Margret Kreimer BSc

Erstellung einer objekt-orientierten Software-Bibliothek für Non-Uniform Rational B-Splines (NURBS)

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

Masterstudium Maschinenbau

eingereicht an der

Technischen Universität Graz

Betreuer

Ao.Univ.-Prof. Mag.rer.nat. Dr.techn. Anton Gfrerrer

Institut für Geometrie

Graz, September 2015

EIDESSTATTLICHE ERKLÄRUNG

AFFIDAVIT

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

14.09.2015
Datum / Date

Kreimer Margret
Unterschrift / Signature

Vorwort

Die vorliegende Arbeit wurde am Institut für Geometrie an der Technischen Universität Graz durchgeführt. Die Aufgabenstellung zu diesem Themengebiet stammt von der Firma Andritz Hydro GmbH in Weiz.

An dieser Stelle möchte ich mich bei meinem Betreuer Herrn Ao.Univ.-Prof. Mag.rer.nat. Dr.techn. Anton Gfrerrer von der TU Graz für die fachliche Betreuung und die ausgezeichnete Unterstützung während der Durchführung meiner Arbeit bedanken.

Weiters bedanke ich mich bei Herrn Dipl.Ing. Dr.techn. Bernhard Karl Bachner für die Möglichkeit der Erstellung dieser Arbeit und für die kompetente Betreuung seitens der Firma Andritz Hydro GmbH.

Ein besonderer Dank gebührt meinen Eltern, Eleonora und Johann, die mir mein Studium ermöglichten und mich während meiner Studienzeit stets unterstützten.

Weiters gilt ein großer Dank meinem Freund René, der mir während meines Studiums immer zur Seite stand und mir dadurch die nötige Kraft spendete.

Graz, September 2015 Margret Kreimer

Kurzfassung

Die Arbeit beschäftigt sich mit der Erstellung einer objekt-orientierten Software-Bibliothek für Non-Uniform Rational B-Splines (NURBS).

Die erstellte Bibliothek bildet die Grundlage eines Geometriemodells, welches in Zukunft als Basis eines Finite-Elemente Modells zur Untersuchung des Schwingungsverhalten einer Stator-Wicklung von Synchrongeneratoren dienen soll. Die Software-Bibliothek wurde in MATLAB implementiert.

Um für verschiedene Geometrievarianten eine möglichst übersichtliche und leicht zu wartende Basis für die Modellbildung zu erstellen, ist eine objekt-orientierte Bibliothek für Computer Aided Geometric Design (CAGD) Voraussetzung. Für die mathematische Beschreibung dieser Objekte haben sich NURBS als Standard etabliert. Daher ist ein Datenaustausch zwischen der erstellten Bibliothek und einer CAD-Software einfach realisierbar.

Die Software-Bibliothek gliedert sich in eine Klasse B-Spline Basisfunktionen zur numerischen Auswertung der Basisfunktionen und deren Ableitungen, in eine Klasse NURBS zur numerischen Berechnung des Tensorproduktes aus den Matrizen der Basisfunktionen und dem Array der Kontrollstruktur. Weiters wurde eine Super-Klasse zur grafischen Beschreibung von Flächen implementiert. Von dieser wurde eine weitere Super-Klasse für Drehflächen abgeleitet und davon wiederum Spezialisierungen für technisch bedeutsame Drehflächen.

Abstract

This master thesis deals with the creation of an object-oriented software library for Non-Uniform Rational B-Splines (NURBS).

The constructed library forms the basis of a geometry model, which in future will be the foundation of a finite element model for investigating the vibration behavior of a stator winding of synchronous generators. The software library has been implemented in MATLAB.

To develop a clear and easy to maintain base of modelling different geometry variants, an object-oriented library for Computer Aided Geometric Design (CAGD) is required. For the mathematical description of these objects, NURBS have become a standard. Therefore, a data exchange between the created library and a CAD software is simple to realize.

The software library contains a class B-spline basis functions for the numerical computation of the basic functions and their derivatives, and a class NURBS for the computation of the tensor product of the matrices of the basic functions and the array of the control structure. In addition, a superclass for the graphical description of surfaces has been implemented. From this superclass, another one for surfaces of revolution has been derived and from this again some specializations for important technical surfaces of revolution.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Grundlagen

1.1 B-Spline Basisfunktionen

In Kapitel 1.1 werden grundlegende Eigenschaften und Definitionen von B-Spline Basisfunktionen beschrieben. Es werden die Begriffe *Knotenvektor* und *Basisfunktionen* definiert und das Zusammenspiel dieser Elemente gezeigt. Weiters wird auf die Ableitungen der B-Spline Basisfunktionen eingegangen. Die Herleitungen und weitere Ausführungen der nachfolgend angeführten Definitionen und Eigenschaften von B-Spline Basisfunktionen findet man z.B. in [4, Kap.2].

1.1.1 Definition der B-Spline Basisfunktionen

Für die Definition der B-Spline Basisfunktionen benötigt man einen sogenannten *Knotenvektor* **U**.

Definition 1.1 *Knoten und Knotenvektor*

Ein Knotenvektor **U** ist eine Folge $u_0, \ldots, u_m$ reeller Zahlen ($u_i \in \mathbb{R}$) mit $u_0 \leq u_1 \leq \cdots \leq u_m$. In weiterer Folge wird dafür die Schreibweise $\mathbf{U} = \{u_0, \ldots, u_m\}$ verwendet. Die Elemente u_i des Knotenvektors werden Knoten genannt. Aufeinanderfolgende Knoten spannen ein halboffenes Teilintervall $[u_i, u_{i+1})$ auf, welches *Knotenintervall* bzw. *i-tes Knotenintervall* genannt wird. Benachbarte Knoten können auch zusammenfallen: $u_i = u_{i+1}$. Es gibt Knotenvektoren mit äquidistanter Unterteilung (uniform) und mit nicht äqui-

distanter Unterteilung (nicht uniform). In *Tab. 1.1* sind Beispiele von Knotenvektoren dargestellt.

	Mehrfachbelegung am Rand	keine Mehrfachbelegung am Rand
uniform	**U**=[0,0,0,1,2,3,4,5,6,7,7,7] **U**=[0,0,0.25,0.5,0.75,1,1,1]	**U**=[3,4,5,6,7,8,9,10,11] **U**=[-6,-4,-2,0,2,4]
nicht uniform	**U**=[0,0,0,1.3,2.4,3.8,4.1,6.1,7,7,7] **U**=[0,0,0.3,0.5,0.9,1,1,1]	**U**=[0,0.3,1,1.5,2.8,3,4.2,6.1,6.5,7.5,8] **U**=[0,1,3.5,5.5,6,6.2,7]

Tabelle 1.1: Uniforme und nicht uniforme bzw. mehrfachbelegte und nicht mehrfachbelegte Knotenvektoren

Definition 1.2 *B-Spline Basisfunktionen*

Gegeben sei eine natürliche Zahl p und ein Knotenvektor $\mathbf{U} = \{u_0, \ldots, u_m\}$. Dann sind für $i = p, \ldots, m-p-1$ die B-Spline Basisfunktionen $N_{i,p}$ wie folgt definiert:

- Falls p = 0:

$$N_{i,0}(u) = \begin{cases} 1 & wenn \quad u_i \leq u < u_{i+1} \\ 0 & sonst \end{cases} \tag{1.1}$$

- Falls $p > 0$:

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{1.2}$$

Eigenschaften 1.1 *B-Spline Basisfunktionen*

- $N_{i,0}(u)$ ist eine Stufenfunktion mit $N_{i,0}(u) = 1$ falls $u \in [u_i, u_{i+1})$, überall sonst gilt $N_{i,0}(u) = 0$.
- Die B-Spline Basisfunktion $N_{i,p}(u)$ vom Grad $p > 0$ ist eine Linearkombination von zwei B-Spline Basisfunktionen vom Grad $p - 1$.
- Die rekursiven Berechnung der Basisfunktion kann in einem Dreieckschema veranschaulicht werden:

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

Abbildung 1.1: Dreieckschema für die rekursive Berechnung der Basisfunktionen.

