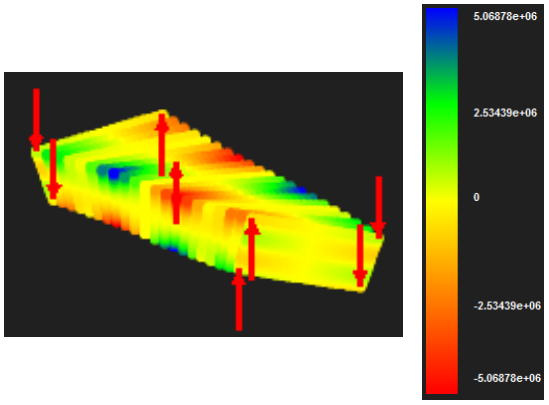


(h) Strain ZX

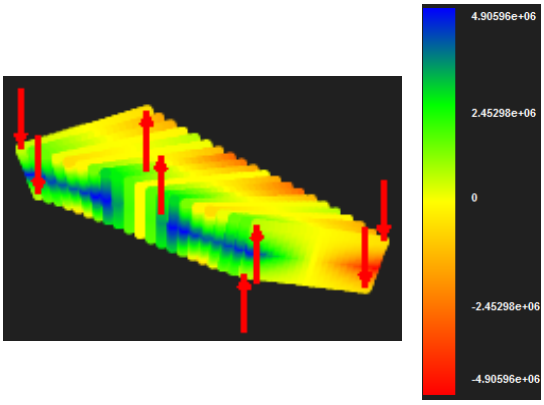
Figure 5.88. Contours for Stresses



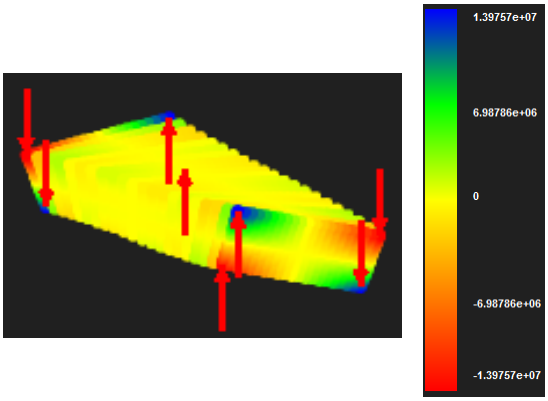
(a) Stress von Mises



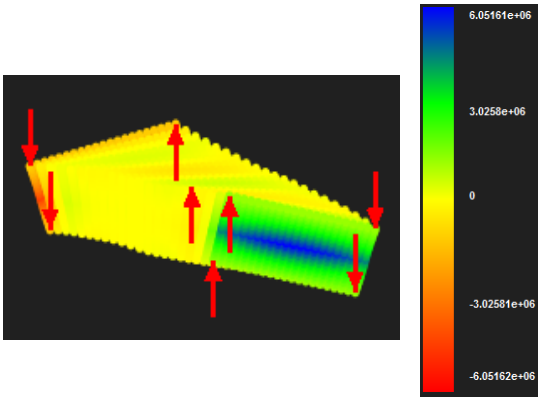
(b) Stress X



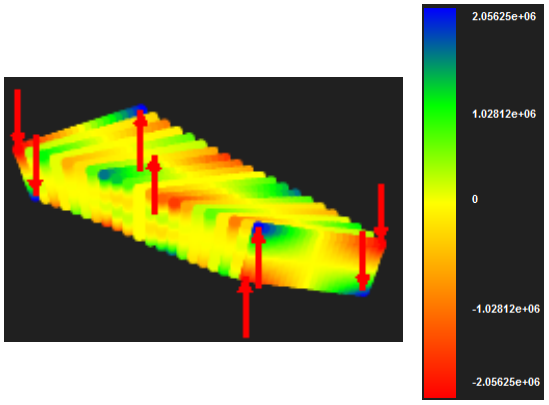
(c) Stress XY



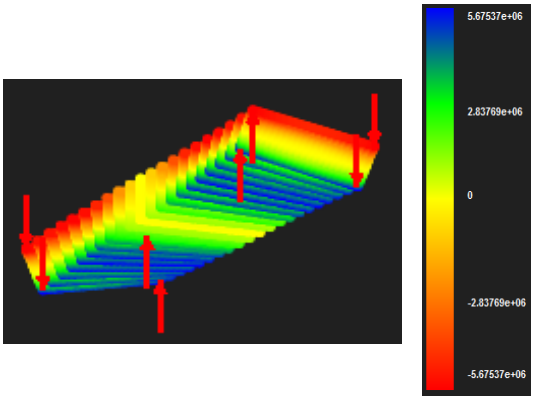
(d) Stress Y



(e) Stress YZ



(f) Stress Z



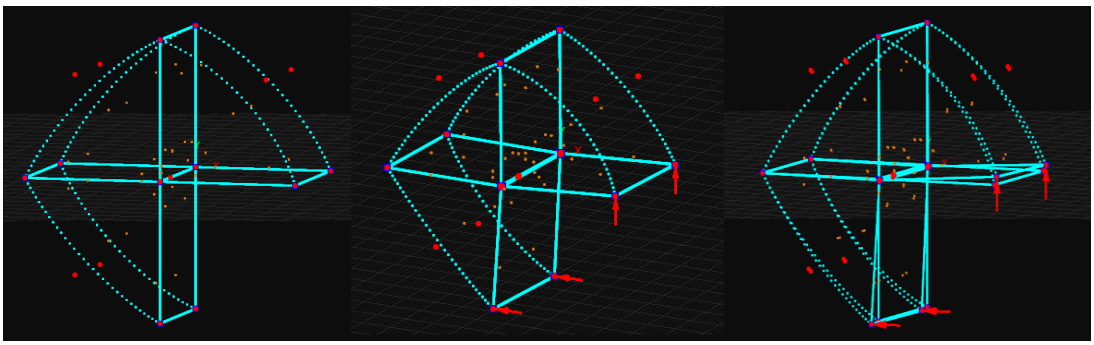
(g) Stress ZX

Figure 5.89. Contours for Stresses

5.7 Camembert

..We will analyze a model that has the shape of a camembert and we will put 4 loads of 400000 *kN* on the control points shown in figure 5.90.

Steel has been chosen as material while as far as parameterization concerned we have used quadratic basis functions on axis ξ and linear basis functions on axes η, ζ.



(a) **(b)** **(c)**

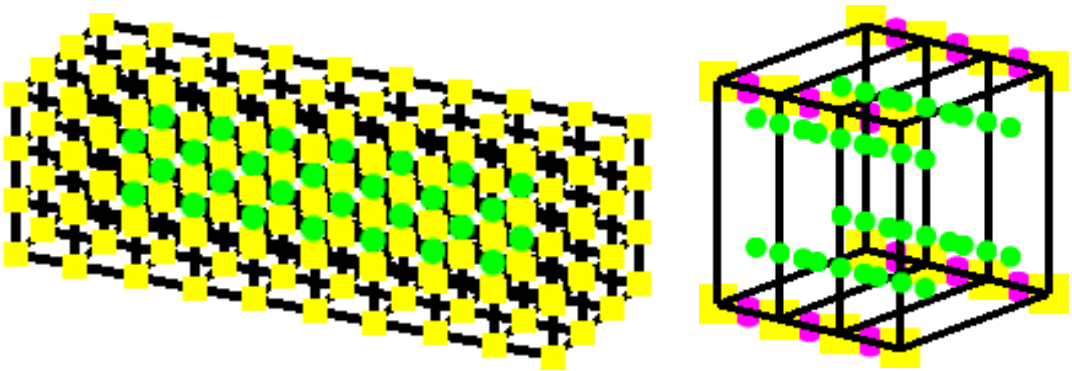
Figure 5.90. Camembert 7x2x2 control points.
(a) Physical space and (b) Physical space deformed for camembert
(c) Physical space deformed & undeformed comparison.

Knot value vector: $\Xi = \{0 \quad 0 \quad 0.333333 \quad 0.333333 \quad 0.666667 \quad 0.666667 \quad 1 \quad 1\}$

Knot value vector $H = \{0 \quad 0 \quad 1 \quad 1\}$

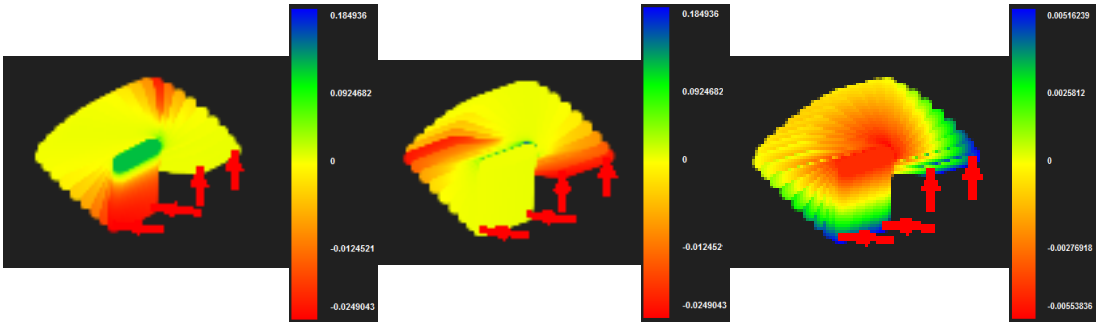
Knot value vector $Z = \{0 \quad 0 \quad 1 \quad 1\}$

There are formed 3 knot Rectangles. There are $n = 7$ control points on ξ, $m = 2$ control points on η and $r = 2$ control points on ζ for a total of 28 points and 84 degrees of freedom.

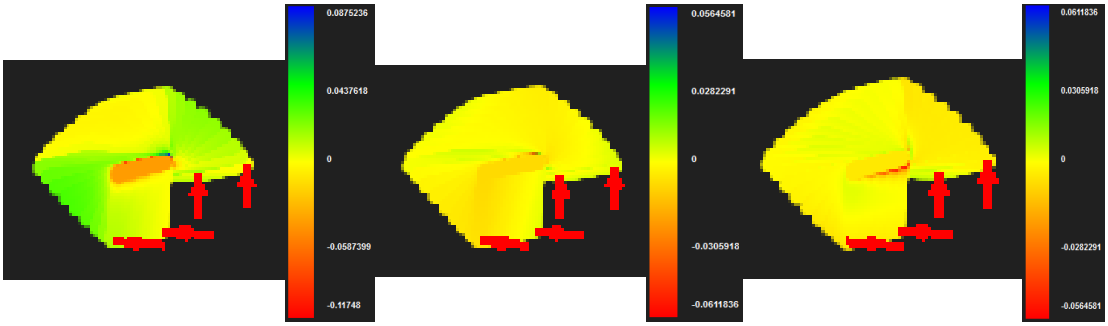


(a) Index space **(b) Parameter space**

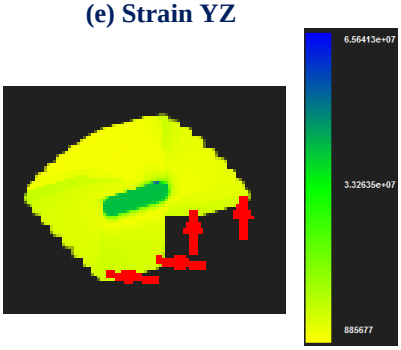
Figure 5.91. Index and parameter space



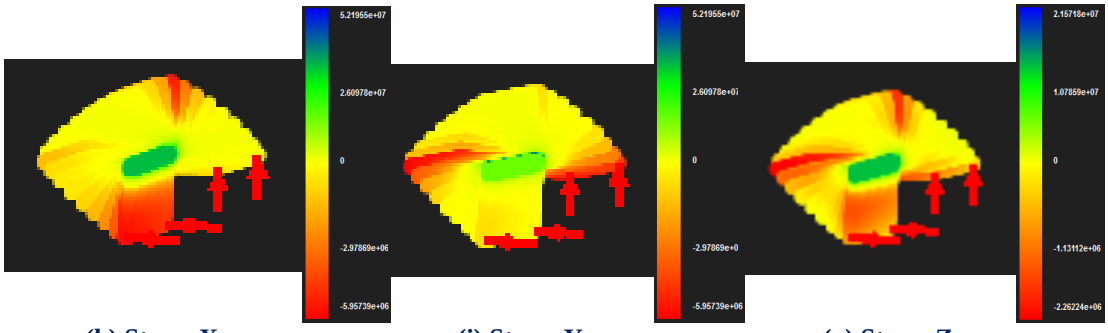
(a) Strain X (b) Strain Y (c) Strain Z



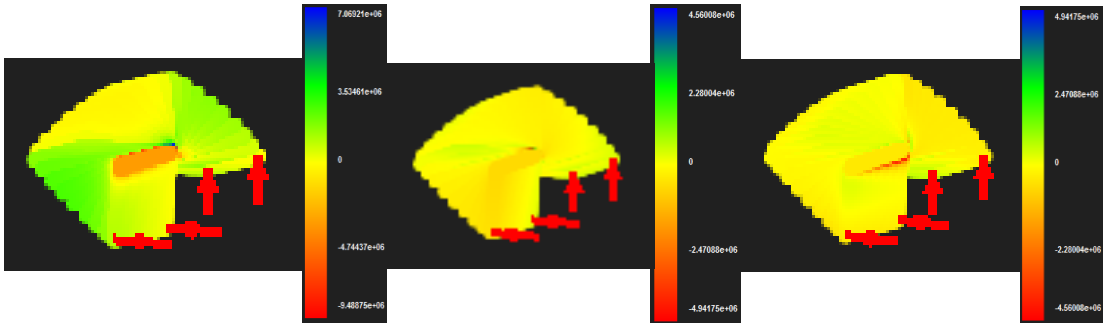
(d) Strain XY (e) Strain YZ (f) Strain ZX



(g) Stress von Mises



(h) Stress X (i) Stress Y (g) Stress Z



(k) Stress XY (l) Stress YZ (m) Stress ZX

Figure 5.92. Contours for strains and stresses.

5.8 Automotive NURBS - Car

Companies in order to increase their competitiveness internationally, strive to limit the «product development design cycle time», reduce the «time to market», limit the cost of design and prototype analysis (optimal use of energy and construction materials) and optimize the quality (strength, functionality) of the final product.

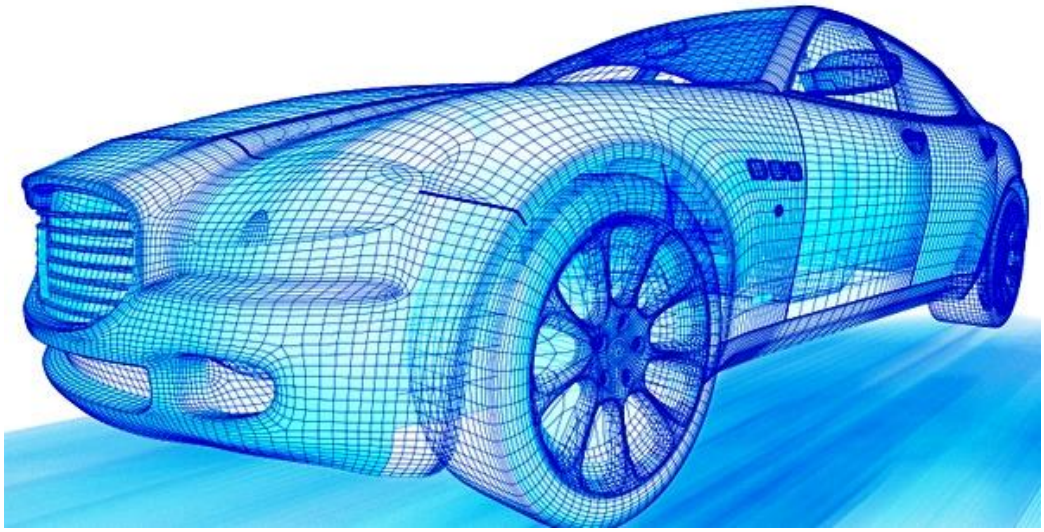


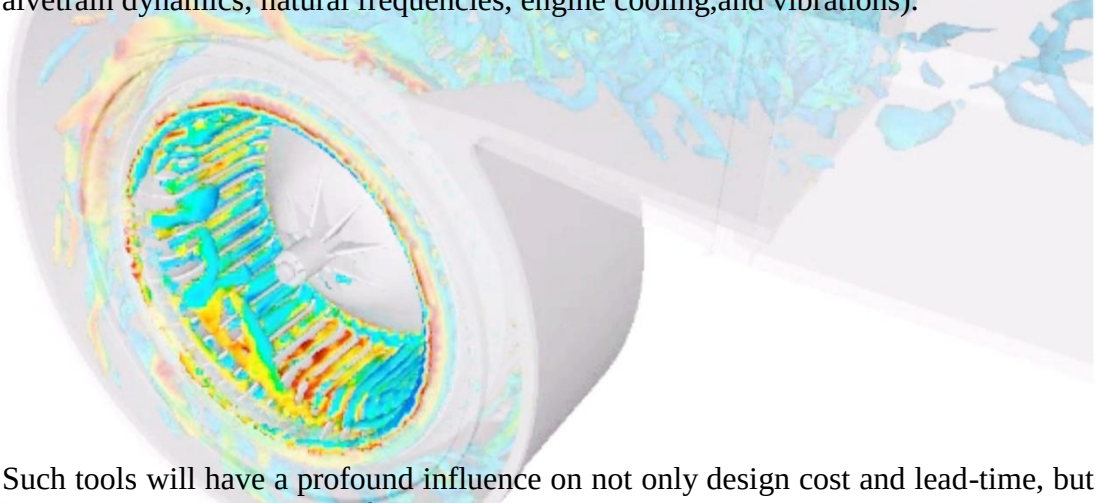
Figure 5.93 Car model, Isogeometric mesh
(http://automotiveschools.com/resources/automotive-designers-in-los-angeles)

This fact, combined with the existing gap between CAD and CAE, which is exacerbated by the rapid development of computational geometry and the increasing demands on speed and accuracy of computational solving engineering problems require the creation of a computational engineering software that will be able to cover the need for combined use of CAD / CAE and will reclaim and utilize the modern technology CAD tools. The lack of such software is particularly noticeable.

Figure 5.94. Car designs
(http://www.theartcareerproject.com/automobile-design-career/584/)
(http://www.cvel.clemson.edu/Projects/cvel_proj_autosafe.html)
(http://restogroupelyon.blogvie.com/2010/11/23/garage-auto-jwls/)

In automotive industries up to now, a sector (group of designers, computational geometry) is responsible for designing the new model (which will launch the company) and another area (group of engineers, computational mechanics) attempts to simulate with finite elements and identify blemishes and problem areas in geometry received from the domain of designers. If the analysis reveals shortcomings, the engineers return the model to designers, designers correct it and send it back to the engineers, the engineers create a new mesh of finite elements based on the new designed geometry, and then they run again the simulation process for the new analysis results. The very lacking communication between the two sectors costs time and money.

According to research that has been conducted by the University of Michigan CAE may permit designs closer to the optimal before creation of the first prototype, however it still lacks the integrated tools for an entire car system modelization (for example, engine, valvetrain, powertrain, and the supporting structure) and analyze it in terms of various design criteria (for example, crashworthiness, combustion, valvetrain dynamics, natural frequencies, engine cooling,and vibrations).



Such tools will have a profound influence on not only design cost and lead-time, but also on product quality. As firms outsource various components and subsystems to suppliers, such integrated tools play a very critical role in ensuring the merit of the overall system design. The problem of not analyzing the hole model as a united entity may have as a result the independent parts of the model not to be able to react correctly which in turn would increase cost, lengthen timing, and yield a quality disaster. We must keep in mind that changes into individual parts of the model may demand changes to other relevant parts that could not so obviously identified by the engineer who is responsible for the analysis procedure. For just these reasons, systems-level integrated CAE tools are necessary to trace the effect of a design change on the entire system.

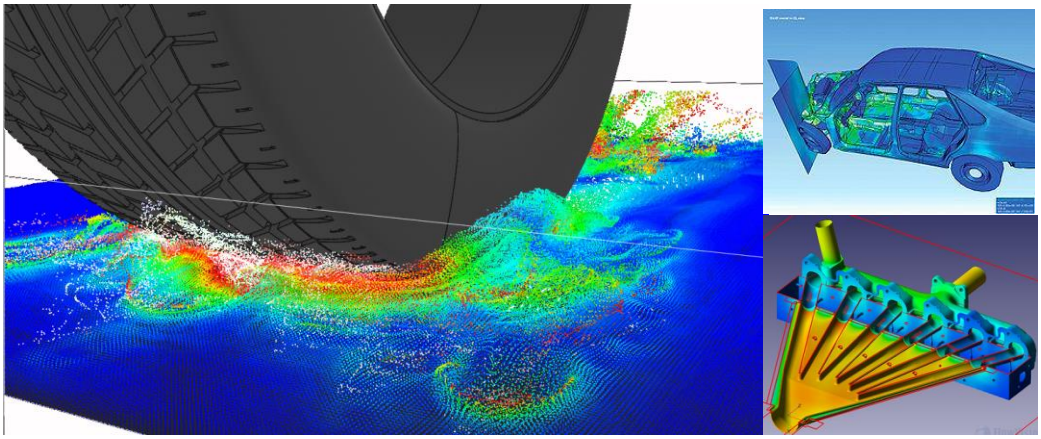


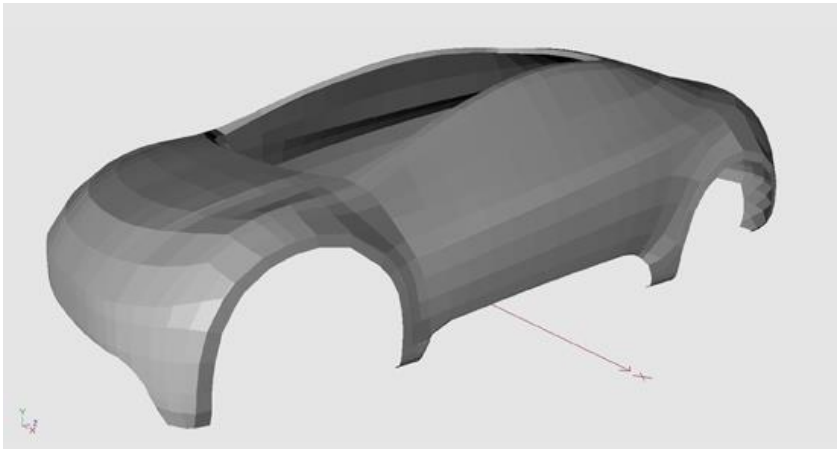
Figure 5.95 Simulation of car patches.
(http://www.mscsoftware.com/industry/automotive, http://fv-cfd.blogspot.gr/ , http://www.kocw.net/home/common/contents3/document/lec/2013/ChonNam/LeeDongWon/3/5.pdf)

One of the most revolutionary fields of Isogeometric analysis is the fact that can significantly eliminate these problems in automotive industry. Any software that will have dual nature CAD / CAE will be able to simultaneously design (geometric design) and analyze (isogeometric analysis) both individual components (modeled as patches to their own material properties and their own geometry) and the entire vehicle as a single kit.

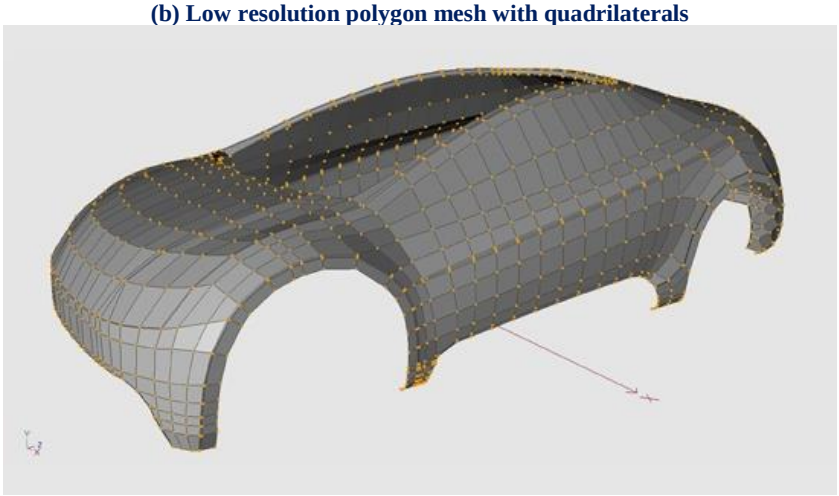
Our next model that we are going to try to analyze is the following car. The procedure that an engineer would follow in order to perform a simulation to his model through a CAE software would be to design it via the graphic user interface and represent it to its (outdated as far as CAD tools concerned possibly) graphic environment.

(a) Real model

The design procedure would be done either by the designer either by the engineer himself by defining his initial mesh through multiple points that would constitute the corners of small polygonal meshes which in turn would define the finite elements that are necessary for the analysis procedure.



(b) Low resolution polygon mesh with quadrilaterals

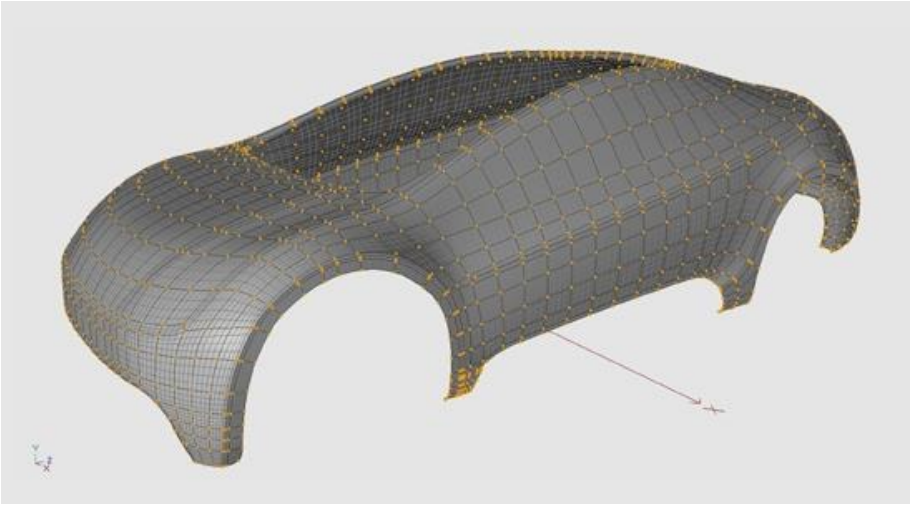


(c) Low resolution polygon mesh, vertex position and finite elements.

Figure 5.96. Coarse polygon mesh
(https://grabcad.com/library/tag/tutorial)

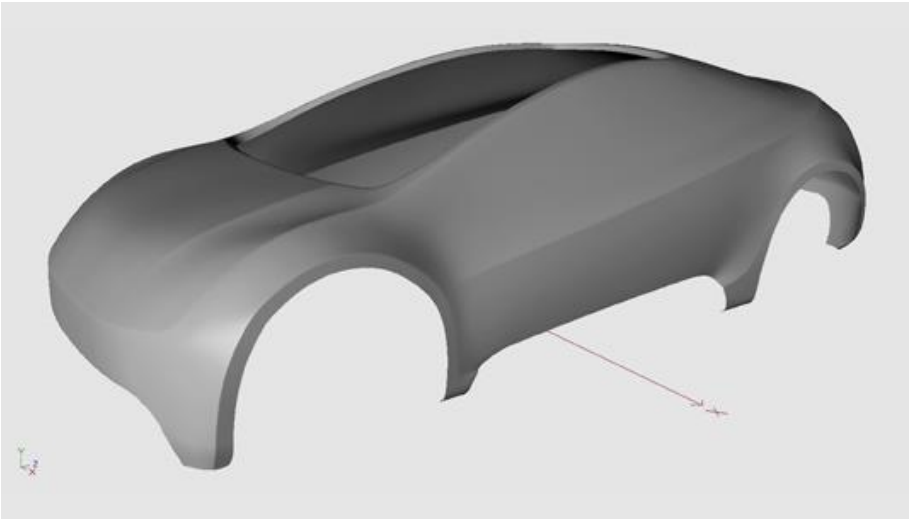
We can see on the above pictures how the designer would have designed the initial mesh if he wanted to analyze the external shell of the car for impact or other kinds of loads to test its strength.

For decades engineers hadn't understand that behind the modern CAD tools were existed hidden and encrypted all the information they needed in order to accomplish a direct analysis procedure. They were unknown of the fact that the model they designed was simultaneously the simulation they needed.



(a) High resolution Nurbs mesh, control points & Isogeometric elements

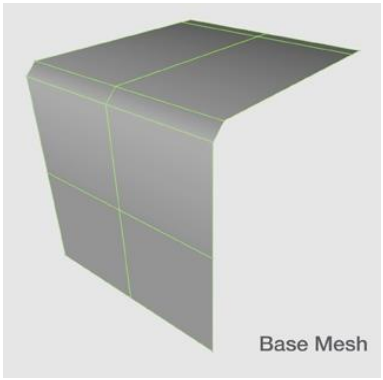
In figure 5.97 (a) we can observe the higher by orders of magnitude resolution of the initial mesh while the time a designer or an expertized engineer would take in order to design-simulate the model would be extremely insignificant.



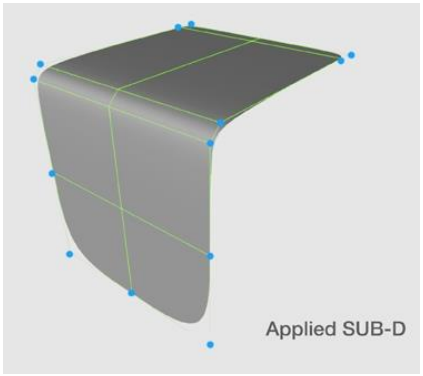
(b) High resolution Nurbs mesh – shaded

Figure 5.97. Nurbs mesh.
(https://grabcad.com/library/tag/tutorial)

In pictures below we can observe the main difference of a single piece of the model depending on the parameterization and the simulation method we have chosen.

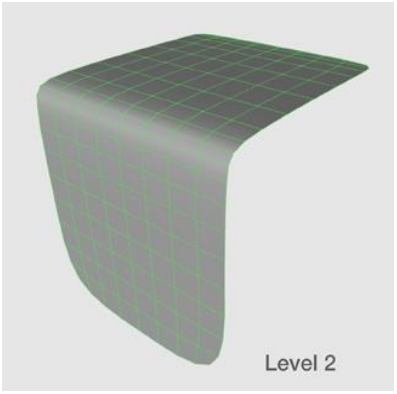


(a) Low resolution polygon mesh

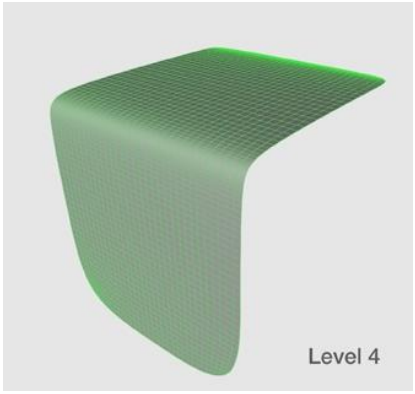


(b) High resolution Sub-Division surfaces

The top left part shows the discretization of the model if we choose the Finite Element method. The top right image shows the exact discretization if we choose to simulate the model with Subdivision Surfaces, similar would be the model if we had chosen Nurbs instead.



(c) Dense refinement



(d) Very dense refinement

Figure 5.98. Polygon mesh and Sub-Division mesh comparison.
(https://grabcad.com/library/tag/tutorial)

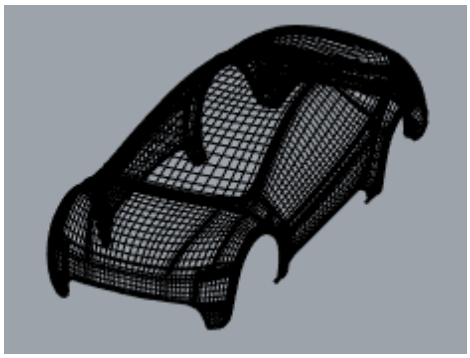
The bottom figures display the isogeometric elements that Subdivision surfaces' discretization leads to. We can certainly observe how complicated can be the mesh for a very high resolution. Such smoothness and complexity could never get reached if we attempted to use polygon rectangles-finite elements to represent our model.

Now if we suppose that we want to analyze a specific part-patch of this model the only necessary information we need from the design to accomplish the analysis procedure are three things **for each parametric axis**:

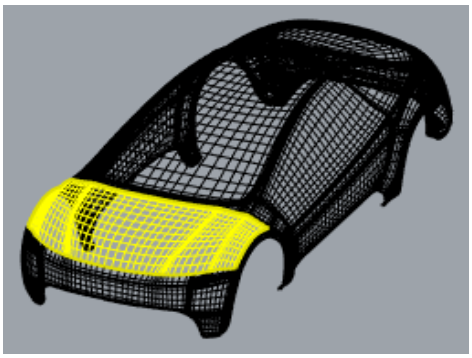
- Degree of the Polynomial basis function
- Knot value vector
- Control point Coordinates

With these three parameters we can analyze any CAD model.

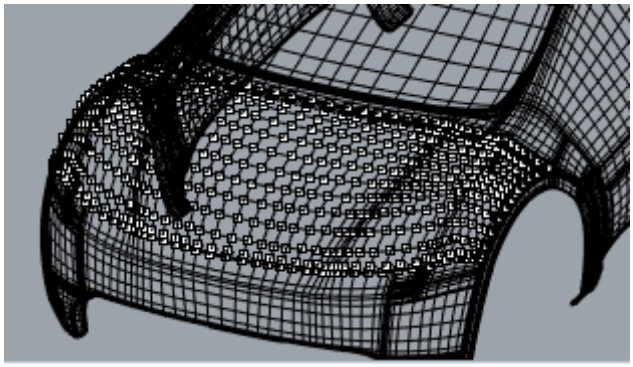
We will analyze the front car hood as a single patch (while our software is not yet capable of solving multiple patches).



(a) High resolution Nurbs mesh

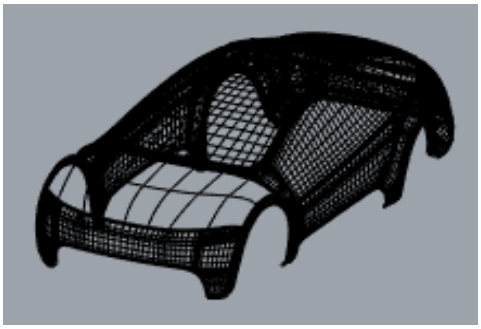


(b) High resolution Nurbs Car hood-patch

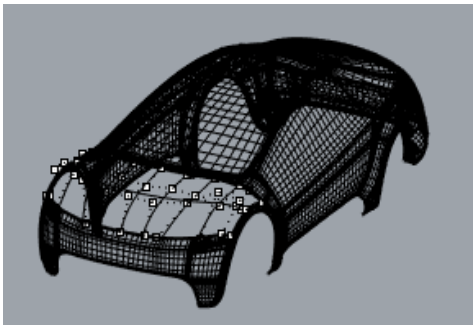


(c) Car hood – patch control points

Because of the complexity (too many control points on axes ξ, η) of the patch we are going to interfere on our model and impose a refinement so as our software can proceed with the analysis results.



(d) h-Refinement on car hood



(e) Car hood 4x11 control points

Figure 5.99. Initial mesh and applied h-Refinement.

But before we manage to move on we have to face one more problem. Every single patch of our model is a 2-Dimensional problem. There are no control points on Parametric axis ζ. This is not strange as there are no softwares globally and moreover designers that design pure 3-Dimensional problems-Solids, as there is no need to do such thing. CAD softwares care mostly for the right representation and proper visualization of a model so a massive model would be completely unnecessary and useles. If a designer wanted to make a 3-Dimensional represantation, for example a cube then he would just draw 6 2-Dimensional surfaces each for every side of the cube. That cube inside would be completely empty with us a result the creation of what we call “polysurface” and not solid.

