TUHH
Technische Universität Hamburg

Numerical surface reconstruction with Non-Uniform Rational B-Splines

Computational solution in Matlab and ANSYS APDL

Project Thesis

for the course of studies Schiffbau and Meerestechnik Master of Science (M. Sc.)
Institute for Ship Structural Design and Analysis (M-10)
Technical University Hamburg

in cooperation with thyssenkrupp Marine Systems GmbH

Written and submitted by: Fabian Krohe
Professor: Prof. DSc. (Tech.) Sören Ehlers
Supervisor from University: Moritz Braun
Supervisor from Company: Josef Pollmanns

Tuesday, 4th August 2020
Hamburg-Harburg

Abstract

Fatigue assessment of cycle loaded components is very important for strength calculations in the shipbuilding sector of steel structures and welded joints, especially when it comes to low cycle fatigue problems with constant plastic deformation with every of the up to several thousands of load cycles. Results can be obtained via experimental data from test specimen as well as from numerical fatigue analysis methods which require computer aided design geometries. Therefore, the modelling of such structures and welds is crucial. Mostly, such assessments are done with models made from the idealized geometries, but they do not take manufacturing imperfections into account and deviations for the numerical fatigue analysis approaches compared to results from experimental tests can occur. This leads to the usage of models made from the actual geometries which can be achieved by generating 3D scanned scatter plots out of the real structures and welds. The conversion from the raw scatter plot to an usable computer aided design geometry is part of the topic of surface reconstruction and describes the process of converting polygon areas into free-form surfaces like non-uniform rational b-splines (NURBS).
This project thesis presents an open and flexible numerical approach of surface reconstruction with NURBS both in the mathematical program Matlab and the programming language ANSYS APDL. An overview about the scientific state of the art of free-forms in fatigue assessments and the influence of the optical measurement technique laser is given. Following, the theoretical principles of parametric free-form descriptions, Bézier curves and surfaces, B-splines and NURBS are explained. At the end, the results show a robust and open numerical approach of surface reconstruction and further investigations in terms of continuation of direct computations of free-forms in ANSYS APDL as well as subsequent topcis like geometry accuracy of 3D scanned components are discussed.

Acknowledgement

I want to thank my professor from the Technical University Hamburg, Prof. Sören Ehlers, for hosting my project thesis in his Institute for Ship Structural Design and Analysis (M-10) and allowing my to work on this topic.

I also want express my gratitude especially to my supervisors Moritz Braun from the Technical University Hamburg, where i graduate at the moment, and Josef Pollmanns from the shipyard and system provider for submarines thyssenkrupp Marine Systems GmbH Kiel, where i am working as dual student. Both of them showed incredible patience when i was ill for several month and could not make any progress. Their input and help was key to get the project thesis done.

The same counts for my group leader at work, Gerrit Herforth, who backed me up during the more difficult phases of the work and giving me the needed time to finish my project thesis.

Special thanks go to Jürgen Sundermeyer who helped me a lot with tips and tricks about ANSYS APDL when i was stuck with programming my macro.

Declaration of Authorship

I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgement, the work presented is entirely my own. I am aware of the University's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism.

Date: Tuesday, 04th August 2020
Location: Hamburg-Harburg
Name: Fabian Krohe

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Motivation . . . 1
- 1.3 Structure . . . 2

2 State of the art **3**

3 Theoretical principles **5**
- 3.1 Free-form curves and surfaces . . . 5
- 3.2 Bézier . . . 6
  - 3.2.1 Bézier curves . . . 6
  - 3.2.2 Properties of Bézier curves . . . 7
  - 3.2.3 Bernstein polynomials . . . 8
  - 3.2.4 De Casteljau algorithm . . . 8
  - 3.2.5 Composite Bézier curves . . . 9
  - 3.2.6 Bézier surfaces . . . 9
- 3.3 B-splines . . . 10
  - 3.3.1 B-spline curves . . . 10
  - 3.3.2 Properties of B-splines . . . 11
  - 3.3.3 Knot vector . . . 11
  - 3.3.4 B-spline surfaces . . . 12
- 3.4 Non-Uniform Rational B-Splines (NURBS) . . . 13
  - 3.4.1 The NURBS curve . . . 13
  - 3.4.2 Properties of NURBS curves . . . 14
  - 3.4.3 The NURBS surface . . . 14
- 3.5 Short overview about differences in the application . . . 14

4 Numerical approach of free-form surface determination **15**
- 4.1 Manual determination of NURBS curve and surface . . . 16
- 4.2 Numerical NURBS computation with Matlab . . . 19
- 4.3 Numerical NURBS computation with ANSYS APDL . . . 22

5 Discussion **26**

6 Conclusion **28**

A Complete Matlab Code **31**

B Complete ANSYS APDL Macro **34**

C Additional NURBS surface plots with Matlab **38**

List of Figures

2.1 Surface reconstruction from 3D scanned geometry to NURBS surfaces . . . 3
3.1 Parametric curve shaped as helix . . . 5
3.2 Parametric surface with isoparametric curves, intersection point and diagonals . . . 6
3.3 Bézier curve with four points P_0 to P_3 and its control polygon . . . 7
3.4 De Casteljau algorithm with b_0^3 obtained from repeated linear interpolation . . . 9
3.5 Composite Bézier curve . . . 9
3.6 Doubled de Casteljau method for Bézier surface . . . 10
3.7 Recursive determination of B-spline base functions . . . 10
3.8 B-spline curve with influence of control points on the shape . . . 11
3.9 Influence of the curve order to the shape . . . 12
3.10 Non-uniform rational B-spline with influence of weight w_3 . . . 13
3.11 Base functions for cubic NURBS curve . . . 14
4.1 Control grid with nine control points . . . 15
5.1 NURBS surface point plot from Matlab for 20 intervals in both parametric directions . . . 26
5.2 NURBS surface triangle plot from Matlab for 20 intervals in both parametric directions . . . 27
5.3 NURBS surface point plot from ANSYS APDL for 20 intervals in both parametric directions 27
5.4 Influence on curve shape with changing the weight from 1 over 2 to 5 . . . 28
C.1 NURBS surface for U_2 and V_2 with $w_{2,2} = 1$ . . . 38
C.2 NURBS surface for U_2 and V_2 with $w_{2,2} = 3$ . . . 38
C.3 NURBS surface for U_2 and V_2 with $w_{2,2} = 10$ . . . 39
C.4 NURBS surface for U_3 and V_3 with $w_{2,2} = 1$ . . . 39
C.5 NURBS surface for U_3 and V_3 with $w_{2,2} = 3$ . . . 40
C.6 NURBS surface for U_3 and V_3 with $w_{2,2} = 10$ . . . 40

List of Tables

5.1 Comparison of B-spline base function results for manual and numerical approach . . . 26

Abbreviations

$\mathbb{N}$	mathematical natural numbers
$\mathbb{P}_n$	vector span of the real polynomials of degree n
$b_{i,n}(t)$	i-th Bernstein polynomial with degree n
$b_{j,m}(t)$	j-th Bernstein polynomial with degree m
$B(t)$	Bézier curve
$B(u,v)$	Bézier surface
$C(u)$	NURBS curve in u-direction
$C(v)$	NURBS curve in v-direction
$f(t)$	function of parametric coordinate t along the x-axis for a parametric curve formulation
$f(x)$	function of cartesian coordinate for an explicit formulation of a free-form curve
$f(x,y)$	function of cartesian coordinates x and y for an implicit formulation of a free-form curve
$f(x,y,z)$	function of cartesian coordinate x, y and z for an explicit formulation of a free-form surface
$g(t)$	function of parametric coordinate t along the y-axis for a parametric curve formulation
$h(t)$	function of parametric coordinate t along the z-axis for a parametric curve formulation
i	index
j	index
k	number of control points in u-direction
K_t	stress concentration factor
l	number of control points in v-direction
m	degree of base functions in u-direction
n	degree of base functions in v-direction
$N_{i,n}$	i-th B-spline base function with degree n
$N_{j,m}$	j-th B-spline base function with degree m
P_i	i-th control point
$P(t)$	B-spline curve
$P(u,v)$	B-spline surface
$R_{i,j}(u,v)$	rational basis function
$S(u,v)$	NURBS surface over control grid in u- and v-direction
t	axis independent parametric coordinate
u	parametric coordinate along the x-axis
U	knot vector in u-direction
v	parametric coordinate along the y-axis
V	knot vector in v-direction
w	parametric coordinate along the z-axis
x	cartesian coordinate along the x-axis
$x(u,w)$	function of parametric coordinates u and w in direction of x for a parametric surface formulation
y	cartesian coordinate along the y-axis
$y(u,w)$	function of parametric coordinates u and w in direction of y for a parametric surface formulation
z	cartesian coordinate along the z-axis
$z(u,w)$	function of parametric coordinates u and w in direction of z for a parametric surface formulation

1 Introduction

In the following chapter, the project thesis and the background for it is shortly introduced. After that, the motivation for the thesis is explained and which approach was chosen to deal with the given problem. The chapter closes with an overview of the following structure of the project thesis.

1.1 Background

For the construction and calculation of the welding from highly stressed out structures due to vibrational loads, fatigue analysis plays a decisive role. Three different forms of fatigue exist: high cycle fatigue (HCF), low cycle fatigue (LCF) and thermal mechanical fatigue (TMF), see [6]. High cycle fatigue can be characterized by low amplitude high frequency elastic strains, while low cycle fatigue is describable by high amplitude low frequency plastic strains. Thermal mechanical fatigue occurs due to large temperature changes resulting in significant thermal expansion and contraction but will not be targeted here any further.
In the shipbuilding sector of steel structures and welding high and low cycle fatigue are the most common types. While the hull of a surface ship is determined under the influence of HCF design criteria for load cycles coming from the sea motion behaviour, other components additionally have to withstand cyclic loads of several thousand application procedures. Under the perspective of strength calculation considerations it would be eligible to design these components completely with loads not exceeding the yield point of the used material, but the economic efficiency would be decreased massively with oversizing the whole structure. That is the reason why it is important to find the optimal point of intersection between strength calculations, fatigue analysis and economical aspects.
When a component or a welded structure is designed for low cycle fatigue, with every new load cycle macroscopic plastic deformations occur at so-called fatigue critical hot-spots like cut-outs, sharp edges and lead-throughs even when the rest of the structure is designed for high cycle fatigue. These plastic deformations lead to fatigue failure after a relatively low number of load cycles. In contrast to such components build in mechanical engineering terms, which are constructed for HCF and therefore seeing load cycles with an occurrence of $10^5...10^6$ and also higher when it comes into the endurance range, low cycle loaded structures are constructed for LCF with around $10^1...10^4$ load cycles. Therefore, LCF criteria play a very important role for dimensioning regarding such structures. This requires a relatively precise fatigue analysis of all the regarding components.

1.2 Motivation

Normally, the fatigue analysis for cyclic loaded components are done with a model made from the idealized computer aided design geometry, but ideally, they are done with a model made from the actual geometry because it has manufacturing imperfections. If the desired target geometry is the idealized and not the actual one, deviations for the fatigue analysis results can occur and lead to adulteration compared to results from experimental fatigue tests. Therefore, the idea is to generate 3D scanned scatter plots from the components to build a model which then can be used for fatigue analysis without sacrificing the consideration of imperfections from the actual geometry. This leads to the topic of surface reconstruction, which is part of the so-called reverse-engineering. Surface reconstruction describes the process of converting polygon areas into free-form surfaces like NURBS, see [18].
Hence, the goal of the project thesis is to explain different mathematical principles of free-form curves and surfaces as well as to create a numerical approach of surface reconstruction. Since commercial computer aided design programs like Rhino3D, see [10], or Vectorworks, see [24], are not open source it is not possible to directly manipulate the computation of free-forms which leads to the demand of creating an open and flexible method working with the most used software for strength and fatigue calculations, ANSYS Mechanical and Workbench. So the project thesis provides all necessary basics of free-form theory and surface reconstruction.

1.3 Structure

The project thesis contains four upcoming chapters and two appendixes with the following topics:

- Chapter 2 "State of the art" puts the numerical approach of surface reconstruction for fatigue assessment presented in this project thesis into relation with other scientific works and studies to compare how other dealt with the same problem and to check what is today's state of the art. Also the accuracy of optical measurement systems is shortly discussed.
- Chapter 3 "Theoretical principles" explains the basic mathematical theory of free-forms, Bézier curves and surfaces, B-splines and non-uniform rational B-splines (NURBS), which are the used as foundation for the free-form generation.
- Chapter 4 "Numerical approach of free-form surface determination" contains the implementation of a numerical computation method for surface reconstruction with creating a NURBS surface out of a given scatter plot. Three different steps are done: A manual approach calculating the NURBS per hand, a numerical approach with the mathematical program Matlab and a numerical approach with the programming language ANSYS APDL.
- Chapter 5 "Discussion" displays the results from the numerical approaches and shows plots for both Matlab and ANSYS APDL. They are verified via comparison with the manual approach.
- Chapter 6 "Conclusion" summarize the project thesis, closes the the work with framing it with the state of the art and makes assumptions about future scientific studies regarding the topic of an open numerical approach of surface reconstruction.
- Appendix A "Complete Matlab Code" contains the complete numerical approach made in the mathematical program Matlab.
- Appendix B "Complete ANSYS APDL Macro" contains the complete numerical approach made in the programming language ANSYS APDL.
- Appendix C "Additional NURBS surface plots with Matlab" shows additional plots of NURBS surfaces with other input parameters to visualize the influence of the chosen knot vectors and weighting factors.

2 State of the art

Most of the existing studies and scientific works dealing directly with the computational determination of free-form curves and surfaces like Bézier, B-Splines or NURBS are made under computer graphic view points, see for example [3], [14] and [20]. Studies about fatigue analysis of actual geometries focus more on the description of fatigue approaches and theories while surface reconstruction approaches are not explained en detail. Markus Ladinek, Robert Lang, Gerhard Lener, et al. are leading engineers and scientists in the topic of the usage of free-forms for fatigue analysis of actual geometries from test specimen. Subsequently, three scientific papers from these authors are shown as state of the art:

- In the paper "Application and comparison of deterministic and stochastic methods for the evaluation of welded components' fatigue lifetime based on real notch stresses" from Lang and Lener, see [12], two advanced methods for modelling the fatigue lifetime of welded components with an irregular distributed geometry are presented: A deterministic method in order to analyse implicit gradient models and a Weibull-based model using a stochastic method. The parameters for both models are determined with an examination of specimens where the geometry of the welded structure was measured with a laser scanning system with high accuracy and resolution. The captured points from the surfaces of the specimens were triangulated into meshes followed by a NURBS-generation to get rid of scan defects. After that, the obtained geometry was transformed into surfaces with C1-continuity and the linear-elastic notch stresses on the real geometries of each specimen were calculated using FEM. The preprocessing from the 3D scanned component as scatter plot over the triangulation towards the NURBS generation is shown in figure 2.1.
- In the paper "A numerical method for determining the fatigue strength of welded joints with a significant improvement in accuracy" from Lener, Lang, et al., see [13], the same two approaches with an implicit gradient model using a deterministic method and a Weibull-based model using a stochastic method are presented and used to examine fatigue strength of actual specimen geometries welded with different welding processes and on different positions. Therefore, the same surface reconstruction determination process like in [12] was used.
- In the paper "The strain-life approach applied to welded joints: Considering the real weld geometry" from Ladinek, Niederwanger, Lang, et al., see [11], the combination of the real weld geometry obtained by 3D laser scanning and the strain-life approach is investigated. The effect of additional stress concentration from the weld profile itself was studied leading to predictions of fatigue lifetime. The strain-life concept, different methods for mean stress and plasticity correction were also taken into account. The same surface reconstruction determination process like in [12] and [13] was used while the geometry then was meshed with SOLID187 tetrahedral elements with quadratic displacement behaviour to create an volume model.