- Im Knotenintervall $[u_i, u_{i+1})$ sind höchstens die $p+1$ Basisfunktionen $N_{i-p,p}(u), \ldots, N_{i,p}(u)$ von null verschieden.
- Für ein beliebiges Knotenintervall gilt die sogenannte *"partition of unity"*: $\sum_{j=i-p}^{i} N_{j,p}(u) \equiv 1$ für alle $u \in [u_i, u_{i+1})$. [4], [2]

In *Abb. 1.2* sind die Graphen von B-Spline Basisfunktionen mit dem Knotenvektor $\mathbf{U} = [0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1]$ für die Grade $p = 0$, 1, 2 und 3 dargestellt.

Die *Abb. 1.3* zeigt die Graphen von B-Spline Basisfunktionen zu den Graden $p = 0$, 1, 2, 3. Der linken Seite liegt der Knotenvektor $\mathbf{U} = [0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1]$ zugrunde, während rechts der Knotenvektor $\mathbf{U} = [0, 0, 0, 0.1, 0.3, 0.3, 0.5, 1, 1, 1]$ ist.

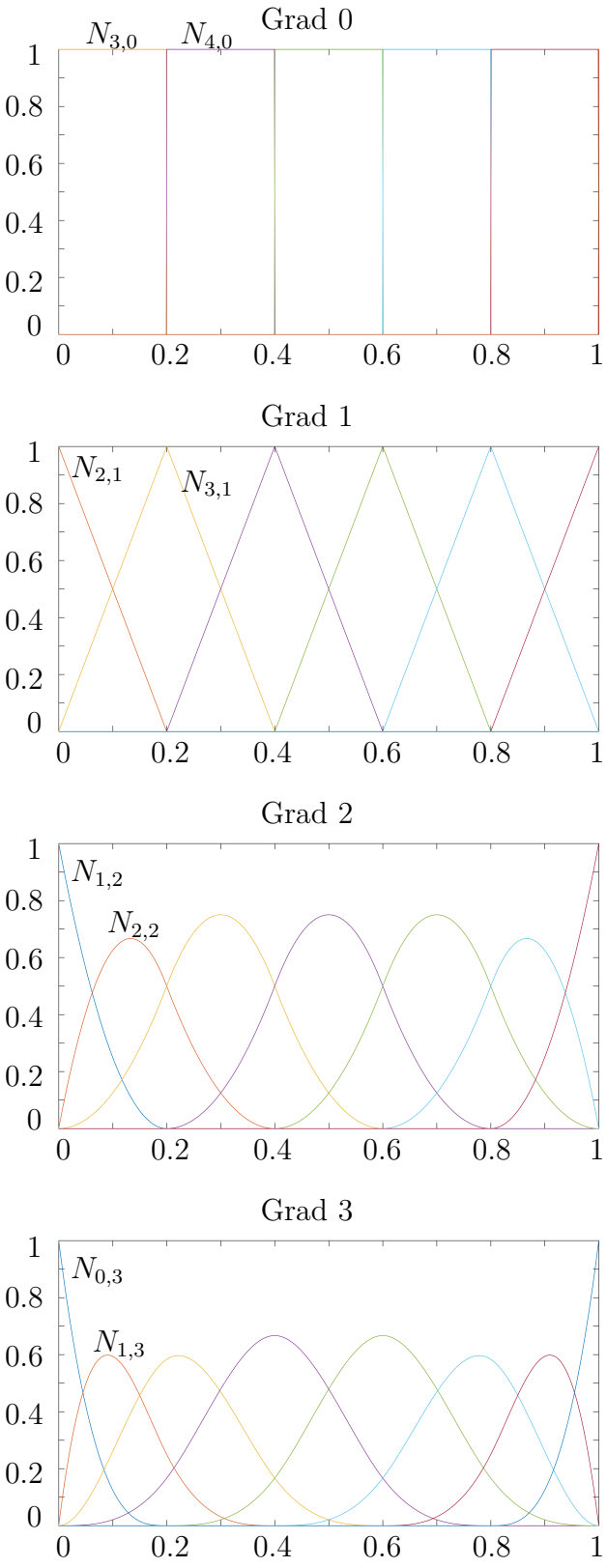

Abbildung 1.2: B-Spline Basisfunktionen zu den Graden $p = 0, 1, 2, 3$.

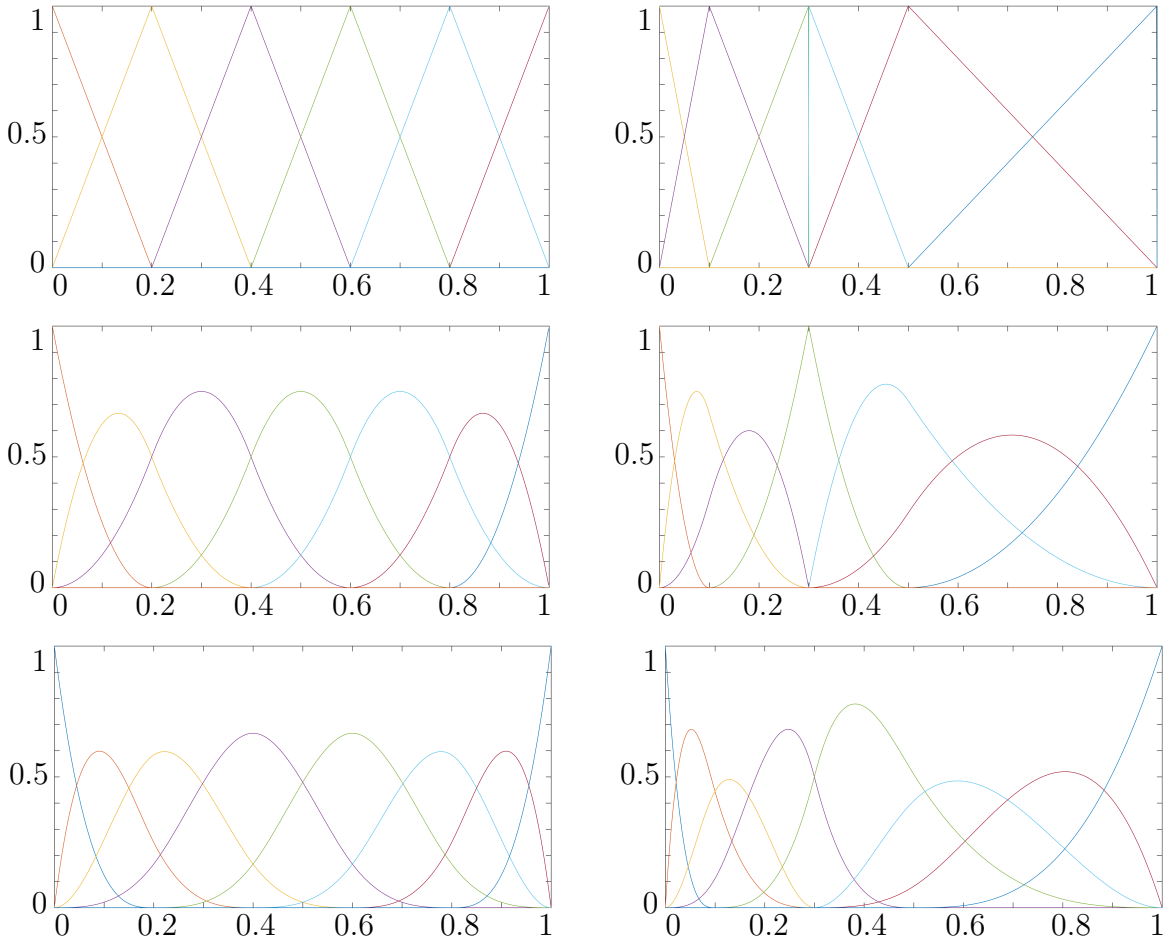

Abbildung 1.3: B-Spline Basisfunktionen mit periodischem (links) und nicht periodischem (rechts) Knotenvektor zu den Graden $p = 1, 2, 3$.