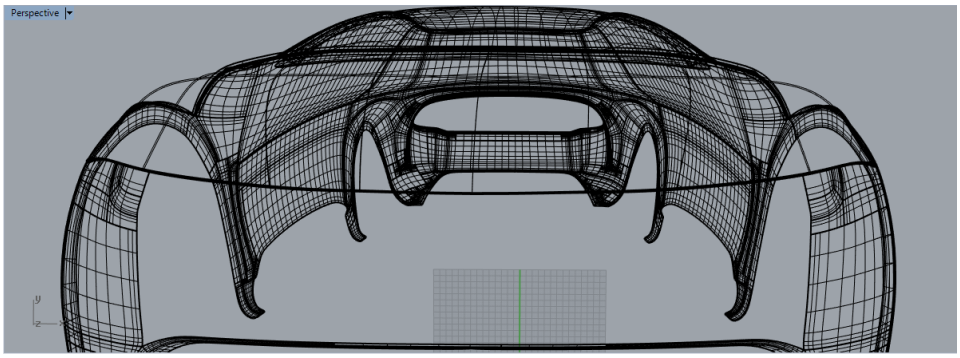
Of course some CAD softwares provide the user the ability to convert that polysurface to a solid but even then the procedure is not so easy and not known to the majority of designers or engineers.

We can use two different ways to analyze our model:

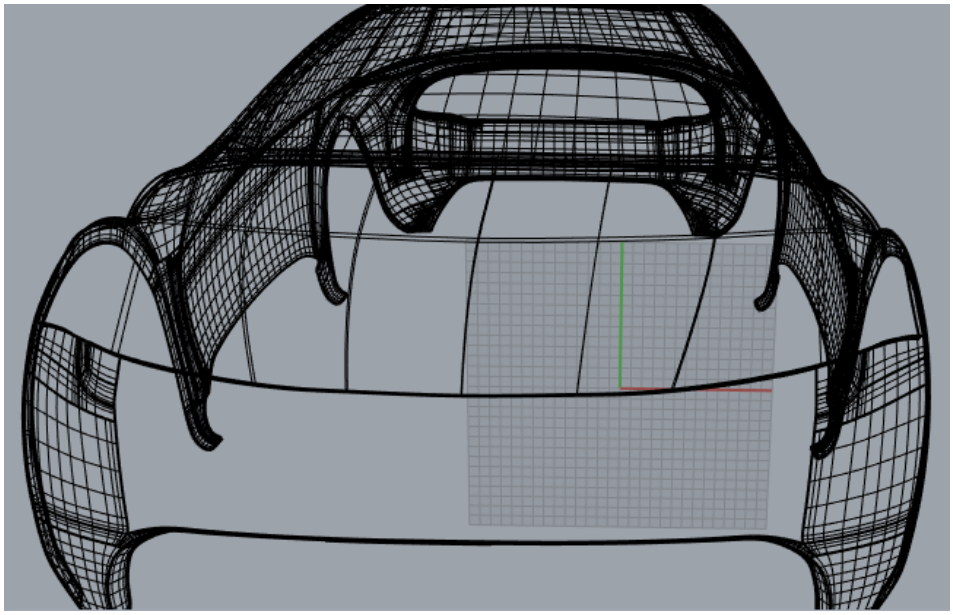
- Use of Shell elements
- Add a third dimension

As far as the first solution concerned our software is incapable of solving shell elements yet. So, the second solution seems to be the only way of proceeding.

We must again interfere on our model and create a new surface above the initial and be careful so that each single point of the initial surface to be sticked out vertically.This procedure was accomplished by using the appropriate CAD tools.



(a) Initial mesh

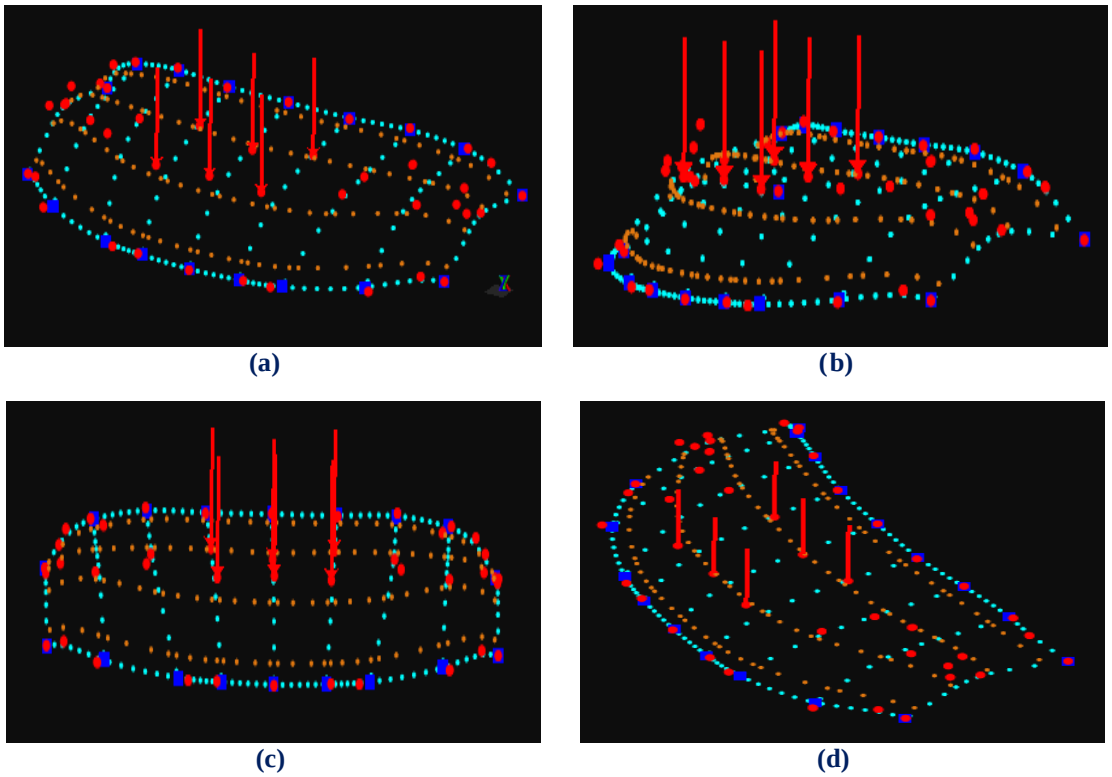


(b) Generated surface on parametric axis ζ

Figure 5.100. (a), (b) Applied p-Refinement

All we have to do now is simply decode the encrypted information of our new surface. If the new created surface has not the same parameterization with the initial one, then we simply apply a refinement so as the final result has the same discretization.

Our model is ready for the analysis procedure. We will put 6 external concentrated loads of 50 *kN* at the control points of the middle as in figure 5.101. Of course I could choose any kind of boundary conditions, but instead of fixing the perimeter I chose to fix the four corners of the model to simplify the analysis procedure.

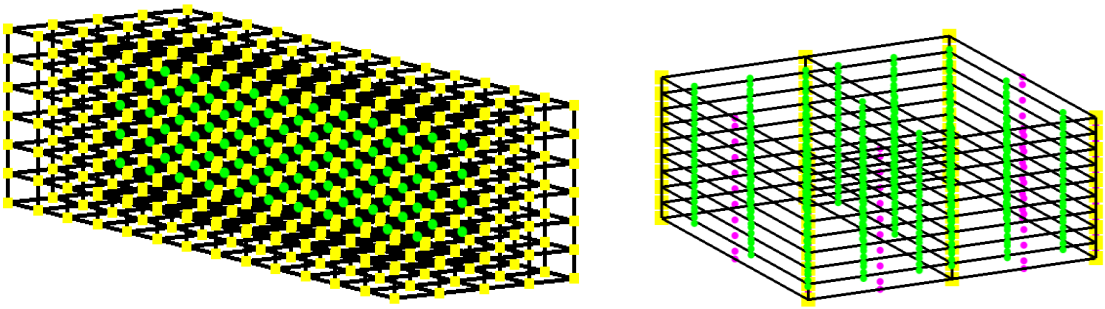


(a) **(b)**

(c) **(d)**

Figure 5.101 (a), (b) Physical space for car hood.

Steel has been chosen as material while as far as parameterization concerned we have used cubic basis functions on axes ξ, η and linear basis functions on axis ζ.



(c) Index space **(d) Parameter space**

Figure 5.102 (c), (d) Mesh representation

There are formed 16 Isogeometric elements. There are $n=4$ control points on ξ, $m=11$ control points on η and $r=2$ control points on ζ for a total of 88 points and 264 degrees of freedom.

The dimensions of our patch are equal to 1.50 m length, 0.60 m width and we chose to offset our surface in a distance of 0.003 m.

In Figure 5.103 we can have a better sence of what exactly are we analyzing.

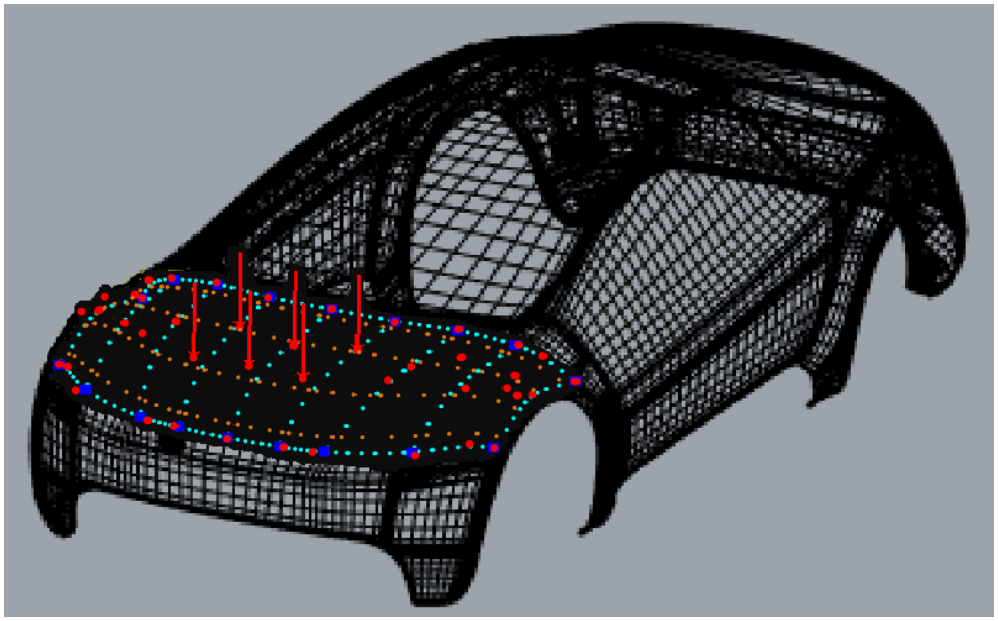
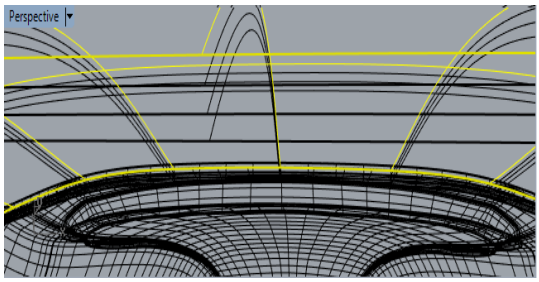


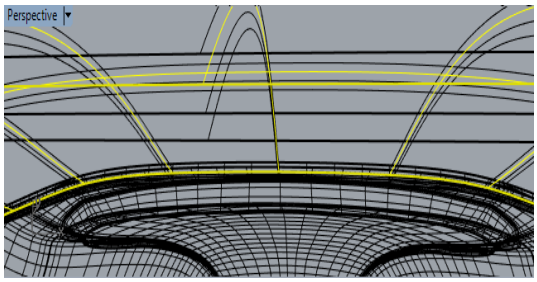
Figure 5.103 Physical space – control point & knot mesh.

Some tests in papers have shown that excellent results were obtained with quartic-NURBS in the structural surface (even for very poor meshes) on elastic linear structures, although it was necessary to use at least cubic-NURBS through the thickness. When linear-NURBS are used through the thickness relatively high errors occur as far as displacement concerned.

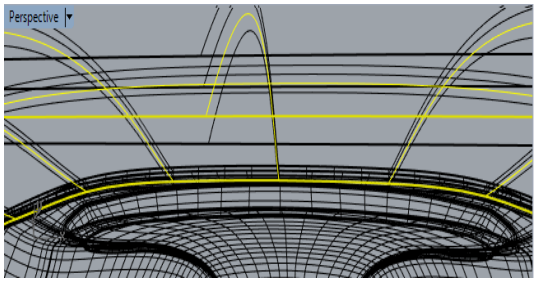
In order to investigate that claim we will aplly two refinements with order elevation. Following the same procedure that we used to create a new surface as a third dimension, we can create more than one surfaces above the initial one and apply quadratic and cubic basis functions exploiting the relative information of each surface.



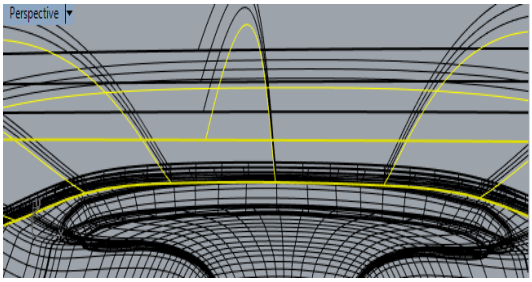
(a) First surface 4x11x4 control points



(b) Second surface 4x11x4 control points



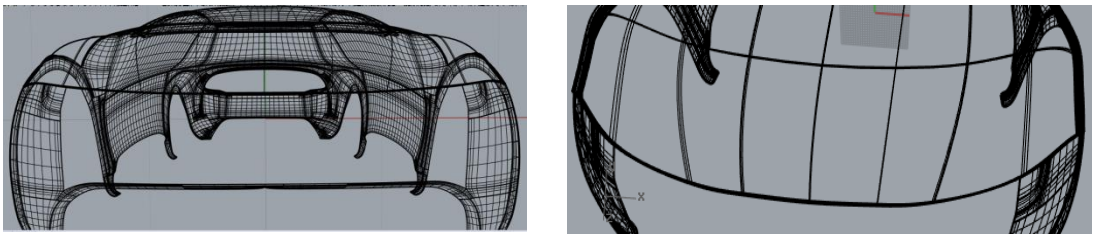
(c) Third surface 4x11x4 control points



(d) Forth surface 4x11x4 control points

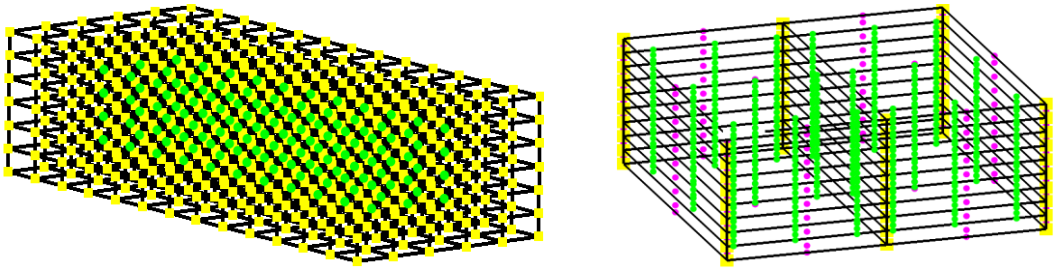
Figure 5.104 Surfaces on ζ direction for cubic basis functions thickness.

The first p-Refinement consists of cubic basis functions on axes ξ, η and quadratic on axis ζ. There are $n = 4$ control points on ξ, $m = 11$ control points on η and $r = 3$ control points on ζ for a total of 132 points and 396 degrees of freedom.



(a) Initial mesh 4x11 control points **(b) First p-Refinement 4x11x3 control points**

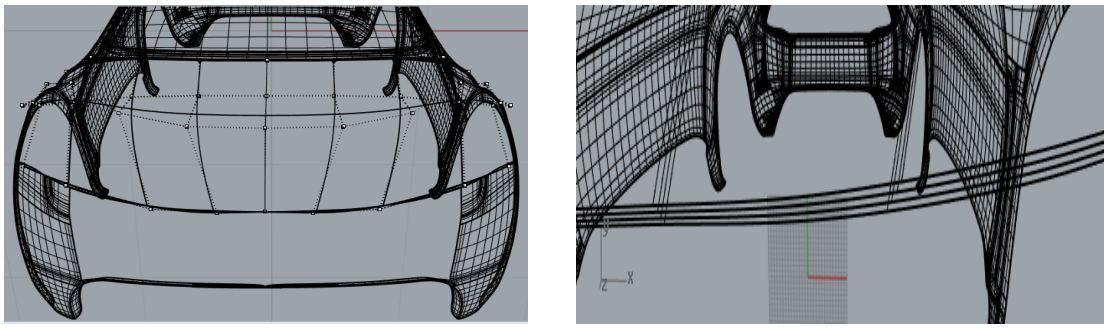
Figure 5.105 Mesh under first h-Refinement



(c) Index space **(d) Parameter space**

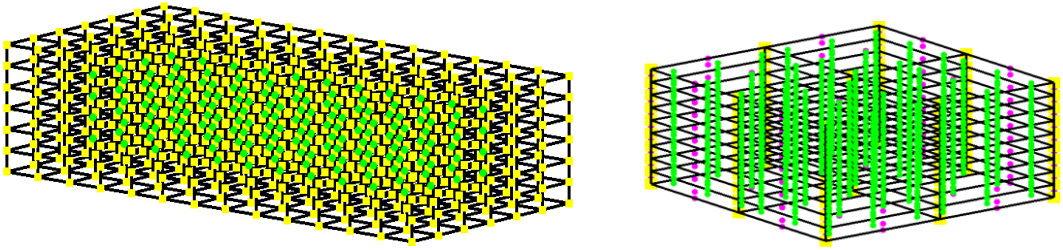
Figure 5.106 Mesh representation.

The second p-refinement consists of cubic basis functions on axes ξ, η ζ. There are $n = 4$ control points on ξ, $m = 11$ control points on η and $r = 4$ control points on ζ for a total of 176 points and 528 degrees of freedom.



(a) Initial Mesh 4x11 control points **(b) Second p-Refinement 4x11x4 control points**

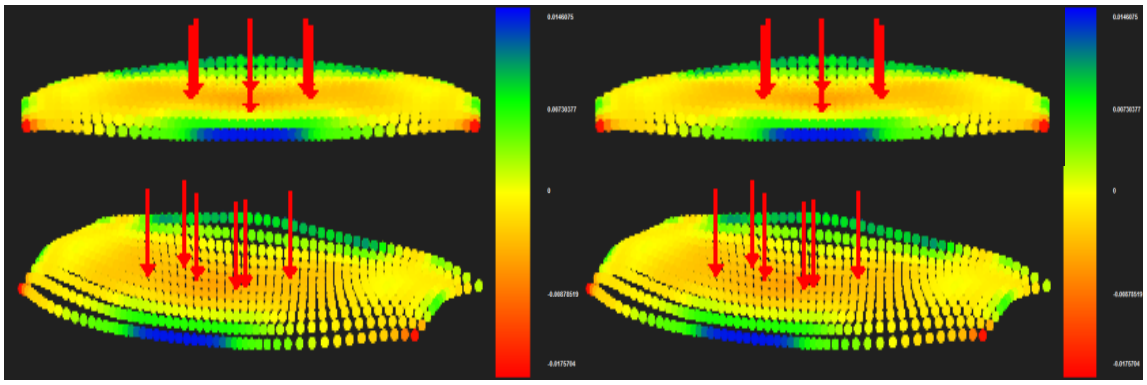
Figure 5.107 Mesh under second h-Refinement



(c) Index space **(d) Parameter space**

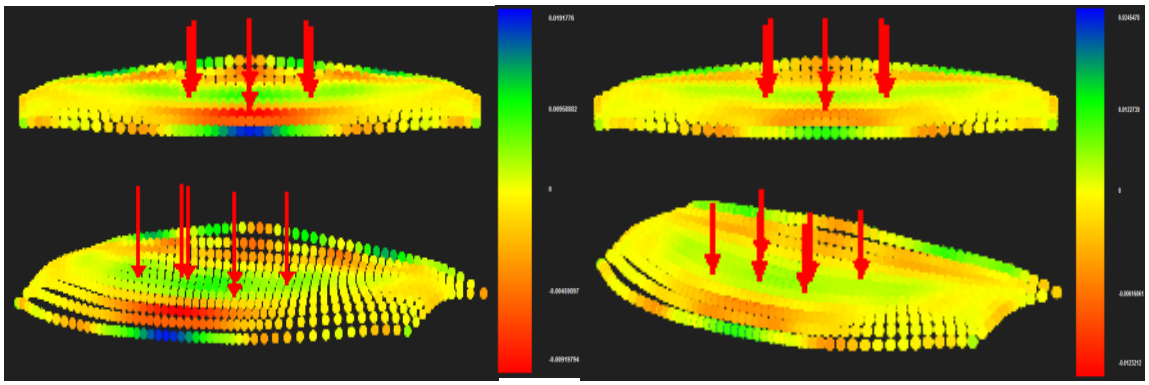
Figure 5.108 Mesh representation.

In Figures 5.108, 5.109, 5.110 contours for Strain X, Y and Z Deformed are presented.



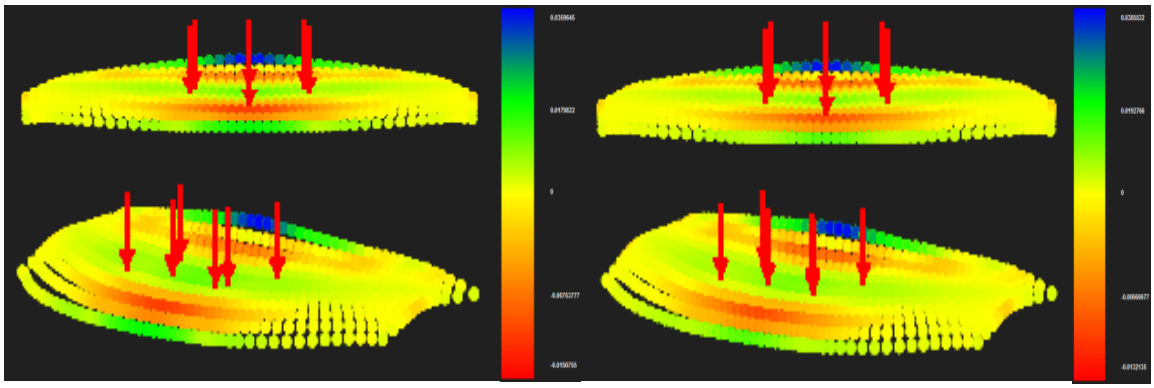
(a) 4x11x2 control points **(b) 4x11x3 control points**

Figure 5.109 Strain X



(a) 4x11x2 control points **(b) 4x11x3 control points**

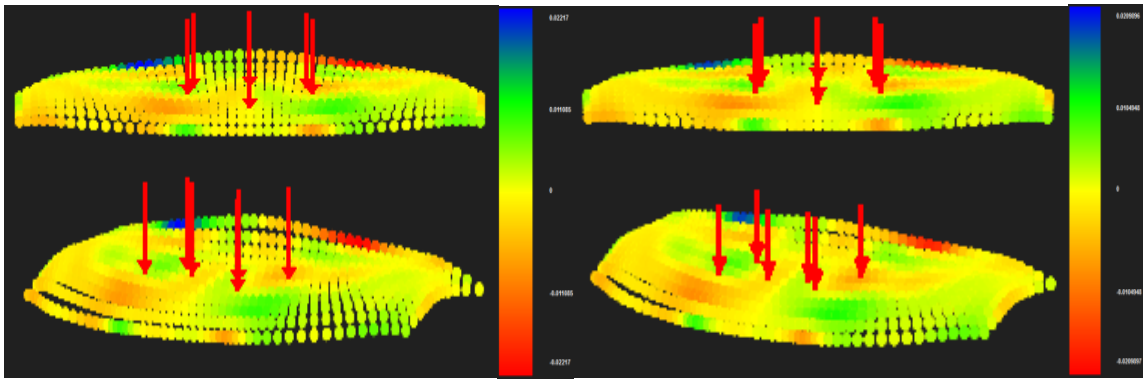
Figure 5.110 Strain Y



(a) 4x11x2 control points **(b) 4x11x3 control points**

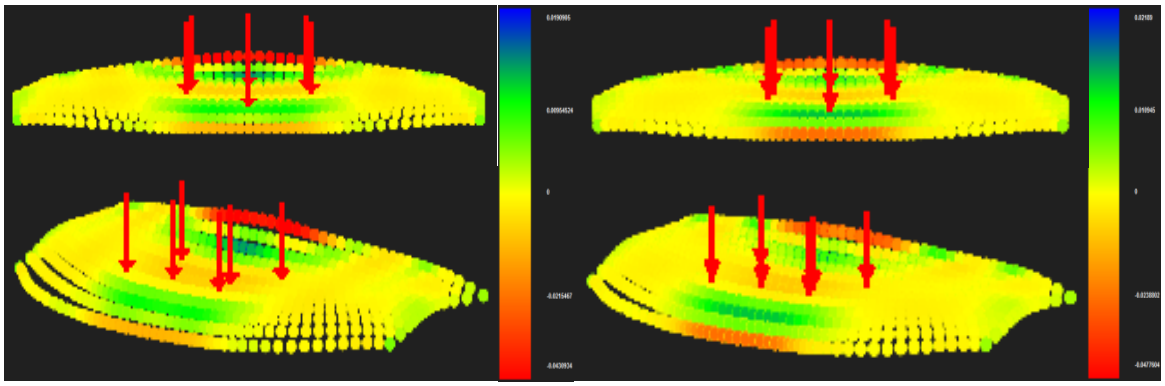
Figure 5.111 Strain Z

In Figures 5.111, 5.112, 5.113 contours for Strain XY, YZ and ZX Deformed are presented.



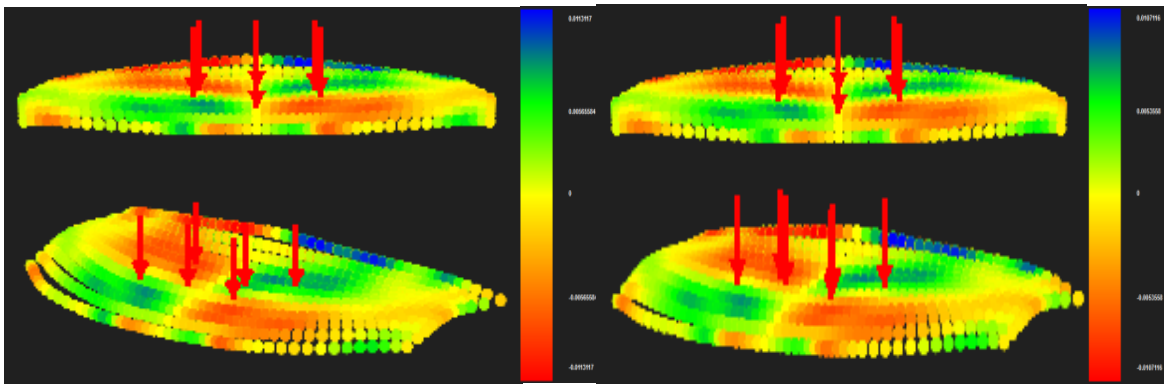
(a) 4x11x2 control points **(b) 4x11x3 control points**

Figure 5.112 Strain XY



(a) 4x11x2 control points **(b) 4x11x3 control points**

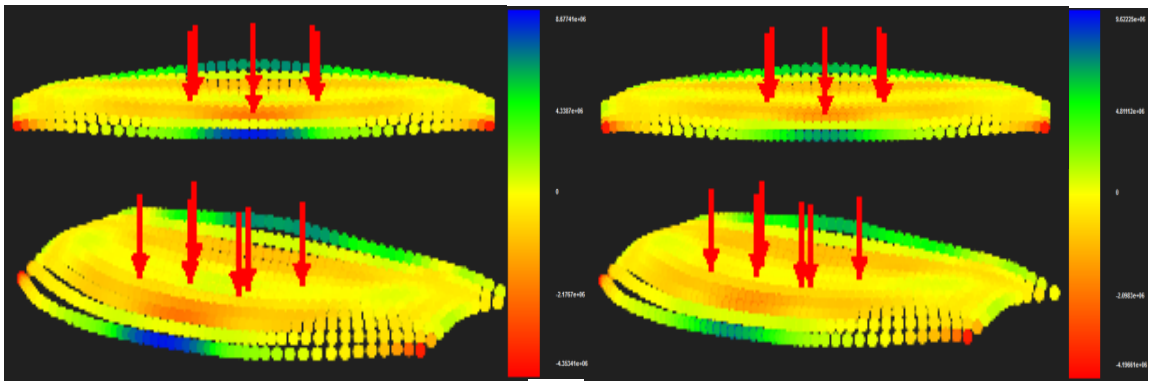
Figure 5.113 Strain YZ



(a) 4x11x2 control points **(b) 4x11x3 control points**

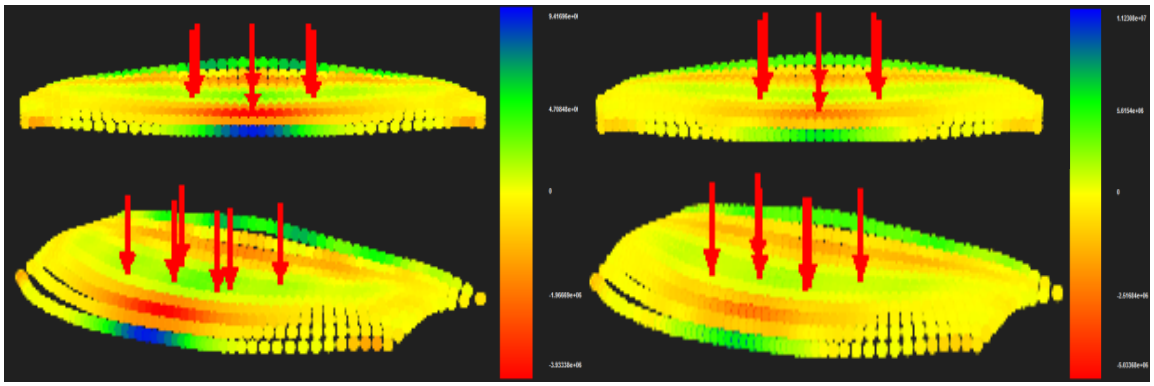
Figure 5.114 Strain ZX

In Figures 5.115, 5.116, 5.117 contours for Stress X, Y and Z Deformed are presented.



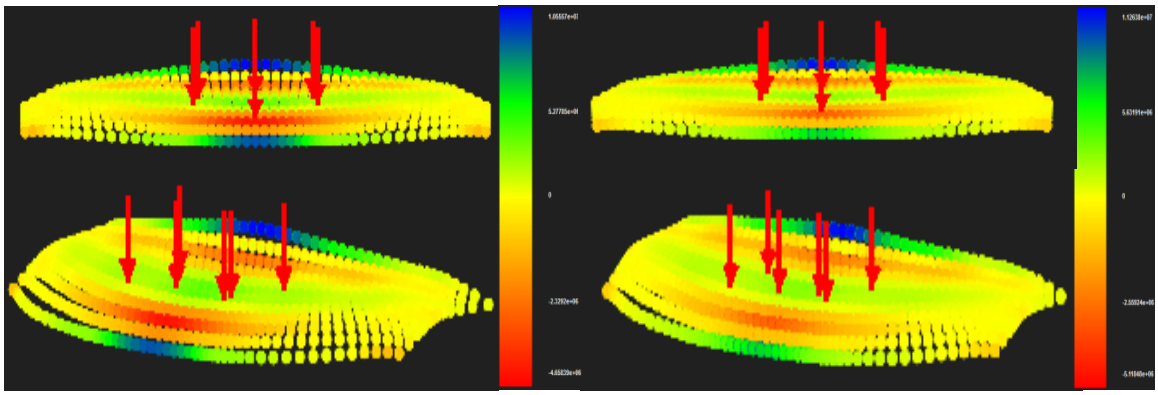
(a) 4x11x2 control points **(b) 4x11x3 control points**

Figure 5.115 Stress X



(a) 4x11x2 control points **(b) 4x11x3 control points**

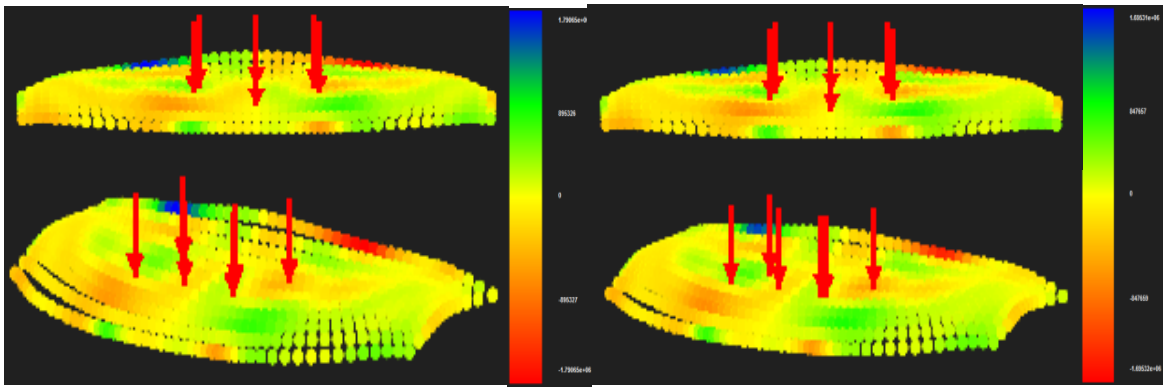
Figure 5.116 Stress Y



(a) 4x11x2 control points **(b) 4x11x3 control points**

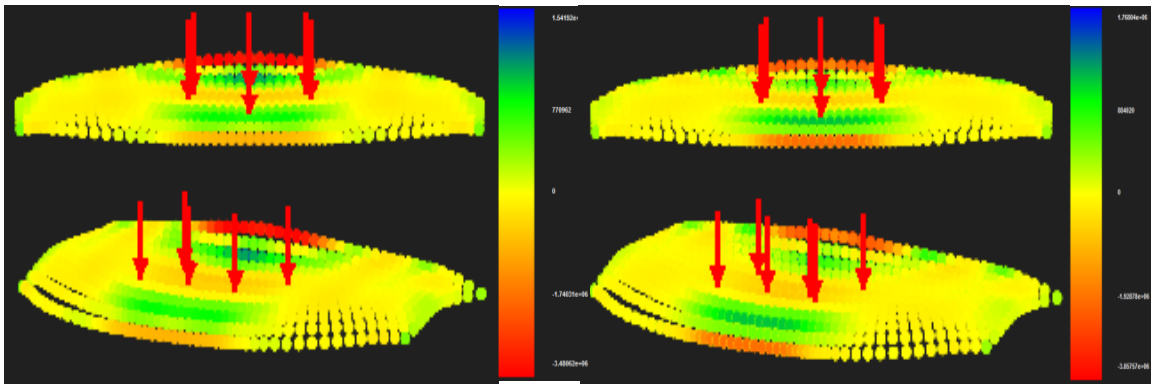
Figure 5.117 Stress Z

In Figures 5.118, 5.119, 5.120 contours for Stress XY, YZ and ZX Deformed are presented.