These papers published between 2016-2018 reveal the advantages of pre-processing 3D scanned scatter plots by using NURBS surfaces as free-form approach, so NURBS are highlighted in the project thesis as solution for an open and flexible surface reconstruction method.

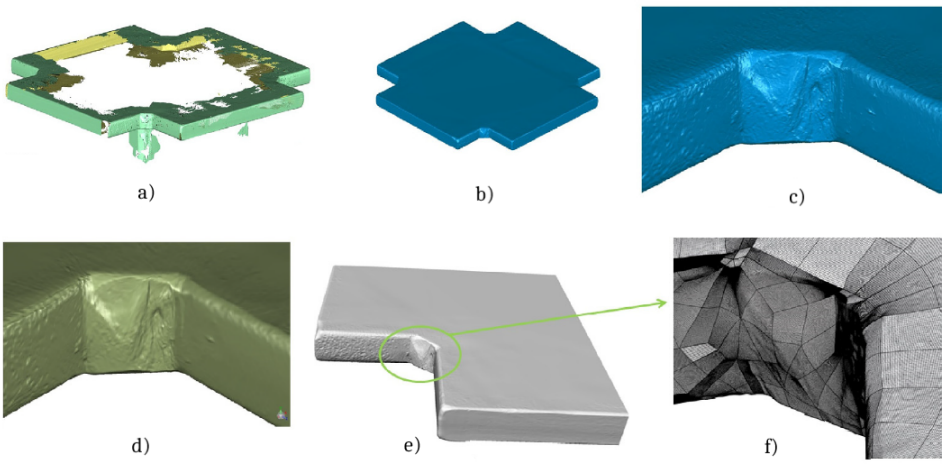

Figure 2.1: Surface reconstruction: a) 3D scanned geometry, b) and c) triangulated mesh, d) generated NURBS, e) and f) closed NURBS surfaces, [12]

Another important topic, which is directly coupled thematically with the surface reconstruction process, is the accuracy evaluation of optical measurement methods like lasers. Exemplary, two scientific studies are presented here about the current research status:

- In the paper "Estimation of fatigue in welded joints based on laser scanning - Correlation between weld quality and fatigue life" from Hultgren and Barsoum, see [9], examined a method based on an experimental analysis to determine the location of fracture initiation for non-load carrying fillet welds which were scanned before. The method was tested with 119 specimen and the fracture surfaces were investigated to find the locations for most probably crack initiation points. The study came to the results, that about 80% of the points of initiation could be determined for the fracture surfaces, that the locations with distance combinations that yield better results with higher correctness levels predicted the initiation locations with a hit rate above 90% and that the leg length has the largest contribution to the proposed algorithm closely followed by the weld toe angle.
- In the paper "Influence of the optical measurement technique and evaluation approach on the determination of local weld geometry parameters for different weld types" from Schubnell, Jung, Le, et al., see [21], different evaluation algorithms and 3D-measurement systems were compared because the influence of the optical measurement system in terms of geometrical accuracy and lateral resolution was not quantified yet. Weld toe radii and flank angles were taken into account for calculating the stress concentration factors which were also compared with the ones determined by formulas. The study came to the results, that the assessment of the weld toe radius is limited by the resolution of the measurement system and has to be mapped with at least 3 points while a weld toe radius of at least 0.1 mm can be assessed with a comparable small relative error, that the flank angle could be measured with a comparably low scatter of 8% or less between the different evaluation methods and that the curvature method leaded to a negligible overestimation of the stress concentration factor K_t of 1.6% in average and the optimization method to an underestimation of K_t of -11.1% in average while single values for both methods showed significant deviation.

These papers published in 2019 reveal the necessity of taking the accuracy of optical measurement methods into account as well as to observe the precise generation of free-forms according to the resolution of the 3d scanned scatter plots. Even if a geometry is laser scanned it does not automatically ensure a more exact fatigue assessment result.

3 Theoretical principles

In the following chapter, the basic principles of free-form curves and surfaces will be presented. The theory of Bézier curves and surfaces as well as B-splines will be used to get to the non-uniform rational B-splines (NURBS), which will be used as foundation for the implementation of free-form generation in both the Matlab and ANSYS APDL code.

3.1 Free-form curves and surfaces

When it comes to the mathematical description of any type of geometrical body or shape, free-form curves and surfaces are the current standard in computer aided design. Typical examples in the industry are automobile bodyworks, aircraft wings and fuselage, ship hulls and daily items like bottles, shoes and household articles, see [19]. Since most of these free-forms can not be described analytically they have to be spanned with given control points in the 2-dimensional space in one direction for curves or in the 3-dimensional space in two directions for surfaces and then can be interpolated or approximated with different algorithms. If a free-form is interpolated, one or multiple polynomials with minimal degree are searched during this process which run through all given points. The resulting curve or surface subtends all control points. If a free-form is approximated, the resulting curve or surface will not run through (all) the control points and therefore has to be manipulated with the position of the given points.
Free-form curves and surfaces can be represented either explicitly, implicitly or parametric, see [19]:

- The form $y = f(x)$ describes the explicit representation of a free-form curve but it lacks the ability of displaying multiple-valued functions (e.g. surfaces) and using constraints with an infinite derivative. Also they can not display vertical lines and circles, [2]. Therefore, they are mostly not used in computer graphics or computer aided design and will not be discussed furthermore.
- The form $f(x, y) = 0$ for free-form curves and $f(x, y, z) = 0$ for free-form surfaces describes the implicit representation and, which already can be seen in the second formula, allows to display multi-valued functions. These are however still axis dependent like for the explicit representation but implicit curve and surface descriptions provide a much more robust formulation and have still applications in computer graphics and computer aided design today.
- Nevertheless, the most common and flexible way of describing free-form curves and surfaces is the parametric form. Bézier curves and surfaces as well as rational and nonrational B-splines, which are explained in the following subsections, are all parametrically represented. The parametric form ensures axis independence, they can easily represent multiple-valued functions and infinite derivatives and have additional degrees of freedom compared to the other two free-form formulations, see [19].

Parametric curves can be represented as

$$x = f(t); \quad y = g(t); \quad z = h(t) \tag{1}$$

with t as parameter. The functions $f(t)$, $g(t)$ $h(t)$ are parametric polynomials.

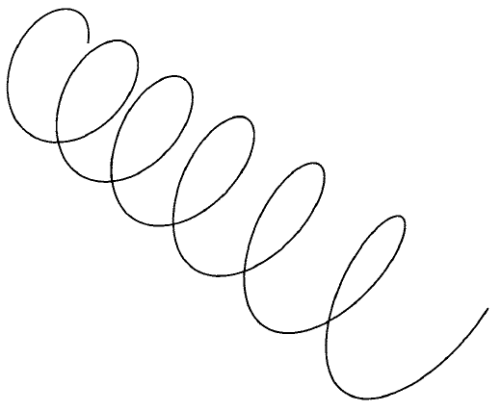

Figure 3.1: Parametric curve shaped as helix, [19]

Parametric surfaces can be represented as

$$x = x(u, w); \quad y = y(u, w); \quad z = z(u, w) \tag{2}$$

with u and w as parameter. So a free-form surface with this formulation is biparametric. If one of these parameters is constant, this describes an isoparametric curve on the surface. At the point of intersection of two isoparametric curves both parameters have to be equal. This can used to describe a single point on a parametric surface. Figure 3.2 shows such curves for u = constant and w = constant with the regarding intersection point and additional diagonals with the needed requirements for the parameters.

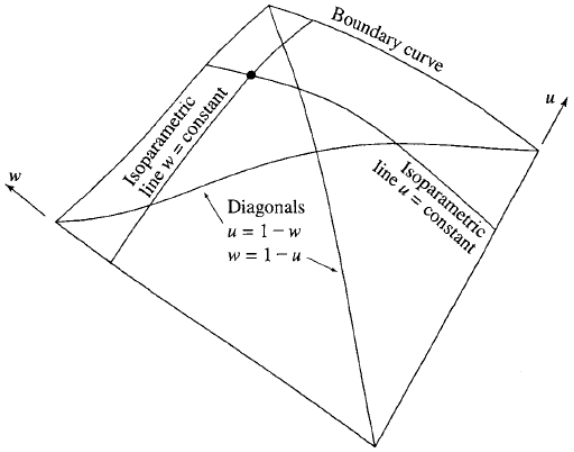

Figure 3.2: Parametric surface with isoparametric curves, intersection point and diagonals, [19]

3.2 Bézier

With the introduction of Bézier curves, splines and surfaces, the mathematical description and industrial usage of parametric shapes in computer graphics and related fields made a huge step forward. Developed by French engineer Pierre Bézier as well as physicist and mathematician Paul de Casteljau, these parametric curves were used during the 1960s for shape forming the bodywork of cars, see [7]. The mathematical basis for Bézier curves are the Bernstein polynomials which were already well-known since 1912 but were not used for such an area of application until the debut of the Bézier curves. While the original mathematical basis came from geometrical considerations, the conformity of these results with the Bernstein polynomials was later proofed, see [19].
These parametric curves were named after Pierre Bézier, who worked for French automaker Groupe Renault and was instrumental in widely publishing the system engineering behind it. Though, the first studies evaluating Bézier curves were created by Paul de Casteljau in 1959 for another French automobile manufacture, in this case Citroën S.A., using his self-made numerically stable de Casteljau algorithm, see [7]. Due to the fact, that Bézier curves are relatively simple to implement mathematically spoken, today they are mostly used for computer aided design to model smooth curves, graphic applications like animations and user interfaces, vector graphics like the creation of illustrations and fonts and for programming motion trajectories in robotics to avoid choppy or unnecessary movement.

3.2.1 Bézier curves

The Bézier curve $B(t)$ is a parametric modelled curve, which is determined through $n + 1$ control points named $P_0, ..., P_n$, which determine the shape of the curve. n is the regarding order, where $n = 1$ specifies a linear shape, $n = 2$ a quadratic shape and so on, see [7]. When connecting the set of control points with lines, it results in a control polygon (also called Bézier polygon), which frames the Bézier curve with its convex hull. Withal, the first and last entry are always the end points of the curve and therefore are lying on it but the intermediate control points are generally not distributed on the curve. The progression of the curve can be modified by changing the control points. These changes contain the shifting, deleting and adding of such points, see [3].

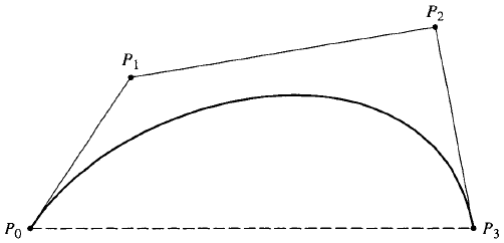

Figure 3.3: Bézier curve with four points P_0 to P_3 and its control polygon

The general definition of Bézier curves can be formulated in two different ways: The recursive and the explicit definition, see [2] and [7].

Recursive definition
First, the recursive definition is shown. A Bézier curve of any degree n can be expressed as a linear interpolation of a pair of corresponding points in two Bézier curves of degree $n-1$, which can be described as a point-to-point linear combination. If $B_{P_0,P_1,...,P_n}(t)$ is the Bézier curve generated by the control points $P_0, P_1, ..., P_n$, then the recursion can be written as:

$$\begin{aligned} &B_{P_0}(t) = P_0, \text{ and} \\ &B(t) = B_{P_0,P_1,...,P_n}(t) = (1-t)B_{P_0,P_1,...,P_{n-1}}(t) + tB_{P_1,P_2,...,P_n}(t) \end{aligned} \tag{3}$$

Explicit definition
Second, the explicit definition is shown. Compared to the recursive expression, the Bézier curve here is determined via binomial coefficients $\binom{n}{i}$, which is equal to:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{4}$$

Now, the formula for the Bézier curve is:

$$\begin{aligned} B(t) &= \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} P_i \\ &= \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} P_i \qquad (0! \equiv 1) \\ &= (1-t)^n P_0 + \binom{n}{1} t(1-t)^{n-1} P_1 + ... + \binom{n}{n-1} t^{n-1}(1-t)P_{n-1} + t^n P_n \qquad 0 \le t \le 1 \end{aligned} \tag{5}$$

Out of this formulation, the Bernstein polynomials can be determined.

3.2.2 Properties of Bézier curves

Due to the fact, that Bézier curves are based on Bernstein polynomials, several properties for Bézier curves immediately can be taken over and are listed here, see [7], [18] and [19]:

- Every Bézier curve begins with control point P_0 and ends at P_n, which is called endpoint interpolation property. These first and the last control points are always lying on the curve.
- Only when all control points are collinear the Bézier curve will form a straight line. In any other case the curve will have a different shape then a straight line.
- The tangent vectors of the starting and finishing points correspond to the connection between the first and second, respectively the penultimate and the last control point.
- Every Bézier curve can be split into two or more subcurves. Then every subcurve is also a Bézier curve.

- The Bézier curve is enframed from the convex envelope of the control polygon which is biggest polygon that can be created through connecting of the control points. In general, the curve follows the direction of the control polygon.
- A single Bézier curve can not be used to shape certain geometrical forms like a circle for example. To achieve that, composite Bézier curves are needed.
- Bézier curves are invariant against affine transformations. Therefore an affine transformation (like scaling, rotation, translation) of a Bézier curve can be done with the transformation of the control polygon.
- The base functions $b_{i,n}(t)$ are real.
- The sum of the base functions are equal one for every given parameter t:

$$\sum_{i=0}^{n} b_{i,n}(t) \equiv 1 \tag{6}$$

- The degree of the Bernstein polynomial corresponds to the number of control points minus one.
- A Bézier curve of an order higher than two may intersect itself or need certain control points.

3.2.3 Bernstein polynomials

The polynomial $b_{i,n}$ is named i-th Bernstein polynomial with degree n, see [7]:

$$\begin{aligned} b_{i,n}(t) &= \begin{pmatrix} n \\ i \end{pmatrix} t^i (1-t)^{n-i} \\ &= \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} \qquad (0! \equiv 1) \end{aligned} \tag{7}$$

Looking back to the explicit definition, every point P_i of the Bézier curve is weighted with one Bernstein polynomial $b_{i,n}$. The Bézier curve then can be written as:

$$B(t) = \sum_{i=0}^{n} b_{i,n}(t) P_i \qquad 0 \leq t \leq 1 \tag{8}$$

All Bernstein polynomials for $0 \leq t \leq 1$ are unequal zero, therefore all control points in this interval influence the shape of the Bézier curve. The Bernstein polynomials act like base functions for the Bézier curve, while the number of control points has an impact on the degree of the Bernstein polynomials. The degree is always equal to the number of control points minus one.
For the computation of Bézier curves the recursive characteristics play an important role, because the de Casteljau algorithm builds on it. If $t \in [0,1]$:

$$\begin{aligned} b_{n,0}(t) &= (1-t) b_{n-1,0}(t) \\ b_{n,i}(t) &= t b_{n-1,i-1}(t) + (1-t) b_{n-1,i}(t) fri = 1...(n-1) \\ b_{n,n}(t) &= t b_{n-1,n-1}(t) \end{aligned} \tag{9}$$

For a fix $n \in \mathbb{N}$ the Bernstein polynomials $b_{n,i}$ for $i = 0...n$ form a base in the vector space $\mathbb{P}_n$ of the real polynomials of degree n.
With this property it is possible to depict every arbitrary polynomial of degree n as linear combination of Bernstein polynomials, see [3]. Thus Bézier curves can be parametrized with given points. Also it is valid nevertheless that all Bernstein polynomials are positive inside the interval $[0,1]$.