1.1.2 Ableitungen der B-Spline Basisfunktionen

Ableitungen von B-Spline Basisfunktionen können als Linearkombination von B-Spline Basisfunktionen eines niedrigeren Grades wie in *Gl. 1.3* gezeigt, dargestellt werden. Wenn $i = 0, \ldots, m$ und $p \geq 1$ gegeben ist, lautet die Ableitung der Basisfunktion:

$$N_{i,p}(u)' = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) \; - \; \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{1.3}$$

Die k-ten Ableitungen der B-Spline Basisfunktionen vom Grad p können gemäß *Gl. 1.4* als Linearkombination der $(k-1)$-ten Ableitungen der B-Spline Basisfunktionen vom Grad $p-1$ ausgedrückt werden.

Weiters ist zu beachten, dass für $k > p$ stets $N_{i,p}^{(k)}(u) \equiv 0$ gilt. [4, S.59ff]

$$N_{i,p}^{(k)}(u) = p\left(\frac{N_{i,p-1}^{(k-1)}(u)}{u_{i+p} - u_i} - \frac{N_{i+1,p-1}^{(k-1)}(u)}{u_{i+p+1} - u_{i+1}}\right) \tag{1.4}$$

In *Abb. 1.4* und *Abb. 1.5* sind die Graphen der Ableitungen der B-Spline Basisfunktionen und der dazugehörigen Basisfunktionen dargestellt. Die *Abb. 1.4* hat den Knotenvektor $\mathbf{U} = [0, 0, 0, 0, 2, 4, 6, 8, 8, 8, 8]$ und den Grad $p = 3$. In *Abb. 1.5* ist der Knotenvektor mit $\mathbf{U} = [0, 0, 0, 0, 2, 4, 4, 6, 6, 6, 8, 8, 8, 8]$ und der Grad mit $p = 3$ definiert. Man beachte, dass einige der Knoten mehrfach belegt sind.

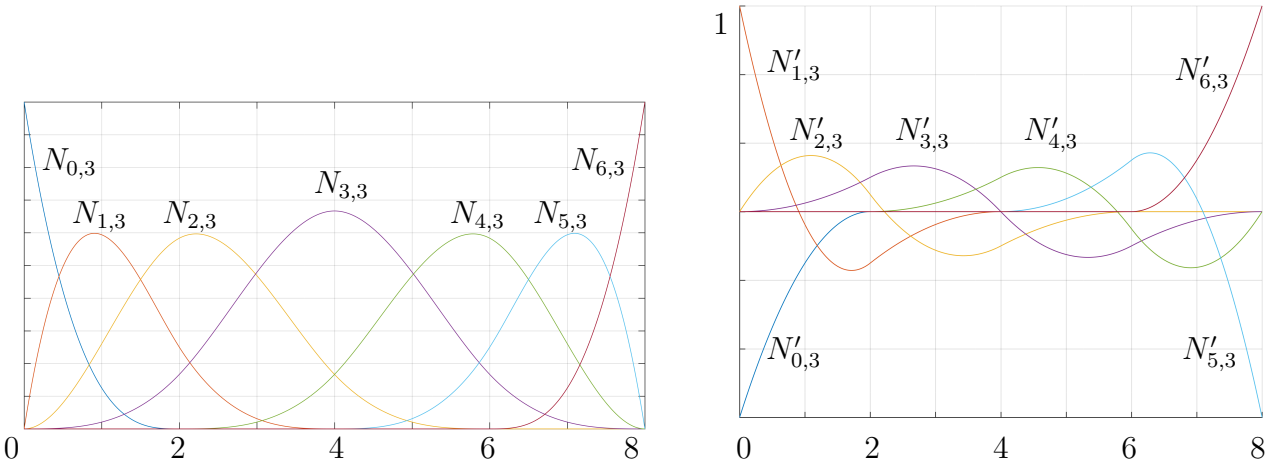

Abbildung 1.4: Basisfunktionen (links) und ihre Ableitungen (rechts); Knotenvektor $\mathbf{U} = [0, 0, 0, 0, 2, 4, 6, 8, 8, 8, 8]$; Grad $p = 3$

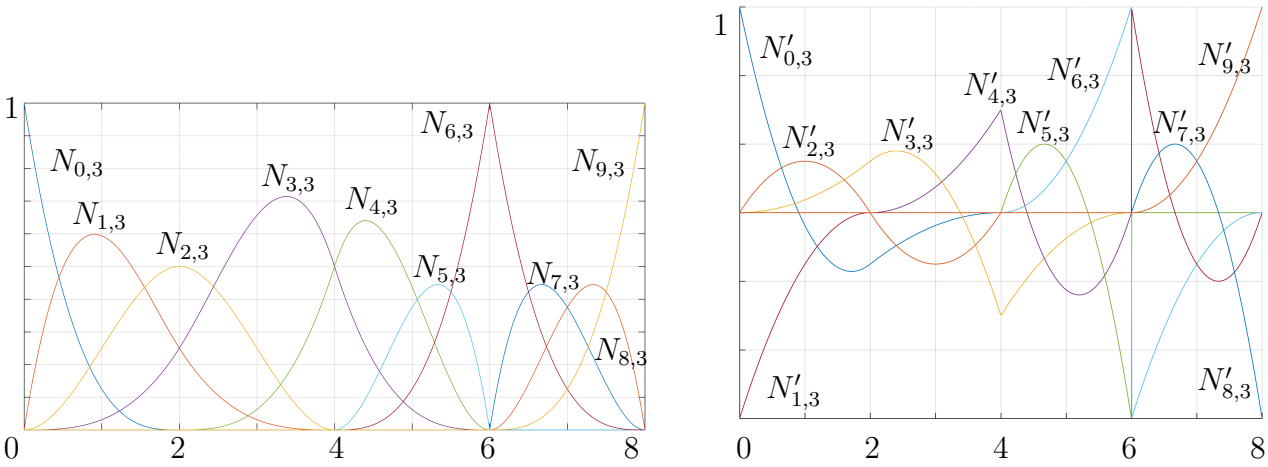

Abbildung 1.5: Basisfunktionen (links) und ihre Ableitungen (rechts); Knotenvektor $\mathbf{U} = [0, 0, 0, 0, 2, 4, 4, 6, 6, 6, 8, 8, 8, 8]$; Grad $p = 3$.