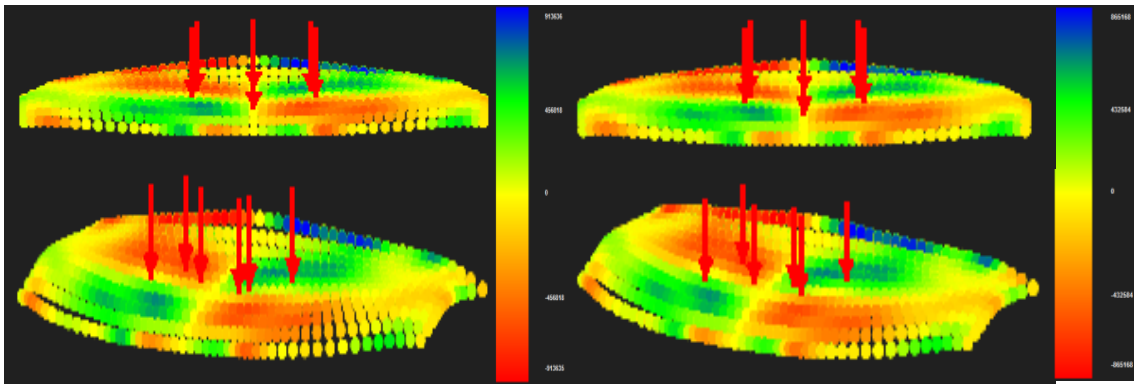
(a) 4x11x2 control points **(b) 4x11x3 control points**

Figure 5.118 Stress XY



(a) 4x11x2 control points **(b) 4x11x3 control points**

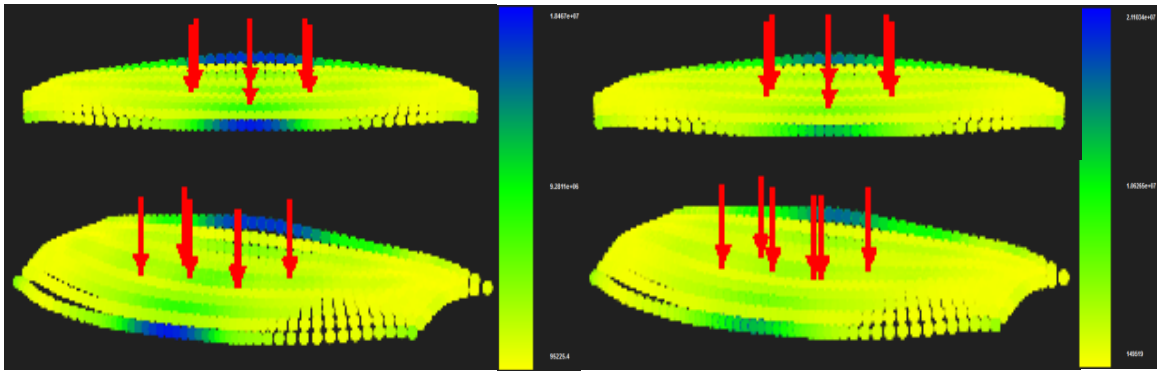
Figure 5.119 Stress YZ



(a) 4x11x2 control points **(b) 4x11x3 control points**

Figure 5.120 Stress ZX

In Figure 5.121 contour for Stress von Mises Deformed is presented.



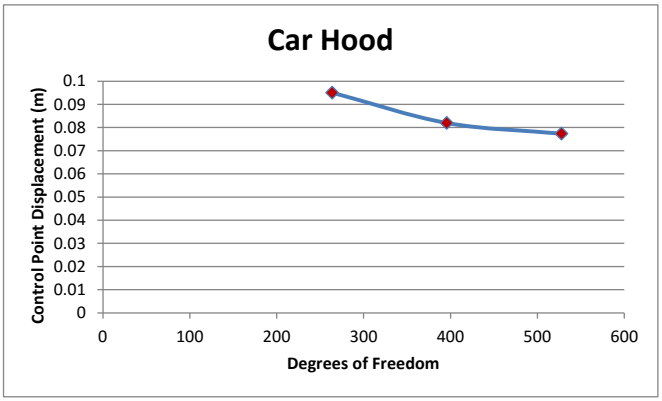
(a) 4x11x2 control points **(b) 4x11x3 control points**

Figure 5.121 Stress von Mises

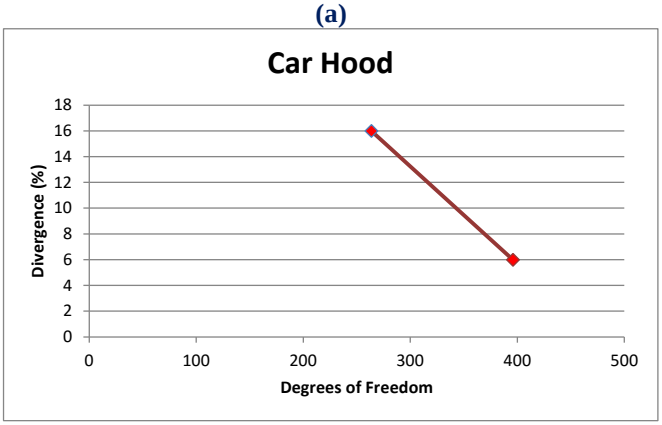
We can see on the above pictures that as parameterization becomes more detailed the greater the difference in contour representation becomes. In figure 5.122 we will ascertain the influence of thickness parameterization in control point displacement, norm displacement and von Mises Stresses.

Control Point Displacement (m):

- (4x11x2 CP)→Degrees of Freedom (264)→0.095045 m
- (4x11x3 CP)→Degrees of Freedom (396)→0.081959 m→Deviation (15.97%)
- (4x11x4 CP)→Degrees of Freedom (528)→0.077329 m → Deviation (5.99%)



(a)

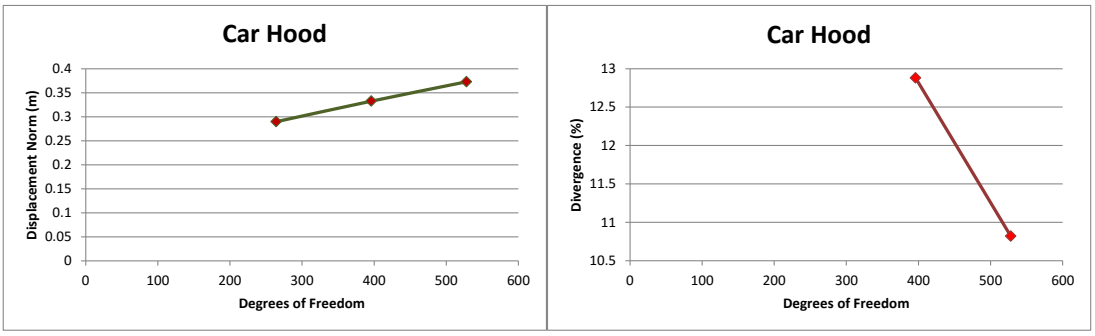


(b)

Figure 5.122 Control point displacement results.
(a) Control point displacement in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

Displacement Norm (m):

- (4x11x2 CP)→Degrees of Freedom (264)→0.289626 m
- (4x11x3 CP)→Degrees of Freedom (396)→0.332448 m→Deviation (12.88%)
- (4x11x4 CP)→Degrees of Freedom (528)→0.372784 m→Deviation (10.82%)

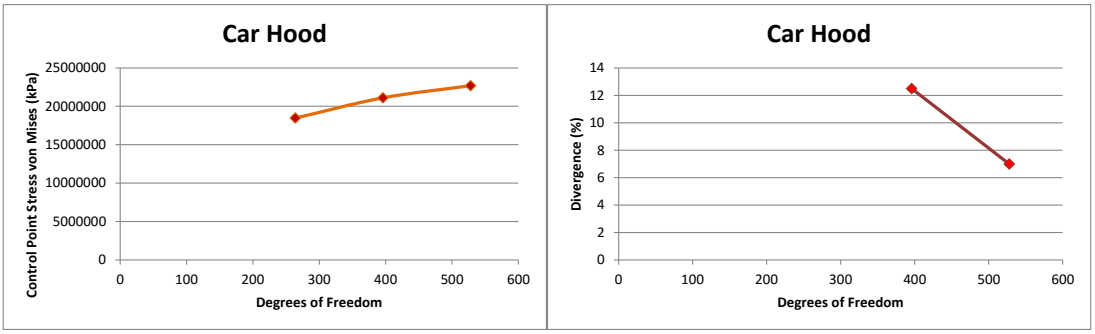


(a) **(b)**

Figure 5.123 Displacement norm results.
(a) Displacement norm in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

Control Point Stress von Mises (kPa)

- (4x11x2 CP)→Degrees of Freedom (264)→18467000
- (4x11x3 CP)→Degrees of Freedom(396)→21103400 →Deviation(12.49%)
- (4x11x4 CP)→Degrees of Freedom (528)→ 22688000 → Deviation (6.98%)



(a) **(b)**

Figure 5.124Stress von Mises results.
(a) Stress von Mises in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

5.9 SNC Dream Chaser

The next model that we are going to analyze is the Dream Chaser. SNC Dream Chaser is a reusable crewed suborbital and orbital lifting-body spaceplane being developed by Sierra Nevada Corporation (SNC) Space Systems. The Dream Chaser is designed to carry up to seven people to and from low Earth orbit. The vehicle would launch vertically on an Atlas V rocket and land horizontally automatically on conventional runways.

Dream Chaser Orbital Spacecraft		
 Dream Chaser Flight Vehicle		
Description		
Role:	Part of NASA's Commercial Crew Program to supply crew and cargo to the International Space Station	
Crew:	Up to 7	
Dimensions		
Length:	9.0 m	29.5 ft
Wing Span:	7.0 m	22.9 ft
Volume:	16.0 m^3	565 cu ft
Mass:	11,300 kg	25,000 lb
Performance		
Endurance:	At least 210 days	
Re-entry:	Less than 1.5 g	

Figure 5.125 SNC Dream Chaser
(https://en.wikipedia.org/wiki/Dream_Chaser)

Dream Chaser would have a built-in launch escape system and could fly autonomously if needed. It could use any suitable launch vehicle but it is planned to be launched on a human-rated Atlas V412 rocket.

The vehicle would be able to return from space by gliding (typically experiencing less than 1.5 g on re-entry) and landing on any airport runway that handles commercial air traffic.

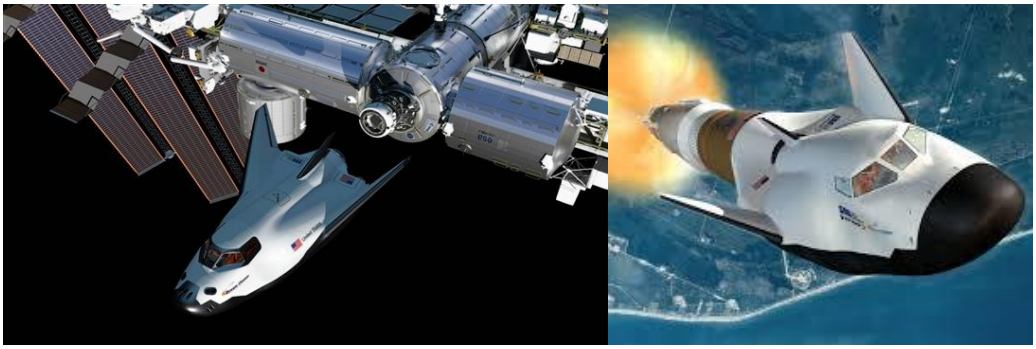


Figure 5.126 SNC Dream Chaser launch.
(http://www.spaceflightinsider.com/)
(http://astronaut.com/nasa-awards-sierra-nevada-corporations-dream-chaser-additional-milestone-funding/)

Its reaction control system thrusters burn ethanol-based fuel, which is not an explosively volatile material, allowing the Dream Chaser to be handled immediately after landing, unlike the Space Shuttle. Its thermal protection system (TPS) is an ablative tile created by NASA's Ames Research Center that would be replaced as a large group rather than tile by tile, and would only need to be replaced after several flights.

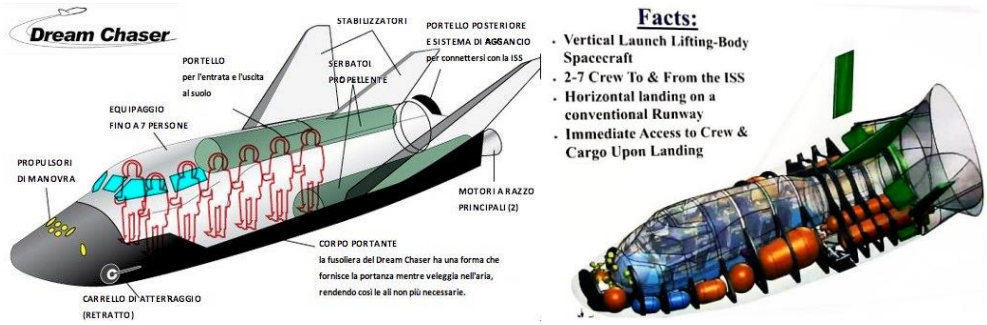


Figure 5.127 SNC Dream Chaser designs.
(http://fjordsofafrica.blogspot.gr/2014_09_01_archive.html)
(http://www.murc.ws/showthread.php?70517-Dream-Chaser-spaceplane-updates-(prototype-images)-(renamed))

On-orbit propulsion of the Dream Chaser is provided by twin hybrid rocket engines. The hybrid rocket motors are fueled with hydroxyl-terminated polybutadiene (HTPB) and nitrous oxide, or more simply put, "rubber and laughing gas". These two substances are both non-toxic and easily stored, making them safer than liquid rocket fuels. Unlike solid rockets, Dream Chaser's hybrid fuel system would allow the motor to stop and start repeatedly, and be throttleable.

Figure 5.128 SNC Dream Chaser model.
(http://www.howtoabout.org/about-the-transport/snc-dream-chaser-for-the-future.html)
(http://www.americaspace.com/?p=86264)

Now in order to analyze the specific model we must find at first the exact CAD model and load it on our software for the necessary geometry information exploitation.

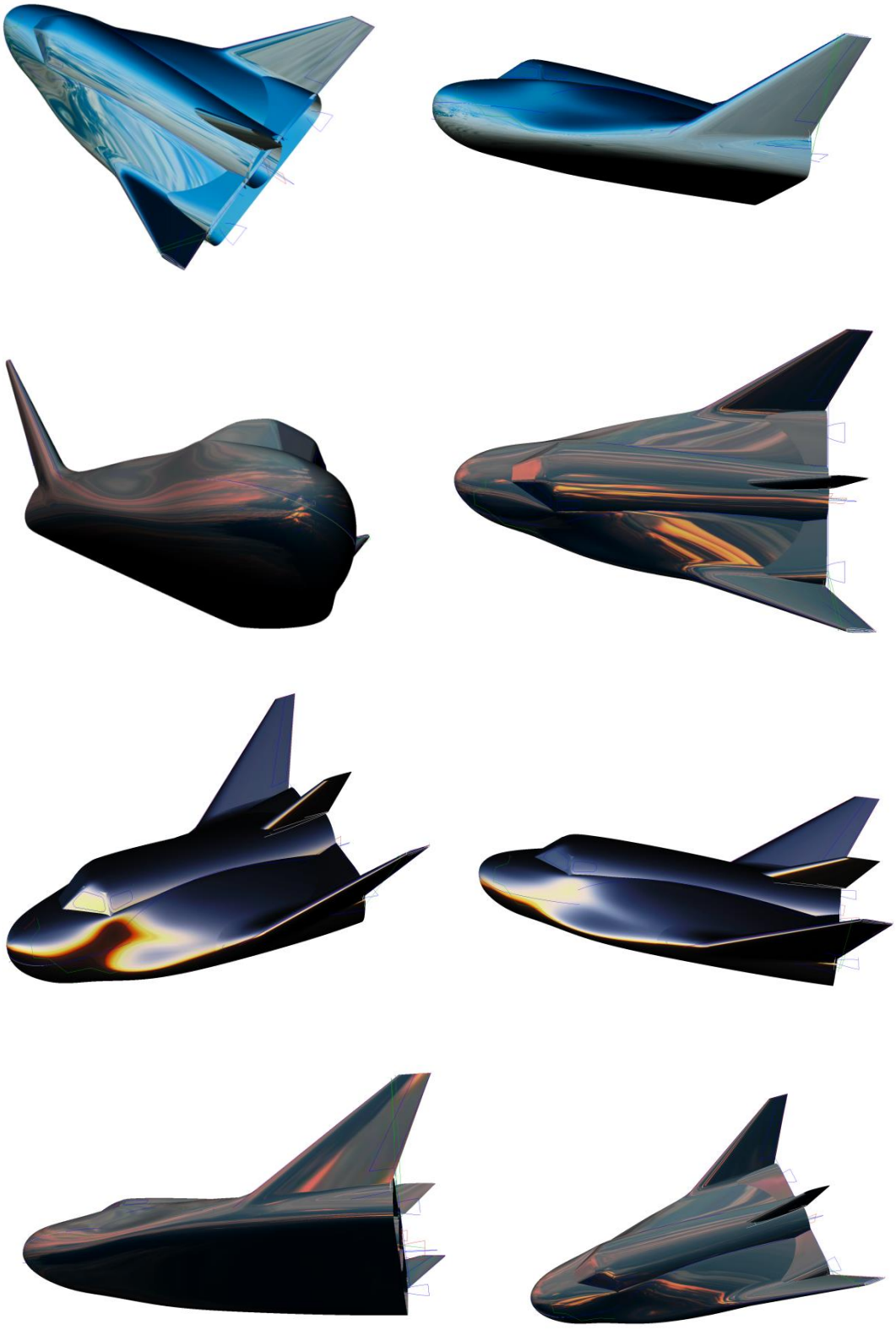
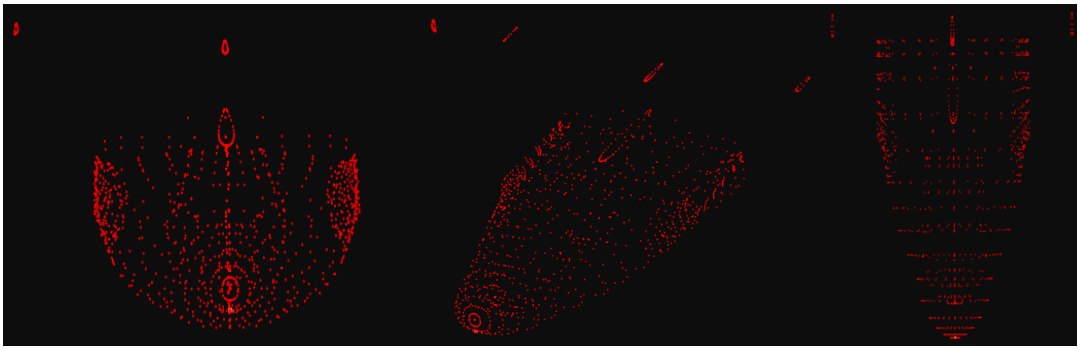
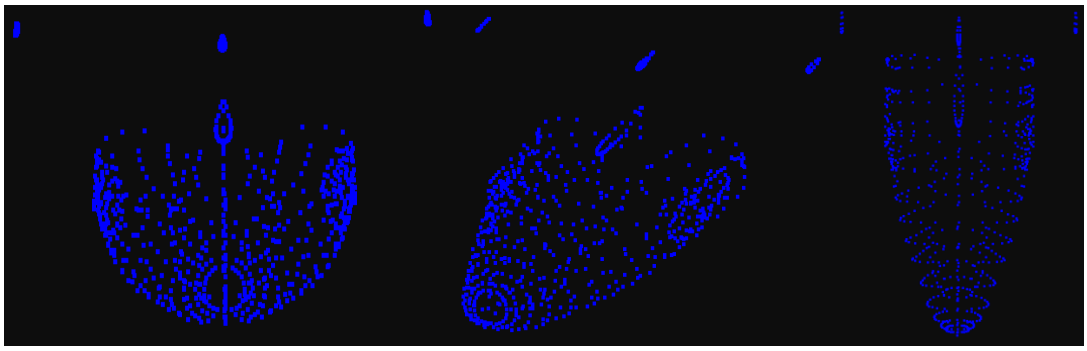


Figure 5.129 SNC Dream Chaser CAD model.
(https://grabcad.com/library/snc-dream-chaser-1)

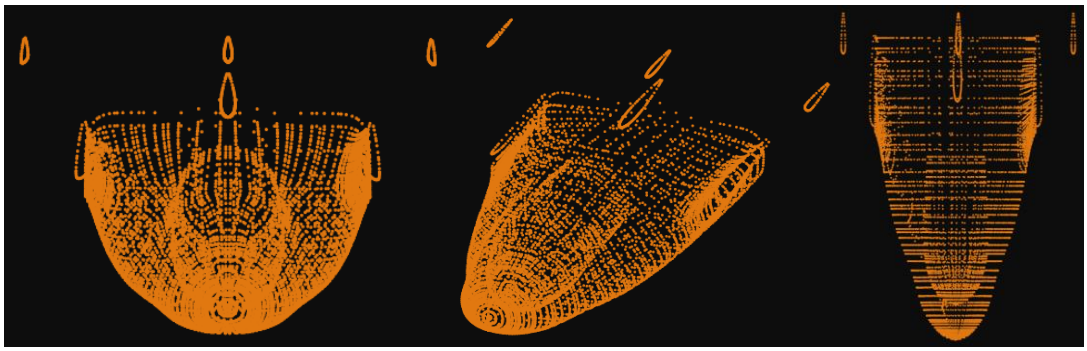
As was mentioned before, we only need 3 parameters from a CAD design for an accomplished and successful analysis procedure. With this information our software can calculate all the necessary tools (knot Coordinates, gauss point Coordinates) to proceed with the simulation and the final analysis results.



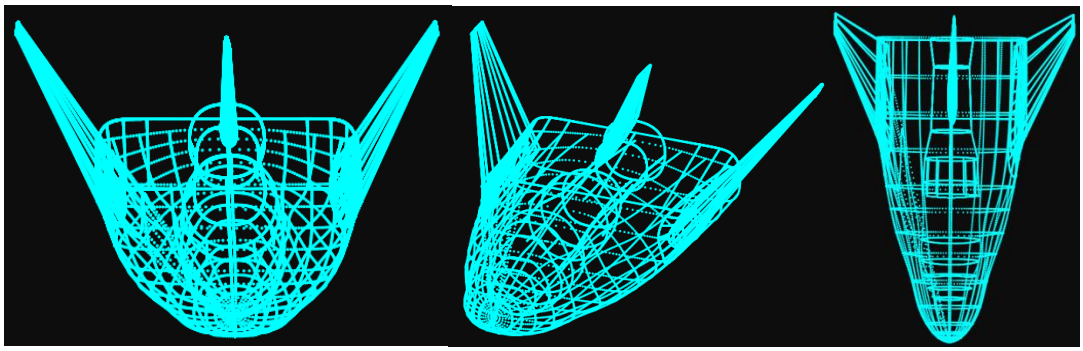
(a) SNC Dream Chaser control points



(b) SNC Dream Chaser knots



(c) SNC Dream Chaser gauss points



(d) SNC Dream Chaser Isogeometric elements

Figure 5.130 Mesh SNC Dream Chaser

The patch that we will choose to analyze is the right back wing of the model. We will use again the same procedure to generate the thickness on the third dimension. We are also going to apply two p-refinements on our model to check the effect of the basis functions' polynomial degree on the analysis results.

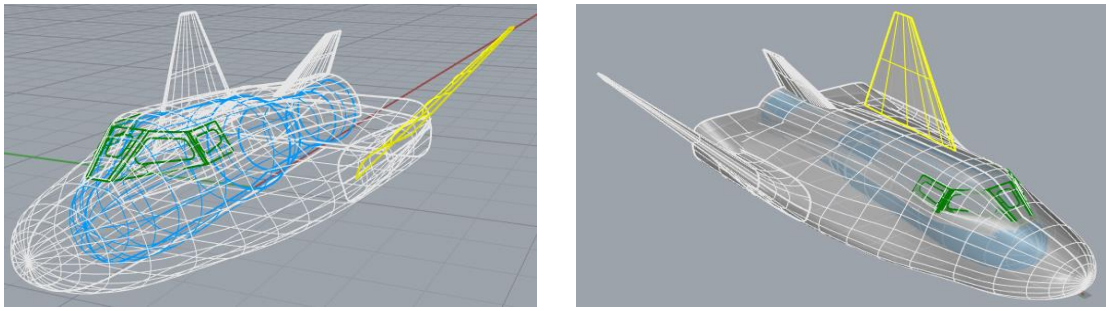


Figure 5.131 Analyzed patch from SNC Dream Chaser.

We will apply 6 external concentrated loads of 5 *kN* on the control points shown in figure 5.132. For reasons of simplicity I will choose to fix the control points that correspond to the connection of the wing and the main body of the spaceship.

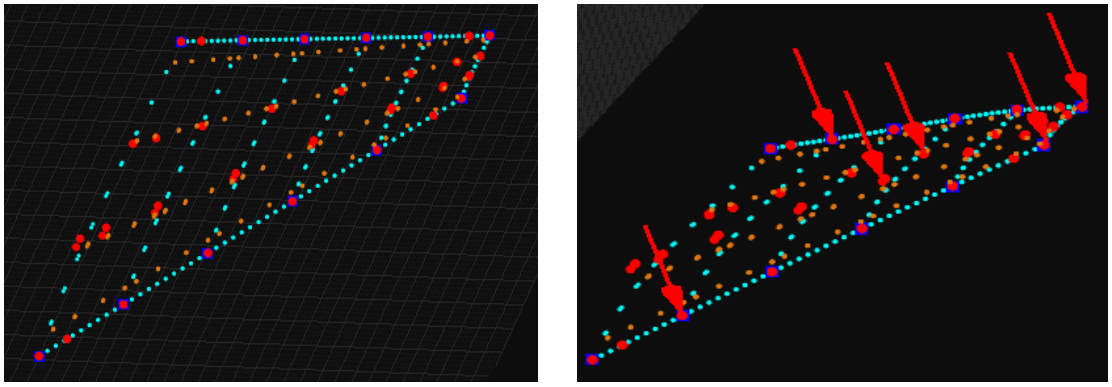


Figure 5.132 SNC Dream Chaser physical space.

From the pictures below we can understand that steel as material would be a logical choice for the simulation. Thickness was chosen to 0.006 m.

Figure 5.133 SNC Dream Chaser material.

(http://www.parabolicarc.com/2010/11/08/pictures-sierra-nevadas-dream-chaser-shuttle-development/)

The initial mesh consists of cubic basis functions on axes ξ,η and linear on axis ζ. There are $n=4$ control points on ξ, $m=8$ control points on η and $r=2$ control points on ζ for a total of 64 points and 192 degrees of freedom.

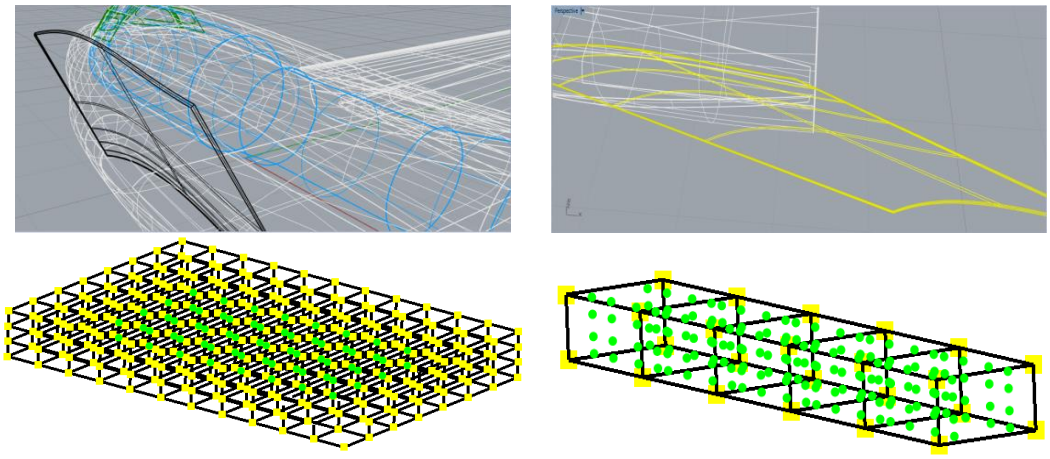


Figure 5.133 SNC Dream Chaser initial mesh, index, parameter space.

The second representation differentiates on the quadratic basis functions on axis ζ. There are $n=4$ control points on ξ, $m=8$ control points on η and $r=3$ control points on ζ for a total of 96 points and 288 degrees of freedom.

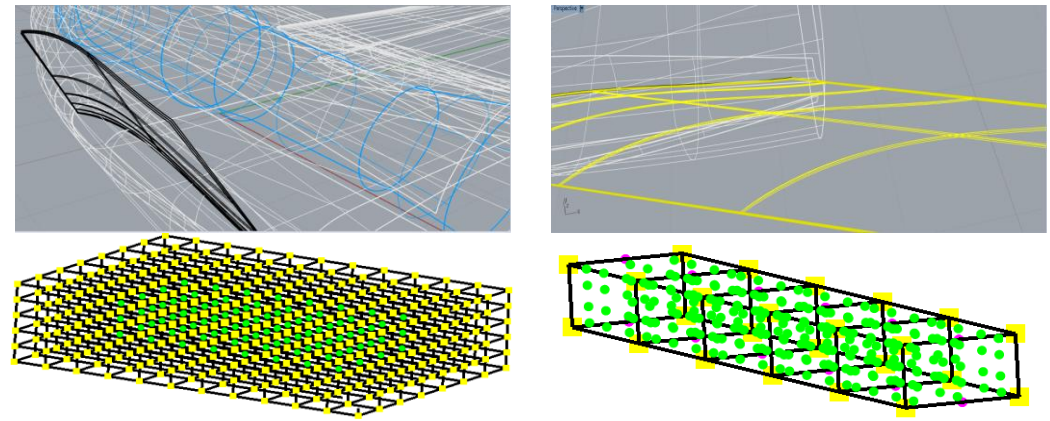


Figure 5.134 SNC Dream Chaser first p-Refinement, index, parameter space.

The second representation differentiates on the quadratic basis functions on axis ζ. There are $n=4$ control points on ξ, $m=8$ control points on η and $r=3$ control points on ζ for a total of 96 points and 288 degrees of freedom.

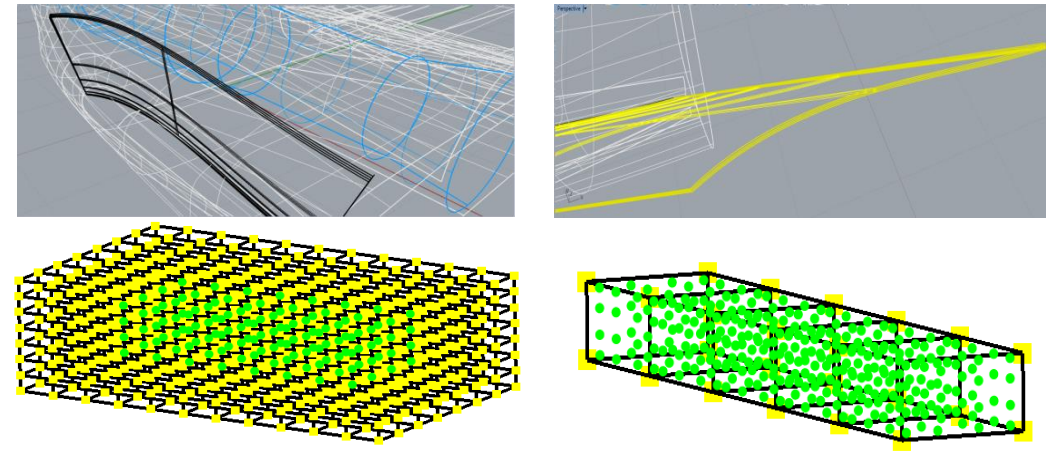
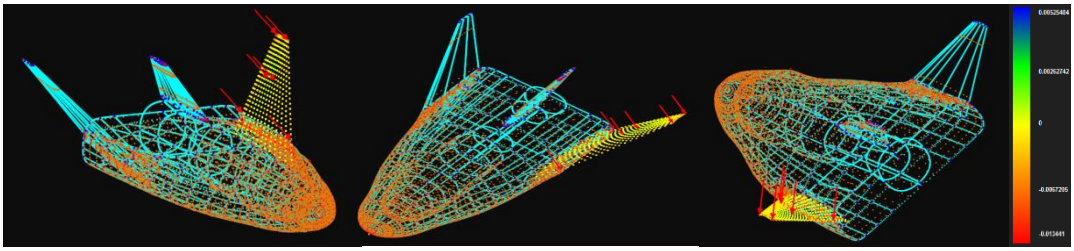
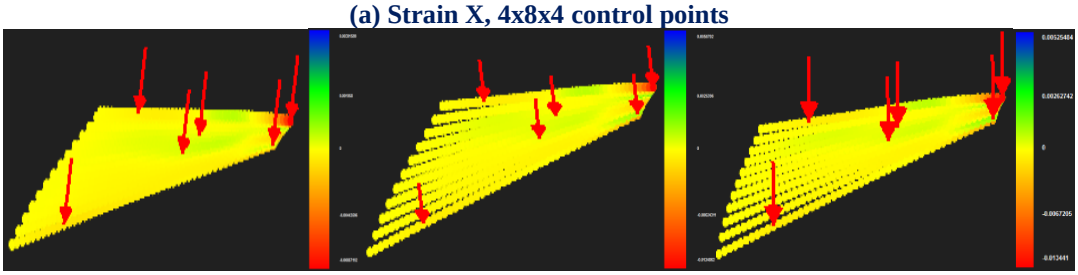


Figure 5.135 SNC Dream Chaser second p-Refinement, index, parameter space.