3.2.4 De Casteljau algorithm

The De Casteljau algorithm is based on the recursivity of the Bernstein polynomials. It is assumed that a point P_0^n on the Bézier curve, which is dependent from t, can be calculated stepwise via the amount of the control points $P_0, ..., P_n$. Thus two consecutive points influence each other. To avoid the extensive evaluation of the Bernstein polynomials, Paul de Casteljau designed an iteration method which approximates the needed point on the curve with the repeatedly generation of partial ratios of the distance between points,

see [7]. This is done with the following formula, where $k = 0, ..., n$ is the particular iteration step (or iteration depth):

$$B(t)_i^{k+1} = (1-t)B(t)_i^k + tB(t)_{i+1}^k \tag{10}$$

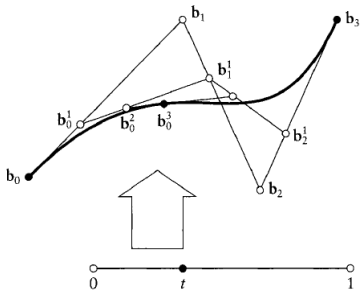

Figure 3.4: De Casteljau algorithm with b_0^3 obtained from repeated linear interpolation, [7]

The de Casteljau algorithm is the theoretical foundation for the base function calculation for B-splines and NURBS, like shown in the subchapter 3.3 and 3.4. For further information about the de Casteljau algorithm and the mathematical ideas behind it, see [7].

3.2.5 Composite Bézier curves

A composite Bézier curve is a piecewise, at least continuous curve defined by a composition of Bézier curves where the starting point of the first and the ending point of the last Bézier curve have equal coordinates, see [7] and [17]. The so completely closed path is called beziergon (or bezigon). In some sources, authors name such a curve also Bézier spline because it has a knot vector format like a B-spline, which is described further on in subchapter 3.3.3. Composite Bézier curves are mostly used for closed outlines and shapes like in modern vector graphics and computer fonts. The mathematical fundamentals are the same as for the normal Bézier curve. Depending on the application, additional smoothness requirements (such as C1 or C2 continuity) can be added. For example, C2 continuous composite cubic Bezier curves are actually cubic B-splines, which are described later in this project thesis, and the other way round.

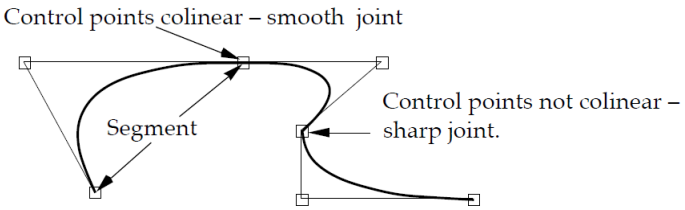

Figure 3.5: Composite Bézier curve, [17]

3.2.6 Bézier surfaces

The Bézier surface $B(u, v)$ of degree (n, m) can be calculated like the Bézier curve but here the parameter space is 2-dimensional and the surface is mapped over a set of control points $P_{i,j}$, see [19]. With the Bernstein polynomials $b_{i,n}(u)$ and $b_{j,m}(v)$ as base functions and the parametric coordinates u and v, the Bézier surface is defined as:

$$B(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} b_{i,n}(u) b_{j,m}(v) P_{i,j} \tag{11}$$

The Bézier surface can be computed via a doubled de Casteljau method, which is visualized in figure 3.6. Therefore it is done first in u-direction and then in v-direction with the points from the first run. Like for the curve, the Bézier surface will lie completely within the convex hull of its control points, while the corner points are equal to the four control points.
Typical Bézier surfaces are bicubic patches ($n = m = 3$) with 16 control points, biquadratic patches ($n = m = 2$) with 9 control points and Bézier triangles.

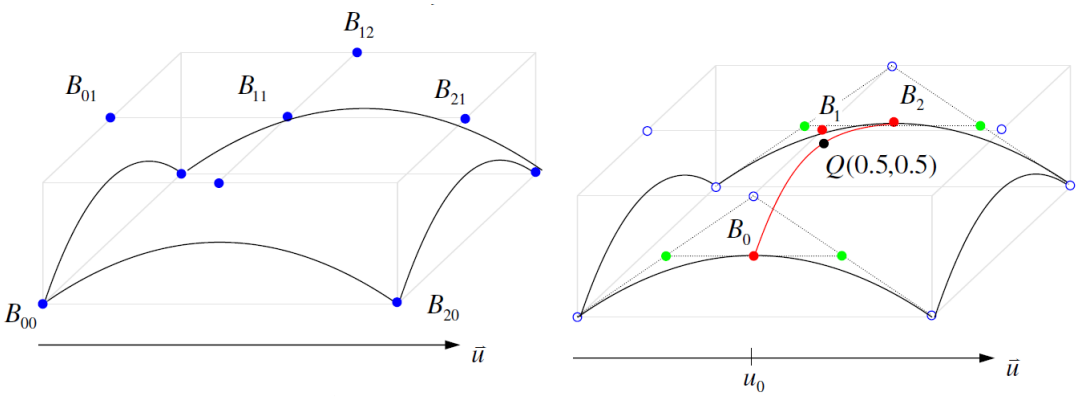

Figure 3.6: Doubled de Casteljau method for Bézier surface, [16]

3.3 B-splines

A spline curve whose description is based in base splines (B-splines) is named base spline curve (B-spline curve). The curve is estimated through so called De-Boor-points which control the shape and appearance of it: The curve always lies inside of the convex envelope of the De-Boor-points, it gets enclosed by them, see [5]. The term spline originate from the profession of shipbuilding and describes a long, thin wooden lath (spline) which is fixed with weights (ducks) at single points to minimize the inner tension which results from the bending of the lath, see [4]. The mathematical idea behind is to define a curve as a set of piecewise simple polynomial functions connected together. Splines were first introduced under their name in 1946 from Isaac J. Schoenberg, see [19].

3.3.1 B-spline curves

Similar to Bézier curves and splines, B-splines are spanned with control points, formulated in a recursive way and constructed piecewise from base functions, see [8]. Therefore B-splines are a generalized formulation of Bézier curves. With the naming of a knot vectors and the degree of the polynom, the influence of the control points can be manipulated or limited for a specific interval. Thus, changes of single control points lead to local changes of the shape of the curve.
A B-spline is defined with the following formula where P_i are the control points with $i = 1, ..., n$ and $N_{i,n}$ are the normalized base functions:

$$P(t) = \sum_{i=1}^{k} P_i N_{i,n}(t) \qquad t_{min} \leq t \leq t_{max} \tag{12}$$

The i-th normalized B-spline base function of degree n can be calculated with the help of the recursive Cox de Boor formula, see see [18], which can be visualized in a truncated triangular table, see figure 3.7:

$$N_{i,1}(t) = \begin{cases} 1, t_i \leq t < t_{i+1} \\ 0, sonst \end{cases}$$

$$N_{i,n}(t) = \frac{t - t_i}{t_{i+n-1} - t_i} N_{i,n-1}(t) + \frac{t_{i+n} - t}{t_{i+n} - t_{i+1}} N_{i+1,n-1}(t) \qquad \left(\frac{0}{0} = 0\right) \tag{13}$$

$$\begin{array}{ccccc} N_{0,0} & & & & \\ & N_{0,1} & & & \\ N_{1,0} & & N_{0,2} & & \\ & N_{1,1} & & N_{0,3} & \\ N_{2,0} & & N_{1,2} & & \\ & N_{2,1} & & N_{1,3} & \\ N_{3,0} & & N_{2,2} & \vdots & \\ & N_{3,1} & \vdots & & \\ N_{4,0} & \vdots & & & \\ \vdots & & & & \end{array}$$

Figure 3.7: Recursive determination of B-spline base functions, [18]

A B-spline of degree n has at least n control points. The values of the knot vectors are represented by t_i in rising order, for which $t_i \leq t_{i+1}$. The number of values t_i of a knot vector are defined as sum of the number of control points and the degree n of the spline, so $n + k$. The curve follows the direction of the control polygon and lies inside of its convex envelope which is defined by n. A point of the curve of degree n lies inside the convex envelope of n neighbour points, see [19].

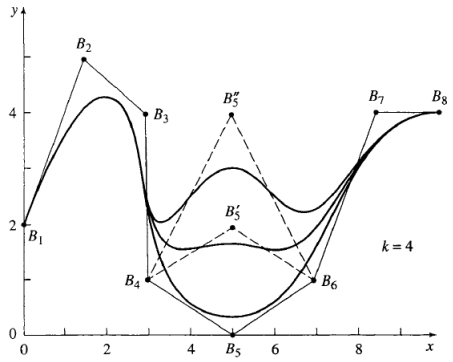

Figure 3.8: B-spline curve with influence of control points on the shape, [19]

3.3.2 Properties of B-splines

Because B-splines are based on the mathematical formulation of Bézier curves and also describe curves, some properties are already fixed, see [19]:

- The curve generally follows the shape of the control polygon and lies within the convex hull of it.
- Any affine transformation is applied to the B-spline curve by transforming the control polygon vertices.
- The curve does not oscillate about any straight line more often than its control polygon oscillates about the line. That means the B-spline does not run through its control polygon more often then it has control points (variation diminishing property).
- Therefore, the maximum order of the curve equals the number of control points. The maximum degree is one less.
- The sum of the B-spline base functions $N_{i,n}(t)$ is always equal one for any parametric value t:

$$\sum_{i=0}^{k} N_{i,n}(t) \equiv 1 \tag{14}$$

- Except for first-order basis functions, $n = 1$, each basis function has precisely one maximum value.
- Each base function is positive or zero for all parametric values t, see equation 13.

3.3.3 Knot vector

The major difference between both parametric approaches of free-form determination, Bézier curves and B-splines, is the usage of a knot vector for the second one, which influences directly the shape of the curve. The knot vector defines the interval in which the polynomials start and stop as the B-spline is created. Equation 13 shows that the choice of the knot vector has a significant influence on the B-spline basis functions $N_{i,n}(t)$ and hence on the resulting B-spline curve. Six rules shall be taken into account when it comes to the determination of a knot vector, see [17]:

1. The number of control points $P_i(t)$ defining the curve must always be equal to or greater than the order n of the curve plus one. That means, a quadratic curve (order $n = 2$) must have at least three points, a cubic curve (order $n = 3$) at least four, and so on.
2. The number of knots u_i in the knot vector U is always equal to the number of control points m plus the order of the curve n plus one. That means, a quadratic curve with three control points has six entries in the knot vector, a cubic curve with four control points has eight entries in the knot vector, and so on.

3. The order n of a curve must be at least two.

4. The values in the knot vector must always be in ascending order, so for example $U = [0, 1, 2, 3]$ is a valid knot vector while $U = [0, 2, 1, 3]$ is not a valid knot vector.

5. The valid parameter range for a curve starts at $u_{min} = u_n - 1$ and goes up to (but does not include) $u_{min} = u_m$, where m is the number of control points. Values less than the minimum parameter, or equal or above the maximum parameter are not defined.

6. The magnitude of the knots does not make any difference, only the ratios of the values to each other counts. For example, the knot vectors $U = [0, 1, 2, 3]$ and $U = [0, 2, 4, 6]$ produce the same curve.

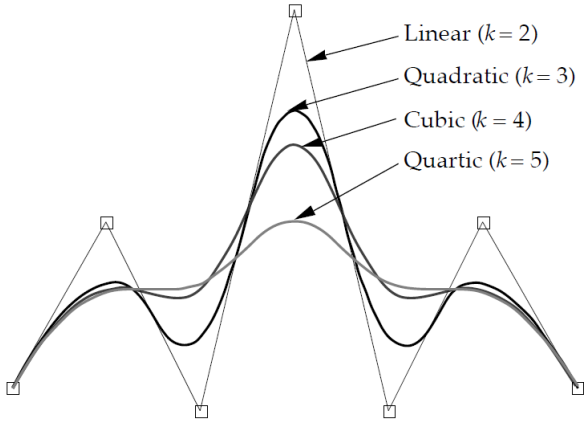

Figure 3.9: Influence of the curve order to the shape, [17]

The knot vector U normally is determined by the program which is used for the creation of NURBS curves and surfaces. There are two different types of knot vectors: uniform and non-uniform, which can be both periodical or non-periodical, see [19].
Uniform knot vectors use an equidistant spacing between their entries, so the step size is constant. The internal knot values are evenly spaced. Mostly, the knot vector then is normalized and runs from 0 to 1. On the opposite side, non-uniform knot vectors either use different spacings between their entries and/or several equal inner values. Periodical, also called open, knot vectors have multiplicity of knot values at the ends equal to the order n of the B-spline basis function while non-periodical, also called closed, knot vectors have every value of the entries only one time.
Mathematically spoken, the knot vector U can be formulated as follows with $a = 0, 1, ..., n$:

$$U = [u_0, u_1, ..., u_{i+n+1}] \quad \text{(periodical)} \tag{15}$$

$$U = [u_0, ..., a * u_0, u_1, ..., u_{i+n}, u_{i+n+1}, ..., a * u_{i+n+1}] \quad \text{(non-periodical)} \tag{16}$$

For every combination for the knot vectors, an example is given for $n = 2$:

$$U = [0, 0, 1, 2, 3, 4, 4] \quad \text{(uniform, periodical)}$$
$$U = [0, 0, 0.7, 2.5, 3.1, 4, 4] \quad \text{(non-uniform, periodical)}$$
$$U = [0, 1, 1, 2, 3, 3, 4] \quad \text{(uniform, non-periodical)}$$
$$U = [0, 0.4, 1, 2.5, 3, 3, 4] \quad \text{(non-uniform, non-periodical)}$$

For the task of surface reconstruction in terms of fatigue assessments, non-uniform non-periodical knot vectors provide the best choice because of their advantages to create a smooth, even distributed curve which is going through the starting and ending point of the interval.

3.3.4 B-spline surfaces

The B-spline surface has a similar formula like the curve, but a second dimension is added. Therefore, $P_{i,j}$ are the points from the control grid with $i = 1, ..., k$ and $j = 1, ..., l$. $N_{i,n}(u)$ and $N_{j,m}(v)$ are the B-spline

base functions in u- and v-direction, see [19]. So the formula is:

$$P(u,v) = \sum_{i=1}^{k} \sum_{j=1}^{l} P_{i,j} N_{i,n}(u) N_{j,m}(v) \tag{17}$$

The number of control points in u-direction therefore are $k+1$ and in v-direction $l+1$. The form of the surface is defined by the knot vectors U and V, which can be open or periodical and uniform or non-uniform. It is not necessary to have the same vectors for both directions.