1.2 Gewöhnliche B-Splines

Das Kapitel 1.2 zeigt die Definitionen von gewöhnlichen B-Spline Kurven und B-Spline Flächen. Es werden die Funktionen zur Berechnung der B-Splines sowie deren wichtigsten Eigenschaften erläutert. Die Herleitungen und weitere Ausführungen der nachfolgend angeführten Definitionen und Eigenschaften von gewöhnlichen B-Splines findet man z.B. in [4, Kap. 3].

1.2.1 Gewöhnliche B-Spline Kurven

Um B-Spline Kurven definieren zu können, ist es notwendig zu dem zuvor beschriebenen Knotenvektor $\mathbf{U}$ und dem Grad p der Kurve zusätzlich, ein Kontrollpolygon $\mathbf{P}_0, \ldots, \mathbf{P}_n$ vorzugeben. Die Punkte $\mathbf{P}_i$ sind entweder aus $\mathbb{R}^2$ oder aus $\mathbb{R}^3$.

Definition 1.3 *B-Spline Kurve*

Gegeben sei eine natürliche Zahl p, ein Knotenvektor $\mathbf{U} = \{u_0, \ldots, u_m\}$ und ein Kontrollpolygon $\mathbf{P}_0, \ldots, \mathbf{P}_n$ wobei $m = n + p + 1$ gelte. Dann ist durch

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\ \mathbf{P}_i, \qquad u \in [u_p, u_{n+1}] \tag{1.5}$$

eine *B-Spline Kurve* vom Grad p zum Kontrollpolygon $\mathbf{P}_0, \ldots, \mathbf{P}_n$ und zum Knotenvektor $\mathbf{U}$ definiert. Hierbei sind die B-Spline Basisfunktionen vom Grad p, gemäß Definition 1.1, S. 1. Die Herleitung der nachfolgend angeführten Eigenschaften von B-Spline Kurven findet man z. B. in [4, S. 81 ff].

Eigenschaften 1.2 *B-Spline Kurve*

- Wird für den Knotenvektor speziell $u_0 = \cdots = u_p = a, u_{m-p} = \cdots = u_m = b$ gewählt, fällt der Anfangspunkt $\mathbf{C}(a)$ der B-Spline Kurve mit dem ersten Punkt $\mathbf{P}_0$ des Kontrollpolygons zusammen, und der Endpunkt $\mathbf{C}(b)$ fällt mit dem letzten Punkt $\mathbf{P}_n$ des Kontrollpolygons zusammen. Außerdem ist die Gerade $\mathbf{P}_0\mathbf{P}_1$ die Tangente von $\mathbf{C}(u)$ in $\mathbf{P}_0$. In diesem Fall sieht der Knotenvektor wie folgt aus:

  $$\mathbf{U} = \{\underbrace{a \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\} \tag{1.6}$$

  In weiterer Folge werden ausschließlich B-Spline Kurven mit einem solchen Knotenvektor verwendet.

- $\mathbf{C}(u)$ ist aufgrund der stückweise polynomialen Basisfunktionen $N_{i,p}(u)$, eine stückweise polynomiale Kurve. Durch die Gleichung

$$m = n + p + 1 \tag{1.7}$$

ist der Zusammenhang zwischen der Anzahl der Knoten $m + 1$, Anzahl der Kontrollpunkte $n + 1$ und dem Grad p der B-Spline Basisfunktionen gegeben.

- Eine B-Spline Kurve $\mathbf{C}(u)$ ist vollständig in der konvexen Hülle ihres Kontrollpolygons enthalten. Dies folgt aus den Eigenschaften bezüglich der Nichtnegativität und aus der *"partition of unity"* Eigenschaft der B-Spline Basisfunktionen (siehe Eigenschaften 1.1 *Basisfunktionen*, S. 2).

- Eine Änderung des Kontrollpunktes $\mathbf{P}_i$ hat nur im Intervall $[u_i, u_{i+p+1})$ eine Auswirkung, dies folgt aus $N_{i,p}(u) = 0$ für $u \notin [u_i, u_{i+p+1})$.

- Das Kontrollpolygon bildet eine stückweise lineare Approximation der Kurve. Eine generelle Regel ist dabei, je niedriger der Grad der Kurve p ist, desto näher liegt die B-Spline Kurve an ihrem Kontrollpolygon.

- Eine B-Spline Kurve ist affin invariant mit ihrem Kontrollpolygon verbunden.

- Innerhalb eines Knotenintervalls $[u_i, u_{i+p+1})$ ist eine B-Spline Kurve $\mathbf{C}$(u) unendlich oft differenzierbar. Ist u_i ein einfacher Knoten, ist $\mathbf{C}$(u) an dieser Stelle $(p-1)$-fach differenzierbar. Ist u_i ein k-facher Knoten (d.h $u_{i-1} \neq u_i = \cdots = u_{i+k-1} \neq u_{i+k}$) dann ist $\mathbf{C}$(u) an dieser Stelle nur $(p-k)$-fach differenzierbar.

1.2.2 Gewöhnliche B-Spline Flächen

Bei den B-Spline Flächen ist die erforderliche Kontrollstruktur ein Rechteckschema

$$\begin{matrix} \mathbf{P}_{0,0} & \cdots & \mathbf{P}_{0,m} \\ \vdots & & \vdots \\ \mathbf{P}_{n,0} & \cdots & \mathbf{P}_{n,m} \end{matrix}$$

von Punkten im $\mathbb{R}^3$. Dieses Rechteckschema wird als *Kontrollnetz* bezeichnet.

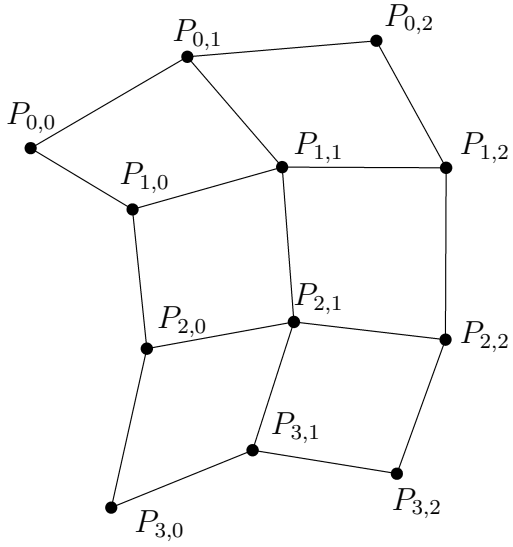

Abbildung 1.6: Kontrollnetz einer B-Spline Fläche.

Abb. 1.6 zeigt ein solches Netz für $n = 3$ und $m = 2$. Die Herleitung der nachfolgend angeführten Eigenschaften von B-Spline Flächen findet man z.B. in [4, S. 100 ff].

Definition 1.4 *B-Spline Fläche*

Gegeben seien die natürlichen Zahlen p und q, die Knotenvektoren $\mathbf{U} = \{u_0, \ldots, u_r\}$ und $\mathbf{V} = \{v_0, \ldots, v_s\}$, ein Kontrollnetz $\mathbf{P}_{i,j}$ mit $i = 0 \ldots, n$ und $j = 0 \ldots, m$. Weiters gelte $r = n + p + 1$ und $s = m + q + 1$. Dann ist durch

$$\mathbf{S}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u)\ N_{j,q}(v)\ \mathbf{P}_{i,j}, \qquad (u, v) \in [u_{p-1}, u_{n+1}] \times [v_{q-1}, v_{m+1}] \tag{1.8}$$

eine *B-Spline Fläche* (*B-Spline patch*) vom Grad p, q zum Kontrollnetz $\mathbf{P}_{i,j}$ und zu den Knotenvektoren $\mathbf{U}$ und $\mathbf{V}$ definiert.