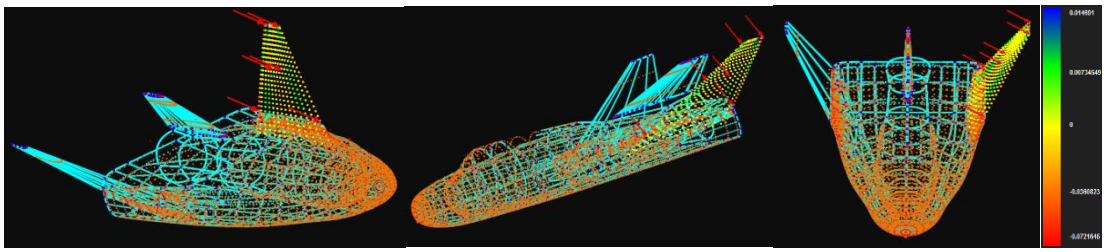


(a) Strain X, 4x8x4 control points

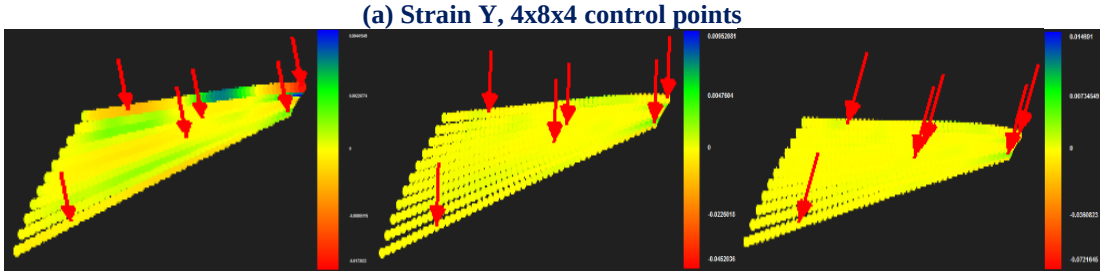


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.136 Strain X

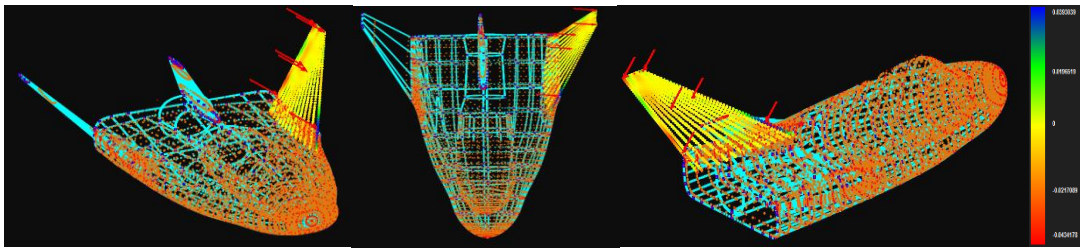


(a) Strain Y, 4x8x4 control points

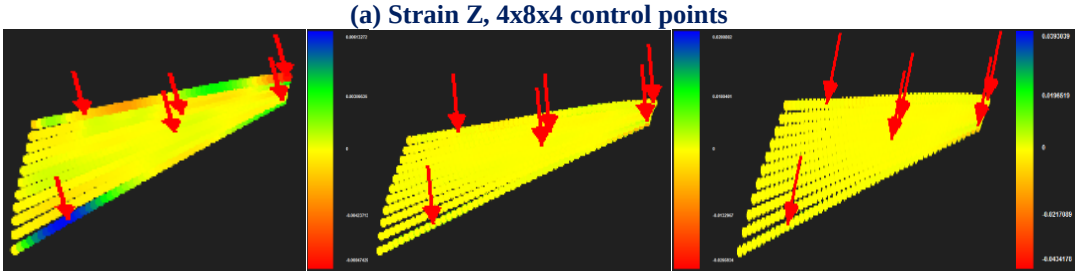


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.137 Strain Y

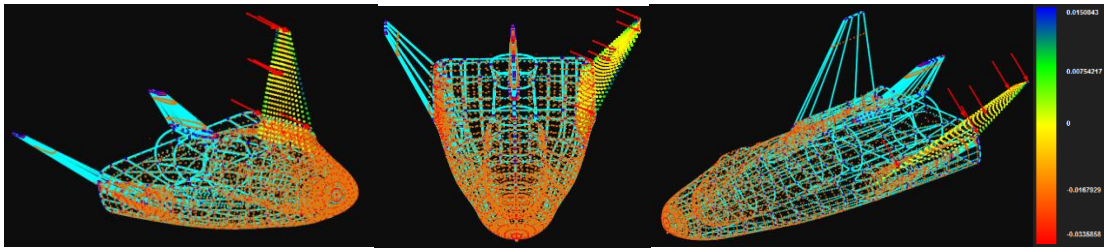


(a) Strain Z, 4x8x4 control points

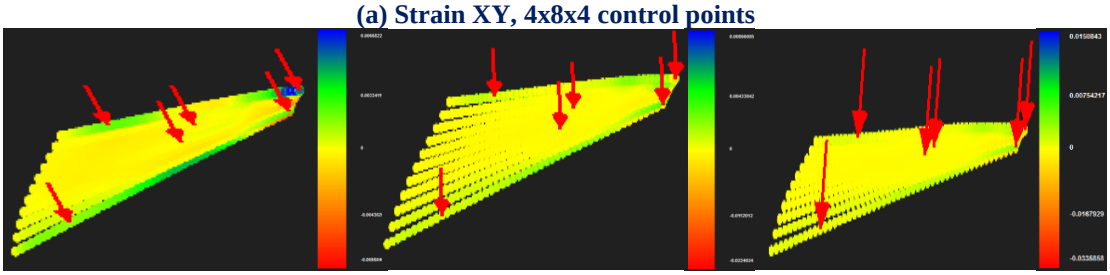


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.138 Strain Z

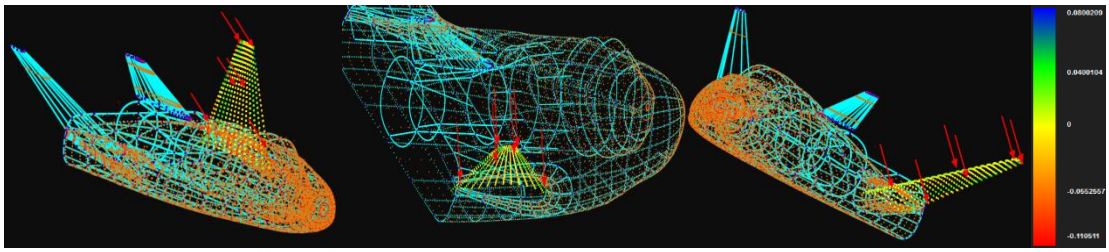


(a) Strain XY, 4x8x4 control points

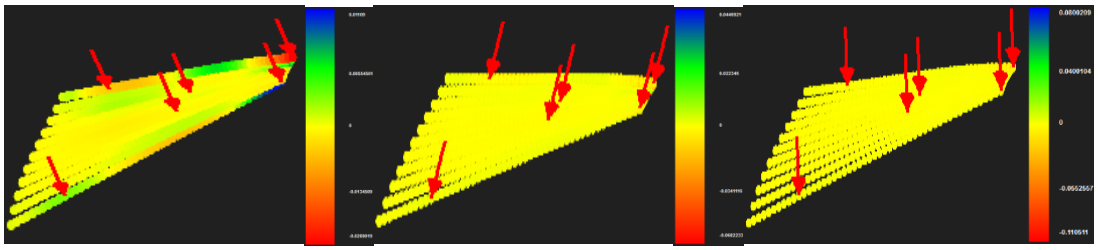


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.139 Strain XY

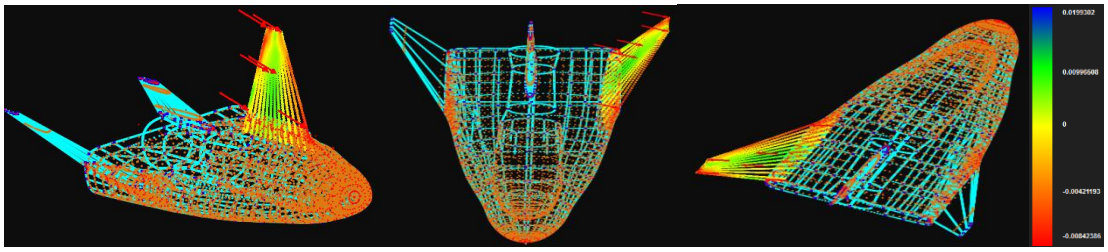


(a) Strain YZ, 4x8x4 control points

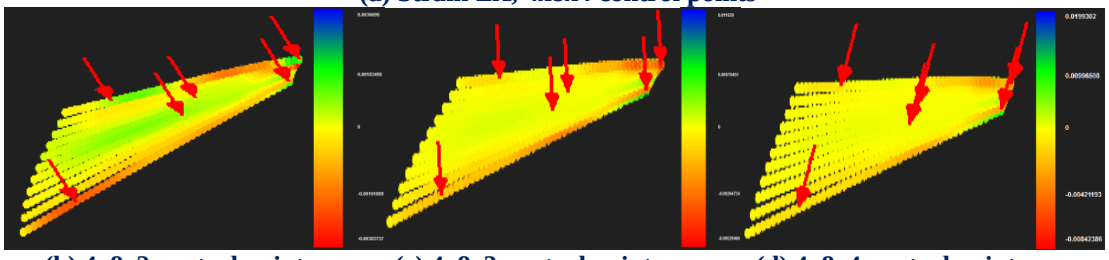


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.140 Strain YZ

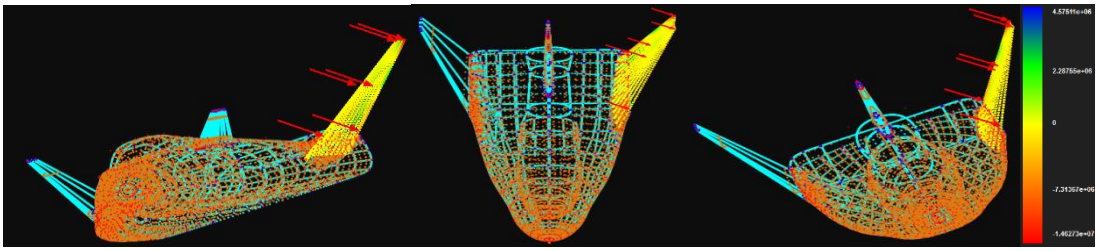


(a) Strain ZX, 4x8x4 control points

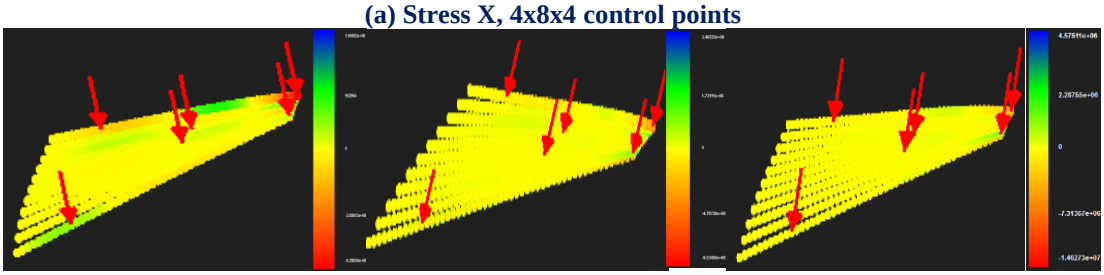


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.141 Strain ZX

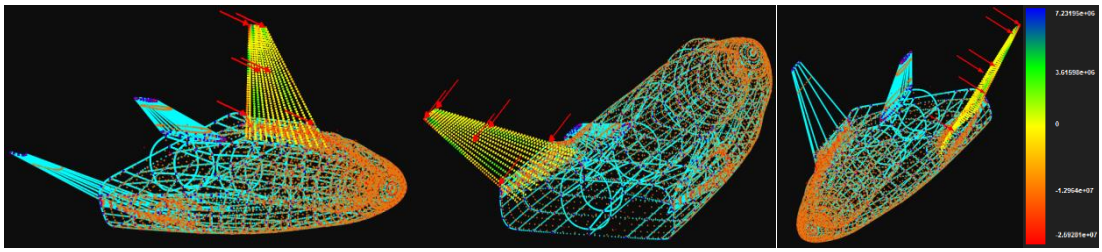


(a) Stress X, 4x8x4 control points

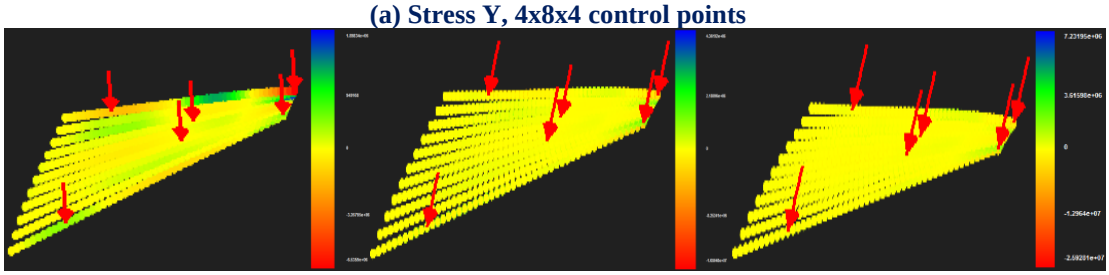


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.142 Stress X

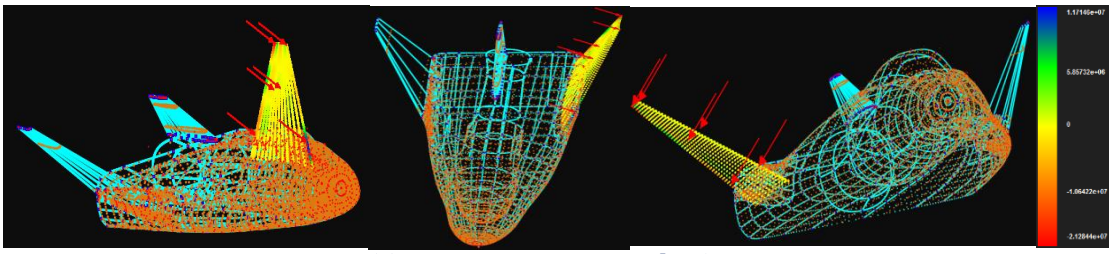


(a) Stress Y, 4x8x4 control points

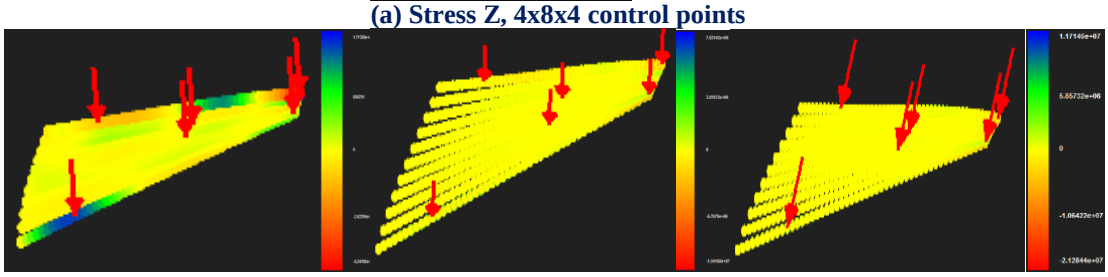


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.143 Stress Y

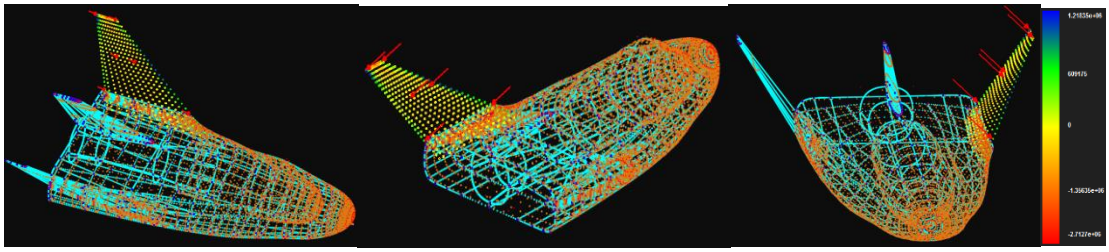


(a) Stress Z, 4x8x4 control points

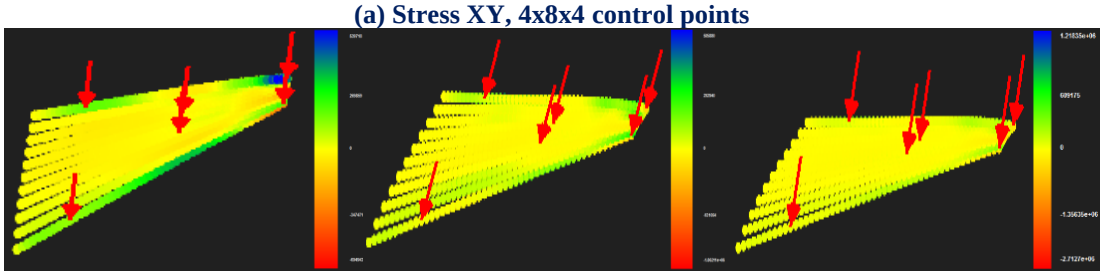


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.144 Stress Z

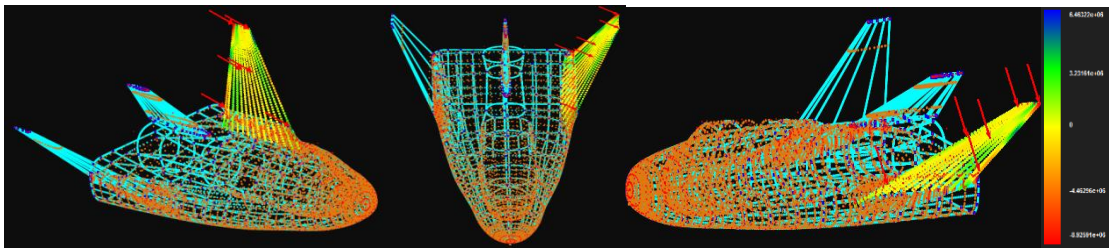


(a) Stress XY, 4x8x4 control points

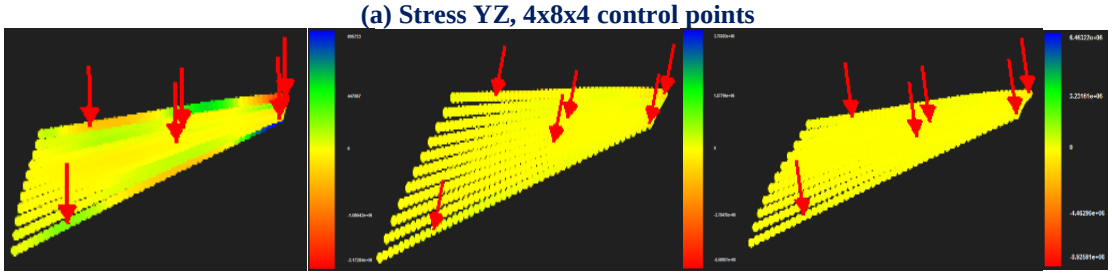


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.145 Stress XY

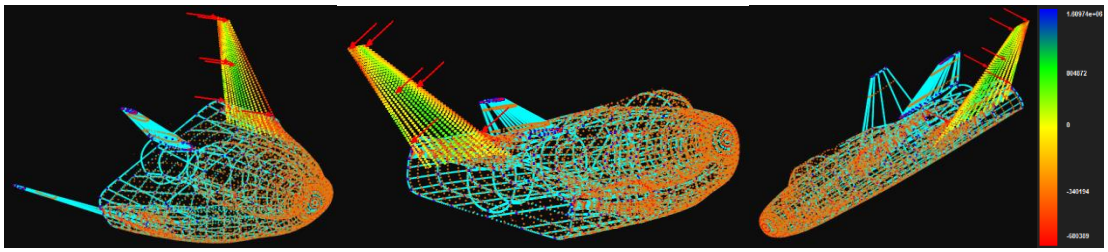


(a) Stress YZ, 4x8x4 control points

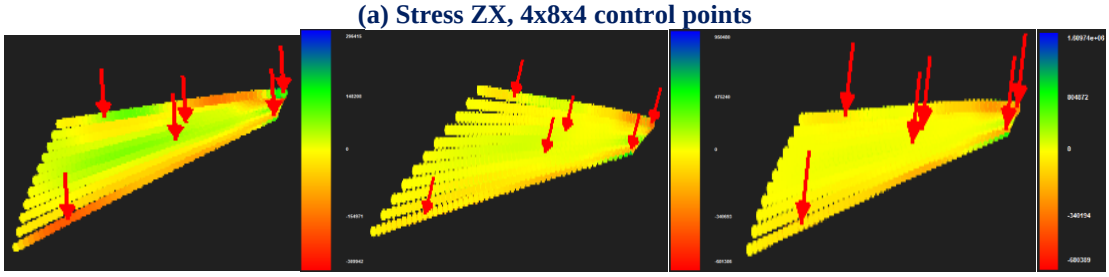


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.146 Stress YZ

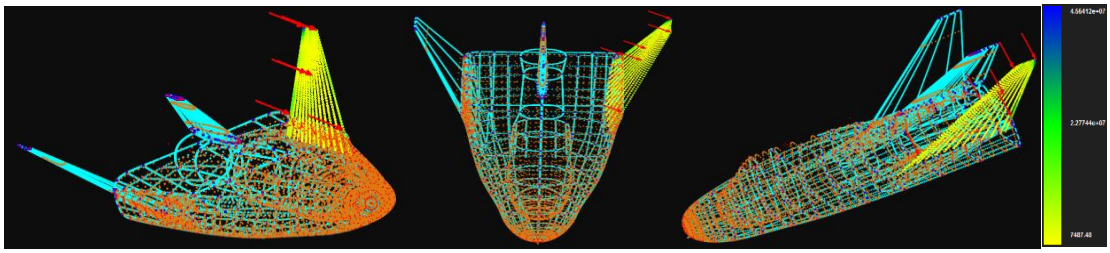


(a) Stress ZX, 4x8x4 control points

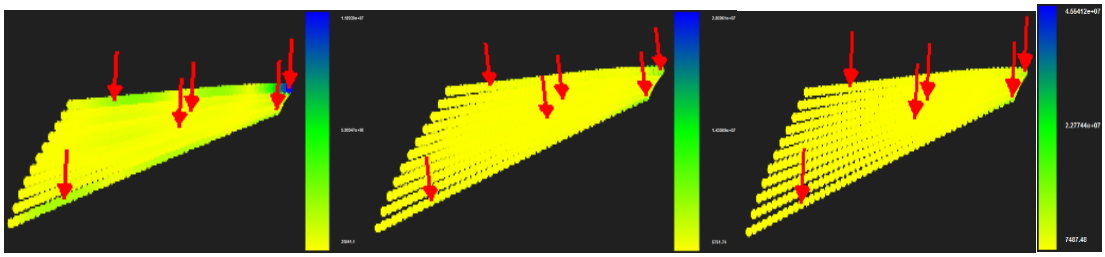


(b) 4x8x2 control points (c) 4x8x3 control points (d) 4x8x4 control points

Figure 5.147 Stress ZX



(a) Stress von Mises, 4x8x4 control points



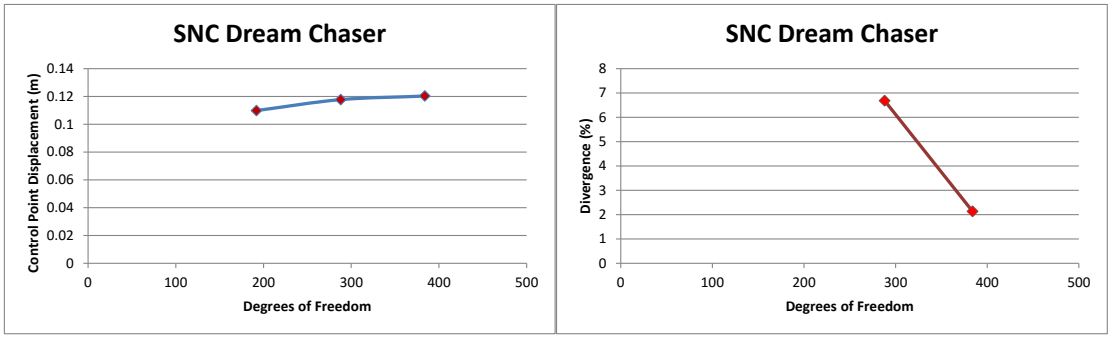
(b) 4x8x2 control points **(c) 4x8x3 control points** **(d) 4x8x4 control points**

Figure 5.148 Stress von Mises

We can see on the above pictures that as parameterization becomes more detailed the greater the difference in contour representation becomes. In figure 5.149 we will ascertain the influence of thickness parameterization in control point displacement, norm displacement and von Mises Stresses.

Control Point Displacement (m):

- Degrees of Freedom (198) → 0.109817 m
- Degrees of Freedom (288) → 0.117677 m → Deviation (6.68%)
- Degrees of Freedom (384) → 0.120239 m → Deviation (2.13%)



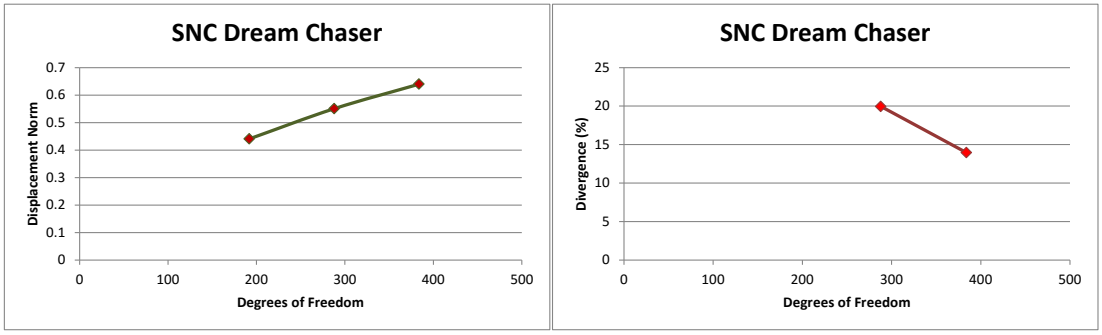
(a) **(b)**

Figure 5.149 Control point displacement results.
(a) Control point displacement in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

As we can see from the results there is no remarkable difference in maximum control point displacement even though it is noticeable that the deviation from the previous solution gets decreased while the degree of the polynomial basis function of the thickness is increased.

Displacement Norm (m):

- Degrees of Freedom (198) → 0.440764 m
- Degrees of Freedom (288) → 0.550663 m → Deviation (19.96%)
- Degrees of Freedom (384) → 0.639944 m → Deviation (13.95%)



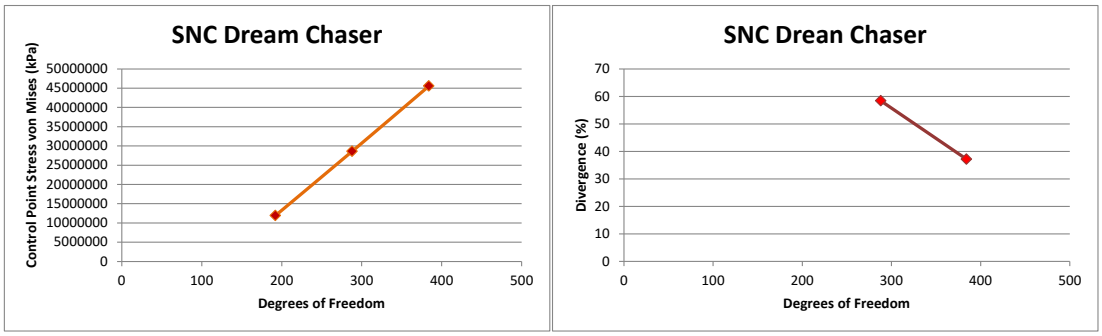
(a) (b)

Figure 5.150 Displacement norm results.
(a) Displacement norm in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

Norm Displacement presents the same behavior as control point displacement diagram does.

Control Point Stress von Mises (kPa):

- Degrees of Freedom (198) → 11893900 kPa
- Degrees of Freedom (288) → 28596100 kPa → Deviation (58.41%)
- Degrees of Freedom (384) → 45541200 kPa → Deviation (37.21%)



(a) (b)

Figure 5.151 Stress von Mises results.
(a) Stress von Mises in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

Unlike previous diagrams, even though deviation from previous von Mises stress is decreased with the applied p-refinements, we are still very far from a final convergence. It would present great interest to observe the behavior of the stress field by using shell elements which as we know are more appropriate for stress analysis results.

5.10 SST Aircraft

A supersonic transport (SST) is a civilian supersonic aircraft designed to transport passengers at speeds greater than the speed of sound. To date, the only SSTs to see regular service have been Concorde and the Tupolev Tu-144. The last passenger flight for the Tu-144 was in June 1978 and it was last flown in 1999 by NASA. Concorde's last commercial flight was in October 2003, with a November 26, 2003 ferry flight being its last airborne operation. Following the permanent cessation of flying by Concorde, there are no remaining SSTs in commercial service.

Figure 5.152 Concorde
(http://imagestack.co/70416911-aircraft-oil-paintings.html)

Supersonic airliners have been the objects of numerous recent and ongoing design studies. Drawbacks and design challenges are excessive noise generation (at takeoff and due to sonic booms during flight), high development costs, expensive construction materials, great weight, and an increased cost per seat over subsonic airliners. Despite these challenges, Concorde was operated profitably in a niche market for over 27 years.

Figure 5.153 Supersonic aircrafts

(http://lab4me8.wix.com/lab4x-air#!robert-l-trout/cqw8)
(http://janio.sarmento.org/365posts-concorde.html)

The desire for a second-generation supersonic aircraft has remained within some elements of the aviation industry, and several concepts emerged quickly following the retirement of Concorde.

Figure 5.154 Second generation supersonic aircrafts.
(http://deredactie.be/cm/vrtnieuws/wetenschap/1.2467729)
(http://stranenotizie.it/dal-mondo/da-londra-a-new-york-in-3-ore-in-aereo-il-tempo-di-un-treno-da-palermo-a-messina/)

In November 2003, EADS (the parent company of Airbus) announced that it was considering working with Japanese companies to develop a larger, faster replacement for Concorde. In October 2005, JAXA, the Japan Aerospace eXploration Agency, undertook aerodynamic testing of a scale model of an airliner designed to carry 300 passengers at Mach 2 (working name *NEXST*). If pursued to commercial deployment, it would be expected to be in service around 2020–25.

Figure 5.155 SAI quiet supersonic aircraft.
(http://www.desktopwallpapers4.me/aircraft/sai-quiet-supersonic-transport-9001/)

In May 2008, it was reported that Aerion Corporation had $3 billion of pre-order sales on its Aerion SBJ supersonic business jet. In late 2010, the project continued with a testbed flight of a section of the wing.

In the 21st century some supersonic airliners and business jets (Aerion SBJ, HyperMach SonicStar, Next Generation Supersonic Transport, Tupolev Tu-444, Gulfstream X-54, LAPCAT, Reaction Engines A2, Spike S-512, Zero Emission Hyper Sonic Transport) were under development.

The final application of this thesis is set to demonstrate the remarkable potential of Isogeometric Analysis. The real revolution of Isogeometric Analysis lies in the ability of analyzing any CAD design of arbitrarily complex geometry. The engineer doesn't have to interfere in the design procedure at all. With this software all he has to do is simply load the archive and begin his analyzation. The engineer will only focus on determining the individual patch that he wants to analyze, define the applied loads on the desired positions and decide the kind of boundary conditions he wants to use.

The CAD model that is going to be analyzed corresponds to the real model of figure 6.156 below.

Figure 5.156 QSST supersonic aircraft.
(http://www.icas.org/ICAS_ARCHIVE/ICAS2014/data/papers/2014_0228_paper.pdf)

The CAD model is shown in figure 6.80, as with the desired patch that we will analyze. The wing consists of two patches. For obvious reasons of simplicity I will analyze the top part of the wing as a single and individual patch ignoring the beneath part.

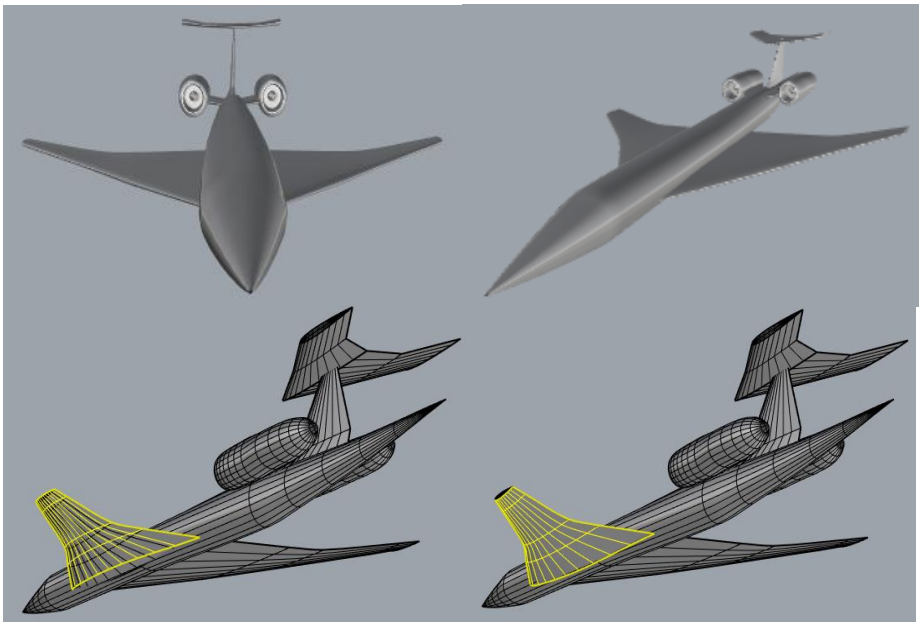
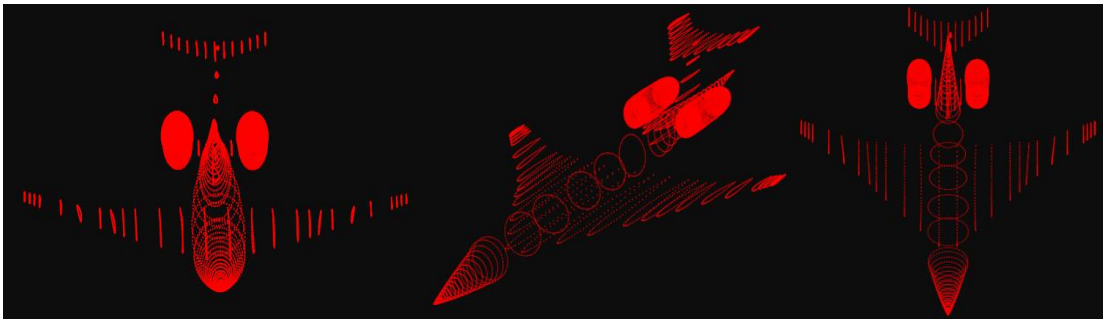
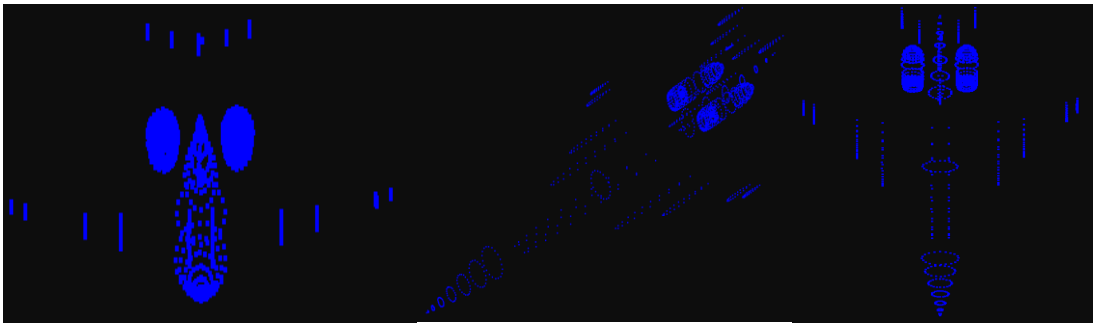


Figure 5.157 CAD model & analyzed patch.
(https://grabcad.com/library/sst-1)

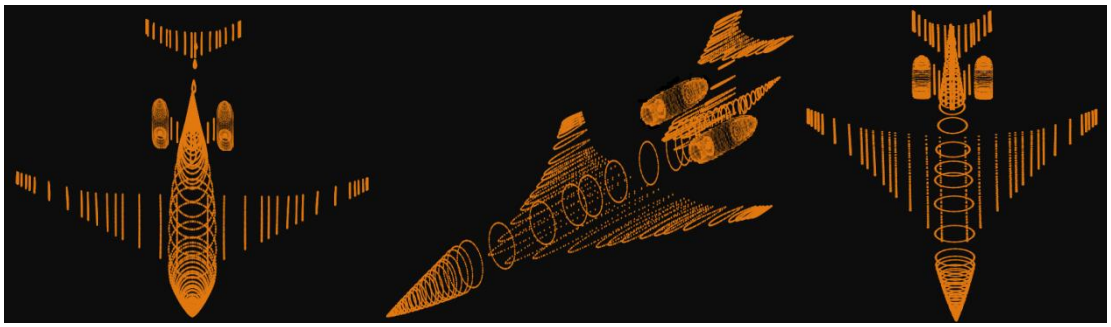
After our model has been loaded to the software all the encrypted information can be seen above.



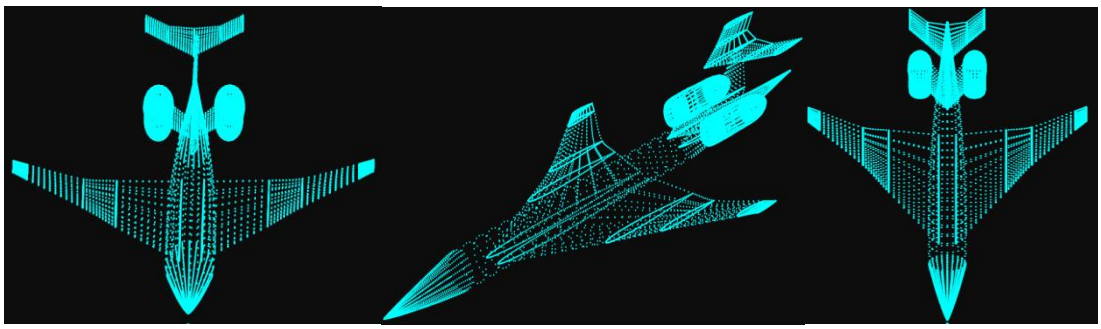
(a) SST aircraft control points



(b) SST aircraft knots



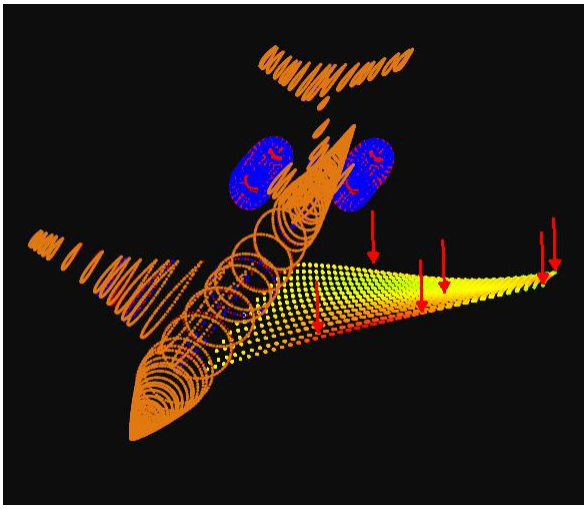
(c) SST aircraft gauss points



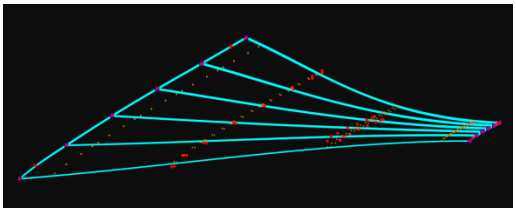
(d) SST aircraft Isogeometric elements

Figure 5.158 Mesh representation.