3.4 Non-Uniform Rational B-Splines (NURBS)

The approximation of a free-form curve or surface can be accomplished with several approaches. One of them operates with so-called Non-Uniform Rational B-Splines, abbreviated as NURBS, which display the generalization of B-Splines and Bézier Curves, see [19].

3.4.1 The NURBS curve

A NURBS curve is mathematically named as $C(u)$ and describes the quotient of two sums of B-Spline base functions $N_{i,n}(u)$ of degree n weighted by the factor w_i, which are defined on a knot vector u and multiplied with the associated control points P_i, also called de Boor points. Changing the weight of a control point only affects it in the interval $[u_i, u_{i+k+1})$, see [1]. The indices i and j are going over the quantity of all existing control points. Therefore the function can be written down as follows, see [7] and [18]:

$$C(u) = \frac{\sum_{i=0}^{k} N_{i,n}(u) w_i P_i}{\sum_{j=0}^{k} N_{j,n}(u) w_j} \tag{18}$$

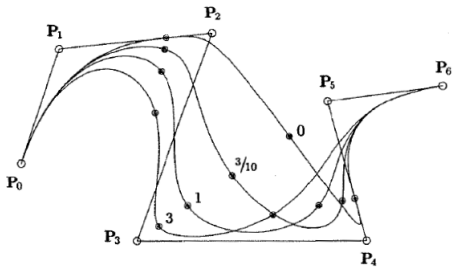

Figure 3.10: Non-uniform rational B-spline with influence of weight w_3, [18]

The base functions for the b-spline, shown in figure 3.11, are calculated recursive. With the Cox de Boor formula they can be written as, see [18] and [22]:

$$N_{i,n}(u) = f_{i,n}(u) N_{i,n-1}(u) + g_{i+1,n}(u) N_{i+1,n-1}(u) \tag{19}$$

The functions are

$$f_{i,n}(u) = \frac{u - k_i}{k_{i+n} - k_i} \tag{20}$$

and

$$g_{i+1,n}(u) = 1 - f_{i+1,n} = \frac{k_{i+n+1} - u}{k_{i+n+1} - k_i} \tag{21}$$

so the b-spline basis function gets:

$$N_{i,n}(u) = \frac{u - u_i}{u_{i+n} - u_i} N_{i,n-1}(u) + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_i} N_{i+1,n-1}(u) \tag{22}$$

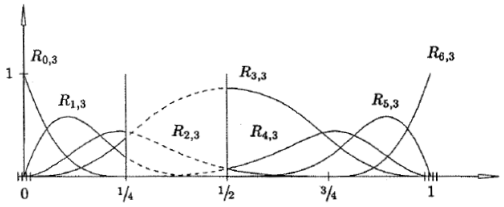

Figure 3.11: Base functions for cubic NURBS curve, [18]

3.4.2 Properties of NURBS curves

Non-uniform rational B-splines carry the same properties like normal B-splines but they have some additional ones, see [19]:

- A rational B-spline curve of order n (degree $n-$) is C2-continuous on the whole curve.
- A rational B-spline curve is invariant with respect to an projective transformation.
- The shape of a rational B-Spline curve is given by its weights which allow a direct manipulation of the curve shape without touching the control points.
- If all weights are equal to one, a NURBS curve reduces to a B-spline curve, see [23].

3.4.3 The NURBS surface

The mathematical description of a NURBS surface $S(u, v)$ builds up on the one from a NURBS curve and is obtained as the tensor product of two NURBS curves using two independent parameters u and v (with indices i and j respectively), see [7] and [18]:

$$S(u,v) = \sum_{i=0}^{k} \sum_{j=0}^{l} R_{i,j}(u,v) P_{i,j} \tag{23}$$

with

$$R_{i,j}(u,v) = \frac{N_{i,n}(u) N_{j,m}(v) w_{i,j}}{\sum_{p=0}^{k} \sum_{q=0}^{l} N_{p,n}(u) N_{q,m}(v) w_{p,q}} \tag{24}$$

as rational basis function.

3.5 Short overview about differences in the application

The formulation of free-forms as Bézier curves and surfaces with parametric coordinates has some advantages over the explicit and implicit free-form approaches: It can be calculated relatively easy, the curves or surfaces lie inside their control grids and hence mirrors its shape properties which is smooth and continuous. On the other side, every control point influences the shape of the curve or surface globally. Every change of the control points therefore also changes the shape of the curve or surface which result in a complete recalculation of the free-form if another shape is wished. Local changes are mostly not possible due to this fact. Additionally, there are no possibilities to control the parameter intervals. So the maximal influences of the control points of a quadratic Bézier curve or surface are lying inside of $t = 0, 0.5, 1$.
B-splines eliminate these disadvantages in regard to the Bézier formulation. They have the ability to get local changes at the curve with the influence of the knot vectors and the degree of the curve itself with the B-spline base functions. So B-splines are generally spoken more flexible in their usage compared to Bézier curves, but they are also more complicated to implement in programs. Also they can not display certain curves like circles or ellipses, see [15].
With non-uniform rational B-splines (NURBS), a very precise, mathematical formulation to represent every geometric from with curves and surfaces is given. In 2D, NURBS can display lines, conic sections, polynoms and so on, while in 3D free-form surfaces of any kind can be determined. With the weighting of the control points, NURBS are giving more freedom to its user then B-splines since the knot vector has not be modified anymore to get local changes in the shape of the curves or surfaces. NURBS are also invariant under rotation, scale, translation and perspective transformations, see [4]. This make NURBS the most flexible but also hardest choice to implement.

4 Numerical approach of free-form surface determination

In the following chapter, the project thesis contains the implementation of a numerical computation of NURBS surfaces out of a given bunch of control points acting as scatter plot. 3D-scanned raw scatter plots can not be directly used for that since they are only facet models containing triangles. The regarding files store three points for each facet and a normal vector to clarify the orientation of the facet which does not allow a direct usage for (low cycle) fatigue analysis calculations. Additionally, the raw data contains little gaps, missing areas and also a lot of not needed, but scanned areas.
To verify the numerical solution of free-form surface determination, it has to be checked with the results from calculation by hand but a manual determination of NURBS curves and surfaces is quite costly. Hence, the control points have to be small in numbers and simple in values to achieve a manual approach. The control grid is displayed by the following nine control points, which also can be seen in figure 4.1:

$$\begin{aligned} P(1,1) &= [0,0,0] \quad & P(2,1) &= [1,0,1] \quad & P(3,1) &= [2,0,0] \\ P(1,2) &= [0,1,1] \quad & P(2,2) &= [1,1,2] \quad & P(3,2) &= [2,1,1] \\ P(1,3) &= [0,2,0] \quad & P(2,3) &= [1,2,1] \quad & P(3,3) &= [2,2,0] \end{aligned}$$

Three knot vectors are chosen. U_1 and V_1 as uniform, non-periodical vectors are used to verify the numerical approach with then manual one while the other knot vectors are used to visualize the influence of different knot vector and weights on the shape of the surface:

$$\begin{aligned} U_1 &= [0.0, 0.5, 1.0, 1.5, 2.0] \quad & U_2 &= [0.0, 0.8, 1.0, 1.2, 2.0] \quad & U_3 &= [0.0, 0.1, 1.0, 1.3, 2.0] \\ V_1 &= [0.0, 0.5, 1.0, 1.5, 2.0] \quad & V_2 &= [0.0, 0.8, 1.0, 1.2, 2.0] \quad & V_3 &= [0.0, 0.1, 1.0, 1.3, 2.0] \end{aligned}$$

Also to have the same influence of all control points on the free-form surface and to simplify the calculation by hand the weights are all set equal to one:

$$\begin{aligned} w_i &= 1 \text{ for } i = 0, ..., k \\ w_j &= 1 \text{ for } j = 0, ..., k \\ &\rightarrow w_{i,j} = 1 \end{aligned}$$

So, three steps have to be completed:

1. A manual calculation is performed where the NURBS curve for the first three control points and the NURBS surface for all nine control points are determined. This step will provide necessary information about the calculation methods and how to implement them later in a numerical computation approach.
2. It is followed by a conversion of the manual calculation into Matlab code. Since Matlab provides a very flexible and simple method to implement mathematical calculations and will help with the closing step because ANSYS APDL is way more complex and inflexible regarding to Matlab.
3. In the end, the numerical computation process will be transformed from Matlab into ANSYS APDL which is planned to be the programming language for any further scientific studies and works. Fatigue analysis then can be performed with ANSYS Mechanical or Workbench.

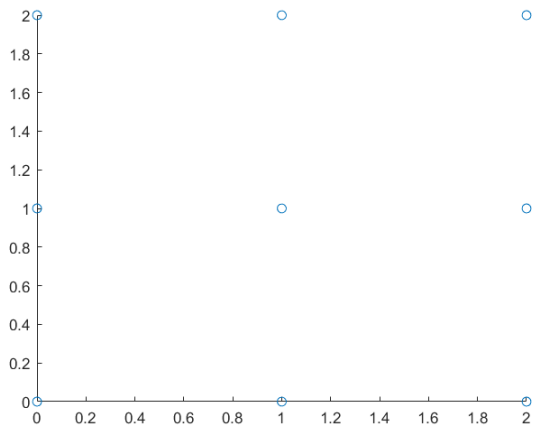

Figure 4.1: Control grid with nine control points

4.1 Manual determination of NURBS curve and surface

In the first step, a NURBS curve and surface will be calculated for the given control grid by hand. As defined in the previous chapter, the formula for a NURBS curve

$$C(u) = \frac{\sum_{i=0}^{k} N_{i,n}(u) w_i P_i}{\sum_{j=0}^{k} N_{j,n}(u) w_j} \tag{25}$$

and for a NURBS surface are used:

$$S(u,v) = \frac{\sum_{i=0}^{k} \sum_{j=0}^{l} N_{i,n}(u) N_{j,m}(v) w_{i,j} P_{i,j}}{\sum_{p=0}^{k} \sum_{q=0}^{l} N_{p,n}(u) N_{q,m}(v) w_{p,q}} \tag{26}$$

The formulas for the base functions are:

$$N_{i,n}(u) = \frac{u - u_i}{u_{i+n} - u_i} N_{i,n-1}(u) + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_i} N_{i+1,n-1}(u) \tag{27}$$

$$N_{j,m}(v) = \frac{v - v_j}{v_{i+m} - v_j} N_{j,m-1}(v) + \frac{v_{j+m+1} - v}{v_{j+m+1} - v_j} N_{j+1,m-1}(v) \tag{28}$$

Since the number of control points and their step size are equal in both directions, the base functions for $n = 0$ and $m = 0$ are also equal. Therefore, the base functions are defined as:

$$N_{i,0}(u) = \begin{cases} 1, & u_i \leq u < u_{i+1} \\ 0, & \text{sonst} \end{cases}$$

$$N_{j,0}(u) = \begin{cases} 1, & v_j \leq v < v_{j+1} \\ 0, & \text{sonst} \end{cases} \tag{29}$$

The NURBS curve will be calculated recursively for the first three points $P(1,1)$, $P(2,1)$ and $P(3,1)$:

$$N_{0,2}(u) = \frac{u - u_0}{u_2 - u_0} N_{0,1}(u) + \frac{u_3 - u}{u_3 - u_1} N_{1,1}(u) \tag{30}$$

$$\hookrightarrow N_{0,1}(u) = \frac{u - u_0}{u_1 - u_0} N_{0,0}(u) + \frac{u_2 - u}{u_2 - u_0} N_{1,0}(u) \tag{31}$$

$$\hookrightarrow N_{1,1}(u) = \frac{u - u_1}{u_2 - u_1} N_{1,0}(u) + \frac{u_3 - u}{u_3 - u_2} N_{2,0}(u) \tag{32}$$

$$N_{1,2}(u) = \frac{u - u_1}{u_3 - u_1} N_{1,1}(u) + \frac{u_4 - u}{u_4 - u_2} N_{2,1}(u) \tag{33}$$

$$\hookrightarrow N_{1,1}(u) = \frac{u - u_1}{u_2 - u_1} N_{1,0}(u) + \frac{u_3 - u}{u_3 - u_2} N_{2,0}(u) \tag{34}$$

$$\hookrightarrow N_{2,1}(u) = \frac{u - u_2}{u_3 - u_2} N_{2,0}(u) + \frac{u_5 - u}{u_5 - u_3} N_{3,0}(u) \tag{35}$$

$$N_{2,2}(u) = \frac{u - u_2}{u_4 - u_2} N_{2,1}(u) + \frac{u_5 - u}{u_5 - u_3} N_{3,1}(u) \tag{36}$$

$$\hookrightarrow N_{2,1}(u) = \frac{u - u_2}{u_3 - u_2} N_{2,0}(u) + \frac{u_5 - u}{u_5 - u_3} N_{3,0}(u) \tag{37}$$

$$\hookrightarrow N_{3,1}(u) = \frac{u - u_3}{u_4 - u_3} N_{3,0}(u) + \frac{u_5 - u}{u_5 - u_4} N_{4,0}(u) \tag{38}$$

For $n = 0$ and $i = 0, 1, ..., 4$ the base functions are:

$$N_{0,0}(u) = \begin{cases} 1, & u_0 \leq u < u_1 \\ 0, & \text{sonst} \end{cases} \tag{39}$$

$$N_{1,0}(u) = \begin{cases} 1, & u_1 \leq u < u_2 \\ 0, & \text{sonst} \end{cases} \tag{40}$$

$$N_{2,0}(u) = \begin{cases} 1, & u_2 \leq u < u_3 \\ 0, & \text{sonst} \end{cases} \tag{41}$$

$$N_{3,0}(u) = \begin{cases} 1, & u_3 \leq u < u_4 \\ 0, & \text{sonst} \end{cases} \tag{42}$$

$$N_{4,0}(u) = \begin{cases} 1, & u_4 \leq u < u_5 \\ 0, & \text{sonst} \end{cases} \tag{43}$$