Eigenschaften 1.3 *B-Spline Fläche*

- $\mathbf{S}(u, v)$ ist aufgrund der stückweise polynomialen Basisfunktionen $N_{i,p}(u)$ und $N_{j,q}(v)$ eine stückweise polynomiale Fläche. Durch die Gleichungen

$$r = n + p + 1 \tag{1.9}$$

$$s = m + q + 1 \tag{1.10}$$

ist der Zusammenhang zwischen der Anzahl $r + 1$ der Knoten, der Anzahl $n + 1$ der Kontrollpunkte und dem Grad p der B-Spline Basisfunktionen in u-Richtung bzw.

zwischen der Anzahl $s+1$ der Knoten, der Anzahl $m+1$ der Kontrollpunkte und dem Grad q in v-Richtung gegeben.

- Wird für die Knotenvektoren speziell $u_0 = \cdots = u_p = a, u_{r-p} = \cdots = u_r = b$ und $v_0 = \cdots = v_q = c, v_{s-q} = \cdots = v_s = d$ gewählt, also

$$\mathbf{U} = \{\underbrace{a \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{r-p-1}, \underbrace{b, \ldots, b}_{p+1}\} \tag{1.11}$$

$$\mathbf{V} = \{\underbrace{c \ldots, c}_{q+1}, v_{q+1}, \ldots, v_{s-q-1}, \underbrace{d, \ldots, d}_{q+1}\}. \tag{1.12}$$

  dann fällt die Ecke $\mathbf{S}(a, c)$ des B-Spline Patches mit dem Punkt $\mathbf{P}_{0,0}$ des Kontrollnetzes zusammen. Außerdem spannen die zwei Strecken $\mathbf{P}_{0,0}$ $\mathbf{P}_{0,1}$ und $\mathbf{P}_{0,0}$ $\mathbf{P}_{1,0}$ die Tangentialebene im Punkt $\mathbf{P}_{0,0} = \mathbf{S}(a, c)$ auf. Analoges gilt bei dieser Wahl der Knoten für die anderen drei Ecken des Patches.

  In weiterer Folge werden ausschließlich B-Spline Flächen mit Knotenvektoren gemäß *Gl. 1.11* und *Gl. 1.12* verwendet.

- Es gilt: $\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)\, N_{j,q}(v) \equiv 1$ für alle $(u, v) \in [u_{p-1}, u_{n+1}] \times [v_{q-1}, v_{m+1}]$

- Eine B-Spline Fläche ist vollständig in der konvexen Hülle ihres Kontrollnetzes enthalten. Dies folgt aus der Nichtnegativität und aus der *"partition of unity"* Eigenschaft der B-Spline Basisfunktionen.

- Eine Änderung des Punktes $\mathbf{P}_{i,j}$ des Kontrollnetzes, wirkt sich nur im Parametergebiet $[u_i, u_{i+p+1}) \times [v_j, v_{j+p+1})$ aus. Dies folgt aus $N_{i,p}(u)\, N_{j,q}(v) = 0$ für $(u, v) \notin [u_i, u_{i+p+1}) \times [v_j, v_{j+p+1})$.

- Das Kontrollnetz bildet eine stückweise lineare Approximation der Fläche. Eine generelle Regel ist dabei, dass je niedriger p und q sind, desto besser wird die B-Spline Fläche durch ihr Kontrollnetz angenähert.

- Eine B-Spline Fläche ist affin invariant mit ihrem Kontrollnetz verbunden.

- $\mathbf{S}$(u,v) ist innerhalb von $[u_i, u_{i+1}] \times [v_j, v_{j+1}]$ eine bivariate polynomiale Funktion und daher existieren in diesem Bereich alle partiellen Ableitungen von $\mathbf{S}(u, v)$. Ist u_i bzw. v_j ein einfacher Knoten, dann ist $\mathbf{S}$(u,v) an dieser Stelle l-fach differenzierbar, wobei $l = min\{p, q\} - 1$. Bei mehrfacher Knotenbelegung sinkt die Differenzierbarkeit entsprechend. [2], [4]

1.3 Rationale B-Splines - NURBS

In diesem Kapitel werden die grundlegenden Eigenschaften und Definitionen von rationalen B-Splines (NURBS) beschrieben, welche auch den Schwerpunkt der vorliegenden Arbeit darstellen. NURBS steht abgekürzt für **N**on **U**niform **R**ational **B**-**S**plines. Bei diesen Kurven und Flächen wird die gesamte Kontrollstruktur (Knotenvektor, Kontrollpolygon oder -netz) durch einen zusätzlichen Satz von Parametern, den so genannten *Gewichten*, erweitert. Durch diese weiteren Parameter wird die Gestaltungsmöglichkeit der Kurven und Flächen wesentlich vielfältiger. Die Gewichte bieten die Möglichkeit die B-Splines lokal zu modellieren. Um diese Gewichte zu definieren werden *homogene Koordinaten* eingeführt. Weiters wird auf die Ableitungen von NURBS Kurven und NURBS Flächen eingegangen.

1.3.1 Verwendung homogener Koordinaten

Um eine rationale Kurve im n-dimensionalen Raum $\mathbb{R}^n$ als Polynomkurve im $(n+1)$-dimensionalen Raum $\mathbb{R}^{n+1}$ darzustellen, bietet die Verwendung von homogene Koordinaten eine effiziente Methode. Die Beschreibung der homogenen Koordinaten wurde in Anlehnung an [4, S. 30ff] erstellt.

Zunächst wird der Fall $n = 2$ betrachtet. Für die homogene Darstellung wird der Punkt $\mathbf{P} = (x, y) \in \mathbb{R}^2$ mit den Gewichten $w \in \mathbb{R}\backslash\{0\}$ auf den Punkt $\mathbf{P}^w = (w\,x, w\,y, w) = (X, Y, Z) \in \mathbb{R}^3$ abgebildet. Um umgekehrt den Punkt $\mathbf{P} = (x, y) \in \mathbb{R}^2$ aus $\mathbf{P}^w := (X, Y, Z) \in \mathbb{R}^3$ zu erhalten, werden die ersten zwei Koordinaten des Punktes $\mathbf{P}^w$ durch die dritte Koordinate, welche die Gewicht darstellt, dividiert. Diese Vorgehensweise ist in *Gl. 1.13* ersichtlich. Die Abbildung $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ ist eine Zentralprojektion H, deren Projektionszentrum im Ursprung O des zugrundeliegenden Koordinatensystems liegt, und deren Bildebene ε die Ebene mit der Gleichung $Z = 1$ ist (siehe *Abb. 1.7*).

$$\mathbf{P} = H(\mathbf{P}^w) = H(wx, wy, w) = H(X,\ Y,\ Z) = \left(\frac{w\,x}{w}, \frac{w\,y}{w}\right) = (x, y) \tag{1.13}$$

Die Punkte $\mathbf{P}^w$ mit den selben x-, y-Werten, aber verschiedenen Gewichten w, liegen auf einem Projektionsstrahl (=Gerade durch O) und liefern daher ein- und denselben Punkt $\mathbf{P}$:

$$\mathbf{P} = H(\mathbf{P}^{w_1}) = H(w_1x, w_1y, w_1) = (x, y) = H(w_2x, w_2y, w_2) = H(\mathbf{P}^{w_2}) \tag{1.14}$$

Definition 1.5

Man bezeichnet X, Y, Z als die homogenen Koordinaten des Punktes $\mathbf{P} \in \mathbb{R}^2$ und x, y als dessen inhomogene (affine) Koordinaten.