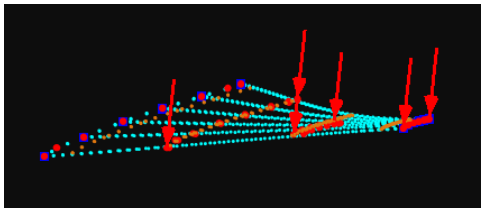
We will apply 6 external concentrated loads of 5 *kN* on the control points shown in figure 6.159. For reasons of simplicity I will choose to fix the control points that correspond to the connection of the wing and the main body of the spaceship.



(a) Physical space, gauss points, control points, knots



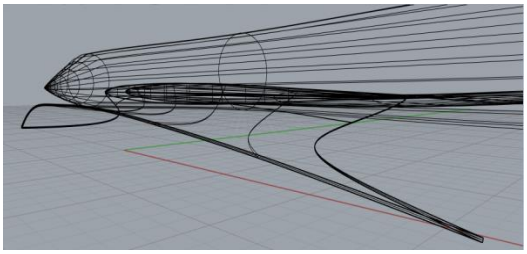
(b) Wing physical space



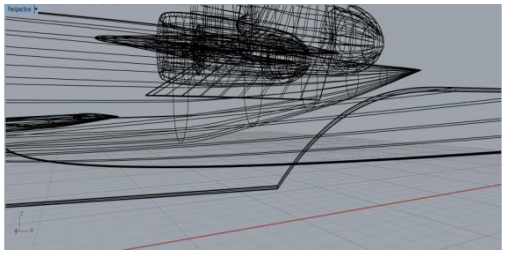
(c) Wing physical space deformed

We will use again the same procedure to generate the thickness on third dimension. We are also going to apply two p-refinements on our model to check the effect of the basis functions' polynomial degree on the analysis results.

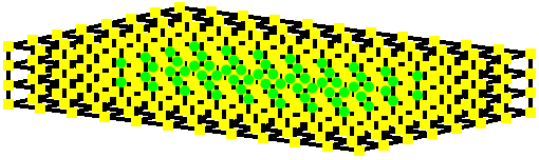
The first representation consists of cubic basis functions on axes ξ, η and linear on axis ζ. There are $n = 4$ control points on ξ, $m = 8$ control points on η and $r = 2$ control points on ζ for a total of 64 points and 192 degrees of freedom.



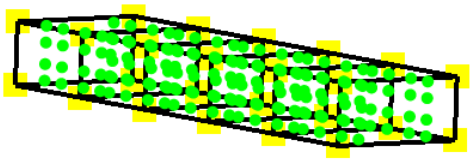
(d) Wing defined by two surfaces



(e) Wing defined by two surfaces



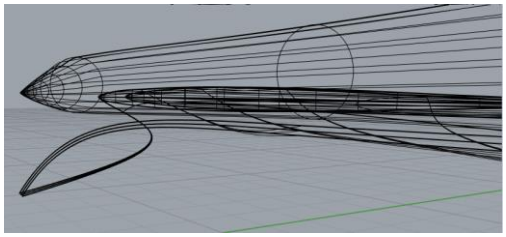
(f) Index space



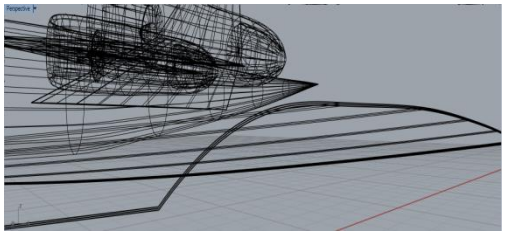
(g) Parameter space

Figure 5.159 Mesh representation, 4x8x2 control points.

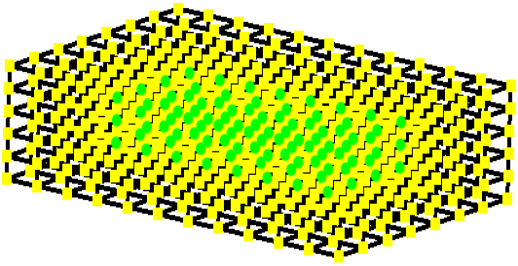
The second representation differentiates on the quadratic basis functions on axis ζ. There are $n=4$ control points on ξ, $m=8$ control points on η and $r=3$ control points on ζ for a total of 96 points and 288 degrees of freedom.



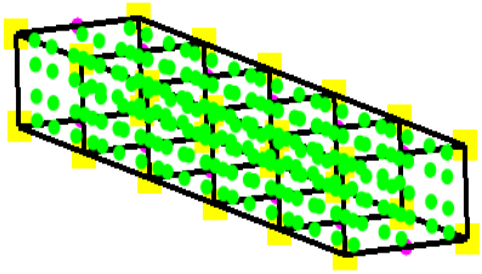
(a) Wing defined by three surfaces



(b) Wing defined by three surfaces



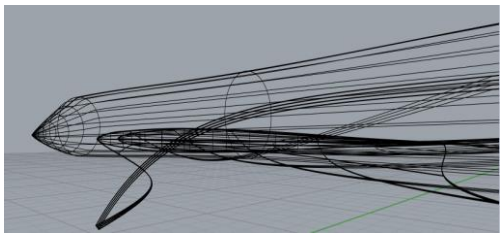
(c) Index space



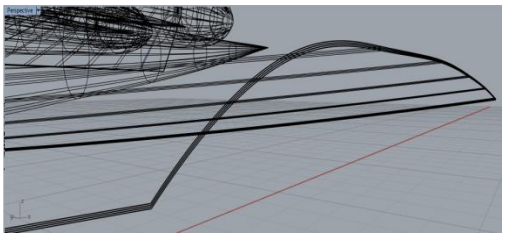
(d) Parameter space

Figure 5.160 Mesh representation, 4x8x3 control points.

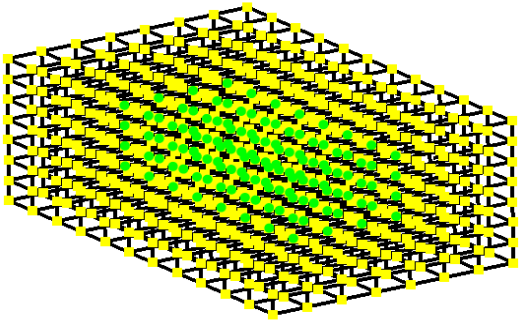
The second representation differentiates on the quadratic basis functions on axis ζ. There are $n=4$ control points on ξ, $m=8$ control points on η and $r=3$ control points on ζ for a total of 96 points and 288 degrees of freedom.



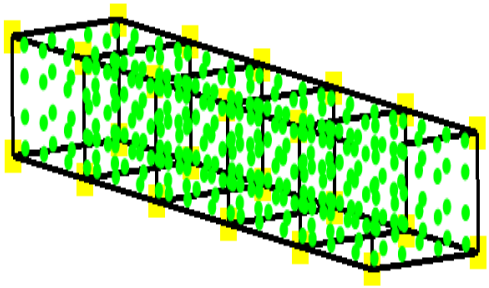
(a) Wing defined by four surfaces



(b) Wing defined by four surfaces



(c) Index space

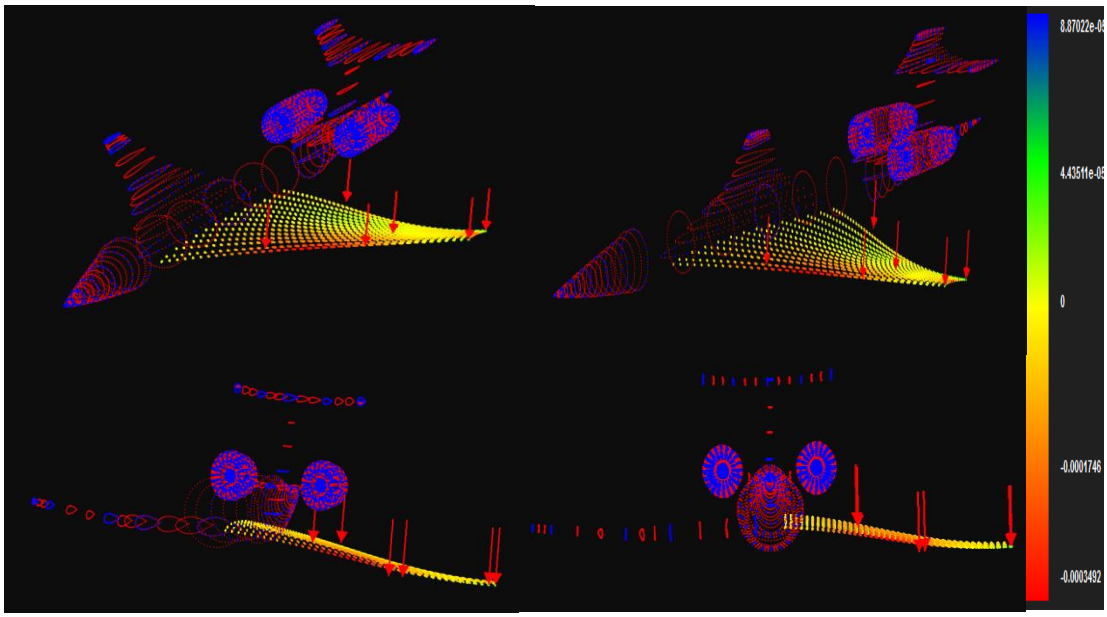


(d) Parameter space

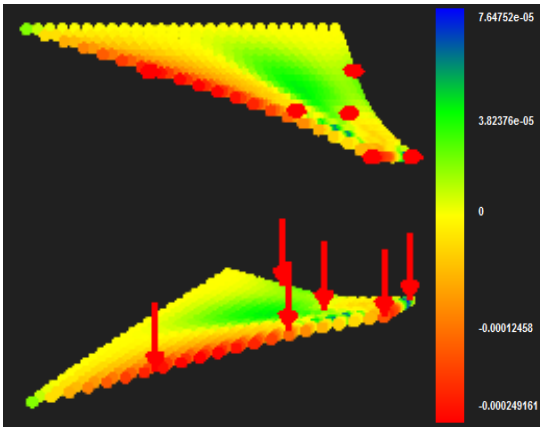
Figure 5.161 Mesh representation, 4x8x4 control points.

Steel has been chosen as material. The analysis results are shown below. In order to have a better sense, the control points and knots of the entire model are shown with the contours

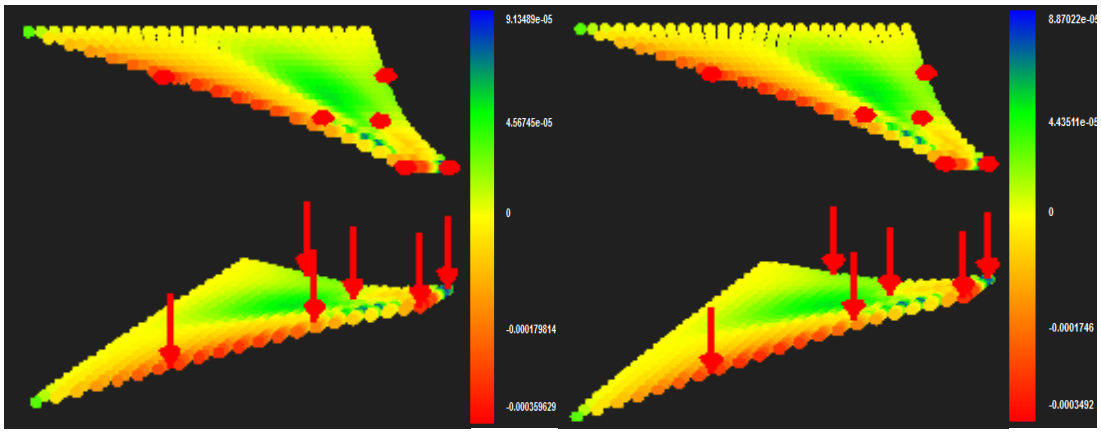
In figure 5.162 contours for Strain X Deformed are presented.



(a) SST aircraft strain X deformed (4x8x4 control points)



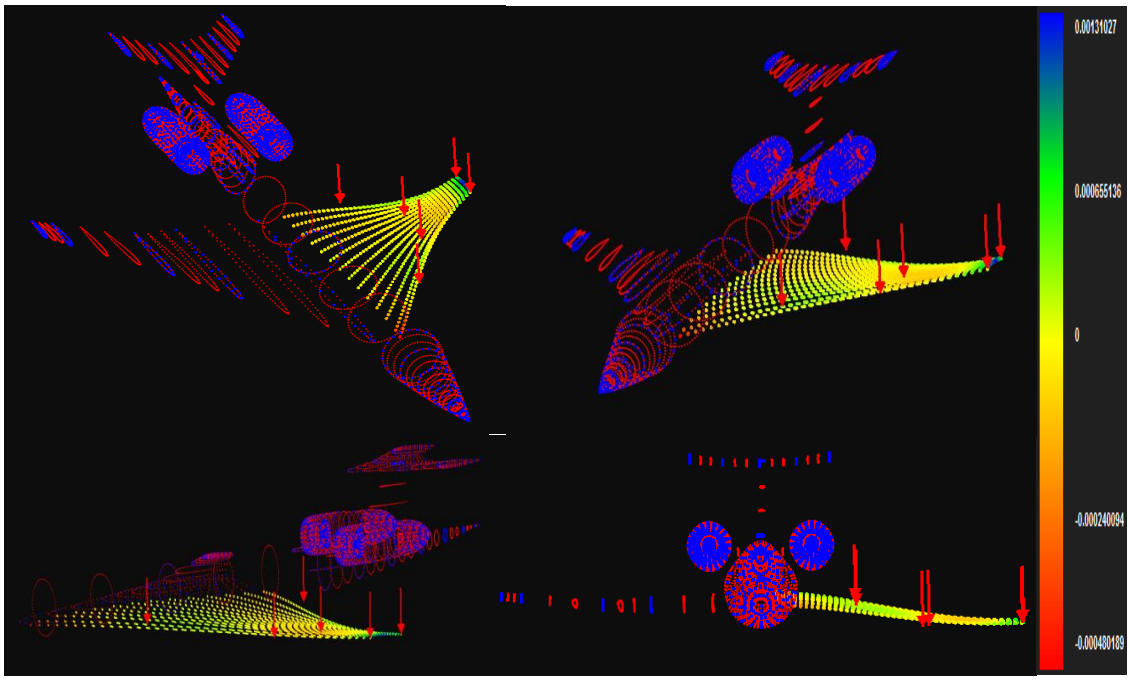
(b) 4x8x2 control points



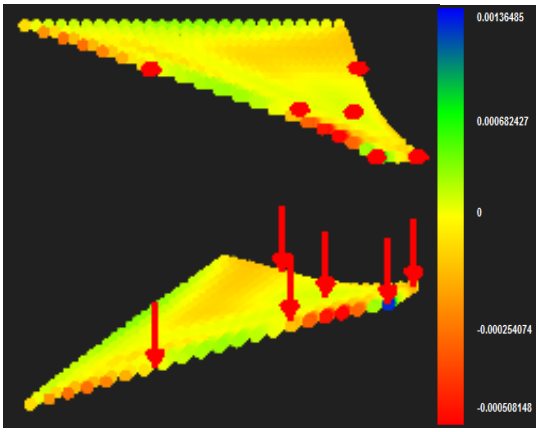
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.162 Strain X

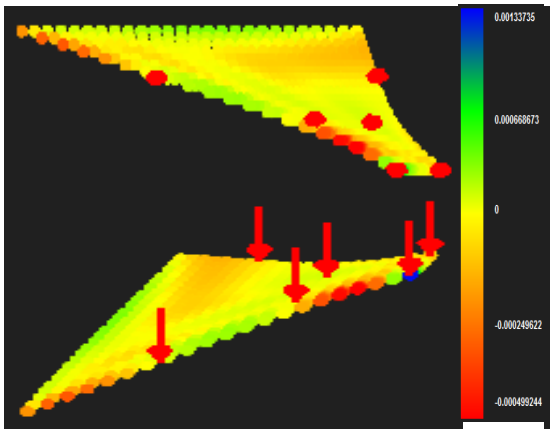
In figure 5.163 contours for Strain Y Deformed are presented.



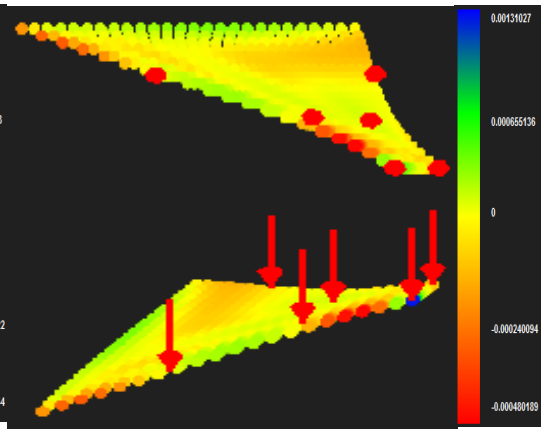
(a) SST aircraft Strain Y deformed (4x8x4 control points)



(b) 4x8x2 control points



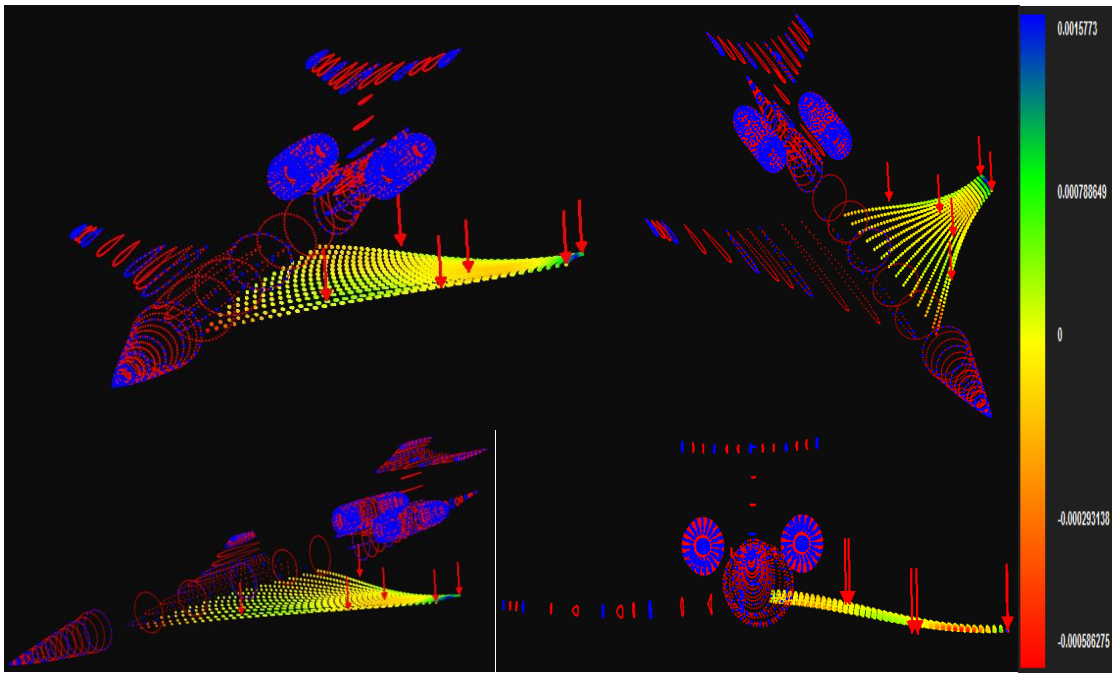
(c) 4x8x3 control points



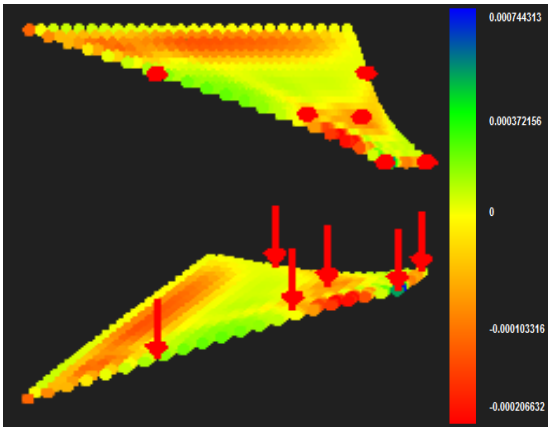
(d) 4x8x4 control points

Figure 5.163 Strain Y

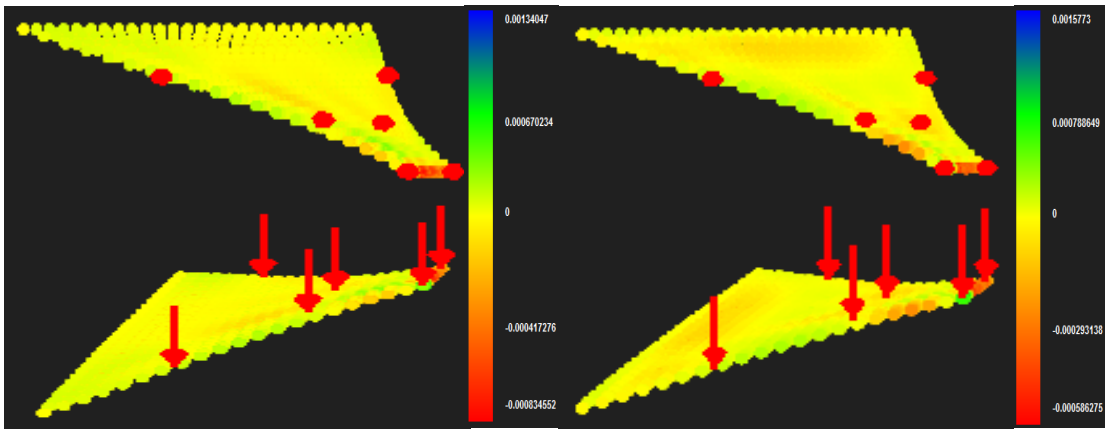
In figure 5.164 contours for Strain Z Deformed are presented.



(a) SST aircraft strain Z deformed (4x8x4 control points)



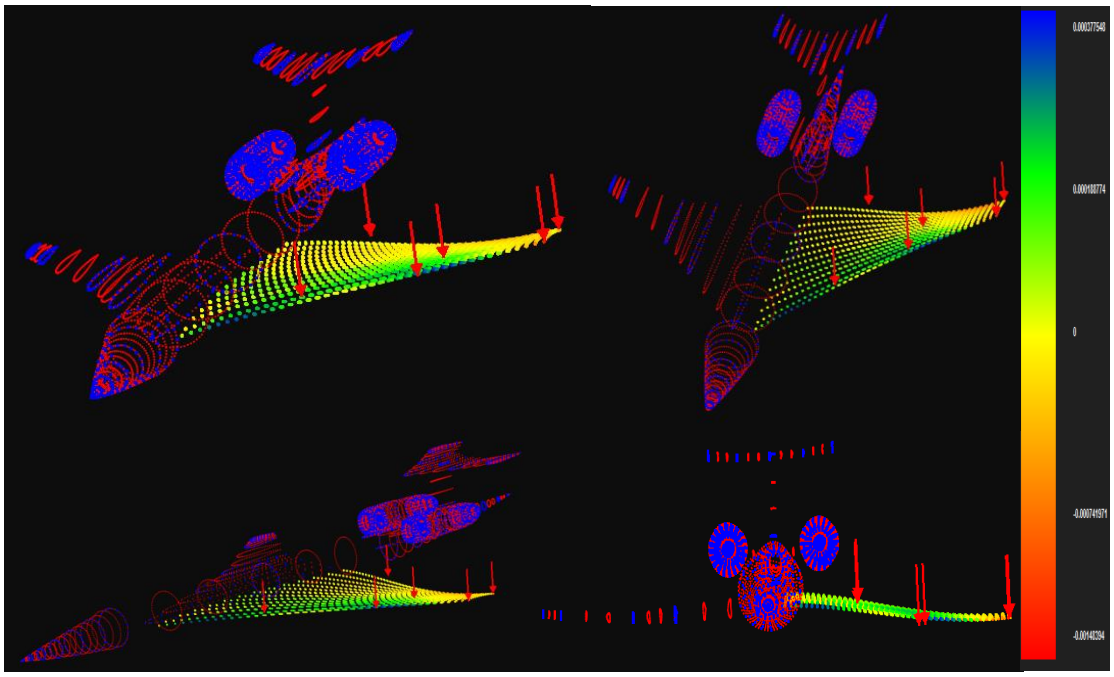
(b) 4x8x2 control points



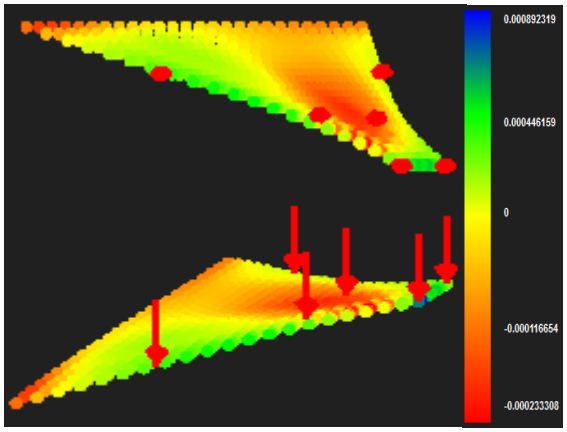
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.164 Strain Z

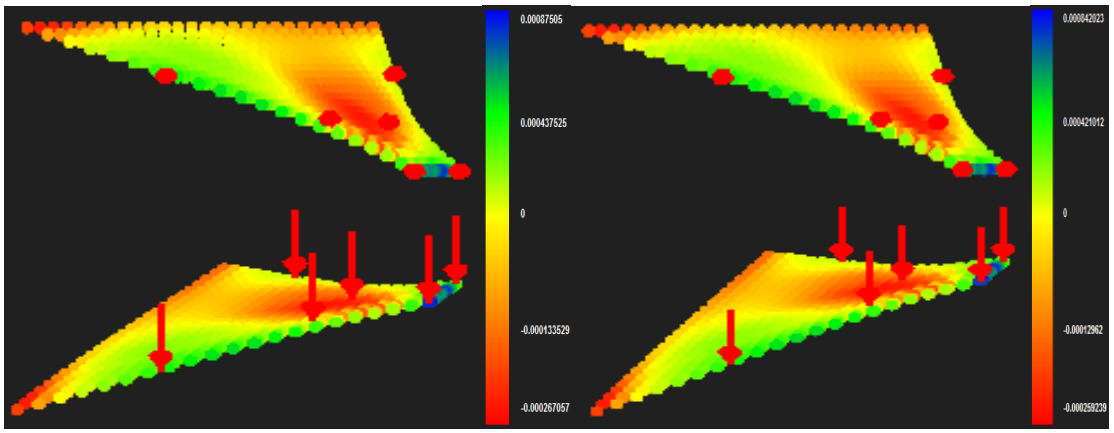
In figure 5.165 contours for Strain XY Deformed are presented.



(a) SST aircraft strain XY deformed (4x8x4 control points)



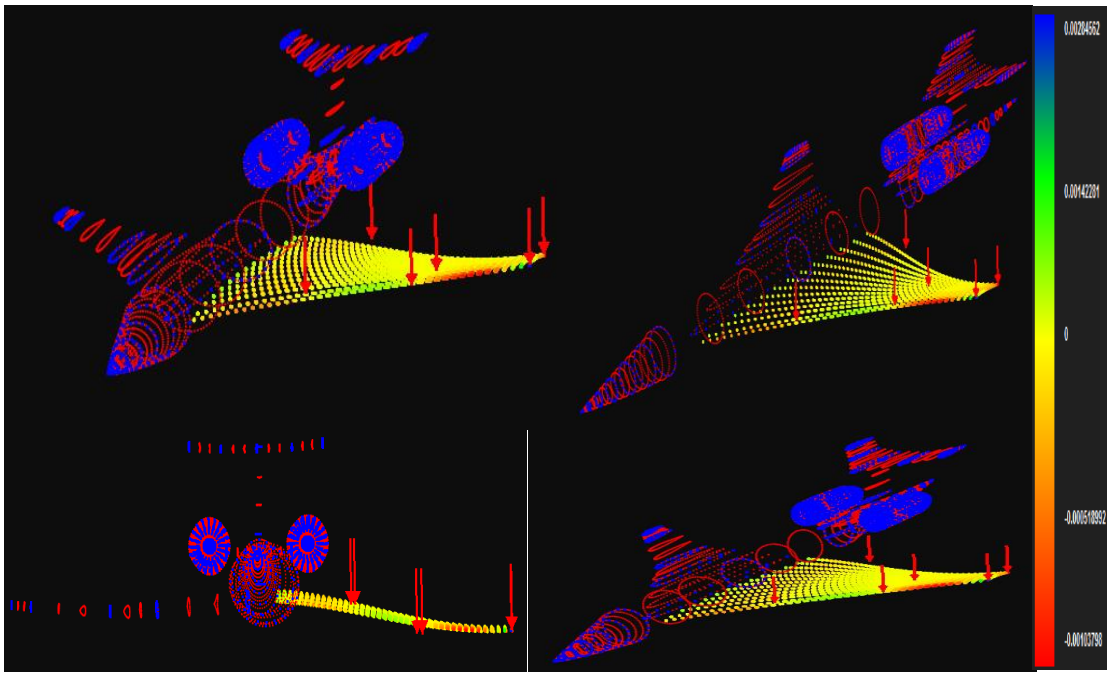
(b) 4x8x2 control points



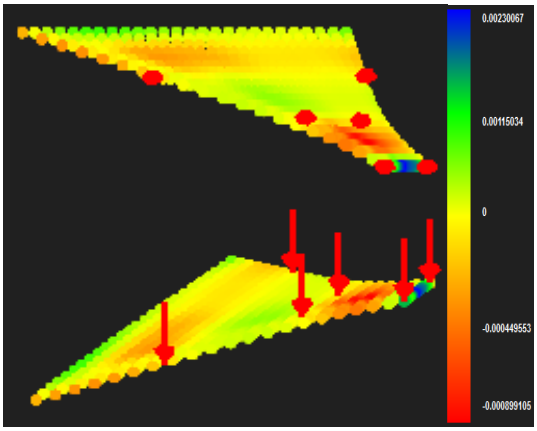
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.165 Strain XY

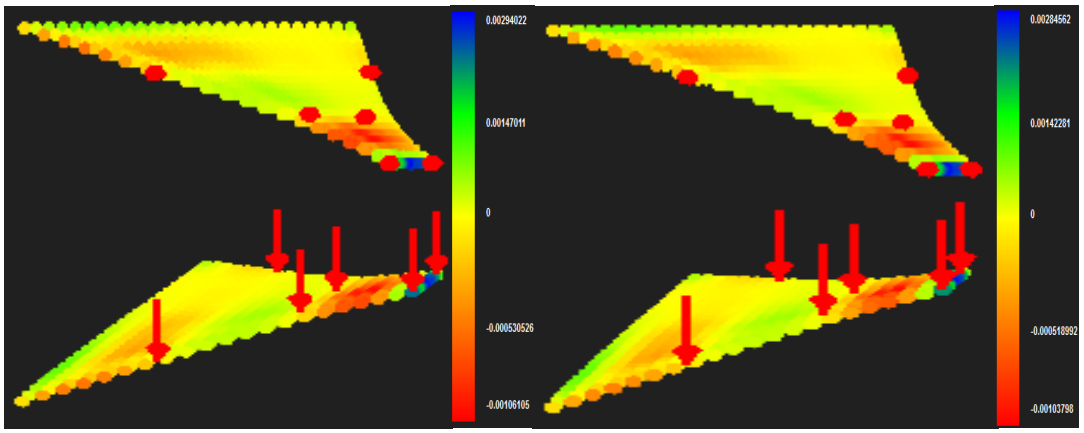
In figure 5.166 contours for Strain YZ Deformed are presented.



(a) SST aircraft strain YZ deformed (4x8x4 control points)



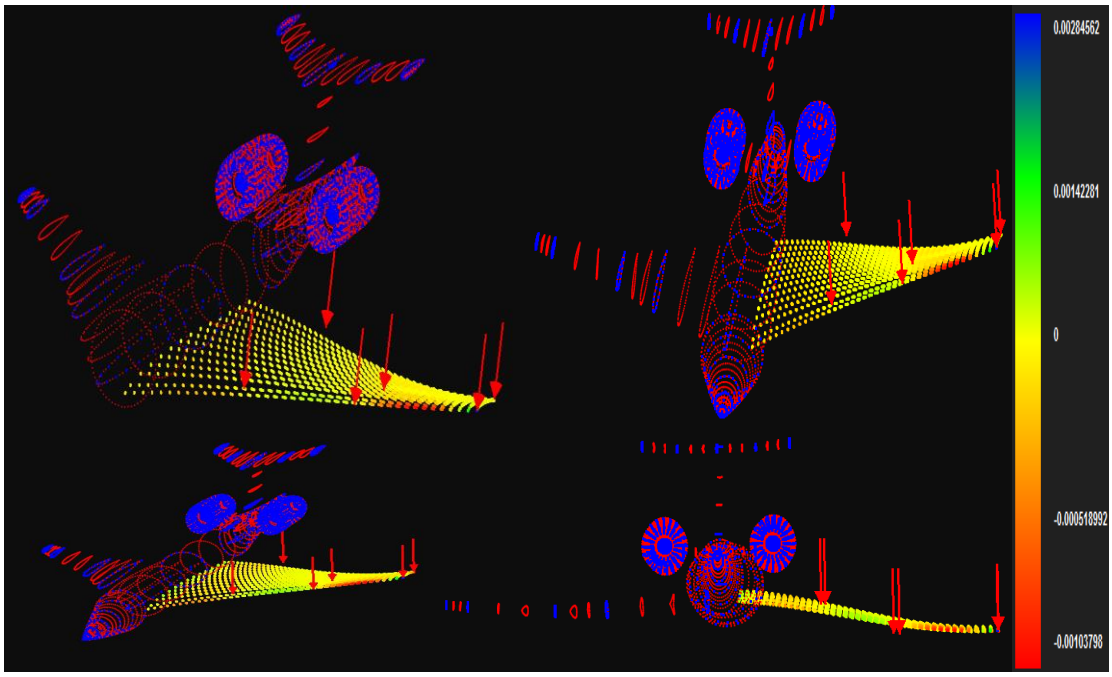
(b) 4x8x2 control points



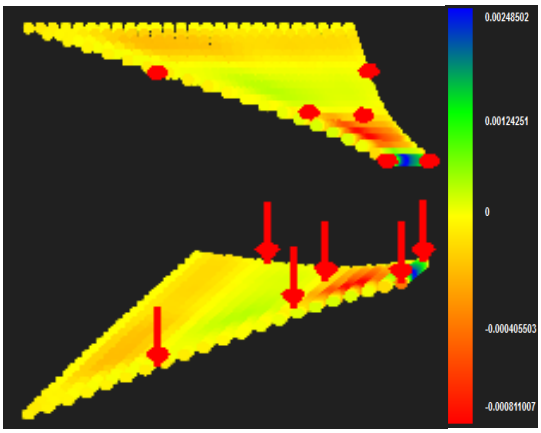
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.166 Strain YZ

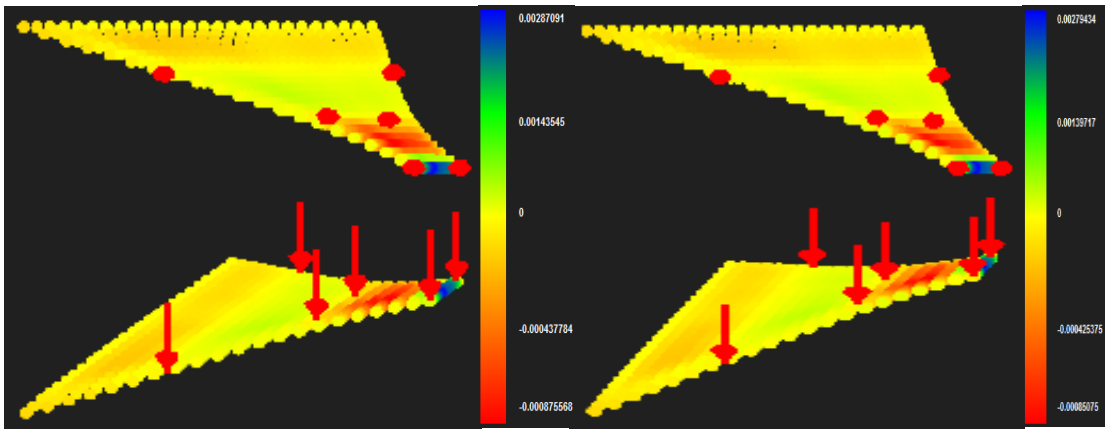
In figure 5.167 contours for Strain ZX Deformed are presented.