With all the base functions written down formally, the needed ones can be calculated:

$$
\begin{aligned}
N_{0,2}(u) &= \frac{u-u_0}{u_2-u_0}\left[\frac{u-u_0}{u_1-u_0}N_{0,0}(u)+\frac{u_2-u}{u_2-u_0}N_{1,0}(u)\right]+\frac{u_3-u}{u_3-u_1}\left[\frac{u-u_1}{u_2-u_1}N_{1,0}(u)+\frac{u_3-u}{u_3-u_2}N_{2,0}(u)\right] \\
&= \frac{u-u_0}{u_2-u_0}\frac{u-u_0}{u_1-u_0}N_{0,0}(u)+\left[\frac{u-u_0}{u_2-u_0}\frac{u_2-u}{u_2-u_0}+\frac{u_3-u}{u_3-u_1}\frac{u-u_1}{u_2-u_1}\right]N_{1,0}(u)+\frac{u_3-u}{u_3-u_1}\frac{u_3-u}{u_3-u_2}N_{2,0}(u) \\
&= \frac{u-0}{1-0}\frac{u-0}{0-0}N_{0,0}(u)+\left[\frac{u-0}{1-0}\frac{1-u}{1-0}+\frac{2-u}{2-0}\frac{u-0}{1-0}\right]N_{1,0}(u)+\frac{2-u}{2-0}\frac{2-u}{2-1}N_{2,0}(u) \\
&= \left(u+\frac{2-u}{2}u\right)N_{1,0}(u)+\frac{2-u}{2}(2u)N_{2,0}(u) \\
&= \left(\frac{-u^2}{2}+2u\right)N_{1,0}(u)+\frac{u^2-4u+4}{2}N_{2,0}(u) \\
&= \left(\frac{-u^2}{2}+2u\right)N_{1,0}(u)+\left(\frac{u^2}{2}-2u+2\right)N_{2,0}(u) \qquad (44)
\end{aligned}
$$

$$
\begin{aligned}
N_{1,2}(u) &= \frac{u-u_1}{u_3-u_1}\left[\frac{u-u_1}{u_2-u_1}N_{1,0}(u)+\frac{u_3-u}{u_3-u_2}N_{2,0}(u)\right]+\frac{u_4-u}{u_4-u_2}\left[\frac{u-u_2}{u_3-u_2}N_{2,0}(u)+\frac{u_5-u}{u_5-u_3}N_{3,0}(u)\right] \\
&= \frac{u-u_1}{u_3-u_1}\frac{u-u_1}{u_2-u_1}N_{1,0}(u)+\left[\frac{u-u_1}{u_3-u_1}\frac{u_3-u}{u_3-u_2}+\frac{u_4-u}{u_4-u_2}\frac{u-u_2}{u_3-u_2}\right]N_{2,0}(u)+\frac{u_4-u}{u_4-u_2}\frac{u_5-u}{u_5-u_3}N_{3,0}(u) \\
&= \frac{u-0}{2-0}\frac{u-0}{1-0}N_{1,0}(u)+\left[\frac{u-0}{2-0}\frac{2-u}{2-1}+\frac{2-u}{2-1}\frac{u-1}{2-1}\right]N_{2,0}(u)+\frac{2-u}{2-1}\frac{u_5-u}{u_5-2}N_{3,0}(u) \\
&= \frac{u^2}{2}N_{1,0}(u)+\left[\frac{u}{2}(2-u)+(2-u)(u-1)\right]N_{2,0}(u) \\
&= \frac{u^2}{2}N_{1,0}(u)+\left(u+\frac{-u^2}{2}-u^2+3u-2\right)N_{2,0}(u) \\
&= \frac{u^2}{2}N_{1,0}(u)+\left(\frac{-3u^2}{2}+4u-2\right)N_{2,0}(u) \qquad (45)
\end{aligned}
$$

$$
\begin{aligned}
N_{2,2}(u) &= \frac{u-u_2}{u_4-u_2}\left[\frac{u-u_2}{u_3-u_2}N_{2,0}(u)+\frac{u_5-u}{u_5-u_3}N_{3,0}(u)\right]+\frac{u_5-u}{u_5-u_3}\left[\frac{u-u_3}{u_4-u_3}N_{3,0}(u)+\frac{u_5-u}{u_5-u_4}N_{4,0}(u)\right] \\
&= \frac{u-u_2}{u_4-u_2}\frac{u-u_2}{u_3-u_2}N_{2,0}(u)+\left[\frac{u-u_2}{u_4-u_2}\frac{u_5-u}{u_5-u_3}+\frac{u_5-u}{u_5-u_3}\frac{u-u_3}{u_4-u_3}\right]N_{3,0}(u)+\frac{u_5-u}{u_5-u_3}\frac{u_5-u}{u_5-u_4}N_{4,0}(u) \\
&= \frac{u-1}{2-1}\frac{u-1}{2-1}N_{2,0}(u)+\left[\frac{u-1}{2-1}\frac{u_5-u}{u_5-2}+\frac{u_5-u}{u_5-2}\frac{u-2}{2-2}\right]N_{3,0}(u)+\frac{u_5-u}{u_5-2}\frac{u_5-u}{u_5-2}N_{4,0}(u) \\
&= (u-1)^2N_{2,0}(u) \\
&= \left(u^2-2u+1\right)N_{2,0}(u) \qquad (46)
\end{aligned}
$$

The NURBS curve for the first three points now is:

$$
\begin{aligned}
C(u) &= \frac{N_{0,2}(u)w_0P_0 + N_{1,2}(u)w_1P_1 + N_{2,2}(u)w_2P_2}{N_{0,2}(u)w_0 + N_{1,2}(u)w_1 + N_{2,2}(u)w_2} \\
&= \frac{\left[\left(\frac{-u^2}{2} + 2u\right)N_{1,0}(u) + \left(\frac{u^2}{2} - 2u + 2\right)N_{2,0}(u)\right]w_0 \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}}{N_{0,2}(u)w_0 + N_{1,2}(u)w_1 + N_{2,2}(u)w_2} + ... \\
&...\frac{\left[\frac{u^2}{2}N_{1,0}(u) + \left(\frac{-3u^2}{2} + 4u - 2\right)N_{2,0}(u)\right]w_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}}{N_{0,2}(u)w_0 + N_{1,2}(u)w_1 + N_{2,2}(u)w_2} + ... \\
&...\frac{\left[\left(u^2 - 2u + 1\right)N_{2,0}(u)\right]w_2 \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}}{N_{0,2}(u)w_0 + N_{1,2}(u)w_1 + N_{2,2}(u)w_2} \\
&= \frac{\left[\frac{u^2}{2}N_{1,0}(u) + \left(\frac{-3u^2}{2} + 4u - 2\right)N_{2,0}(u)\right]w_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \left[\left(u^2 - 2u + 1\right)N_{2,0}(u)\right]w_2 \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}}{N_{0,2}(u)w_0 + N_{1,2}(u)w_1 + N_{2,2}(u)w_2} \\
&= \begin{pmatrix} \frac{\left[\frac{u^2}{2}N_{1,0}(u) + \left(\frac{-3u^2}{2} + 4u - 2\right)N_{2,0}(u)\right]w_1 + 2\left[\left(u^2 - 2u + 1\right)N_{2,0}(u)\right]w_2}{N_{0,2}(u)w_0 + N_{1,2}(u)w_1 + N_{2,2}(u)w_2} \\ 0 \\ \frac{\left[\frac{u^2}{2}N_{1,0}(u) + \left(\frac{-3u^2}{2} + 4u - 2\right)N_{2,0}(u)\right]w_1}{N_{0,2}(u)w_0 + N_{1,2}(u)w_1 + N_{2,2}(u)w_2} \end{pmatrix}
\end{aligned}
\tag{47}
$$

The NURBS surface spanned over whole control grid is:

$$
\begin{aligned}
S(u,v) = &\frac{N_{0,2}(u)N_{0,2}(v)w_{0,0}P_{0,0} + N_{1,2}(u)N_{0,2}(v)w_{1,0}P_{1,0} + N_{2,2}(u)N_{0,2}(v)w_{2,0}P_{2,0} + ...}{N_{0,2}(u)N_{0,2}(v)w_{0,0} + N_{1,2}(u)N_{0,2}(v)w_{1,0} + N_{2,2}(u)N_{0,2}(v)w_{2,0} + ...} \\
&\frac{...N_{0,2}(u)N_{1,2}(v)w_{0,1}P_{0,1} + N_{1,2}(u)N_{1,2}(v)w_{1,1}P_{1,1} + N_{2,2}(u)N_{1,2}(v)w_{2,1}P_{2,1} + ...}{...N_{0,2}(u)N_{1,2}(v)w_{0,1} + N_{1,2}(u)N_{1,2}(v)w_{1,1} + N_{2,2}(u)N_{1,2}(v)w_{2,1} + ...} \\
&\frac{...N_{0,2}(u)N_{2,2}(v)w_{0,2}P_{0,2} + N_{1,2}(u)N_{1,2}(v)w_{1,2}P_{1,2} + N_{2,2}(u)N_{2,2}(v)w_{2,2}P_{2,2}}{...N_{0,2}(u)N_{2,2}(v)w_{0,2} + N_{1,2}(u)N_{1,2}(v)w_{1,2} + N_{2,2}(u)N_{2,2}(v)w_{2,2}}
\end{aligned}
\tag{48}
$$

The manual determination makes clear, that the calculation of a NURBS curve for as small amount of about three control points can be done manually, but already a small surface for nine control points requires so much calculation effort, that it is only economically justifiable via a numerical program. Therefore, an ANSYS APDL macro is coded for the project thesis. To verify the overall calculation method for NURBS surfaces and to get an easier entrance into the topic, Matlab was chosen as mathematical software first. The next two subchapters 4.2 and 4.3 present the programming of both Matlab code and ANSYS APDL macro.

4.2 Numerical NURBS computation with Matlab

In the second step, the structure of the Matlab code for the numerical NURBS computation approach is presented. It starts with definition of the control points $P_(i,j)$, which were assigned at the beginning of chapter 3:

```
P(:,:,1,1) = [0 0 0];
P(:,:,1,2) = [1 0 1];
P(:,:,1,3) = [2 0 0];
P(:,:,2,1) = [0 1 1];
P(:,:,2,2) = [1 1 2];
P(:,:,2,3) = [2 1 1];
P(:,:,3,1) = [0 2 0];
P(:,:,3,2) = [1 2 1];
P(:,:,3,3) = [2 2 0];
```

Due to the fact, that the Cartesian coordinates x, y and z of the control points are 3-dimensional while the control grid itself with the parametric coordinates u and v is only 2-dimensional, they have to be written inside 4d-arrays. So to speak, the control points are saved as arrays inside an array. This provides the possibility to call them as $P_(i,j)$ over their indices i and j regarding the direction of both parametric coordinates u and v from the control grid.

Next, the all needed variables and input parameters for the computation are defined which will be needed for the later upcoming free-form surface determination:

```
n = 2;                      % degree of base funtions in u-direction
m = 2;                      % degree of base funtions in v-direction
k = 2;                      % number of control points in u-direction
p = 2;                      % number of control points in v-direction
m_u = n + k +1;             % number of knot points in u-direction
m_v = m + k +1;             % number of knot points in v-direction
U = [0 0.5 1 1.5 2];        % knot points in u-direction
V = [0 0.5 1 1.5 2];        % knot points in v- direction
w = [1 1 1; 1 1 1; 1 1 1];% weights for control points
intervals_u = 200;          % number of intervals in u-direction
intervals_v = 200;          % number of intervals in v-direction
s = 1;                      % counter for NURBS surface value memory
```

Only Matlab can plot functions over an interval without getting values for it at first, but ANSYS APDL do not have a possibility to implement functions very easy like in Matlab. Therefore, the number of interval steps have to be given in this case. The more interval steps for each direction the more control points are used for the determination of the NURBS surface which will create a more dense saturation.

The calculation of the NURBS surface at a given point (u,v) is done inside two for-loops which provide the necessary parametric coordinates:

```
for u = 0:U(m_u)/intervals_u:U(m_u)
        for v = 0:V(m_v)/intervals_v:V(m_v)
                ...
        end
end
```

The increments in both directions are controlled via the last entry of the knot vectors divided by the interval values.

Inside this two for-loops, the actual approach of the NURBS surface generation is implemented. The NURBS base functions are calculated each for u and v-direction. Since the structure for both code blocks is equal it is only shown for the first parametric coordinate u being valid for the other one as well. The calculation of the base functions is divided into two parts. The first one contains the calculation for $n = 0$ taken from formula 29:

```
for i = 1:1:m_u-1
        if u >= U(i) && u < U(i+1)
                N_u(i,1) = 1;
        else
                N_u(i,1) = 0;
        end
end
for i = m_u
        if u >= U(i)
                N_u(i,1) = 1;
        else
                N_u(i,1) = 0;
        end
end
```

Because the determination of $N(4,0)$ normally requires u_5 which is not defined in the knot vector and so will be set to zero in a manual approach, the numerical approach differs between them. For $N(4,0)$ only u_4 is taken into account. This prevents the code from getting an error message.
Then, the base functions for $n = 2, ..., n+1$ are calculated. Like many other mathematical programs, Matlab always starts with an index of one and not with zero. The formulas for the NURBS surface determination normally work with a starting index of one, so some slight adjustments have to be done:

```
for n = 2:1:3
        x = m_u-1; % counter to only determine the needed base
           functions
        for i = 1:1:x
                if i+n-1 > 5 || U(i+n-1)-U(i) == 0
                        a = 0;
                else
                        a = (u-U(i))/(U(i+n-1)-U(i));
                end
                if i+n > 5 || U(i+n)-U(i+1) == 0
                        b = 0;
                else
                        b = (U(i+n)-u)/(U(i+n)-U(i+1));
                end
                        N_u(i,n) = a * N_u(i,n-1) + b * N_u(i+1,n-1);
        end
        x = x-1;
end
```

Normally, the B-spline base functions are determined recursively. In a numerical approach this is not possible. Therefore, the determination is coded explicitly.

The NURBS surface is determined in the following part:

```
for i = 1:1:3
        for j = 1:1:3
                Q(:,:,i,j) = N_u(i,n)*N_v(j,m)*w(i,j)*P(:,:,i,j);
                R(i,j) = N_u(i,n)*N_v(j,m)*w(i,j);
        end
end

if sum(sum(R,1),2) == 0
        S(s,:) = [0 0 0];
else
        S(s,:) = sum(sum(Q,3),4)/sum(sum(R,1),2);
end
s = s+1; % counting up the memory for NURBS surface
```

The NURBS surface determination is split up into the numerator and denominator which are calculated separately for every control point. After that, they are summed up and divided with each other to get the surface point at the given parametric coordinates.

In the end, the NURBS surface is plotted over the control grid:

```
plot3(S(:,1), S(:,2), S(:,3), 'o')
```

To get a better visual the NURBS surface points are plotted with Delaunay triangulation which connects all points to triangle surfaces:

```
S_c = unique(S(:,1:3),'rows')
plot3(S_c(:,1),S_c(:,2),S_c(:,3),'.-')
tri = delaunay(S_c(:,1),S_c(:,2)); % Delaunay triangulation
plot(S_c(:,1),S_c(:,2),'.')
[r,c] = size(tri); % Number of triangles
disp(r)
h = trisurf(tri,S_c(:,1),S_c(:,2),S_c(:,3)); % Plot it with TRISURF
axis vis3d
```

Both scatter plots are presented in chapter 5.

The complete Matlab code for the generation of a NURBS surface for a given control grid can be found in Appendix A.

4.3 Numerical NURBS computation with ANSYS APDL

In the third step, the structure of the ANSYS APDL macro for the numerical NURBS computation approach is presented. The code starts with the control points $P(i,j)$ which are defined inside of an 3x3x3-array:

```
*DIM,P,ARRAY,3,3,3
P(1,1,1) = 0
P(1,1,2) = 0
P(1,1,3) = 0
P(2,1,1) = 1
P(2,1,2) = 0
P(2,1,3) = 1
P(3,1,1) = 2
P(3,1,2) = 0
P(3,1,3) = 0
P(1,2,1) = 0
P(1,2,2) = 1
P(1,2,3) = 1
P(2,2,1) = 1
P(2,2,2) = 1
P(2,2,3) = 2
P(3,2,1) = 2
P(3,2,2) = 1
P(3,2,3) = 1
P(1,3,1) = 0
P(1,3,2) = 2
P(1,3,3) = 0
P(2,3,1) = 1
P(2,3,2) = 2
P(2,3,3) = 1
P(3,3,1) = 2
P(3,3,2) = 2
P(3,3,3) = 0
```

Since the points are 3-dimensional, but the control grid is only 2-dimensional, they have to be written inside 3d-arrays. ANSYS APDL does not have the ability to store arrays as entries inside an array so this is the only option to provide the possibility to call them as $P(i,j)$ inside the code.