Bemerkung 1.1

Die homogenen Koordinaten eines Punktes $\mathbf{P}$ des $\mathbb{R}^2$ sind nur bis auf einen Faktor w bestimmt (siehe *Gl. 1.14*).

In *Abb. 1.7* wird die Zentralprojektion H eines Punktes $P^w = (X, Y, Z) \in \mathbb{R}^3$ auf die Ebene ε gezeigt. Neben dem räumlichen Koordinatensystem mit dem Ursprung in O und den Achsen X, Y und Z wird in der Bildebene ε das Koordinatensystem mit dem Ursprung Q und den Achsen x, y ($Q = Z \cap \varepsilon, x \parallel X, y \parallel Y$) verwendet.

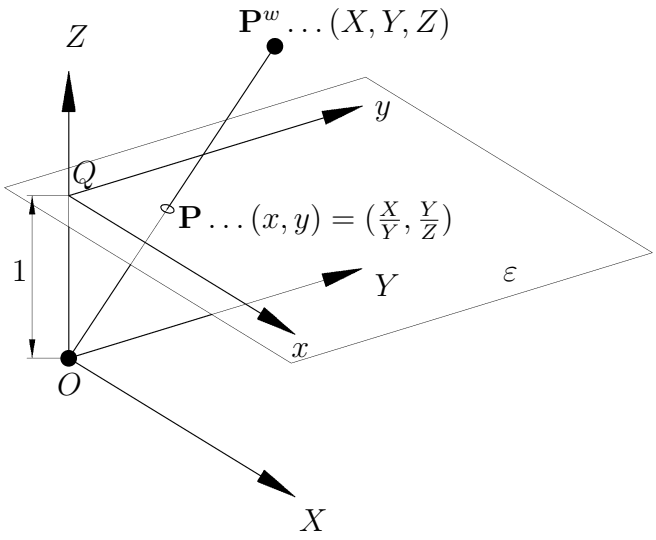

Abbildung 1.7: Zentralprojektion eines Punktes P^w des $\mathbb{R}^3$ auf die Ebene in $\varepsilon \ldots Z = 1$.

Diese Vorgangsweise lässt sich nun auch für höhere Dimensionen verallgemeinern. In der weiteren Ausführung dieser Arbeit wird nur der Fall $n = 3$ betrachtet.

Wir betrachten nun einen Punkt $\mathbf{P} = (x, y, z) \in \mathbb{R}^3$ und ein Gewicht $w \in \mathbb{R}\backslash\{0\}$, und weisen diesem Paar $(\mathbf{P}, w)$ den Punkt $\mathbf{P}^w = (w\,x, w\,y, w\,z, w) = (X, Y, Z, W) \in \mathbb{R}^4$

zu. Um umgekehrt den Punkt $\mathbf{P} = (x, y, z) \in \mathbb{R}^3$ aus der homogenen Darstellung $\mathbf{P}^w = (X, Y, Z, W) \in \mathbb{R}^4$ zu erhalten, werden die ersten drei Koordinaten des Punktes $\mathbf{P}^w$ durch die vierte Koordinate dividiert, siehe *Gl. 1.15*. Bei dieser Abbildung handelt es sich wiederum um eine Zentralprojektion H. Das Projektionszentrum liegt im Koordinatenursprung des zugrundeliegenden Koordinatensystems mit vier Koordinaten, die zugehörige Bildebene ist eine Hyperebene.

$$\mathbf{P} = H(\mathbf{P}^w) = H(wx, wy, wz, w) = H(X, Y, Z, W) = \left(\frac{w\,x}{w}, \frac{w\,y}{w}, \frac{w\,z}{w}\right) = (x, y, z) \tag{1.15}$$

Definition 1.6

Man bezeichnet X, Y, Z, W als die homogenen Koordinaten des Punktes $\mathbf{P} \in \mathbb{R}^4$ und x, y, z als dessen inhomogene (affine) Koordinaten.

Bemerkung 1.2

Die homogenen Koordinaten eines Punktes des $\mathbb{R}^3$ sind nur bis auf einen Faktor w bestimmt (siehe *Gl. 1.15*).

Die Verwendung von homogenen Koordinaten und die Zentralprojektion H spielt bei der Einführung von NURBS Kurven und NURBS Flächen eine wesentliche Rolle (siehe Abschnitt 1.3.2, S. 15 und Abschnitt 1.3.7, S. 23).

1.3.2 NURBS Kurven

Die Herleitungen und weiteren Ausführungen der nachfolgend angeführten Definitionen und Eigenschaften von NURBS Kurven findet man z. B. in [4, Kap. 4].

Definition 1.7 *NURBS Kurve*

Gegeben sei eine natürliche Zahl p, ein Knotenvektor $\mathbf{U} = \{u_0, \ldots, u_m\}$, ein Kontrollpolygon $\mathbf{P}_0, \ldots, \mathbf{P}_n$ mit $m = n + p + 1$ und Gewichte $w_0, \ldots, w_n \in \mathbb{R}^+$. Dann ist durch

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u)\; w_i\; \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u)\; w_i}, \qquad u_p \le u \le u_{n+1} \tag{1.16}$$

eine Parameterdarstellung einer *NURBS Kurve* vom Grad p zum Kontrollpolygon $\mathbf{P}_0, \ldots, \mathbf{P}_n$ mit den Gewichten $w_0, \ldots, w_n$ und dem Knotenvektor $\mathbf{U}$ definiert.

Eigenschaften 1.4 *NURBS Kurve*

Wird für den Knotenvektor speziell $u_0 = \cdots = u_p = a, u_{m-p} = \cdots = u_m = b$ gewählt, fällt der Anfangspunkt $\mathbf{C}(a)$ der NURBS Kurve mit dem ersten Punkt $\mathbf{P}_0$ des Kontrollpolygons zusammen, und der Endpunkt $\mathbf{C}(b)$ fällt mit dem letzten Punkt $\mathbf{P}_n$ des Kontrollpolygons zusammen.

Abgesehen von der Tatsache, dass eine NURBS Kurve stückweise rational statt stückweise polynomial ist, gelten alle unter *Kap. 1.2* , S.7 für gewöhnliche B-Spline Kurven formulierten Eigenschaften auch für NURBS Kurven. Außerdem gilt für NURBS Kurven:

- Eine Änderung des Kontrollpunktes $\mathbf{P}_i$ und bzw. oder des Gewichtes w_i hat nur im Intervall $[u_i, u_{i+p+1})$ einen Einfluss auf die NURBS Kurve. Durch die Gewichte gibt es bei den NURBS Kurven eine weitere Möglichkeit die lokale Form der Kurve zu beeinflussen. Je größer das Gewicht w_i eines Kontrollpunktes $\mathbf{P}_i$ gewählt wird, desto mehr nähert sich die Kurve dem Kontrollpunkt $\mathbf{P}_i$. In *Abb. 1.8* wird dies dargestellt.