(a) SST aircraft strain ZX deformed (4x8x4 control points)



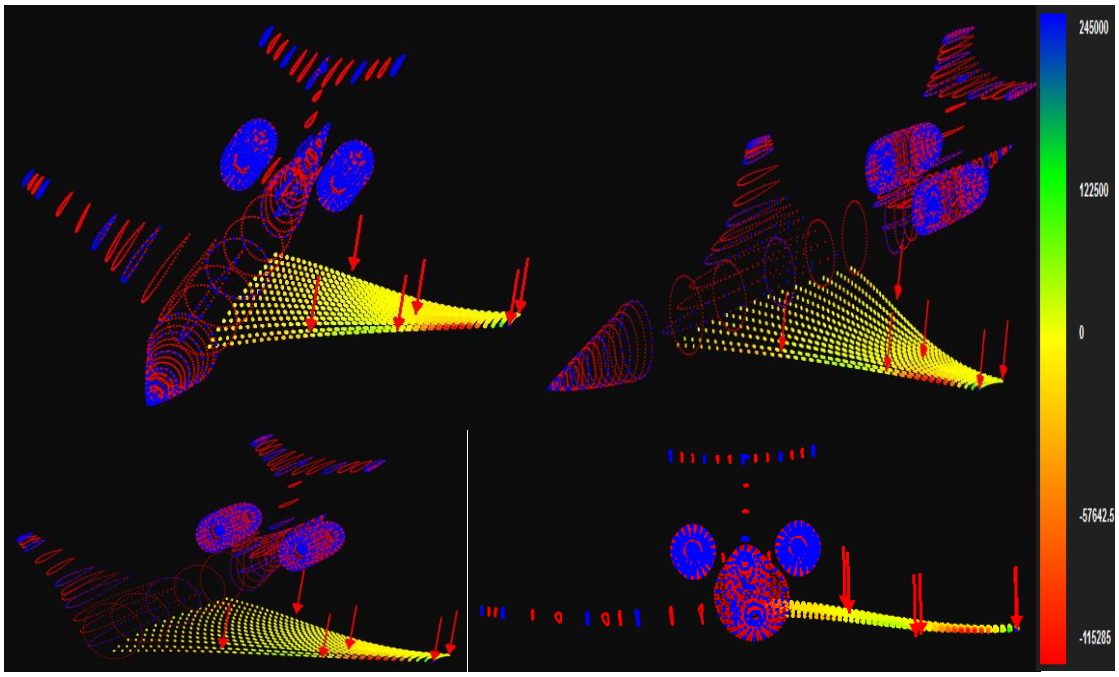
(b) 4x8x2 control points



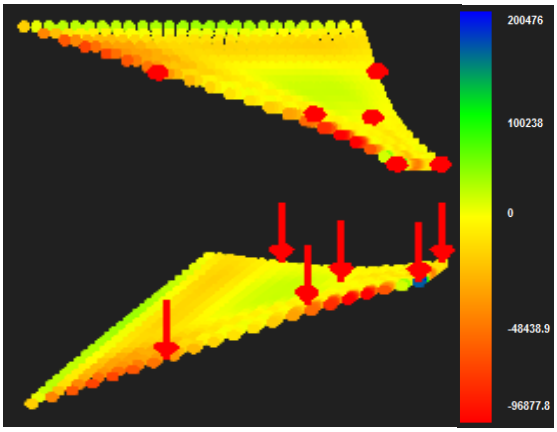
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.167 Strain ZX

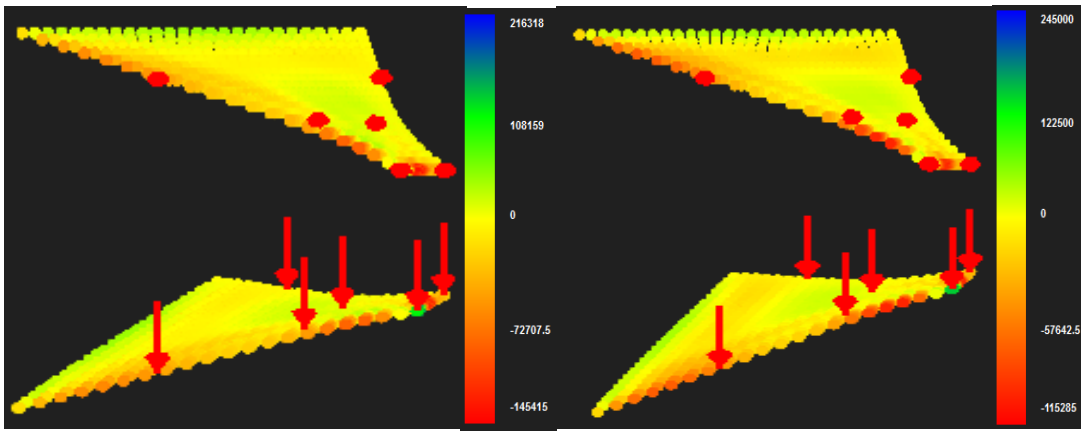
In figure 5.168 contours for Stress X Deformed are presented.



(a) SST aircraft stress X deformed (4x8x4 control points)



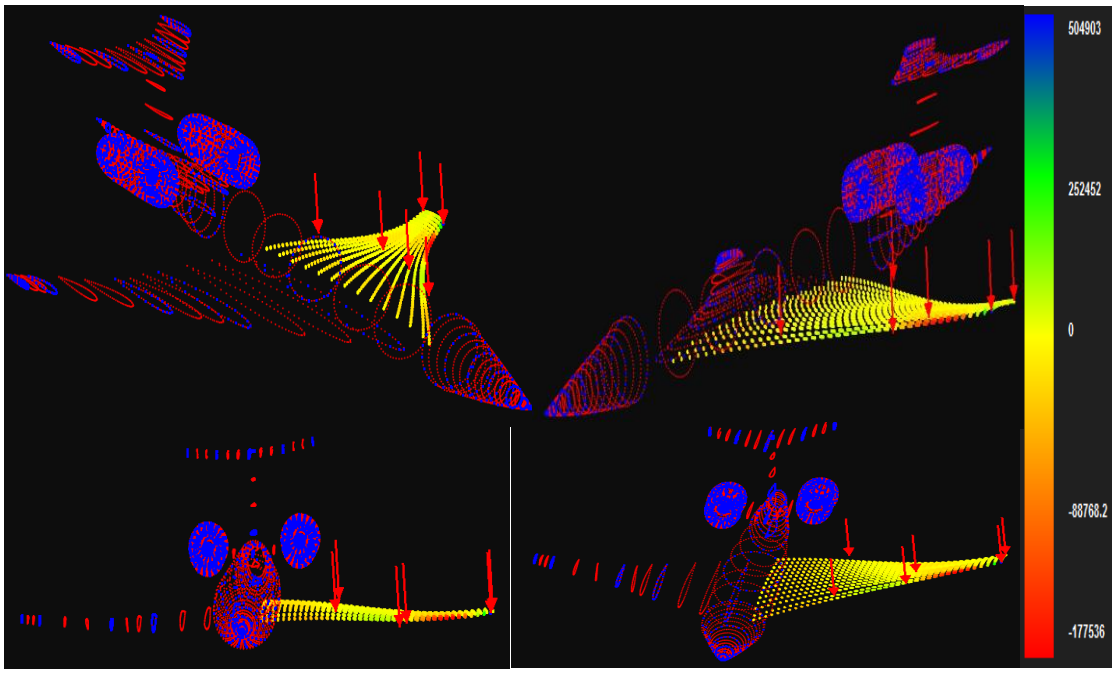
(b) 4x8x2 control points



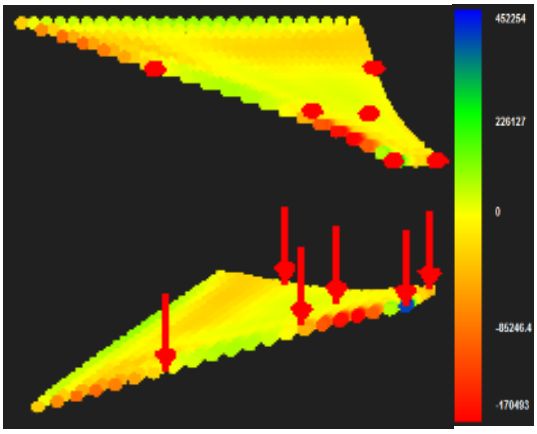
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.168 Stress X

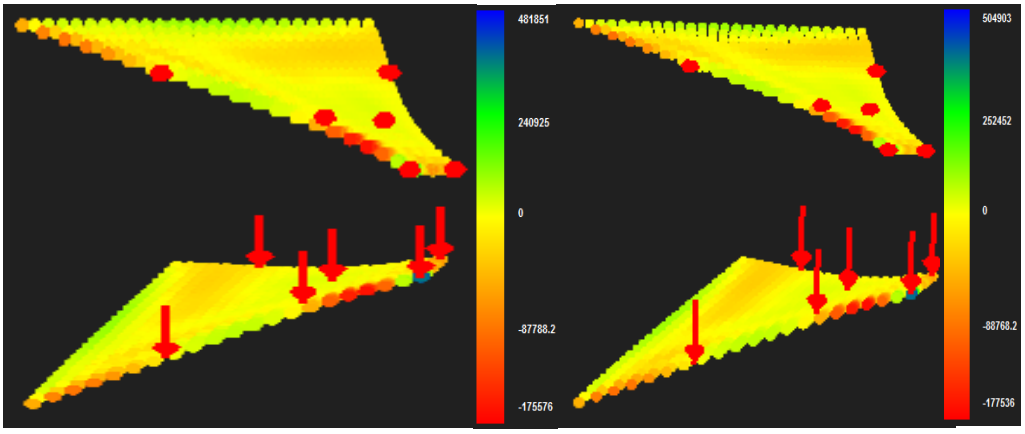
In figure 5.169 contours for Stress Y Deformed are presented.



(a) SST aircraft stress Y deformed (4x8x4 control points)



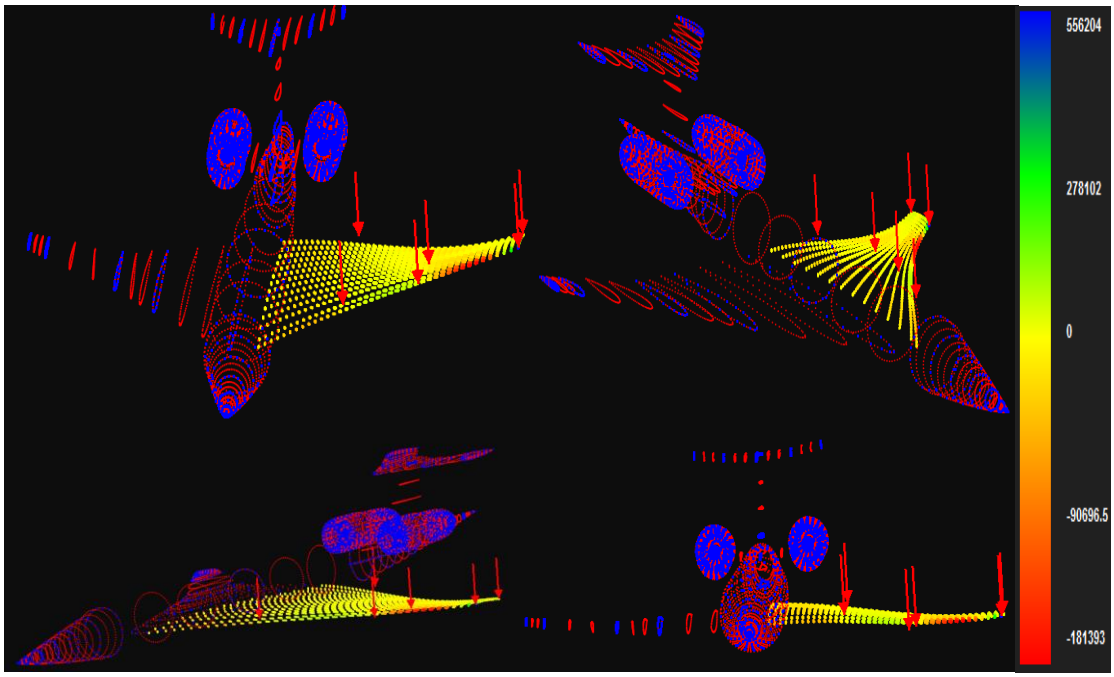
(b) 4x8x2 control points



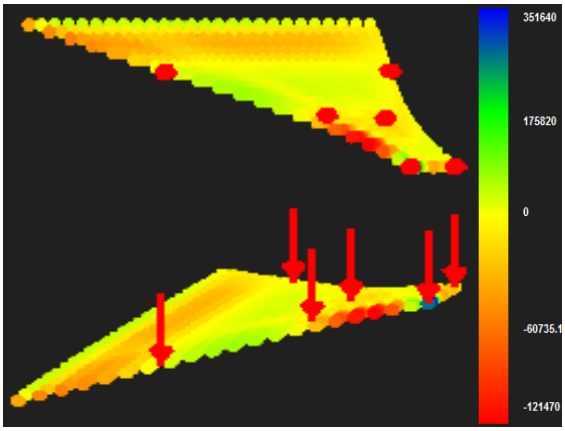
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.169 Stress Y

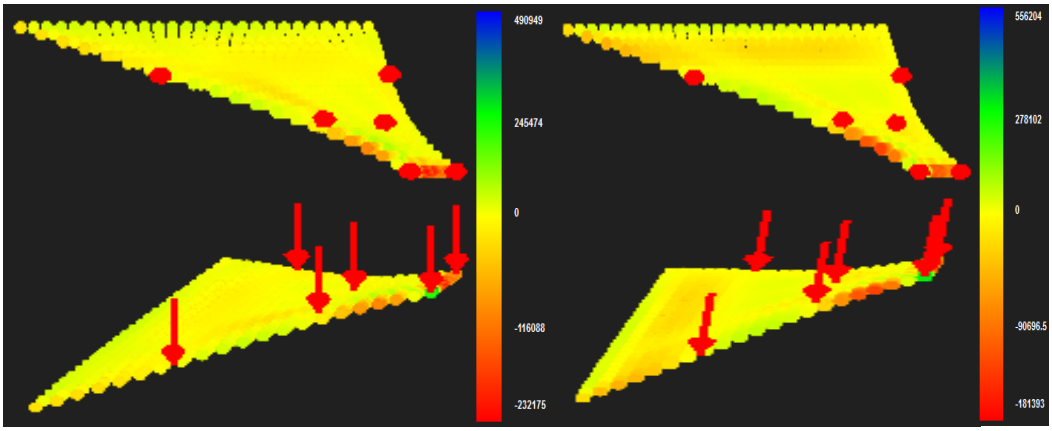
In figure 5.170 contours for Stress Z deformed are presented.



(a) SST aircraft stress Z deformed (4x8x4 control points)



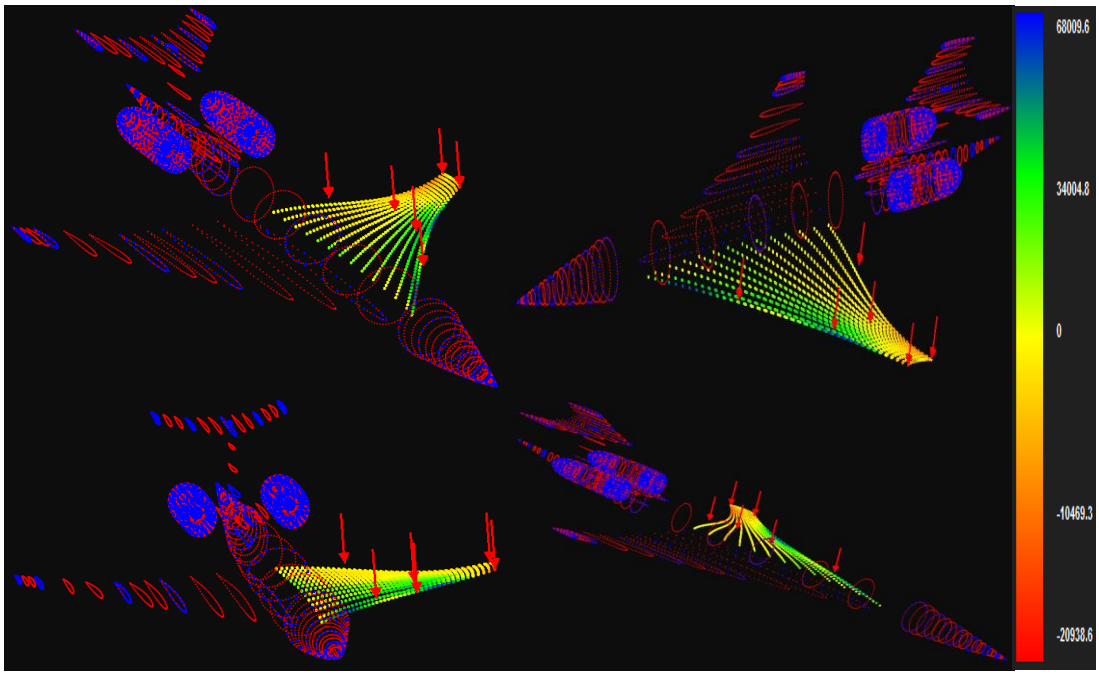
(b) 4x8x2 control points



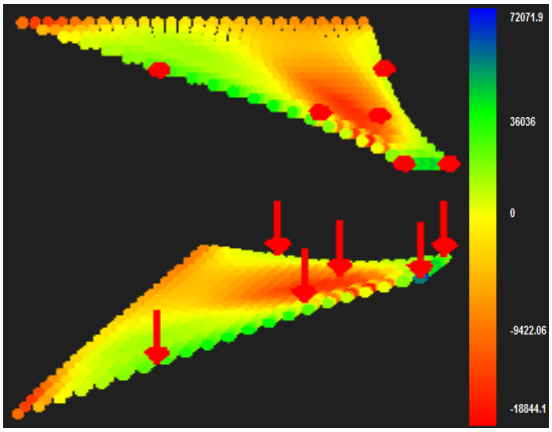
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.170 Stress Z

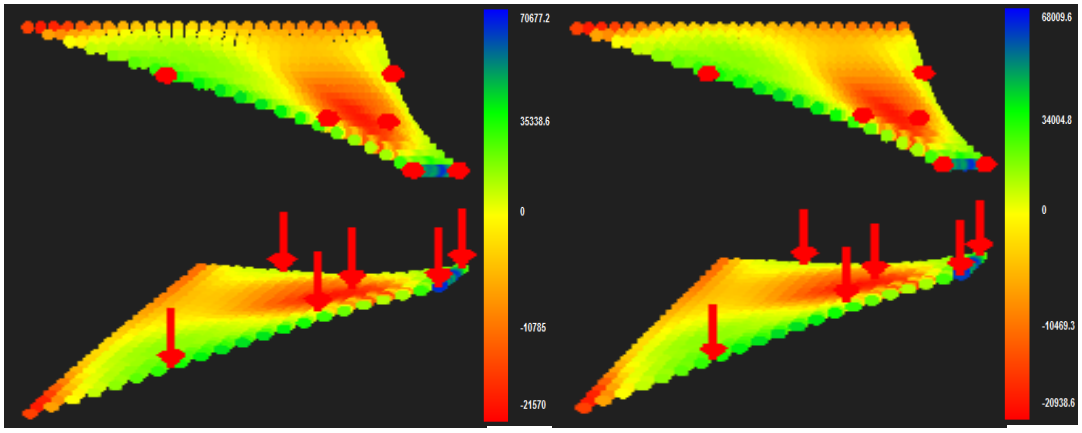
In figure 5.171 contours for Stress XY Deformed are presented



(a) SST aircraft stress XY deformed (4x8x4 control points)



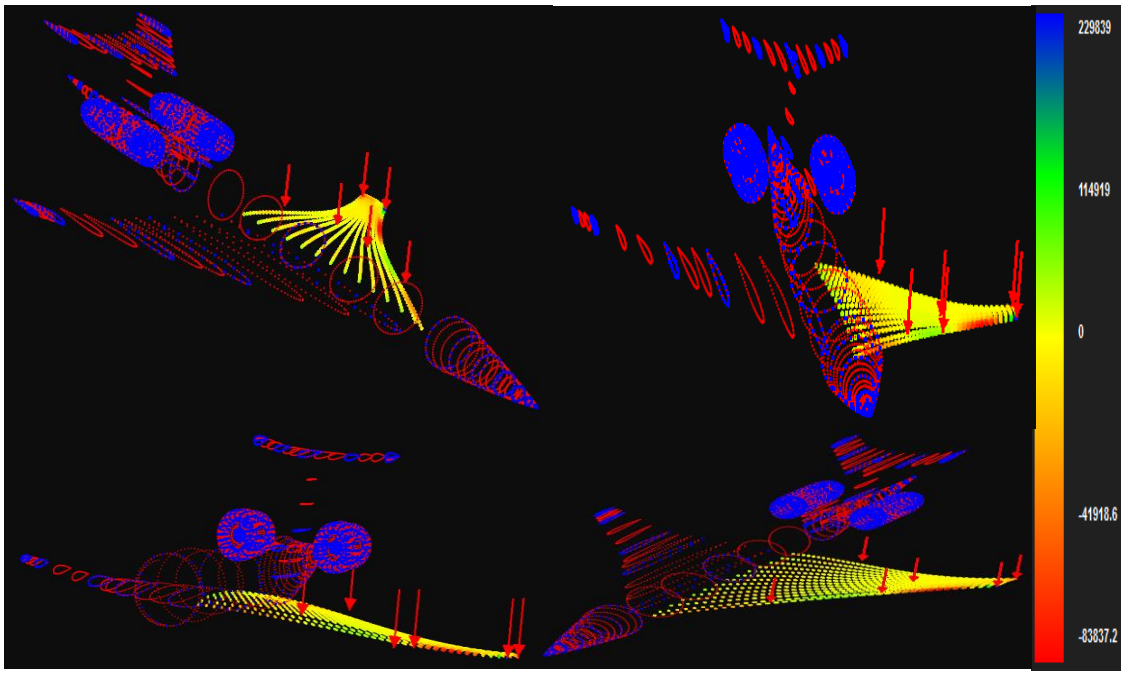
(b) 4x8x2 control points



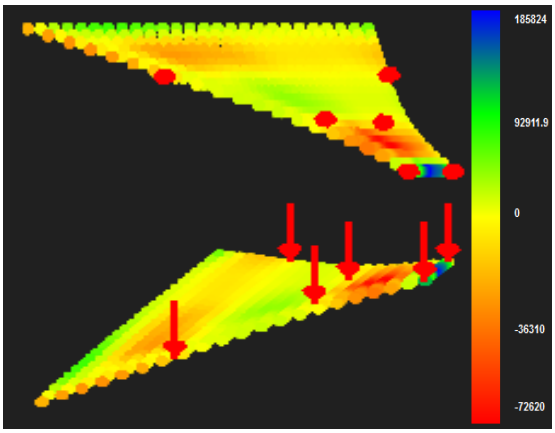
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.171 Stress XY

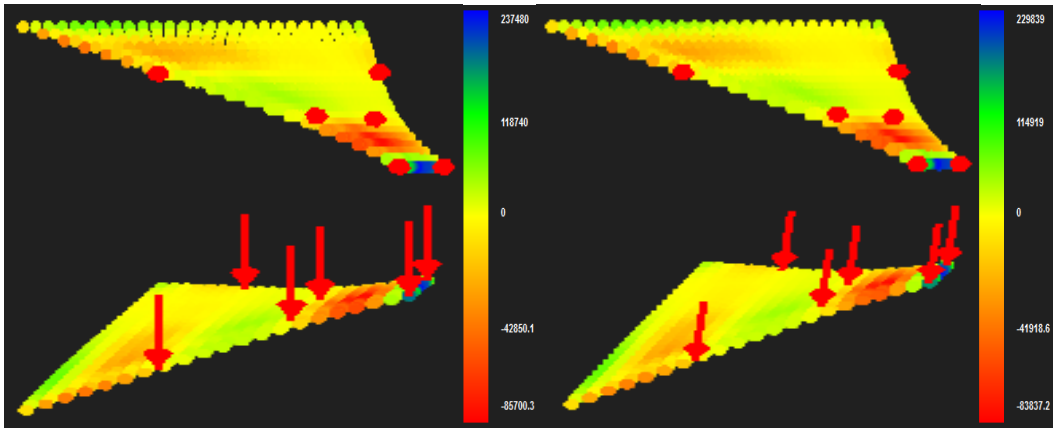
In figure 5.172 contours for Stress YZ Deformed are presented.



(a) SST aircraft stress YZ deformed (4x8x4 control points)



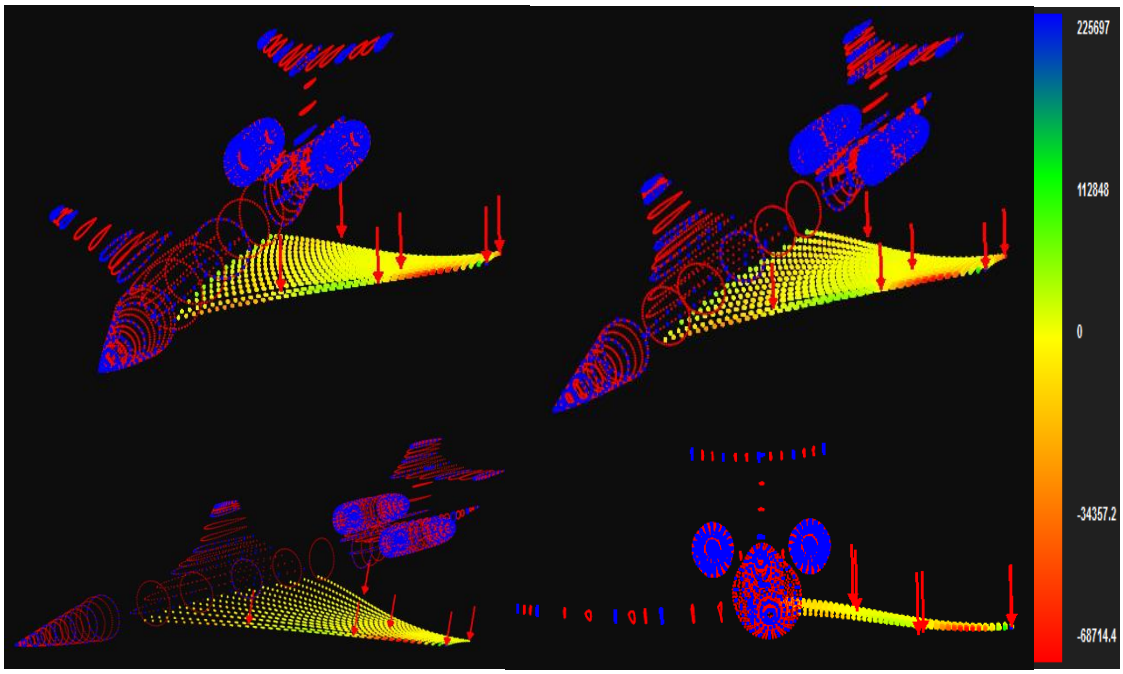
(b) 4x8x2 control points



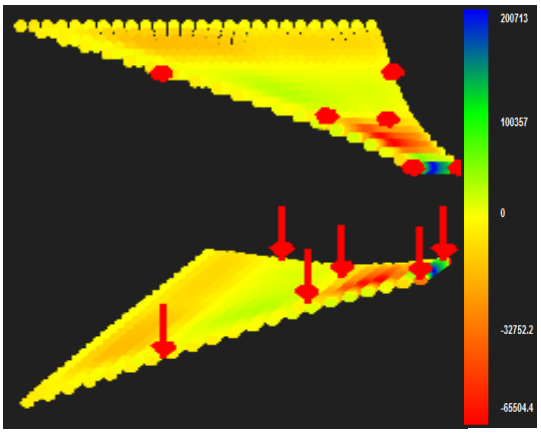
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.172 Stress YZ

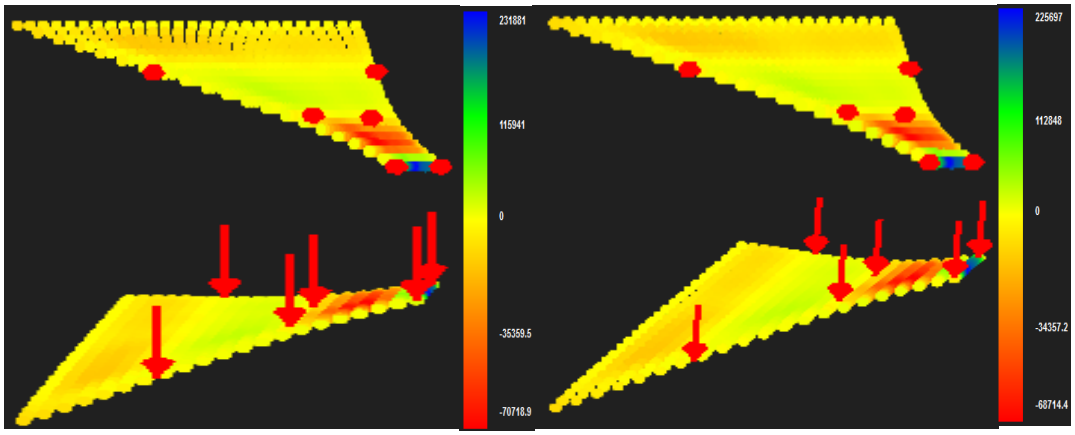
In figure 5.173 contours for Stress ZX Deformed are presented



(a) SST aircraft stress ZX deformed (4x8x4 control points)



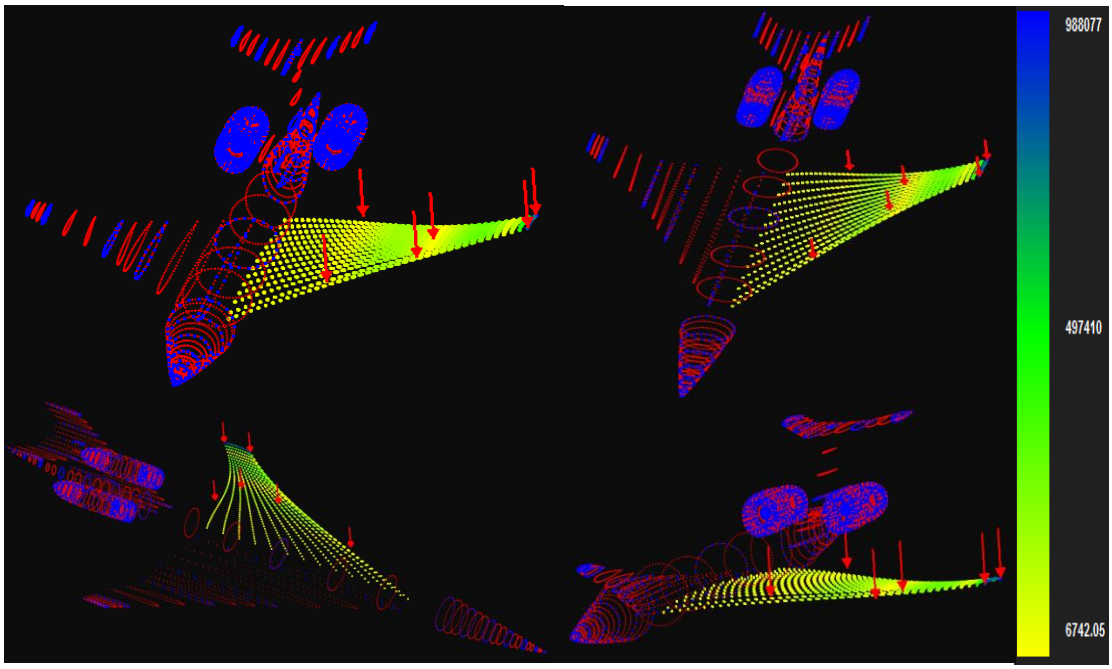
(b) 4x8x2 control points



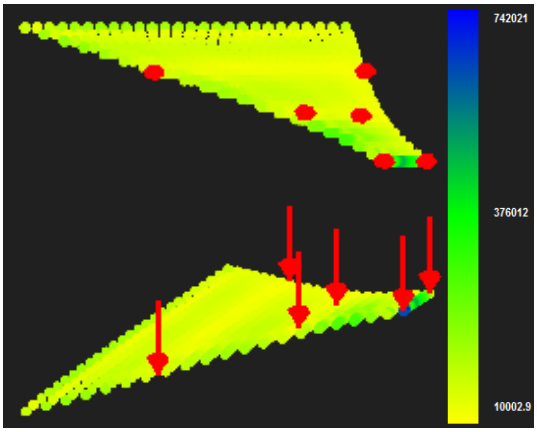
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.173 Stress ZX

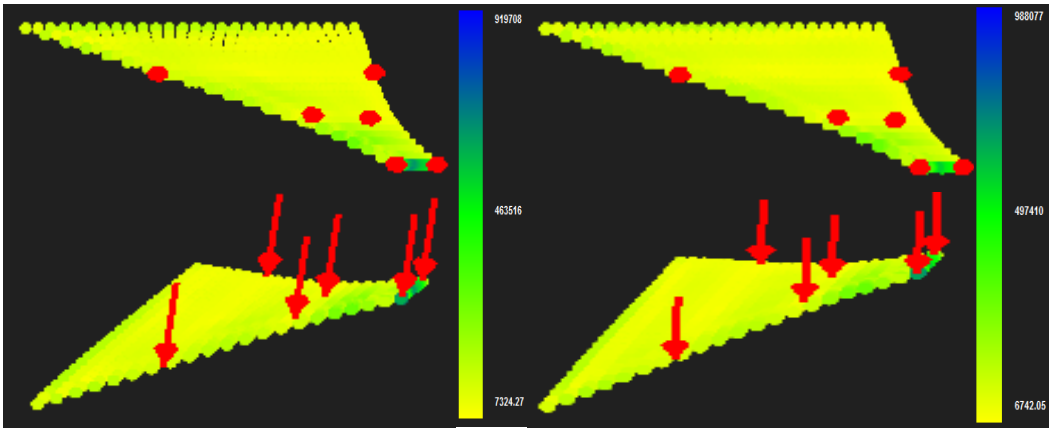
In figure 5.174 contours for Stress von Mises Deformed are presented



(a) SST aircraft stress von Mises deformed (4x8x4 control points)



(b) 4x8x2 control points



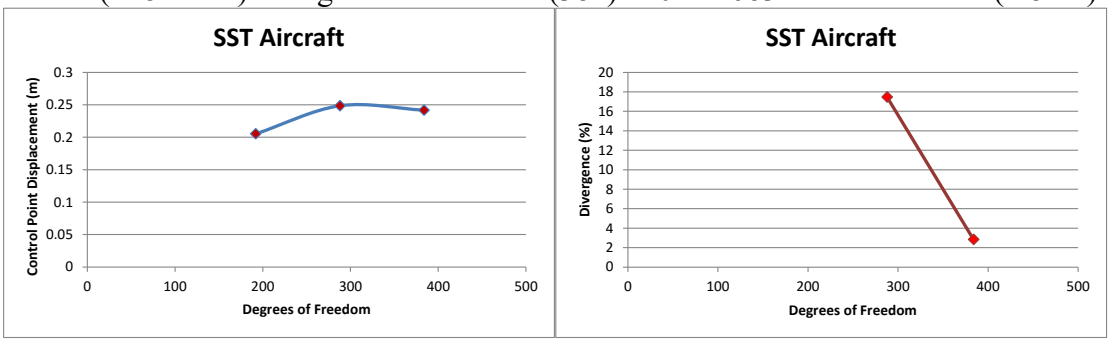
(c) 4x8x3 control points **(d) 4x8x4 control points**

Figure 5.174 Stress von Mises

In diagrams below we can see that for Displacement Norm and Stress von Mises we get nice results but in control point Displacement the use of Linear basis functions gives unacceptable results.