Next is the definition paragraph of all needed variables:

```
n_bf = 2                    ! degree of base funtions in u-direction
m_bf = 2                    ! degree of base funtions in v-direction
k_cp = 2                    ! number of control points in u-direction
p_cp = 2                    ! number of control points in u-direction
m_u = n_bf + k_cp + 1       ! number of knot points in u-direction
m_v = m_bf + p_cp + 1       ! number of knot points in v-direction
*DIM,U,ARRAY,5,1            ! knot points in u-direction
U(1,1) = 0,0.5,1,1.5,2
*DIM,V,ARRAY,5,1            ! knot points in v-direction
V(1,1) = 0,0.5,1,1.5,2
*DIM,w,ARRAY,3,3            ! weights for control points
w(1,1) = 1,1,1
w(1,2) = 1,1,1
w(1,3) = 1,1,1
intervals_u = 20            ! number of intervals in u-direction
intervals_v = 20            ! number of intervals in v-direction
n_K = 1                     ! counter for keypoint ID number
```

The only difference regarding the Matlab code is that a keypoint identification number fot the later needed

keypoints which will represent the calculated points from the NURBS surface.

Additional variables stored in arrays for the macro are defined as well:

```
! Definition of used array parameters inside the macro body
*DIM,N_u,ARRAY,5,3                        ! base function matrix for u-direction
*DIM,N_v,ARRAY,5,3                        ! base function matrix for v-direction
*DIM,S_num,ARRAY,3,1                ! numerator of NURBS surface vector for one point
*DIM,S,ARRAY,3,1                        ! NURBS surface vector for one point
```

Because ANSYS APDL needs the *DIM command for intializing array parameters, these ones are predefined before called inside the calculation routine.

The calculation of the NURBS surface at a given point (u, v) is done inside two DO-loops:

```
! Begin of main macro body
*DO,u,0,U(m_u),U(m_u)/intervals_u
        *DO,v,0,V(m_v),V(m_v)/intervals_v
                ...
        *ENDDO
*ENDDO
```

Inside this two *DO-loops, the actual code body of the NURBS surface generation is implemented.

The NURBS base functions are calculated each for u- and v-direction, here only presented for the first one since both code blocks are equal in their structure:

```
*DO,i,1,m_u,1
        *IF,i+1,LE,m_u,THEN
                *IF,uu,GE,U(i,1),AND,uu,LT,U(i+1,1),THEN
                        N_u(i,1) = 1
                *ELSE
                        N_u(i,1) = 0
                *ENDIF
        *ELSE
                *IF,uu,GE,U(i,1),THEN
                        N_u(i,1) = 1
                *ELSE
                        N_u(i,1) = 0
                *ENDIF
        *ENDIF
*ENDDO
```

The explicit determination of the B-spline base function calculation is also split up into two part like in the Matlab code. One of the biggest challanges was to find out, that ANSYS APDL handles "a*b" not the same as "a * b" which leads to wrong results at first. Furthermore, the *IF-conditions had to be split up into single conditions giving out a value for each possible statement. Summarized condition checks like in Matlab did not work.

```
*DO,n,2,n_bf+1,1
*DO,i,1,m_u,1
        *IF,i+n-1,GT,m_u,THEN
                f = 0
        *ELSE
                *IF,U(i+n-1,1)-U(i,1),EQ,0,THEN
                        f = 0
                *ELSE
                        f = (uu-U(i,1))/(U(i+n-1,1)-U(i,1))
                *ENDIF
        *ENDIF
        *IF,i+n,GT,m_u,THEN
                g = 0
        *ELSE
                *IF,(U(i+n,1)-U(i+1,1)),EQ,0,THEN
                        g = 0
                *ELSE
                        g = (U(i+n,1)-uu)/(U(i+n,1)-U(i+1,1))
                *ENDIF
        *ENDIF
        *IF,i+1,GT,m_u,THEN
                *IF,N_u(i,n-1),EQ,0,THEN
                        N_u(i,n) = 0
                *ELSE
                        N_v(i,n) = f*N_u(i,n-1)
                *ENDIF
        *ELSE
                *IF,N_u(i,n-1),EQ,0,AND,N_u(i+1,n-1),EQ,0,THEN
                        N_u(i,n) = 0
                *ELSE
                        *IF,N_u(i,n-1),NE,0,AND,N_u(i+1,n-1),EQ,0,THEN
                                N_u(i,n) = f*N_u(i,n-1)
                        *ELSE
                                *IF,N_u(i,n-1),EQ,0,AND,N_u(i+1,n-1),NE,0,THEN
                                        N_u(i,n) = g*N_u(i+1,n-1)
                                *ELSE
                                        N_u(i,n) = f*N_u(i,n-1) + g*N_u(i+1,n-1)
                                *ENDIF
                        *ENDIF
                *ENDIF
        *ENDIF
*ENDDO
*ENDDO
```

The NURBS surface is determined in the following part:

```
S_num(1,1) = 0,0,0  ! initialisation of numerator for fraction of NURBS surface term
S_denom = 0         ! initialisation of denominator for fraction of NURBS surface term

*DO,i,1,k_cp+1,1
        *DO,j,1,p_cp+1,1
                S_num(1,1) = S_num(1,1) + N_u(i,n_bf+1)*N_v(j,m_bf+1)*w(i,j)*P(i,j,1)
                S_num(2,1) = S_num(2,1) + N_u(i,n_bf+1)*N_v(j,m_bf+1)*w(i,j)*P(i,j,2)
                S_num(3,1) = S_num(3,1) + N_u(i,n_bf+1)*N_v(j,m_bf+1)*w(i,j)*P(i,j,3)
                S_denom = S_denom + N_u(i,n_bf+1)*N_v(j,m_bf+1)*w(i,j)
        *ENDDO
*ENDDO

*DO,xyz,1,3,1
        *IF,S_denom,EQ,0,THEN
                S(xyz,1) = 0
        *ELSE
                S(xyz,1) = S_num(xyz,1)/S_denom
        *ENDIF
*ENDDO
```

Like in the Matlab code, the NURBS surface derivate is split up into the numerator and the denominator which are calculated for each parametric coordinate and summed up. After that, they get divided by each other.

In the end, the NURBS surface is plotted over the control grid which has to be defined over key points:

```
K,n_K,S(1,1),S(2,1),S(3,1)
n_K = n_K + 1
```

The scatter plot of the keypoints is presented in chapter 5.

The complete ANSYS APDL code for the generation of a NURBS surface for a given control grid can be found in Appendix B.

5 Discussion

To verify the correctness of both Matlab and ANSYS APDL approach they are compared with the manual approach and the knot vectors U_1. Therefore, several values for the parametric coordinate u_i are taken and are used to calculate NURBS curve both manual and numerical. The determination proecedure for the B-spline base functions is tested, see table 5.1.

u_i	$N_{i,n}$ for manual approach			$N_{i,n}$ for numerical approach		
1.4	0	0	0.02	0	0	0.02
	0	0.2	0.66	0	0.2	0.66
	1	0.8	0.32	1	0.8	0.32
	0	0	0	0	0	0
	0	0	0	0	0	0
1.5	0	0	0	0	0	0
	0	0	0.5	0	0	0.5
	0	1	0.5	0	1	0.5
	1	0	0	1	0	0
	0	0	0	0	0	0
1.75	0	0	0	0	0	0
	0	0	0.125	0	0	0.125
	0	0.5	0.375	0	0.5	0.375
	1	0.5	0	1	0.5	0
	0	0	0	0	0	0
2.0	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	1	0	0	1	0	0

Table 5.1: Comparison of B-spline base function results for manual and numerical approach

The results are the same showing that the computational surface reconstruction works as planned.
Both numerical approaches for the surface reconstruction with non-rational uniform B-splines were tested with the same input parameters like shown in the previous subchapters. The following two surface point plots are made with the Matlab code. First, the raw scatter plot without any further postprocessing:

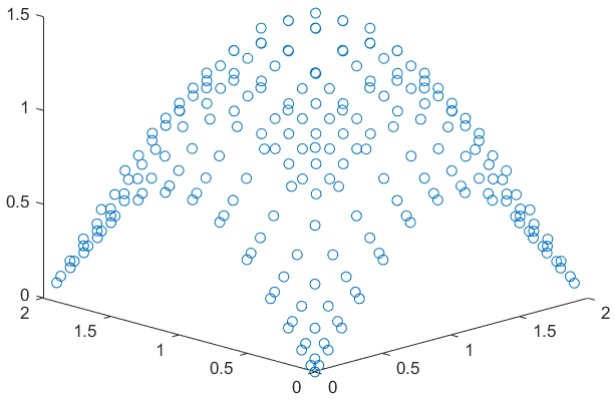

Figure 5.1: NURBS surface point plot from Matlab for 20 intervals in both parametric directions

Second, the triangulated surface plot:

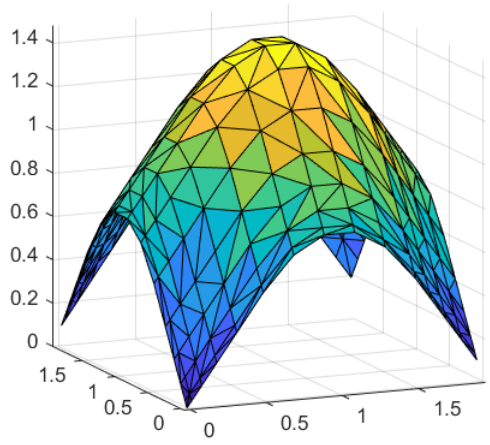

Figure 5.2: NURBS surface triangle plot from Matlab for 20 intervals in both parametric directions

The surface point plot from the ANSYS APLD macro shows the same values like the Matlab code which can be verified by comparing the coordinates of all determined surface points.

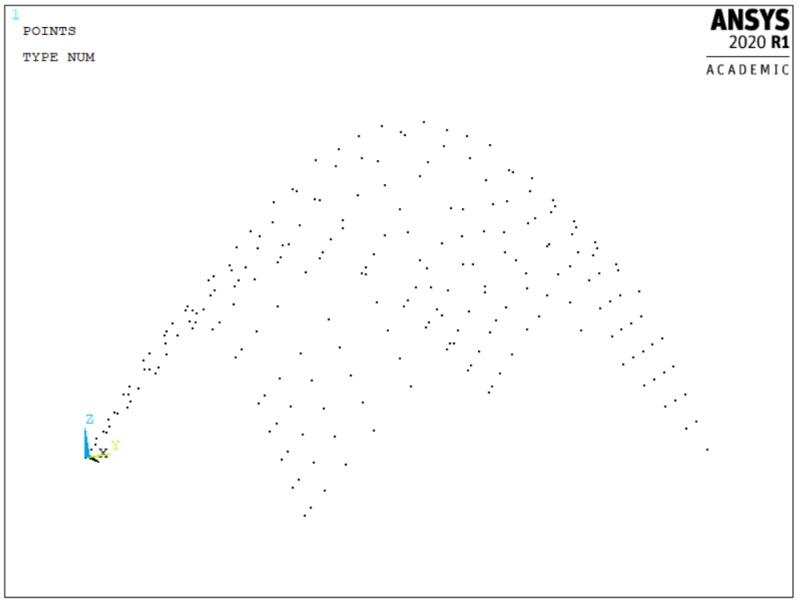

Figure 5.3: NURBS surface point plot from ANSYS APDL for 20 intervals in both parametric directions

In the Appendix C, additional scatter plots made in Matlab are presented with 200 intervals in each direction, the non-uniform and non-periodical knot vectors U_2 and V_2 as well as U_3 and V_3 and different weights $w_{i,j}$ which are mentioned in the captions. Changing knot vector entries and weighting factors lead to different shape behaviour, see figure 5.4. The higher the weight, the more pointy the NURBS surface get at the regarding control point. Furthermore, the distribution of NURBS surface points created at the points of intersection from the curves changes with different knot vectors. The knot vector has direct influence on how the control grid is saturated with NURBS surface points. So it is proofed, that the programmed numerical approach computes the same shape changes and NURBS surface point saturation over the control grid as expected.

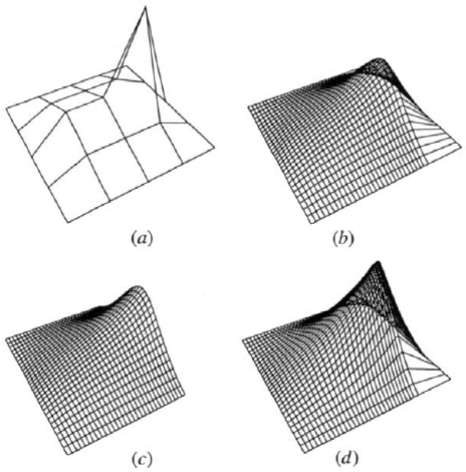

Figure 5.4: Influence on curve shape with changing the weight from 1 over 2 to 5, [19]

6 Conclusion

While scientific studies like [11], [12] and [13] focus on the usage of NURBS as free-forms for fatigue assessment, this project thesis dealt with the numerical surface reconstruction and the mathematical principles behind it. It was examined which free-form will provide the most flexible and robust way of creating free-form surfaces out of a given bunch of points. During this task, NURBS were identified as best option which is equal to the assumptions made in state of the art approaches. The precise modelling of cycle loaded structure, especially in the low cycle fatigue section, is necessary to get correct results for fatigue assessments. Hence NURBS provides the best overall formulation since it is also possible to use them as Bézier surfaces for example. Their complexe, but also general formulation make NURBS the most universal free-form determination method.
Experimental tests with specimen presented in actual scientific papers like [9] or [21] proof that 3D scanned geometries do not automatically provide better fatigue assessment results. Therefore, additional work will be needed to determine the right parameters for the NURBS surface generation to ensure the best possible accuracy while reconstructing scatter plots. In combination with the analysis method, the chosen surface reconstruction approach can be crucial for the results.
The ANSYS APDL macro which was programmed during the project thesis only works with predefined set of control points and only creates keypoints on the surface. It will need a lot of additional work to program a completely working ANSYS APDL macro routine which can process 3D scanned components in form of raw scatter plots and combine the keypoints to areas which then can transformed into an usable mesh for further fatigue analysis. Also it has to be examined if ANSYS APDL will provide the best enviroment for surface reconstruction approaches since the computational time is a lot lower then in Matlab for example. The more points in the scatter plot the longer the surface determination procedure. Furthermore, a parameter study is needed to verify how different input affects the surface reconstruction process and how the quality of the approximated surfaces are. The plots from Appendix C give a first impression of the shape changes and surface point saturations for different knot vectors and weighting factors, but for further studies a complete parameter analysis will become necessary.