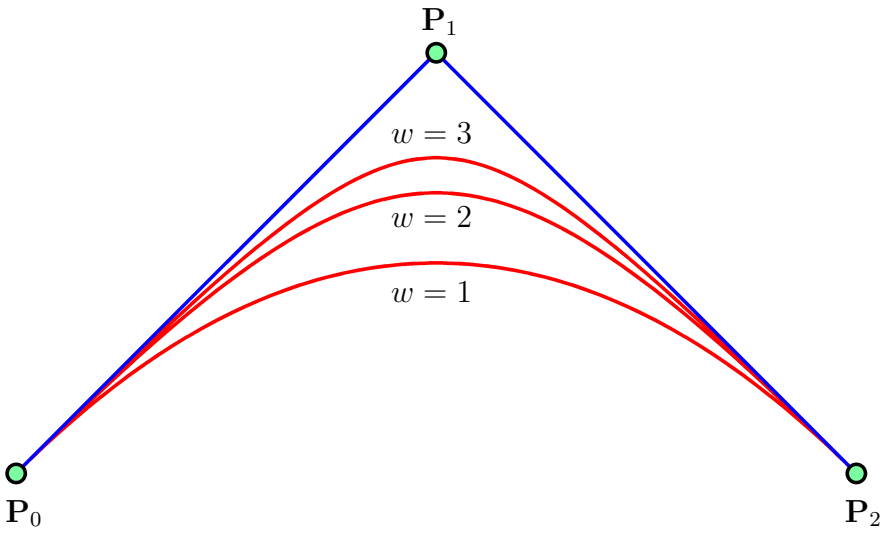

Abbildung 1.8: Darstellung verschiedener Gewichte des Kontrollpunktes $\mathbf{P}_1$.

- Wenn alle Gewichte den gleichen Wert besitzen ($w_0 = \cdots = w_n = w$) ist die NURBS Kurve ident mit der gewöhnlichen B-Spline Kurve zum selben Kontrollpolygon und demselben Kontenvektor. Dies folgt aus der "*partition of unity*" Eigenschaft: [2, S. 111]

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u)\ w_i\ \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u)\ w_i} = \frac{w_i \sum_{i=0}^{n} \mathbf{P}_i\ N_{i,p}(u)}{w_i \underbrace{\sum_{i=0}^{n} N_{i,p}(u)}_{1}} = \sum_{i=0}^{n} N_{i,p}(u)\ \mathbf{P}_i \tag{1.17}$$

Die in Kapitel 1.3.1 (S. 11) eingeführten homogenen Koordinaten bieten eine effiziente Methode um NURBS Kurven darzustellen. Jede NURBS Kurve $\mathbf{C}(u)$ des $\mathbb{R}^3$ entsteht als Zentralprojektion $\mathbf{C}(u) = H(\mathbf{C}^w(u)) = H\{\sum_{i=0}^{n}\ N_{i,p}(u)\ \mathbf{P}_i^w\}$ der gewöhnlichen B-Spline Kurve $\mathbf{C}^w(u)$ (*Gl. 1.18*) des $\mathbb{R}^4$.

$$\mathbf{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u) \begin{pmatrix} w_i \mathbf{P}_i \\ w_i \end{pmatrix} := \sum_{i=0}^{n} N_{i,p}(u)\ \mathbf{P}_i^w \tag{1.18}$$

Analog entsteht jede NURBS Kurve $\mathbf{C}(u)$ des $\mathbb{R}^2$ als Zentralprojektion $\mathbf{C}(u) = H(\mathbf{C}^w(u))$ einer gewöhnlichen B-Spline Kurve $\mathbf{C}^w(u)$ des $\mathbb{R}^3$. Dieser Sachverhalt ist in *Abb. 1.9* illustriert. Bei dieser Zentralprojektion H werden auch die Kontrollpunkte $\mathbf{P}_i^w$ der B-Spline Kurve $\mathbf{C}^w(u)$ auf die Kontrollpunkte $\mathbf{P}_i$ der NURBS Kurve $\mathbf{C}(u)$ projiziert: $\mathbf{P}_i = H(\mathbf{P}_i^w)$ [4, S.121]

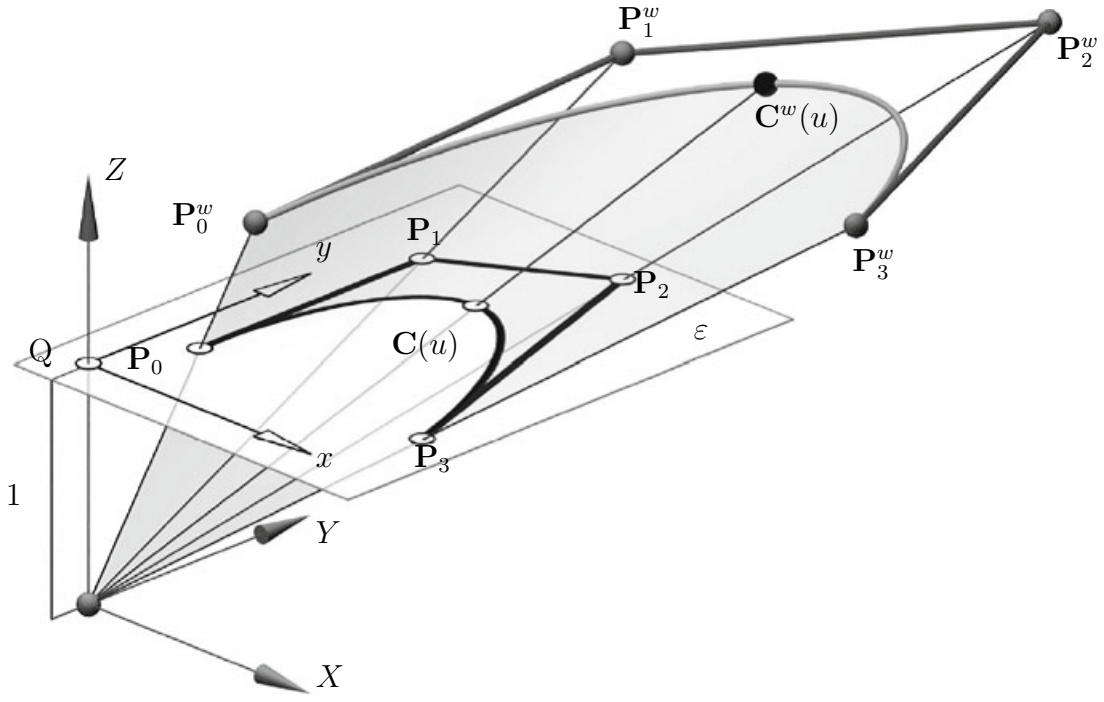

Abbildung 1.9: Zentralprojektion einer gewöhnlichen B-Spline Kurve $\mathbf{C}^w(u) \in \mathbb{R}^3$ auf eine ebene NURBS Kurve $\mathbf{C}(u) \in \mathbb{R}^2$. [2, S.109]

1.3.3 Kreise als NURBS Kurven

Da eine NURBS Kurve vom Grad $p = 2$ eine stückweise rationale Parameterdarstellung vom Grad 2 besitzt, besteht sie aus Kegelschnittsbögen (Stücke von Ellipsen, Hyperbeln, Parabeln) und in Ausnahmefällen aus Geradenstücken. Da ein Kreis eine Sonderform einer Ellipse darstellt und im Laufe dieser Arbeit auch Drehflächen in NURBS Darstellung erzeugt werden sollen, wird zunächst erläutert, wie Kreisbögen durch NURBS Kurven repräsentiert werden können. [4, S.29]

Einen exakten Viertelkreis (siehe *Abb. 1.10*) als NURBS Kurve erhält man durch folgende Vorgangsweise.