Control Point Displacement (m):

- (4x8x2 CP)→Degrees of Freedom (198)→ 0.205158 m
- (4x8x3 CP)→Degrees of Freedom (288)→ 0.248493 m→ Deviation (17.44%)
- (4x8x4 CP)→Degrees of Freedom (384)→ 0.241683 m → Deviation (2.82%)

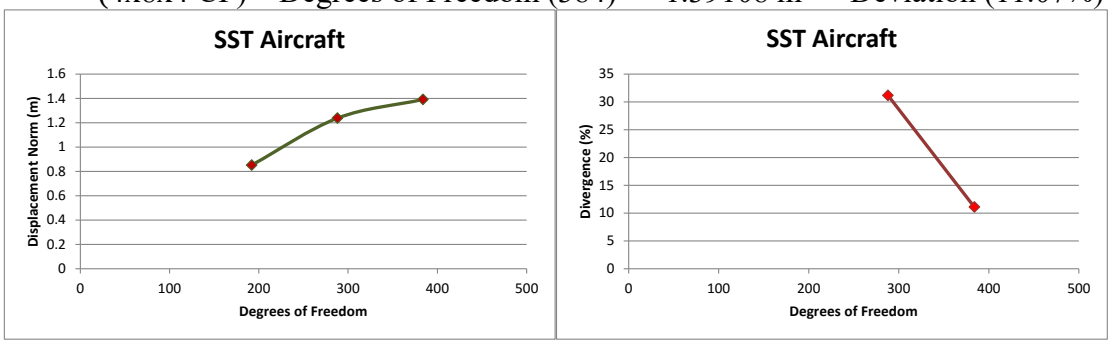


(a) **(b)**

Figure 5.175 Control point displacement results.
(a) Control point displacement in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

Norm Displacement (m):

- (4x8x2 CP)→Degrees of Freedom (198) → 0.851492 m
- (4x8x3 CP)→Degrees of Freedom (288) → 1.23705 m → Deviation (31.17%)
- (4x8x4 CP)→Degrees of Freedom (384) → 1.39108 m → Deviation (11.07%)



(a) **(b)**

Figure 5.176 Displacement norm results.
(a) Displacement norm in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

Control Point Stress von Mises (kPa):

- (4x8x2 CP)→Degrees of Freedom (198) → 742021 kPa
- (4x8x3 CP)→Degrees of Freedom (288)→ 919708 kPa→ Deviation (19.32%)
- (4x8x4 CP)→Degrees of Freedom (384) → 988077 kPa → Deviation (6.92%)

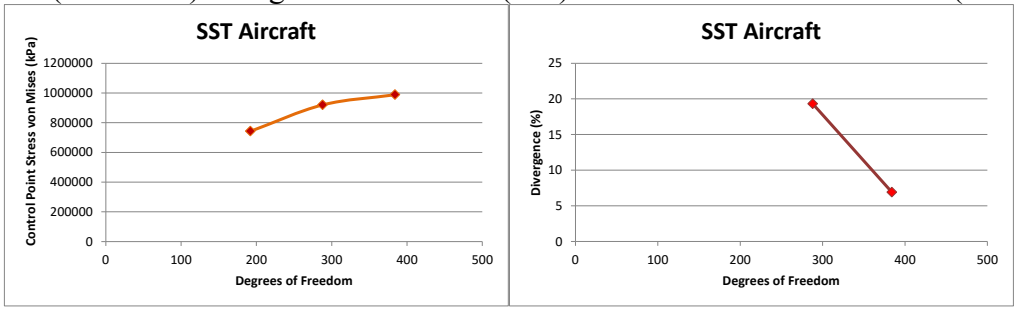


Figure 5.177 Stress von Mises results.
(a) Stress von Mises in comparison with total degrees of freedom,
(b) Divergence from previous solution in comparison with total degrees of freedom.

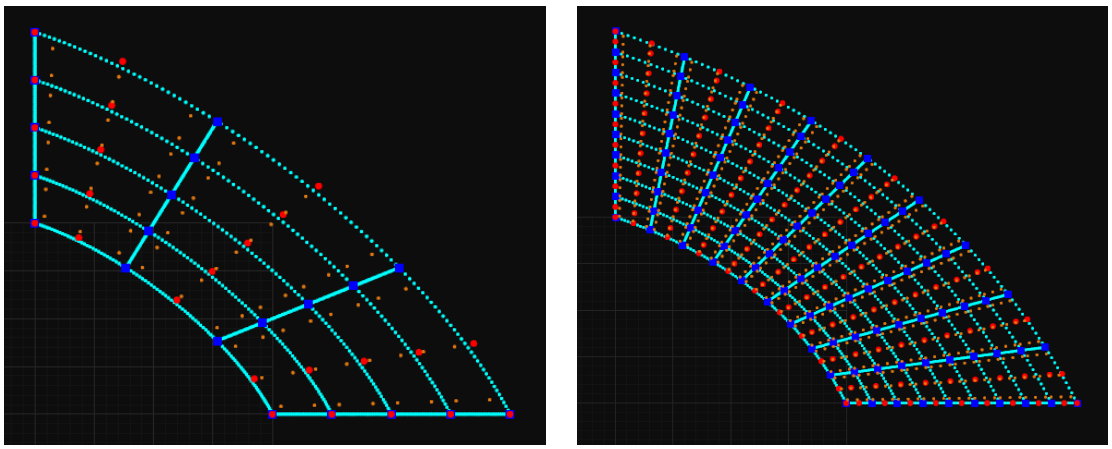
6 Conclusions

Exact Geometry

Isogeometric Analysis enables exact representation for the analysis process. The mathematical model has the exact same features as the design model. Therefore, transition from design to analysis model does not produce any geometrical errors and the accuracy is instantly increased.

Improved refinement schemes

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. Refinements preserve the exact geometry and parametric mapping while providing greater accuracy for analysis. FEM, on the contrary, creates 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 combination also applies in IGA. k-Refinement is proving to be more competitive than its counterparts.

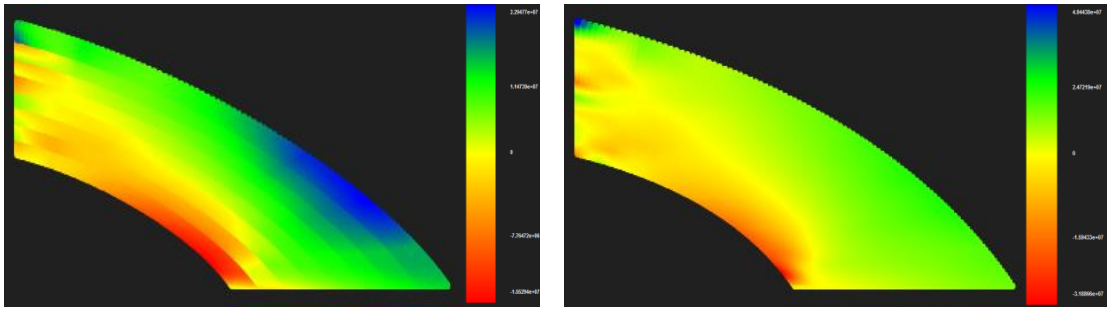


(a) **(b)**

Figure 6.1. (a) Coarse mesh and (b) application of k-Refinement.

Element interconnectivity

Isogeometric shape functions hold an overlapping that leads to greater interconnectivity between elements. Derivatives are continuous across element boundaries, therefore stress and strain fields are continuous as well. The use of corrective methods such as extrapolation is applied only to special cases of C^0 Continuity.



(a) **(b)**

Figure 6.2. Stress contour σ_{xy} for (a) C^0 (b) C^1 continuity.

Patch Utilitization in IGA

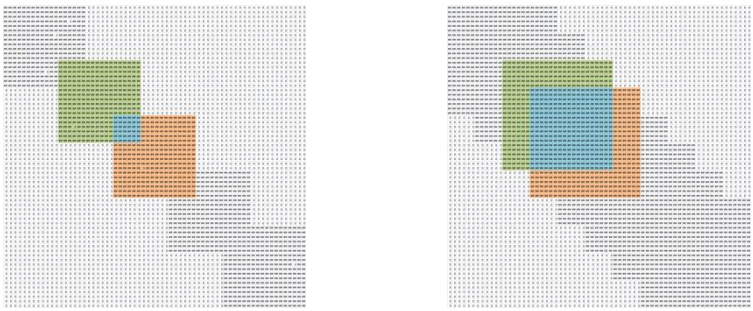
Patches are used in Isogeometric Analysis when a significant change in Geometry or Material occurs. Similarly to Finite Elements, patches possess C^{-1} Continuity on the boundary and all shape functions are contained in their interior. During Stiffness Matrix Formulation, a local Stiffness Matrix is produced for every patch. Afterwards, the global Stiffness Matrix is built through patch connectivity.

Each patch has a unique mapping from its own parameter space to the physical space. Refinement is applied separately for every patch. NURBS patch connection is not always water-tight; boundary inconsistencies are possible to exist.

Patch boundary can also be created through a single knot value vector, when the separate parameter spaces of the patches can be merged into one. control points on the boundaries are fused into one. The corresponding C^{-1} Continuous shape functions are combined into one, with C^0 Continuity across that edge. C^0 Continuity is also useful for the creation of interpolatory control points at a specific part of the model.

Stiffness Matrix Formulation

Stiffness Matrix creation for IGA follows the same general principles as in FEM. Enhanced shape function overlapping leads to a matrix with greater bandwidth. As more non-zero elements are created and interconnectivity is increased, the resulting Stiffness Matrix is a more accurate representation of the natural problem. However, this overlapping makes the formulation process time-consuming.



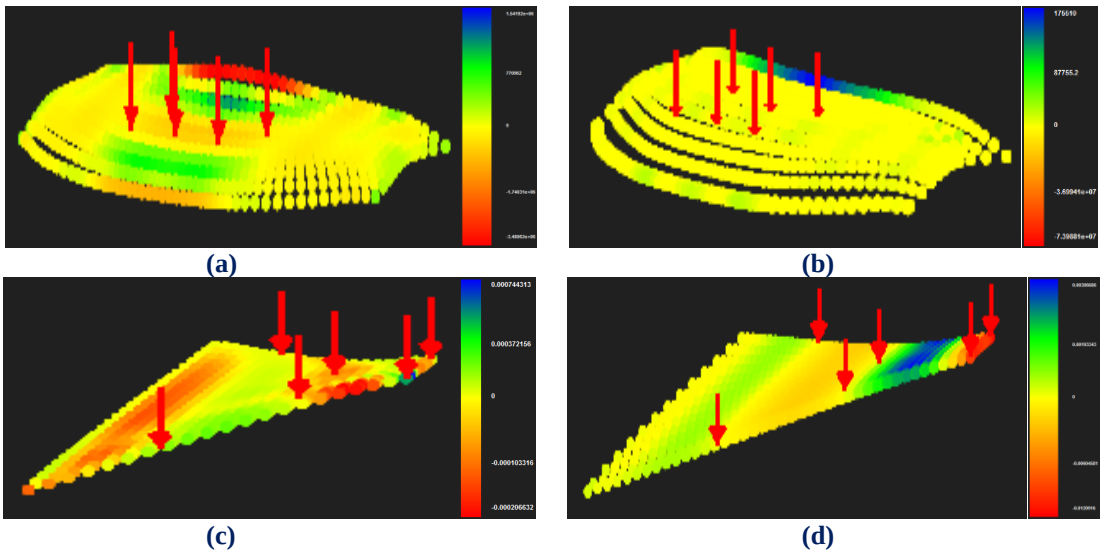
(a) **(b)**

Figure 6.3. Stiffness matrices with various interconnectivities. (a) C^0 and (b) C^2 continuity.

The role of the Engineer

The collective experience from all the years of FEM practice is definitely useful in Isogeometric Analysis as well. The engineer should be able to maintain a thorough understanding of the methods and techniques involved, as no machine can replace the creativity and important role of the human mind.

The technique of generating a new surface under the old one in order to face a 2-Dimensional problem as a 3-D one is not always an easy procedure. Attention must be paid so that the parameterization of the new surface is the right one. In case of wrong parameterization, wrong results are received, if the appropriate CAD tools are not used in the right way.



(a) **(b)**

(c) **(d)**

Figure 6.4. (a), (c) Right parameterization and (b), (d) Wrong parameterization.

Appendix A

A.1 Introduction (Plug-in Creation)

Despite the fact that global CAD and CAE industries grow at impressive compound annual growth rate (2012-2016), they minimally communicate with each other and can each benefit from the latest technological achievements of the other. The gap between them troubled for decades the scientific and professional community of engineers and due to the fact that researchers know little of CAD computational mechanics and researchers of CAE slightly from computational geometry. The main purpose and target of this thesis is to investigate a way of bridging these two industries. If a successful interconnection between CAD and CAE software is accomplished then the entire theory of Isogeometric Analysis will be incorporated into real and sophisticated engineering projects.

Engineers will finally manage to exploit the encrypted information that remain decoded inside geometry and get exempted from the laborious and time consuming simulation procedure. In this chapter I will attempt to create a plug-in on one of the most popular CAD Software, AutoCAD, so that any engineering project of this application software can get into position to be used in an Isogeometric analysis procedure.

A.2 AutoCAD Geometry Exploitation

AutoCAD is a commercial software application for 2D and 3D computer-aided design (CAD) and drafting. It is available since 1982 as a desktop application and since 2010 as a mobile web and cloud-based app marketed as AutoCAD 360. It was developed and marketed by Autodesk, Inc., AutoCAD and it runs on microcomputers with internal graphics controllers.

Prior to the introduction of AutoCAD, most commercial CAD programs ran on mainframe computers or minicomputers, with each CAD operator (user) working at a separate graphics terminal.

AutoCAD is used across a wide range of industries and professionals such as:

- Architects
- Project managers
- Engineers
- Graphic designers
- and other professionals.

It is supported by 750 training centers worldwide as of 1994.

As Autodesk's flagship product, by March 1986 AutoCAD had become the most ubiquitous CAD program worldwide. As of 2015, AutoCAD is in its thirtieth generation, and collectively with all its variants, continues to be the most widely used CAD program throughout most of the world.

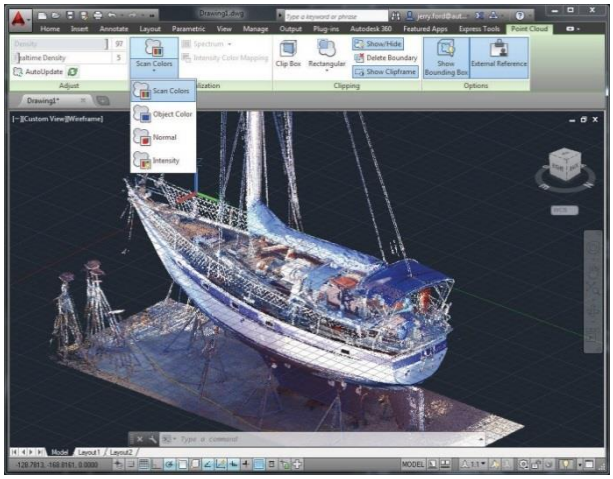


Figure A.1 AutoCAD models (www.autodeskforum.hu)

Despite the fact that AutoCAD is, if not the best, one of the most appropriate software applications in order to create a 2D or 3D model by architects designers and engineers, it is quite difficult if not impossible to exploit the hidden data that underlie in geometry and contain all the significant information about the model creation and the geometry description. It is possible to maintain only basic characteristics of B-Splines and NURBS Surfaces, such us the degree or the coordinates of the position of the control points that describe the model. Even then though, the software does not give the user the ability to export such information to other file formats that could be used by CAE software applications that deal with the analysis of the designed models.

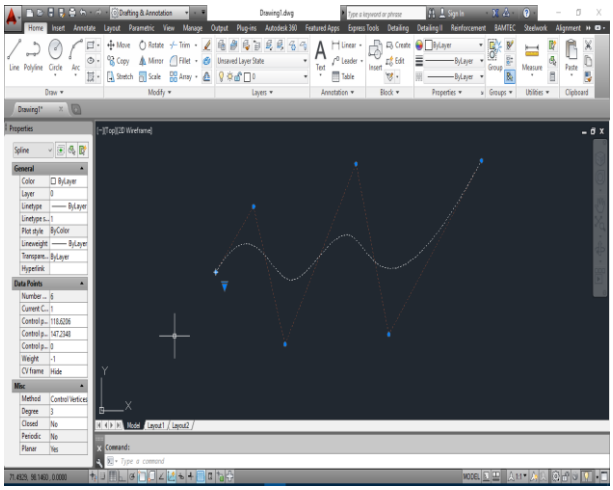


Figure A.2. B-Spline curve. (created by George Karaiskos with AutoCAD)

This disadvantage does not allow to Isogeometric Analysis to bridge the existing gap between CAD and CAE. As a result, the analysis procedure is not able to fully exploit the data and the mesh of the geometry, not even to utilize the same shape functions for the creation of the geometry and also for the determination of the numerical solution of the problem.

In order such problems and difficulties to be solved, programmers are needed, who will have the ability to specialize in learning the programming language of each CAD software application and create plug-in source codes. These plug-ins will link with the main application and will have the ability to export from the software all the necessary information for the analysis procedure.

A.3 AutoLISP

AutoLISP is a dialect of Lisp programming language built specifically for use with the full version of AutoCAD and its derivatives, which include AutoCAD Map 3D, AutoCAD Architecture and AutoCAD Mechanical. Neither the application programming interface, nor the interpreter to execute AutoLISP code are included in the AutoCAD LT product line. AutoLISP is a small, dynamically scoped, dynamically typed LISP dialect with garbage collection, immutable list structure and settable symbols, lacking in such regular LISP features as macro system, records definition facilities, arrays, functions with variable number of arguments or let bindings. Aside from the core language, most of the primitive functions are for geometry, accessing AutoCAD's internal DWG database, or manipulation of graphical entities in AutoCAD. The properties of these graphical entities are revealed to AutoLISP as association lists in which values are paired with AutoCAD "group codes" that indicate properties such as definitional points, radii, colors, layers, linetypes, etc. AutoCAD loads AutoLISP code from .LSP files.

```
1 (defun hello ()
2    (princ "\nHello World!")
3 )
```

Figure A.3. Simple hello example.
(https://en.wikipedia.org/wiki/AutoLISP)

AutoLISP code can interact with the user through Autocad's graphical editor by use of primitive functions that allow user to pick points, choose objects on screen, input numbers and other data. AutoLisp also has a built-in GUI mini-language, the Dialog control Language, for creating modal dialog boxes with automated layout, within AutoCAD.

```
1 (defun C:POINTLABEL ( / pt)
2    (setq pt (getpoint "\nPick point: "))
3    (command "POINT" pt)
4    (command "TEXT" (polar pt 0 0.6) 0 (strcat "X:" (rtos (car pt)) " Y:" (rtos (cadr pt))))
5    (princ)
6 )
```

Figure A.4. More complex example.
(https://en.wikipedia.org/wiki/AutoLISP)

In order to export the necessary data from AutoCAD software application to .txt file format, so that they could be used from a CAE software application for the analysis procedure, I specialized in AutoLISP and I created a plug-in with two versions according to the case and reason of exportation. Furthermore, it allows the user to maintain and use all the info for the purpose of the analysis of 1-Dimension problems.

A.4 AutoCAD Plug-In First Version

First Version Exports:

- Number of Spline Objects
- Spline Object ID
- Degree
- Parametric Coordinates of knots
- Number of control points
- Cartesian Coordinates and Weights of control points

Source Code

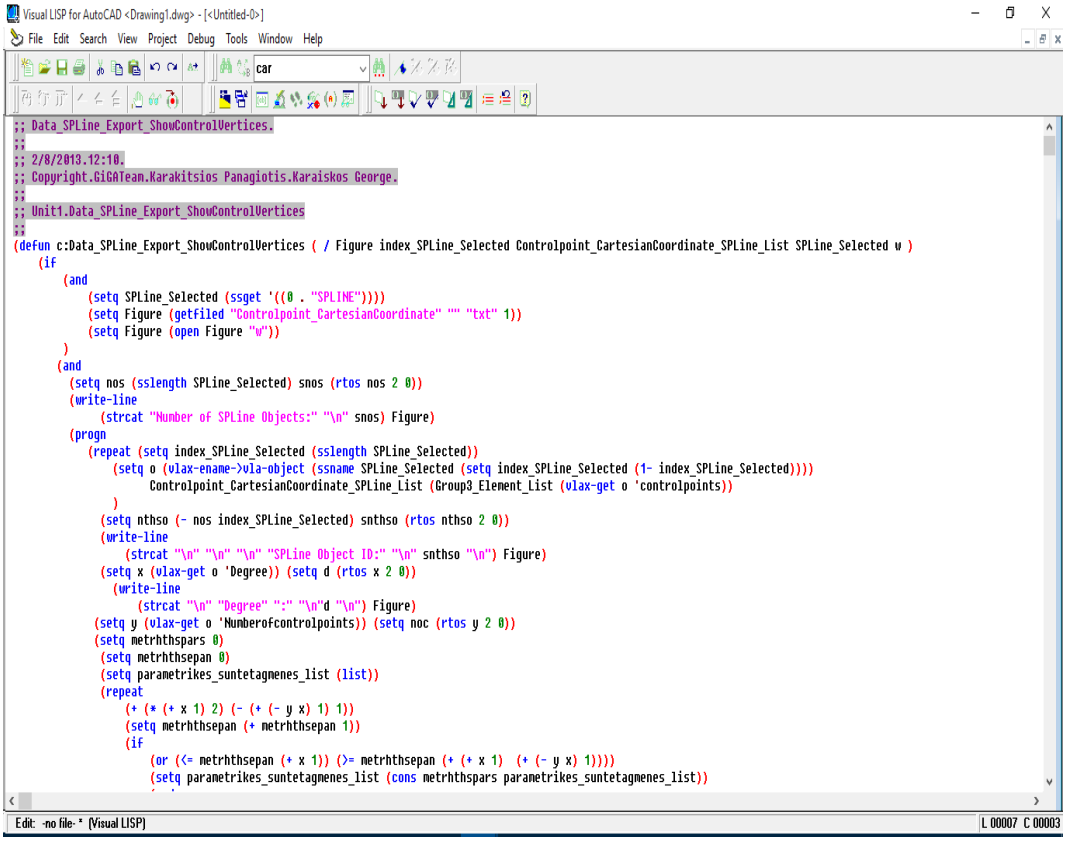


Figure A.5. Source code.
(created by George Karaiskos with AutoLISP)

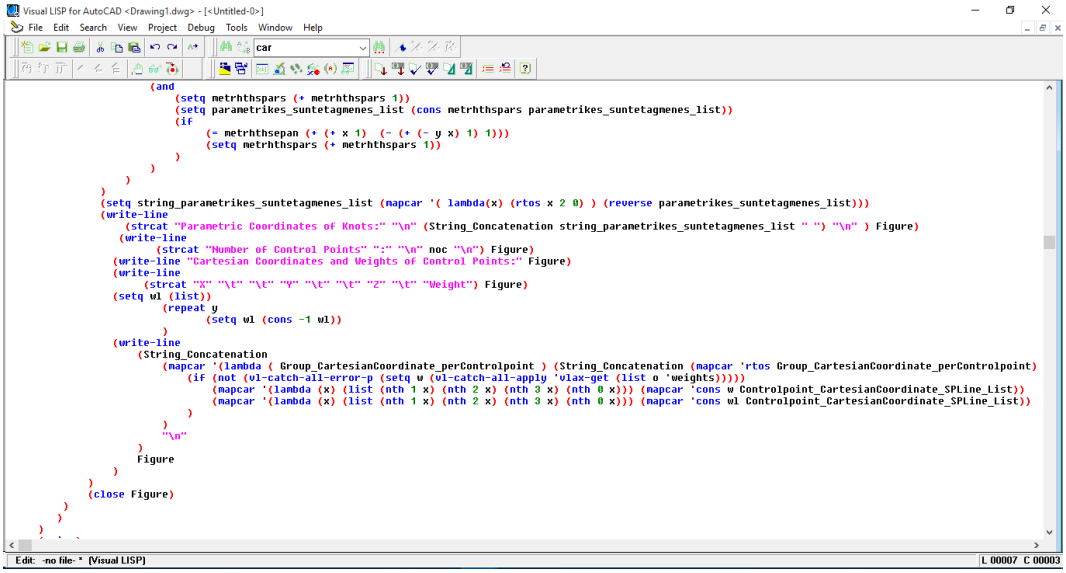


Figure A.6 Source code
(created by George Karaiskos with AutoLISP)

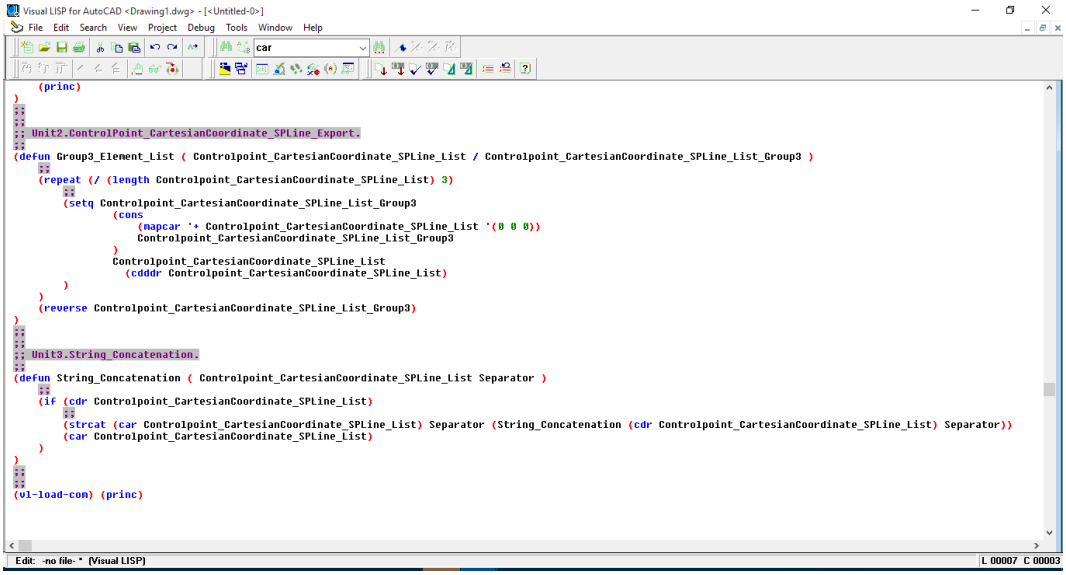


Figure A.7 Source Code
(created by George Karaiskos with AutoLISP)

A.5 AutoCAD Plug-In Second Version

Second Version Exports:

- Number of Spline Objects
- Spline Object ID
- Degree
- Parametric Coordinates of knots
- Number of knots
- Cartesian Coordinates and of knots
- Number of control points
- Cartesian Coordinates and Weights of control points

Source Code

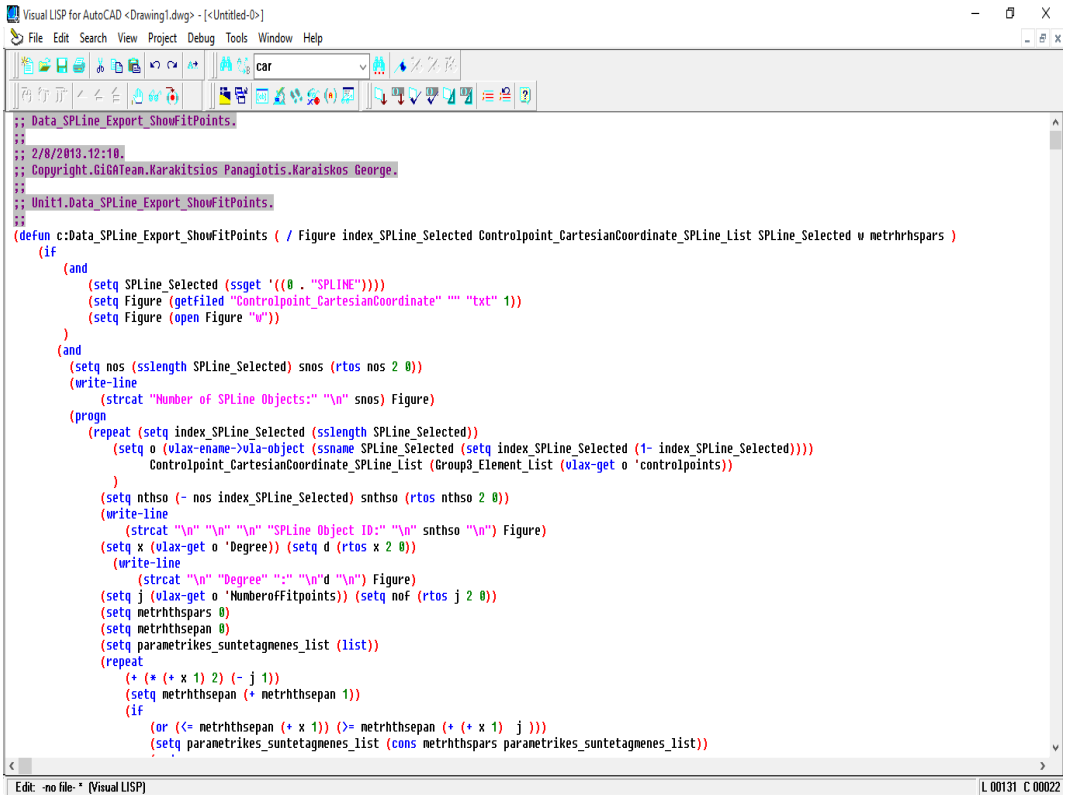
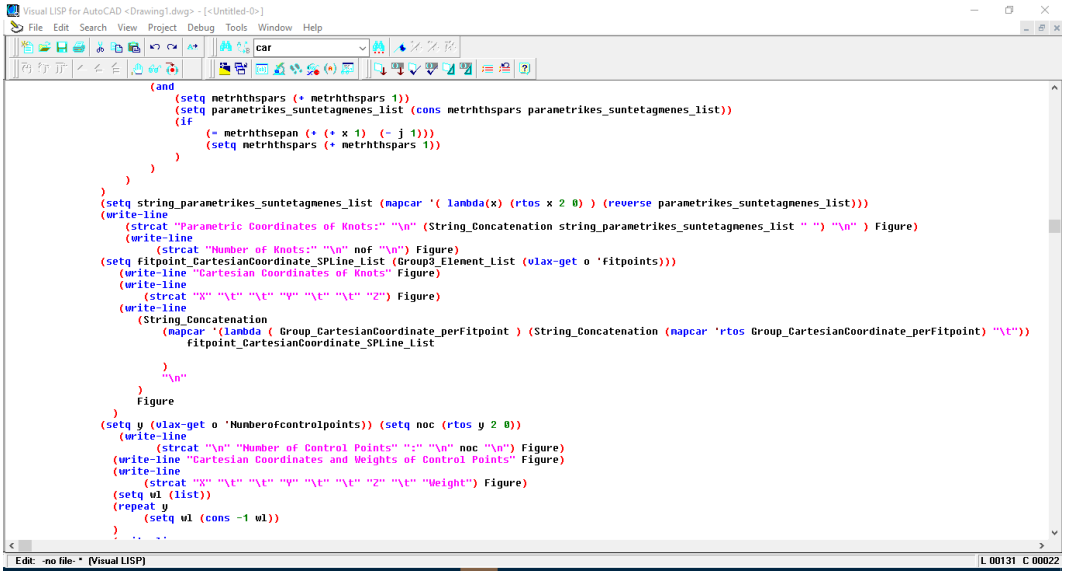


Figure A.8 Source code
(created by George Karaiskos with AutoLISP)

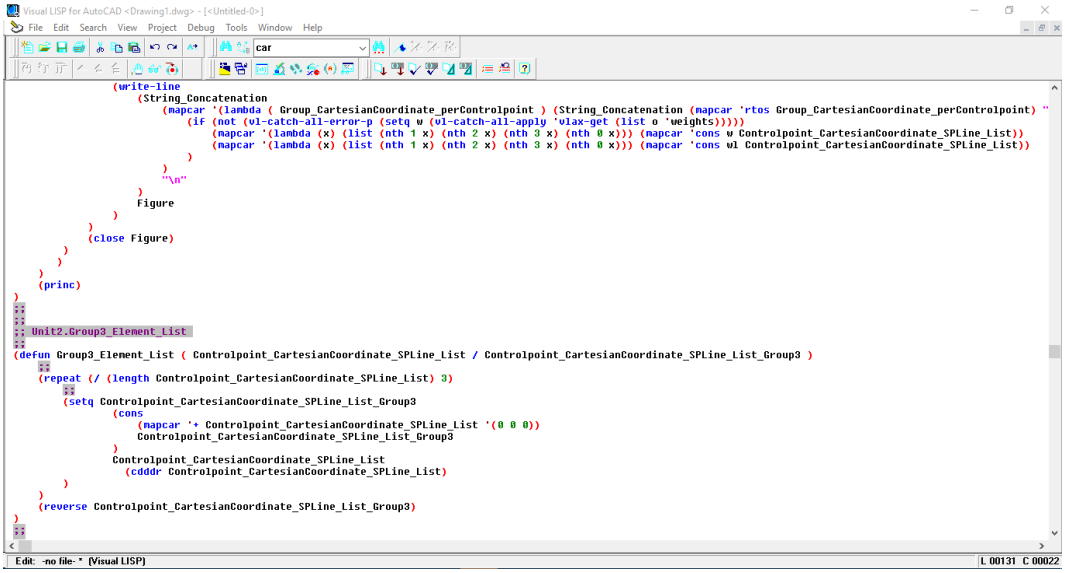


```
                    (and
                        (setq metrhthspars (+ metrhthspars 1))
                        (setq parametrikes_suntetagmenes_list (cons metrhthspars parametrikes_suntetagmenes_list))
                        (if
                            (= metrhthsepan (+ (+ x 1)  (- j 1)))
                            (setq metrhthspars (+ metrhthspars 1))
                        )
                    )
                )
            )
            (setq string_parametrikes_suntetagmenes_list (mapcar '( lambda(x) (rtos x 2 8) ) (reverse parametrikes_suntetagmenes_list)))
            (write-line
                (strcat "Parametric Coordinates of Knots:" "\n" (String_Concatenation string_parametrikes_suntetagmenes_list " ") "\n" ) Figure)
                (write-line
                    (strcat "Number of Knots:" "\n" nof "\n") Figure)
            (setq fitpoint_CartesianCoordinate_SPLine_List (Group3_Element_List (vlax-get o 'fitpoints)))
                (write-line "Cartesian Coordinates of Knots" Figure)
                (write-line
                    (strcat "X" "\t" "\t" "Y" "\t" "\t" "Z") Figure)
                (write-line
                    (String_Concatenation
                        (mapcar '(lambda ( Group_CartesianCoordinate_perFitpoint ) (String_Concatenation (mapcar 'rtos Group_CartesianCoordinate_perFitpoint) "\t"))
                            fitpoint_CartesianCoordinate_SPLine_List

                        )
                        "\n"
                    )
                    Figure
                )
            (setq y (vlax-get o 'Numberofcontrolpoints)) (setq noc (rtos y 2 0))
                (write-line
                    (strcat "\n" "Number of Control Points" ":" "\n" noc "\n") Figure)
              (write-line "Cartesian Coordinates and Weights of Control Points" Figure)
              (write-line
                    (strcat "X" "\t" "\t" "Y" "\t" "\t" "Z" "\t" "Weight") Figure)
              (setq wl (list))
              (repeat y
                    (setq wl (cons -1 wl))
              )
```