References

[1] Altmann, Markus. *About Nonuniform Rational B-Splines - NURBS*. Worcester Polytechnic Institute, Computer Science department, online, accessed on 24th june 2020. URL https://web.cs.wpi.edu/~matt/courses/cs563/talks/nurbs.html

[2] Assarsson, Ulf. *Curves and Surfaces: Hermite/Bezier Curves, (B-)Splines and NURBS*. Computer Graphics Course, University of Gothenburg, Department of Computer Science and Engineering, December 2018. URL http://www.cse.chalmers.se/edu/course/TDA362/CurvesandSurfaces.pdf

[3] Bast, Liam Oliver. *Rendering von Freiformflächen*. Bachelorarbeit im Studiengang Computervisualistik (Bachelor of Science), Universität Koblenz-Landau, Fachbereich 4: Informatik, April 2018.

[4] Breen, David and Regli, William and Peysakhov, Maxim. *B-Splines and NURBS*. Lecture: Computer Graphics I, Course: Geometric and Intelligent Computing Laboratory, Department of Computer Science, Drexel University, October 2009. URL https://www.cs.drexel.edu/~david/Classes/CS430/Lectures/L-09_BSplines_NURBS.pdf

[5] De Boor, Carl. *A Practical Guide to Splines Revised Edition*. Springer Verlag New York, 1978. ISBN 0-387-95366-3

[6] DeLuca, D.P.. *Understanding Fatigue*. United Technologies Pratt & Whitney, October 2007. URL https://pdfs.semanticscholar.org/6569/d309015513cedaa412d91af458177afe3789.pdf

[7] Farin, Gerald. *Curves and Surfaces for CAGD - A Practical Guide Fifth Edition*. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, 2002. ISBN 1-55860-737-4

[8] Guntermann, Klaus. *B-Spline-Kurve und -Basisfunktionen*. used in lecture Maschinelles Lernen Universität Hamburg, original from FG Systemprogrammierung TU Darmstadt, April 2003. URL https://tams.informatik.uni-hamburg.de/lehre/2006ss/vorlesung/maschinelles_lernen/folien/06-16-bw.pdf

[9] Hultgren, Gustav and Barsoum, Zuheir. *Estimation of fatigue in welded joints based on laser scanning - Correlation between weld quality and fatigue life*. International Institute of Welding IW-Doc. XIII-2783-19 (2019).

[10] Issa, Rajaa. *Parametric Curves and Surfaces* Rhino Developer Docs, online, accessed on 24th june 2020. URL https://developer.rhino3d.com/guides/general/essential-mathematics/parametric-curves-surfaces/#31-parametric-curves

[11] Ladinek, Markus. Niederwanger, Alexander. Lang, Robert. Schmid, Johannes. Timmers, Ralph. Lener, Gerhard. *The strain-life approach applied to welded joints: Considering the real weld geometry*. Journal of Constructional Steel Research 148 (2018) 180–188, April 2018. URL https://doi.org/10.1016/j.jcsr.2018.04.024

[12] Lang, Robert and Lener, Gerhard. *Application and comparison of deterministic and stochastic methods for the evaluation of welded components' fatigue lifetime based on real notch stresses*. International Journal of Fatigue 93 (2016) 184–193, August 2016. URL https://dx.doi.org/10.1016/j.ijfatigue.2016.08.023

[13] Lener, Gerhard. Lang, Robert. Ladinek, Markus. Timmers, Ralph. *A numerical method for determining the fatigue strength of welded joints with a significant improvement in accuracy*. Procedia Engineering 213 (2018) 359–373, November 2017. URL https://doi.org/10.1016/j.proeng.2018.02.036

[14] Markgraf, Richard. *Raytracing von NURBS*. Bachelorarbeit im Studiengang Computervisualistik (Bachelor of Science), Universität Koblenz-Landau, Fachbereich 4: Informatik, Mai 2019.

[15] Müller, S. *(4) NURBS*. Vorlesung „Computergraphik III", Universität Koblenz-Landau, Juni 2013. URL https://userpages.uni-koblenz.de/~cg/ss13/cg3/04_nurbs.pdf

[16] Müller, S. *(5) Freiformflächen*. Vorlesung „Computergraphik III", Universität Koblenz-Landau, Juni 2013. URL https://userpages.uni-koblenz.de/~cg/ss13/cg3/05_flaechen.pdf

[17] Peterson, John. *No. 5: How to use Knot Vectors*. Albert Technical Memo, June 1990. URL https://saccade.com/writing/graphics/KnotVectors.pdf

[18] Piegel, Les and Tiller, Wayne. *The NURBS Book 2nd Edition*. Springer-Verlag New York, 1997. ISBN 3-540-61545-8.

[19] Rogers, David F. *An Introduction to NURBS: With Historical Perspective*. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, Elsevier Science, 2001. ISBN 978-1-55860-669-2.

[20] Schollmeyer, Andre. *Efficient and High-Quality Rendering of Higher-Order Geometric Data Representations*. Dissertation zur Erlangung des akademischen Grades (Dr. rer. nat.), Fakultät Medien, Bauhaus-Universität Weimar, November 2018.

[21] Schubnell, Jan. Jung, Matthias. Le, Chanh Hieu. Farajian, Majid. Braun, Moritz. Ehlers, Sören. Fricke, Wolfgang. Garcia, Martin. Nussbaumer, Alain. Baumgartner, Jörg. *Influence of the optical measurement technique and evaluation approach on the determination of local weld geometry parameters for different weld types*. Welding in the World, 64 (2020) 301-316. URL https://doi.org/10.1007/s40194-019-00830-0

[22] Shene, Dr. Ching-Kuang. *De Boor's Algorithm*. Department of Computer Science, Michigan Technological University, online, accessed on 24th june 2020. URL https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/de-Boor.html

[23] Shene, Dr. Ching-Kuang. *NURBS: Definition*. Department of Computer Science, Michigan Technological University, online, accessed on 24th june 2020. URL https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/NURBS/NURBS-def.html

[24] Vectorworks. *NURBS-Kurve*. Vectorworks®-Handbuch, online, accessed on 24th june 2020. URL http://vectorworks-hilfe.computerworks.eu/2017/VW_2017_Handbuch_Vectorworks/21_VW_WerkzBef_N-O/Vectorworks_16-.htm#XREF_12217_5_Gewicht_Der_hier

A Complete Matlab Code

```
%% Computational surface reconstruction with NURBS

% Control point 4d-array
P(:,:,1,1) = [0 0 0];
P(:,:,1,2) = [1 0 1];
P(:,:,1,3) = [2 0 0];
P(:,:,2,1) = [0 1 1];
P(:,:,2,2) = [1 1 2];
P(:,:,2,3) = [2 1 1];
P(:,:,3,1) = [0 2 0];
P(:,:,3,2) = [1 2 1];
P(:,:,3,3) = [2 2 0];

% Input parameters
n = 2;                          % degree of base funtions in u-direction
m = 2;                          % degree of base funtions in v-direction
k = 2;                          % number of control points in u-direction
p = 2;                          % number of control points in u-direction
m_u = n + k +1;                 % number of knot points in u-direction
m_v = m + k +1;                 % number of knot points in v-direction
U = [0 0.5 1 1.5 2];            % knot points in u-direction
V = [0 0.5 1 1.5 2];            % knot points in v- direction
w = [1 1 1; 1 1 1; 1 1 1];      % weights for control points
intervals_u = 20;               % number of intervals in u-direction
intervals_v = 20;               % number of intervals in v-direction
s = 1;                          % counter for NURBS surface value memory

% Begin of the computational section
for u = 0:U(m_u)/intervals_u:U(m_u)
for v = 0:V(m_v)/intervals_v:V(m_v)

% B-Spline base functions for u-direction
    for i = 1:1:m_u-1
        if u >= U(i) && u < U(i+1)
            N_u(i,1) = 1;
        else
            N_u(i,1) = 0;
        end
    end
    for i = m_u
        if u >= U(i)
            N_u(i,1) = 1;
        else
            N_u(i,1) = 0;
        end
    end
    for n = 2:1:3
        x = m_u-1; % counter to only determine the needed base functions
            for i = 1:1:x
                if i+n-1 > 5 || U(i+n-1)-U(i) == 0
                    a = 0;
                else
                    a = (u-U(i))/(U(i+n-1)-U(i));
                end
                if i+n > 5 || U(i+n)-U(i+1) == 0
                    b = 0;
```

```
                else
                    b = (U(i+n)-u)/(U(i+n)-U(i+1));
                end
            N_u(i,n) = a * N_u(i,n-1) + b * N_u(i+1,n-1);
            end
        x = x-1;
    end

% B-Spline base functions for v-direction
    for j = 1:1:m_v-1
        if v >= V(j) && v < V(j+1)
            N_v(j,1) = 1;
        else
            N_v(j,1) = 0;
        end
    end
    for j = m_v
        if v >= V(j)
            N_v(j,1) = 1;
        else
            N_v(j,1) = 0;
        end
    end
    for m = 2:1:3
        y = m_v-1; % counter to only determine the needed base functions
            for j = 1:1:y
                if j+m-1 > 5 || V(j+m-1)-V(j) == 0
                    c = 0;
                else
                    c = (v-V(j))/(V(j+m-1)-V(j));
                end
                if j+m > 5 || V(j+m)-V(j+1) == 0
                    d = 0;
                else
                    d = (V(j+m)-v)/(V(j+m)-V(j+1));
                end
            N_v(j,m) = c * N_v(j,m-1) + d * N_v(j+1,m-1);
            end
        y = y-1;
    end

% NURBS surface determination
for i = 1:1:3
    for j = 1:1:3
        Q(:,:,i,j) = N_u(i,n)*N_v(j,m)*w(i,j)*P(:,:,i,j);
        R(i,j) = N_u(i,n)*N_v(j,m)*w(i,j);
    end
end

if sum(sum(R,1),2) == 0
    S(s,:) = [0 0 0];
else
    S(s,:) = sum(sum(Q,3),4)/sum(sum(R,1),2);
end
s = s+1; % counting up the memory for NURBS surface

end
end
```

```
%% Making Point Plots
plot3(S(:,1), S(:,2), S(:,3), 'o')

%% Making Surface Plots From Scatter Data
S_c = unique(S(:,1:3),'rows')
plot3(S_c(:,1),S_c(:,2),S_c(:,3),'.-')
tri = delaunay(S_c(:,1),S_c(:,2)); % Delaunay triangulation
plot(S_c(:,1),S_c(:,2),'.')
[r,c] = size(tri); % Number of triangles
disp(r)
h = trisurf(tri,S_c(:,1),S_c(:,2),S_c(:,3)); % Plot it with TRISURF
axis vis3d
```

B Complete ANSYS APDL Macro

```
! Test surface reconstruction with a NURBS surface
finish
/clear
/PREP7

! Control points
*DIM,P,ARRAY,3,3,3
P(1,1,1) = 0
P(1,1,2) = 0
P(1,1,3) = 0
P(2,1,1) = 1
P(2,1,2) = 0
P(2,1,3) = 1
P(3,1,1) = 2
P(3,1,2) = 0
P(3,1,3) = 0
P(1,2,1) = 0
P(1,2,2) = 1
P(1,2,3) = 1
P(2,2,1) = 1
P(2,2,2) = 1
P(2,2,3) = 2
P(3,2,1) = 2
P(3,2,2) = 1
P(3,2,3) = 1
P(1,3,1) = 0
P(1,3,2) = 2
P(1,3,3) = 0
P(2,3,1) = 1
P(2,3,2) = 2
P(2,3,3) = 1
P(3,3,1) = 2
P(3,3,2) = 2
P(3,3,3) = 0

! Input parameters
n_bf = 2                    ! degree of base funtions in u-direction
m_bf = 2                    ! degree of base funtions in v-direction
k_cp = 2                    ! number of control points in u-direction
p_cp = 2                    ! number of control points in u-direction
m_u = n_bf + k_cp + 1       ! number of knot points in u-direction
m_v = m_bf + p_cp + 1       ! number of knot points in v-direction
*DIM,U,ARRAY,5,1            ! knot points in u-direction
U(1,1) = 0,0.5,1,1.5,2
*DIM,V,ARRAY,5,1            ! knot points in v-direction
V(1,1) = 0,0.5,1,1.5,2
*DIM,w,ARRAY,3,3            ! weights for control points
w(1,1) = 1,1,1
w(1,2) = 1,1,1
w(1,3) = 1,1,1
intervals_u = 20            ! number of intervals in u-direction
intervals_v = 20            ! number of intervals in v-direction
n_K = 1                     ! counter for keypoint ID number

! Definition of used array parameters inside the macro body
*DIM,N_u,ARRAY,5,3          ! base function matrix for u-direction
```

```
*DIM,N_v,ARRAY,5,3          ! base function matrix for v-direction
*DIM,S_num,ARRAY,3,1        ! numerator of NURBS surface vector for one point
*DIM,S,ARRAY,3,1            ! NURBS surface vector for one point


! Begin of main macro body
*DO,uu,0,U(m_u,1),U(m_u,1)/intervals_u
*DO,vv,0,V(m_v,1),V(m_v,1)/intervals_v

! B-Spline base functions for u-direction
*DO,i,1,m_u,1
    *IF,i+1,LE,m_u,THEN
        *IF,uu,GE,U(i,1),AND,uu,LT,U(i+1,1),THEN
            N_u(i,1) = 1
        *ELSE
            N_u(i,1) = 0
        *ENDIF
    *ELSE
        *IF,uu,GE,U(i,1),THEN
            N_u(i,1) = 1
        *ELSE
            N_u(i,1) = 0
        *ENDIF
    *ENDIF
*ENDDO

*DO,n,2,n_bf+1,1
*DO,i,1,m_u,1
    *IF,i+n-1,GT,m_u,THEN
        f = 0
    *ELSE
        *IF,U(i+n-1,1)-U(i,1),EQ,0,THEN
            f = 0
        *ELSE
            f = (uu-U(i,1))/(U(i+n-1,1)-U(i,1))
        *ENDIF
    *ENDIF
    *IF,i+n,GT,m_u,THEN
        g = 0
    *ELSE
        *IF,(U(i+n,1)-U(i+1,1)),EQ,0,THEN
            g = 0
        *ELSE
            g = (U(i+n,1)-uu)/(U(i+n,1)-U(i+1,1))
        *ENDIF
    *ENDIF
    *IF,i+1,GT,m_u,THEN
        *IF,N_u(i,n-1),EQ,0,THEN
            N_u(i,n) = 0
        *ELSE
            N_v(i,n) = f*N_u(i,n-1)
        *ENDIF
    *ELSE
        *IF,N_u(i,n-1),EQ,0,AND,N_u(i+1,n-1),EQ,0,THEN
            N_u(i,n) = 0
        *ELSE
            *IF,N_u(i,n-1),NE,0,AND,N_u(i+1,n-1),EQ,0,THEN
                N_u(i,n) = f*N_u(i,n-1)
```

```
            *ELSE
                *IF,N_u(i,n-1),EQ,0,AND,N_u(i+1,n-1),NE,0,THEN
                    N_u(i,n) = g*N_u(i+1,n-1)
                *ELSE
                    N_u(i,n) = f*N_u(i,n-1) + g*N_u(i+1,n-1)
                *ENDIF
            *ENDIF
        *ENDIF
    *ENDIF
*ENDDO
*ENDDO


! B-Spline base functions for v-direction
*DO,j,1,5,1
    *IF,j+1,LE,m_v,THEN
        *IF,vv,GE,V(j,1),AND,vv,LT,V(j+1,1),THEN
            N_v(j,1) = 1
        *ELSE
            N_v(j,1) = 0
        *ENDIF
    *ELSE
        *IF,vv,GE,V(j,1),THEN
            N_v(j,1) = 1
        *ELSE
            N_v(j,1) = 0
        *ENDIF
    *ENDIF
*ENDDO

*DO,m,2,m_bf+1,1
*DO,j,1,m_v,1
    *IF,j+m-1,GT,m_v,THEN
        f = 0
    *ELSE
        *IF,V(j+m-1,1)-V(j,1),EQ,0,THEN
            f = 0
        *ELSE
            f = (vv-V(j,1))/(V(j+m-1,1)-V(j,1))
        *ENDIF
    *ENDIF
    *IF,j+m,GT,m_v,THEN
        g = 0
    *ELSE
        *IF,(V(j+m,1)-V(j+1,1)),EQ,0,THEN
            g = 0
        *ELSE
            g = (V(j+m,1)-vv)/(V(j+m,1)-V(j+1,1))
        *ENDIF
    *ENDIF
    *IF,j+1,GT,m_v,THEN
        *IF,N_v(j,m-1),EQ,0,THEN
            N_v(j,m) = 0
        *ELSE
            N_v(j,m) = f*N_v(j,m-1)
        *ENDIF
    *ELSE
        *IF,N_v(j,m-1),EQ,0,AND,N_v(j+1,m-1),EQ,0,THEN
```

```
            N_v(j,m) = 0
        *ELSE
            *IF,N_v(j,m-1),NE,0,AND,N_v(j+1,m-1),EQ,0,THEN
                N_v(j,m) = f*N_v(j,m-1)
            *ELSE
                *IF,N_v(j,m-1),EQ,0,AND,N_v(j+1,m-1),NE,0,THEN
                    N_v(j,m) = g*N_v(j+1,m-1)
                *ELSE
                    N_v(j,m) = f*N_v(j,m-1) + g*N_v(j+1,m-1)
                *ENDIF
            *ENDIF
        *ENDIF
    *ENDIF
*ENDDO
*ENDDO

! NURBS surface determination
S_num(1,1) = 0,0,0        ! initialisation of numerator for fraction of NURBS surface term
S_denom = 0                ! initialisation of denominator for fraction of NURBS surface term
*DO,i,1,k_cp+1,1
    *DO,j,1,p_cp+1,1
        S_num(1,1) = S_num(1,1) + N_u(i,n_bf+1)*N_v(j,m_bf+1)*w(i,j)*P(i,j,1)
        S_num(2,1) = S_num(2,1) + N_u(i,n_bf+1)*N_v(j,m_bf+1)*w(i,j)*P(i,j,2)
        S_num(3,1) = S_num(3,1) + N_u(i,n_bf+1)*N_v(j,m_bf+1)*w(i,j)*P(i,j,3)
        S_denom = S_denom + N_u(i,n_bf+1)*N_v(j,m_bf+1)*w(i,j)
    *ENDDO
*ENDDO

*DO,xyz,1,3,1
    *IF,S_denom,EQ,0,THEN
        S(xyz,1) = 0
    *ELSE
        S(xyz,1) = S_num(xyz,1)/S_denom
    *ENDIF
*ENDDO

! Keypoint generation
K,n_K,S(1,1),S(2,1),S(3,1)
n_K = n_K + 1

! End of main macro body
*ENDDO
*ENDDO
```

C Additional NURBS surface plots with Matlab

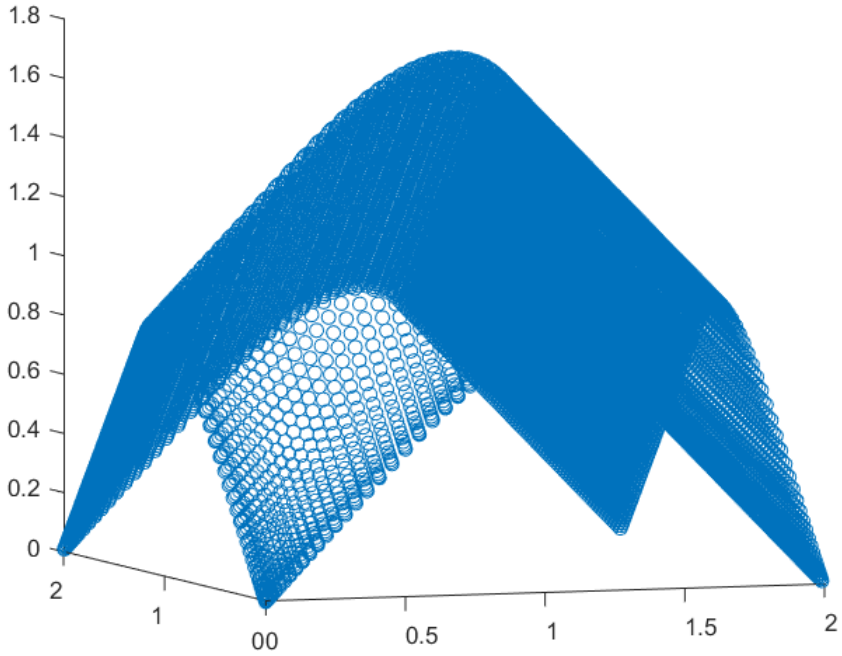

Figure C.1: NURBS surface for U_2 and V_2 with $w_{2,2} = 1$

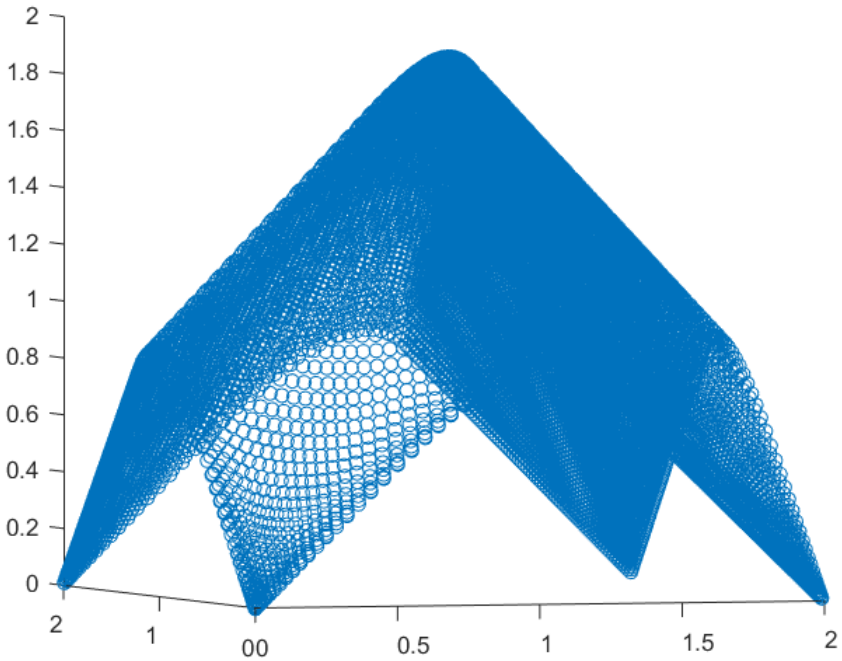

Figure C.2: NURBS surface for U_2 and V_2 with $w_{2,2} = 3$

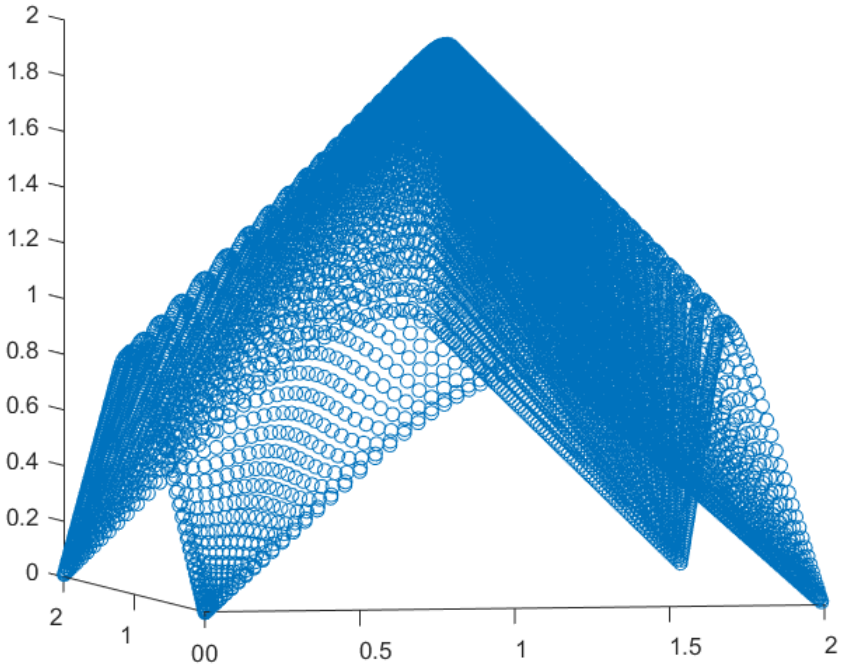

Figure C.3: NURBS surface for U_2 and V_2 with $w_{2,2} = 10$

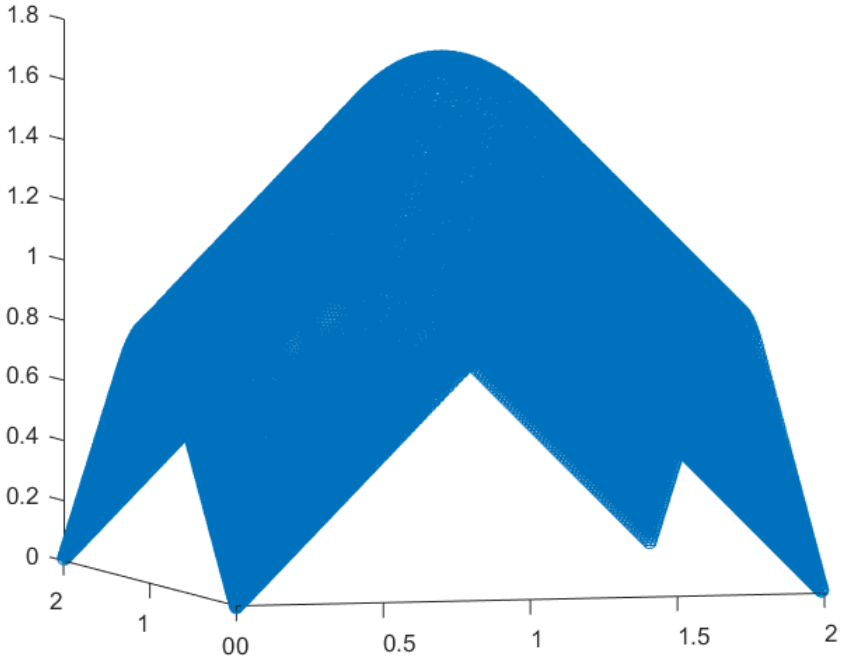

Figure C.4: NURBS surface for U_3 and V_3 with $w_{2,2} = 1$

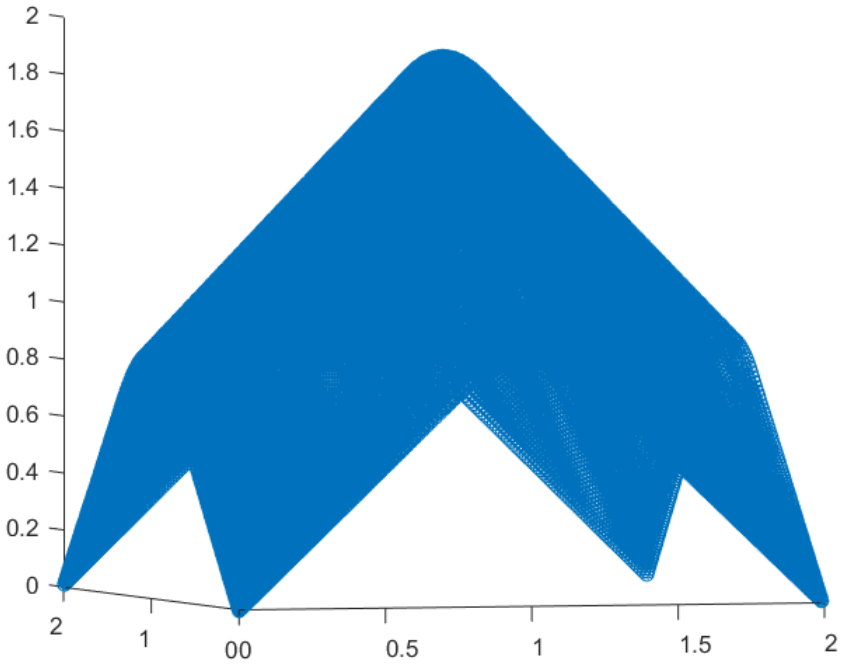

Figure C.5: NURBS surface for U_3 and V_3 with $w_{2,2} = 3$

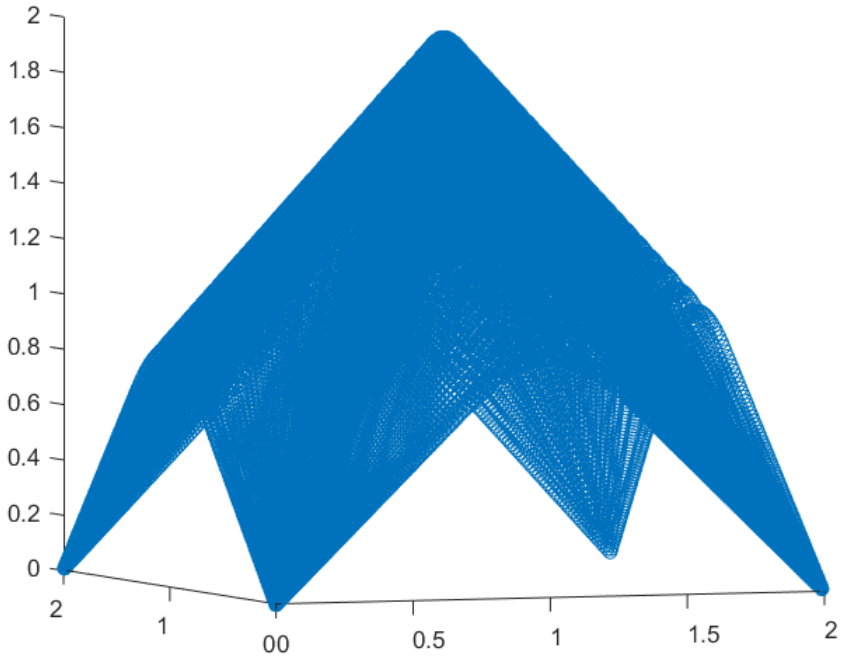

Figure C.6: NURBS surface for U_3 and V_3 with $w_{2,2} = 10$