Edit: -no file- * (Visual LISP) L 00131 C 00022

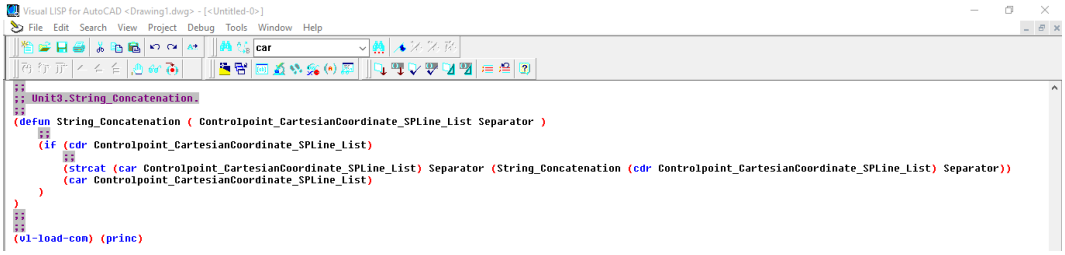
Figure A.9 Source code.
(created by George Karaiskos with AutoLISP)



```
                (write-line
                    (String_Concatenation
                        (mapcar '(lambda ( Group_CartesianCoordinate_perControlpoint ) (String_Concatenation (mapcar 'rtos Group_CartesianCoordinate_perControlpoint) "
                            (if (not (vl-catch-all-error-p (setq w (vl-catch-all-apply 'vlax-get (list o 'weights)))))
                                (mapcar '(lambda (x) (list (nth 1 x) (nth 2 x) (nth 3 x) (nth 0 x))) (mapcar 'cons w Controlpoint_CartesianCoordinate_SPLine_List))
                                (mapcar '(lambda (x) (list (nth 1 x) (nth 2 x) (nth 3 x) (nth 0 x))) (mapcar 'cons wl Controlpoint_CartesianCoordinate_SPLine_List))
                            )
                        )
                        "\n"
                    )
                    Figure
                )
            )
            (close Figure)
        )
      )
    )
    (princ)
)
;;
;;
;; Unit2.Group3_Element_List
;;
(defun Group3_Element_List ( Controlpoint_CartesianCoordinate_SPLine_List / Controlpoint_CartesianCoordinate_SPLine_List_Group3 )
    ;;
    (repeat (/ (length Controlpoint_CartesianCoordinate_SPLine_List) 3)
        ;;
        (setq Controlpoint_CartesianCoordinate_SPLine_List_Group3
                (cons
                    (mapcar '+ Controlpoint_CartesianCoordinate_SPLine_List '(0 0 0))
                    Controlpoint_CartesianCoordinate_SPLine_List_Group3
                )
                Controlpoint_CartesianCoordinate_SPLine_List
                  (cdddr Controlpoint_CartesianCoordinate_SPLine_List)
        )
    )
    (reverse Controlpoint_CartesianCoordinate_SPLine_List_Group3)
)
;;
```

Edit: -no file- * (Visual LISP) L 00131 C 00022

Figure A.10 Source code.
(created by George Karaiskos with AutoLISP)



```
;;
;; Unit3.String_Concatenation.
;;
(defun String_Concatenation ( Controlpoint_CartesianCoordinate_SPLine_List Separator )
    ;;
    (if (cdr Controlpoint_CartesianCoordinate_SPLine_List)
        ;;
        (strcat (car Controlpoint_CartesianCoordinate_SPLine_List) Separator (String_Concatenation (cdr Controlpoint_CartesianCoordinate_SPLine_List) Separator))
        (car Controlpoint_CartesianCoordinate_SPLine_List)
    )
)
;;
;;
(vl-load-com) (princ)
```

Figure A.12 Source code.
(created by George Karaiskos with AutoLISP)

A.6 Plug-In Using Process

The procedure that is required in order to export the necessary data from AutoCAD software application to .txt file format is the following:

1. **Load of the Plug-In**

Since AutoCAD software application has been launched, the user grabs the .lsp archive plug-in (Data_SPLine_Export_ShowControlVertices) and throws it in the graphic environment as in the Figure 3.13

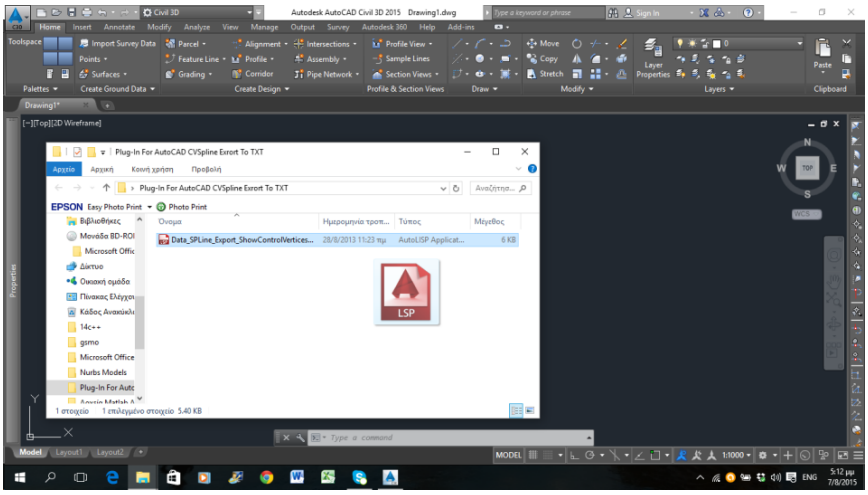


Figure A.13.lsp archive plugin
(created by George Karaiskos with AutoLISP)

Then a File Loading-Security Concern message is shown up informing us that our plug-in is about to be loaded. We press Load to confirm the Load procedure as in Figure 3.4

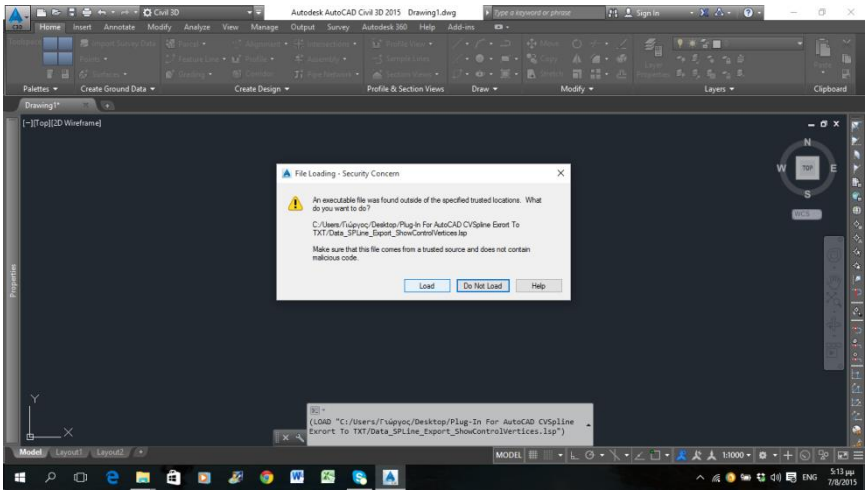


Figure A.14 File Loading-Security Concern message
(created by George Karaiskos with AutoLISP)

2. Model Creation

We choose Draw, then Spline CV and we draw as many Splines we want by defining the coordinates position of the control points that describe our 1- Dimension problem.

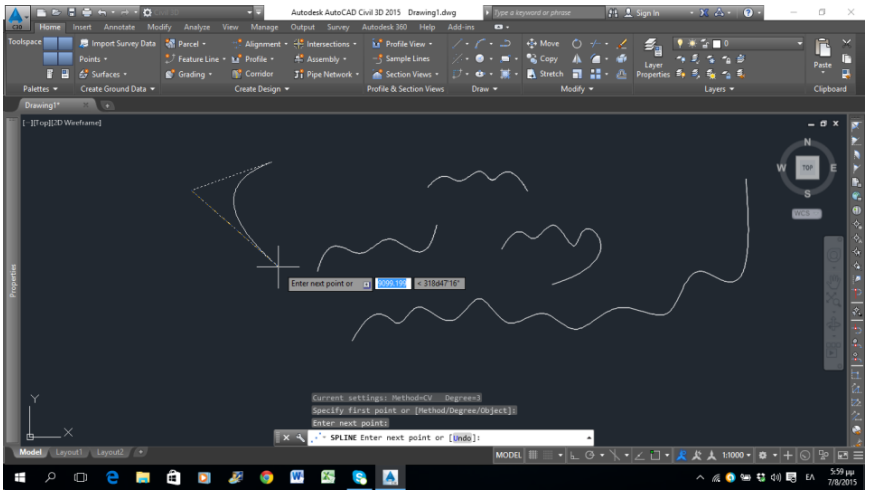


Figure A.15 Definition of coordinates position (created by George Karaiskos with AutoLISP)

If we want to see the position of the control points that describe our spline along with the control net that connects them or the knots that can be found by the name “Fit Points” from the program, we click on the arrow of one spline object and we choose the control Vertices, or knots option, according to our desire.

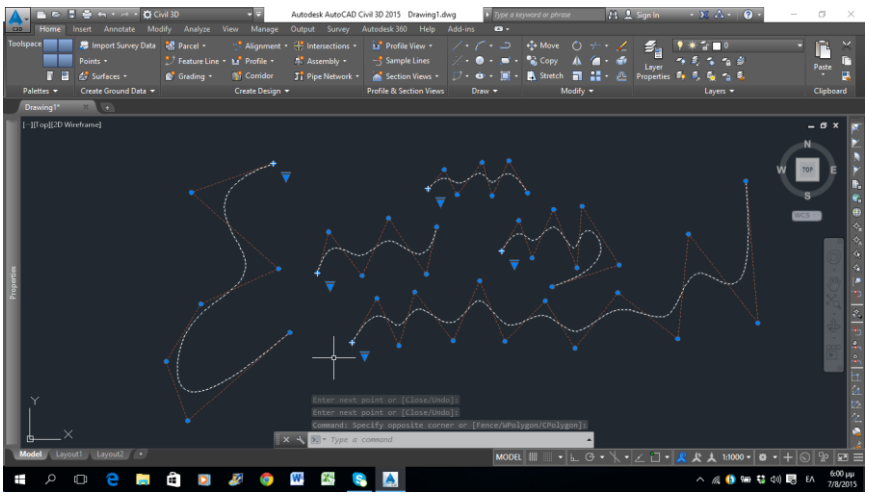


Figure A.16 Position of control points or knots (created by George Karaiskos with AutoLISP)

3. Export Data

Since our spline objects have been drawn we may then export them to a .txt file.

We select our designed spline objects and we type on the command window "Data_SPLine_Export_ShowControlVertices" or we can begin by typing the word "Data…" and our command is showing up as one of the potential commands we may type. We choose our command and we press Enter.

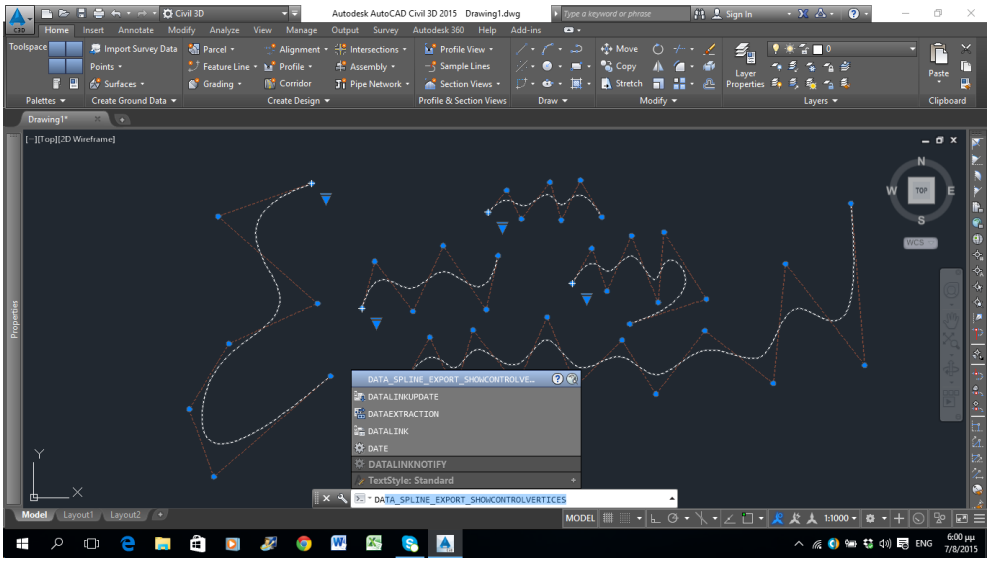


Figure A.17 Data_SPLine_Export_ShowControlVertices (created by George Karaiskos with AutoLISP)

Then a window is showing up in which we must define the name of the .txt file that contains all the data, as also the directory in which the. txt file will be saved. We type our choices and we press Save.

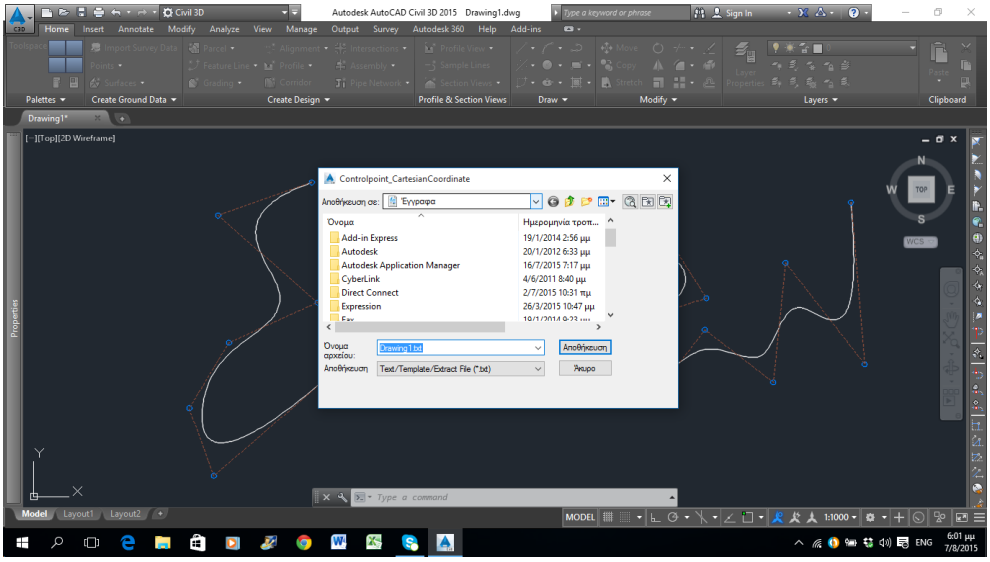
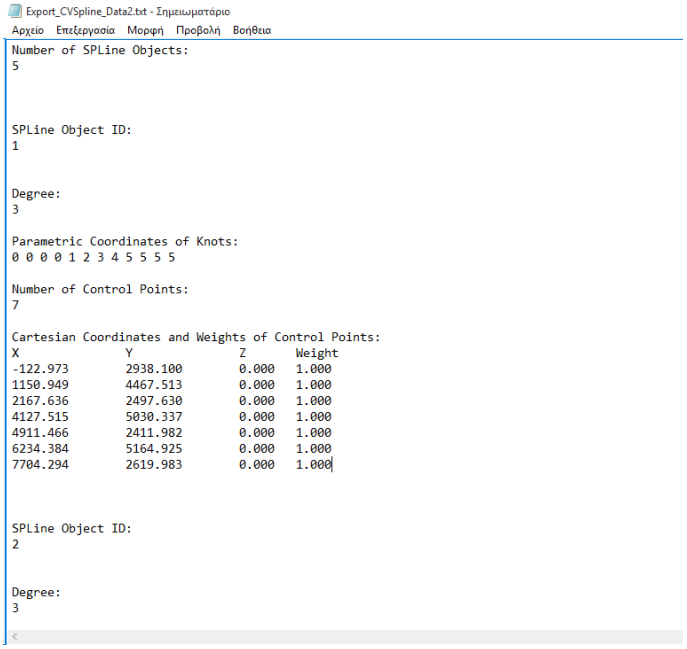


Figure A.18 Saving the .txt file that contains all the data (created by George Karaiskos with AutoLISP)

4. Open exported File Data

After “Save” has been pressed we can maintain our data by opening the txt file that has been created to the chosen directory.

Figure A.19 .txt file in the chosen directory
(created by George Karaiskos with AutoLISP)



Export_CVSpline_Data2.txt - Σημειωματάριο

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια

```
Number of SPLine Objects:
5


SPLine Object ID:
1


Degree:
3

Parametric Coordinates of Knots:
0 0 0 0 1 2 3 4 5 5 5 5

Number of Control Points:
7

Cartesian Coordinates and Weights of Control Points:
X               Y               Z       Weight
-122.973        2938.100        0.000   1.000
1150.949        4467.513        0.000   1.000
2167.636        2497.630        0.000   1.000
4127.515        5030.337        0.000   1.000
4911.466        2411.982        0.000   1.000
6234.384        5164.925        0.000   1.000
7704.294        2619.983        0.000   1.000


SPLine Object ID:
2


Degree:
3
```

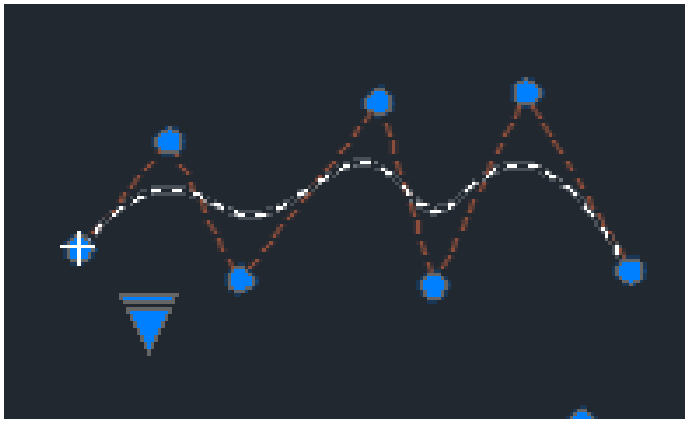
Figure A.20 Data of the .txt file
(created by George Karaiskos with AutoLISP)

A.7 First Version Plug-In Export Example

This is an example of five different spline objects that have been drawn in AutoCAD software application and their data have been exported to a txt file named "Export_CVSpline_Data" using the first version plug-in.

Number of SPLine Objects:						
5						
SPLine Object ID:						
1						
Degree:						
3						
Parametric Coordinates of Knots:						
0 0 0 0 1 2 3 4 5 5 5 5						
Number of Control Points:						
7						
Cartesian Coordinates and Weights of Control Points:						
X		Y		Z		Weight
-122.97		2938.10		0.00		1
1150.95		4467.51		0.00		1
2167.64		2497.63		0.00		1
4127.52		5030.34		0.00		1
4911.47		2411.98		0.00		1
6234.38		5164.93		0.00		1
7704.29		2619.98		0.00		1

(a)

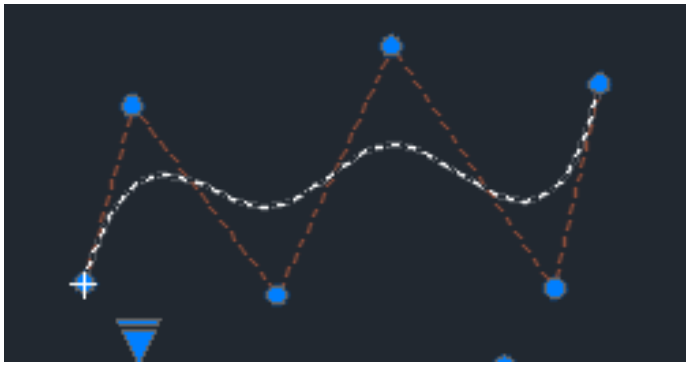


(b)

Figure A.21. Example drawn in AutoCAD. object 1 (created by George Karaiskos with AutoCAD)

SPLine Object ID:						
2						
Degree:						
3						
Parametric Coordinates of Knots:						
0 0 0 0 1 2 3 4 4 4 4						
Number of Control Points:						
6						
Cartesian Coordinates and Weights of Control Points:						
X		Y		Z		Weight
-8814.03		-3657.88		0.00		1
-7945.67		-436.18		0.00		1
-5309.55		-3874.72		0.00		1
-3231.67		648.05		0.00		1
-254.41		-3750.81		0.00		1
551.94		-33.47		0.00		1

(a)

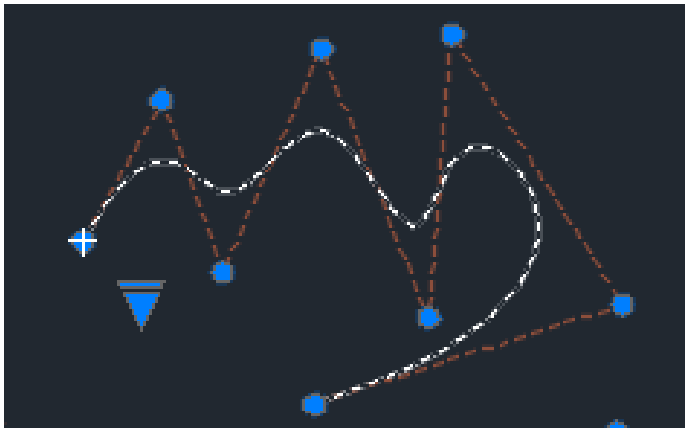


(b)

Figure A.22 Example drawn in AutoCAD object 2
(created by George Karaiskos with AutoCAD)

SPLine Object ID:						
3						
Degree:						
3						
Parametric Coordinates of Knots:						
0 0 0 0 1 2 3 4 5 6 6 6 6						
Number of Control Points:						
8						
Cartesian Coordinates and Weights of Control Points:						
X		**Y**		**Z**		**Weight**
5669.11		-1954.09		0.00		1
7002.67		462.18		0.00		1
8057.12		-2480.72		0.00		1
9762.84		1329.56		0.00		1
11592.62		-3255.16		0.00		1
11995.79		1577.38		0.00		1
14911.02		-3038.32		0.00		1
9638.79		-4742.10		0.00		1

(a)



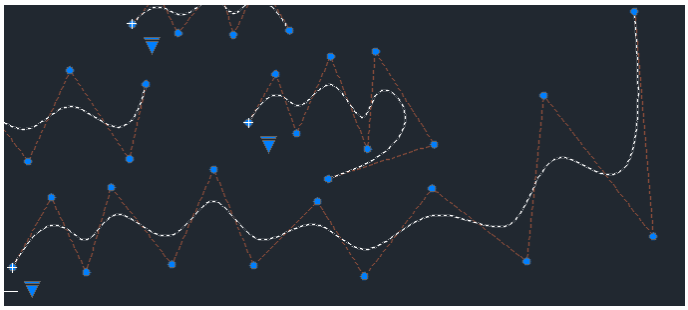
(b)

Figure A.23 Example drawn in AutoCAD object 3 (created by George Karaiskos with AutoCAD)

SPLine Object ID:
4
Degree:
3
Parametric Coordinates of Knots:
0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 12 12 12
Number of Control Points:
14
Cartesian Coordinates and Weights of Control Points:

X	Y	Z	Weight
-6081.33	-9127.71	0.00	1
-4146.85	-5659.51	0.00	1
-2410.78	-9375.43	0.00	1
-1170.73	-5164.06	0.00	1
1855.00	-8979.07	0.00	1
3938.29	-4272.24	0.00	1
5922.37	-9028.61	0.00	1
9096.90	-5857.70	0.00	1
11428.20	-9573.62	0.00	1
14801.14	-5213.60	0.00	1
19513.33	-8830.43	0.00	1
20356.57	-605.86	0.00	1
25812.79	-7591.79	0.00	1
24870.35	3555.97	0.00	1

(a)

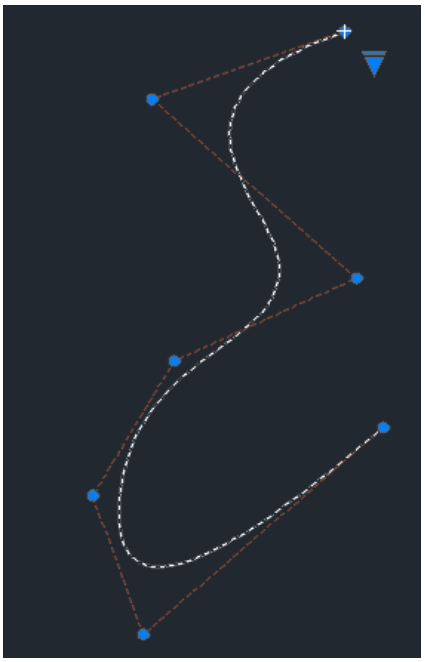


(b)

Figure A.24 Example drawn in AutoCAD object 4 (created by George Karaiskos with AutoCAD)

SPLine Object ID:						
5						
Degree:						
3						
Parametric Coordinates of Knots:						
0 0 0 0 1 2 3 4 5 5 5 5						
Number of Control Points:						
7						
Cartesian Coordinates and Weights of Control Points:						
X		Y		Z		Weight
-12281.59		4943.25		0.00		1
-18729.86		2664.15		0.00		1
-11884.77		-3330.87		0.00		1
-17985.83		-6105.42		0.00		1
-20713.94		-10614.07		0.00		1
-19027.47		-15271.36		0.00		1
-10991.94		-8334.98		0.00		1

(a)



(b)

Figure A.25 Example drawn in AutoCAD object 5 (created by George Karaiskos with AutoLISP)

A.8 Second Version Plug-In Export Example

This is an example of three different spline objects that have been drawn in AutoCAD software application and their data have been exported to a .txt file named "Export_CVSpline_Data" using the second version plug-in.

Number of SPLine Objects:
3

SPLine Object ID:
1

Degree:
3

Parametric Coordinates of Knots:
0 0 0 0 1 2 3 4 5 6 7 8 9 9 9

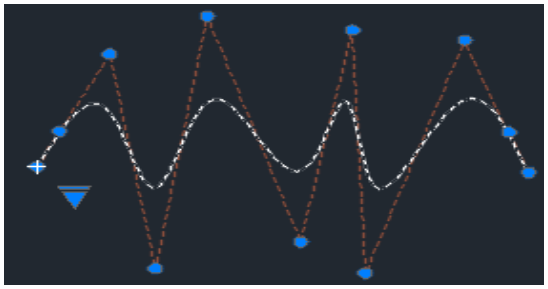
Number of Knots:
9

Cartesian Coordinates of Knots		
X	Y	Z
1746.91	4227.73	0.00
2516.03	5225.31	0.00
3158.88	3883.74	0.00
3905.05	5317.04	0.00
4892.28	4158.94	0.00
5443.30	5294.11	0.00
5822.12	3895.21	0.00
6889.71	5305.58	0.00
7647.36	4136.00	0.00

Number of Control Points:
11

Cartesian Coordinates and Weights of Control Points			
X	Y	Z	Weight
1746.91	4227.73	0.00	1
2020.84	4800.43	0.00	1
2618.30	6049.47	0.00	1
3169.12	2590.11	0.00	1
3791.65	6654.34	0.00	1
4916.35	3013.05	0.00	1
5537.89	6434.51	0.00	1
5684.09	2512.33	0.00	1
6883.55	6269.03	0.00	1
7413.73	4788.44	0.00	1
7647.36	4136.00	0.00	1

(a)



(b)

Figure A.26 Example drawn in AutoCAD object 1 (created by George Karaiskos with AutoCAD)

SPLine Object ID:
2

Degree:
3

Parametric Coordinates of Knots:
0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 11 11 11

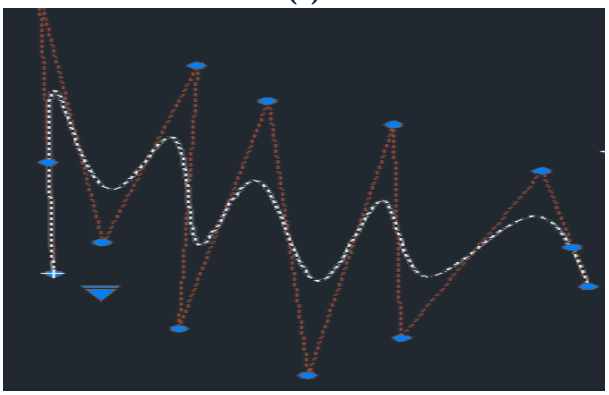
Number of Knots:
11

Cartesian Coordinates of Knots		
X	Y	Z
-3670.90	1111.81	0.00
-3578.68	5459.66	0.00
-3043.81	3304.16	0.00
-2564.26	4593.77	0.00
-2232.27	1848.73	0.00
-1752.72	3488.39	0.00
-1088.73	927.58	0.00
-480.08	2990.96	0.00
-55.86	1038.12	0.00
903.23	2548.81	0.00
1511.89	780.19	0.00

Number of Control Points:
13

Cartesian Coordinates and Weights of Control Points			
X	Y	Z	Weight
-3670.90	1111.81	0.00	1
-3725.50	3953.36	0.00	1
-3807.97	8246.04	0.00	1
-3201.12	1909.28	0.00	1
-2284.81	6433.08	0.00	1
-2457.76	-302.30	0.00	1
-1598.86	5528.26	0.00	1
-1209.89	-1491.24	0.00	1
-376.93	4925.18	0.00	1
-297.98	-535.72	0.00	1
1061.72	3739.12	0.00	1
1359.63	1780.95	0.00	1
1511.89	780.19	0.00	1

(a)



(b)

Figure A.27 Example drawn in AutoCAD. object 2.
(created by George Karaiskos with AutoCAD)

SPLine Object ID:
3

Degree:
3

Parametric Coordinates of Knots:
0 0 0 0 1 2 3 4 5 6 7 8 8 8

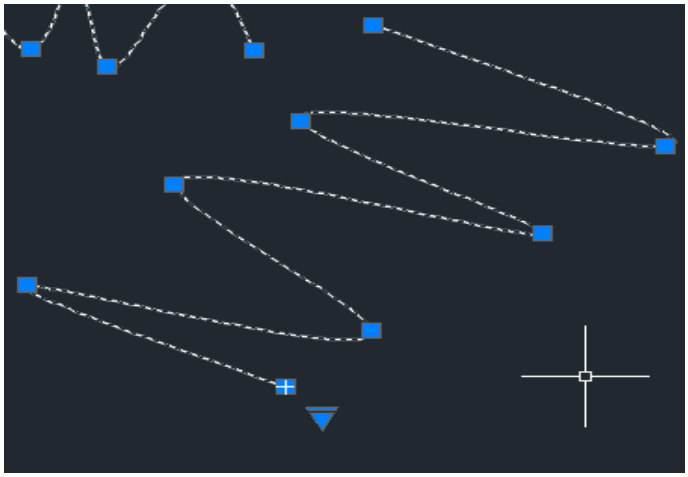
Number of Knots:
8

Cartesian Coordinates of Knots		
X	Y	Z
8016.26	-1080.61	0.00
4847.90	492.51	0.00
9072.38	-210.77	0.00
6663.68	2065.63	0.00
11184.62	1306.83	0.00
8201.54	3046.52	0.00
12703.95	2657.87	0.00
9109.43	4527.11	0.00

Number of Control Points:
10

Cartesian Coordinates and Weights of Control Points			
X	Y	Z	Weight
8016.26	-1080.61	0.00	1
6218.04	-284.29	0.00	1
2242.78	1476.11	0.00	1
12073.80	-1694.20	0.00	1
3371.50	3713.18	0.00	1
14729.15	-94.30	0.00	1
4835.72	4244.69	0.00	1
15667.24	1728.88	0.00	1
11214.38	3628.92	0.00	1
9109.43	4527.11	0.00	1

(a)



(b)

Figure A.28. Example drawn in AutoCAD. object 3.
(created by George Karaiskos with AutoLISP)

After these examples we can verify that our aim for creating a successful plug-in for 1-Dimensional problems has been accomplished. But after several tries and further research I reached to the conclusion that it was so hard to find the appropriate commands for the relative information exploitation of 2-Dimensional and 3-Dimensional problems. So I decided to follow a different path to accomplish my target. I decided to join a research team whose vision was to create a new hybrid software that could be used not only as a pure design application software, but also as a powerful computational engineering tool for solving difficult engineering projects that takes full advantage of all the features of Isogeometric Analysis. This difficult and demanding project led me to object-oriented programming. I chose the appropriate programming language for the realization of graphic imaging and understanding the code of the modern designing tools, that are in fact the heart of Isogeometric Analysis.

References

Published Books and Scientific Papers

[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 fully ‘‘locking-free’’ isogeometric approach for plane linear elasticity problems - Auricchio, Veiga, 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] An introduction to Isogeometric Analysis applied to solid mechanics-Luis F. R. Espatha, Alexandre L. Braunb and Armando M. Awruchb, Argentina, 1-4 November 2011

[16] Forecast and Analysis of the North American Auto Industry Trends through 2007- Office for the Study of Automotive Transportation University of Michigan Transportation Research Institute - 1998

[17] C++ All-in-One For Dummies, 3rd Edition-John Wiley & Sons, Inc., Hoboken, New Jersey- 2014

[18] OpenGL Programming Guide Seventh Edition-Dave Shreiner

The Khronos OpenGL ARB Working Group-2010

[19] Foundations of Qt Development-Johan Thelin-2007

[20] Getting Started with OpenGL-Peter Grogono-2003

Websites

[21] http://amresearch.blogspot.gr/2006/01/in-memoriam-to-professor-john-h.html

[22 http://en.wikipedia.org/wiki/Abaqus

[23] http://en.wikipedia.org/wiki/Nastran

[24] http://www.cervenka.cz/products/

[25] http://www.adina.com/products.shtml

[26] http://en.wikipedia.org/wiki/Femap

[27] http://www.pi.gr/CMSnew/index.php

[28] http://www.sofistik.com/en/

[29] http://lhlogismiki.gr/index_gr.php

[30] http://www.mactech.com/articles/develop/issue_25/schneider.html

[31] http://en.wikipedia.org/wiki/Computer-aided_geometric_design#History

[32] http://isicad.net/articles.php?article_num=14940

[33] http://www.bogotobogo.com/Qt/Qt5_QStatusBar.php

[34] http://sys-exit.blogspot.gr/2013/02/qt-menu-mdi-child-window-example.html

[35] http://www.digitalfanatics.org/projects/qt_tutorial/chapter14.html

[36] http://qt-project.org/wiki/DatabaseConnectionDialog

[37] https://www.opengl.org/discussion_boards/showthread.php/168825-interactive-mouse-drawing-points

[38] http://www.opengl-tutorial.org/beginners-tutorials/tutorial-6-keyboard-and-mouse/

[39] http://www.falloutsoftware.com/tutorials/gl/gl6.htm

[40] http://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm

[41] https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_Examples.html#zz-1.

[42] http://www.ics.com/blog/qt-and-opengl-part-1-loading-3d-model-open-asset-import-library-assimp

[43] http://en.wikibooks.org/wiki/OpenGL_Programming

[44] http://nehe.gamedev.net/tutorial/multiple_viewports/20002/

[45] http://openglsamples.sourceforge.net/projects/index.php/blog/index/

[46] http://www.et.byu.edu/~merk/me570/qtRedExamples.html

[47] http://netization.blogspot.gr/2014/10/loading-obj-files-in-opengl.html

[48] https://forum.qt.io/topic/22389/how-to-make-multiple-qglwidgets-share-the-same-qglcontext/