- Man wählt als Grad $p = 2$ die Anzahl der Kontrollpunkte minus eins mit $n = 2$ (damit ergibt sich $m = n + p + 1 = 5$),
- den Knotenvektor $\mathbf{U} = \{0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$,
- das Kontrollpolygon $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$ als drei aufeinanderfolgende Ecken eines Quadrates, wobei die Quadratseitenlänge der Radius des Kreises ist,
- die Gewichte der 3 Kontrollpunkte zu $w_0 = 1, w_1 = \frac{1}{\sqrt{2}}, w_2 = 1$.

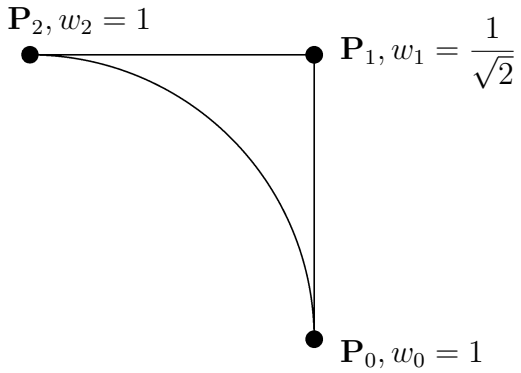

Abbildung 1.10: NURBS Viertelkreis mit drei Kontrollpunkten.

Einen Vollkreis (*Abb. 1.11*) als NURBS Kurve erhält man etwa wie folgt:

- Man wählt als Grad $p = 2$ der NURBS Kurve, die Anzahl der Kontrollpunkte minus eins mit $n = 8$ (damit ergibt sich $m = n + p + 1 = 11$),
- den Knotenvektor $\mathbf{U} = \{0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1\}$,

- das Kontrollpolygon $\mathbf{P}_0, \ldots, \mathbf{P}_8$. $\mathbf{P}_1, \mathbf{P}_3, \mathbf{P}_5, \mathbf{P}_7$ sind die Ecken eines dem Kreis umschriebenen Quadrates und $\mathbf{P}_0 = \mathbf{P}_8, \mathbf{P}_2, \mathbf{P}_4, \mathbf{P}_6$ sind die Seitenhalbierungspunkte dieses Quadrates (siehe *Abb. 1.11*).
- Für die Punkte mit geradem Index ist als Gewicht $w_i = 1$ zu wählen, und für die Punkte mit ungeradem Index $w_i = \frac{1}{\sqrt{2}}$.

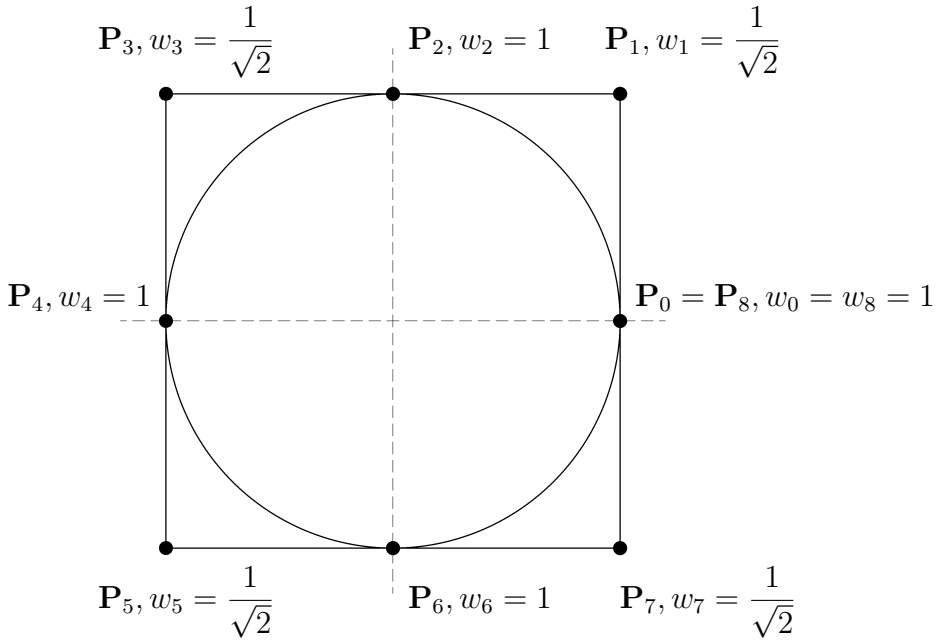

Abbildung 1.11: NURBS Vollkreis mit neun Kontrollpunkten.

Wir wollen nun für den Fall des Viertelkreises (siehe *Abb. 1.10*) die oben beschriebene rationale Parametrisierung berechnen. Hierbei wählen wir $r = 1$ als Radius des Kreises und $O(0, 0)$ als Mittelpunkt. Es handelt sich also um den in O zentrierten Einheitskreis c mit der Gleichung $x^2 + y^2 - 1 = 0$ Es wird jenes Viertel des Kreises parametrisiert, welches im 1. Quadranten liegt:
Die Kontrollpunkte sind daher $\mathbf{P}_0 = (1, 0), \mathbf{P}_1 = (1, 1), \mathbf{P}_2 = (0, 1)$ und somit erhalten wir $\mathbf{P}_0^w = (1, 0, 1), \mathbf{P}_1^w = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}), \mathbf{P}_2^w = (0, 1, 1)$. Außerdem erhalten wir gemäß *Gl. 1.1* und *Gl. 1.2* (S. 2) für den gegebenen Knotenvektor $\mathbf{U} = \{0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$ und den Grad $p = 2$ die folgenden drei B-Spline Basisfunktionen:

$$N_{0,2}(u) = (1 - 4u)^2, \; N_{1,2}(u) = 8t(1 - 4u), \; N_{2,2}(u) = 16u^2 \tag{1.19}$$

Für $u \in [0, \frac{1}{4}]$ ist daher

$$\mathbf{C}^w(u) = \sum_{i=0}^{2} N_{i,2}(u)\, \mathbf{P}_i^w = N_{0,2}(u) \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + N_{1,2}(u) \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} + N_{2,2}(u) \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \tag{1.20}$$

$$\begin{bmatrix} N_{0,2}(u) + \frac{1}{\sqrt{2}} N_{1,2}(u) \\ \frac{1}{\sqrt{2}}\ N_{1,2}(u) + N_{2,2}(u) \\ N_{0,2}(u) + \frac{1}{\sqrt{2}} N_{1,2}(u) + N_{2,2}(u) \end{bmatrix}.$$

Mit *Gl. 1.19* erhält man

$$\mathbf{C}^w(u) = \begin{bmatrix} (1-4t)^2 + 4\sqrt{2}\, u(1-4u) \\ 4\sqrt{2}\, u(1-4u) + 16u^2 \\ (1-4t)^2 + 4\sqrt{2}\, u(1-4u) + 16u^2 \end{bmatrix} =: \begin{bmatrix} f(u) + g(u) \\ g(u) + h(u) \\ f(u) + g(u) + h(u) \end{bmatrix}, \tag{1.21}$$

wobei wir

$$f(u) := (1-4u)^2, \qquad g(u) := 4\sqrt{2}\, u\,(1-4u), \qquad h(u) := 16u^2 \tag{1.22}$$

gesetzt haben. Die inhomogene Parametrisierung des Viertelkreises ist daher:

$$\mathbf{C}(u) \;=\; \begin{bmatrix} x(u) \\ y(u) \end{bmatrix} \;=\; \frac{1}{f(u) + g(u) + h(u)} \begin{bmatrix} f(u) + g(u) \\ g(u) + h(u) \end{bmatrix} \tag{1.23}$$

Man überprüft nun leicht, dass *Gl. 1.23* tatsächlich eine Parametrisierung des in $O(0,0)$ zentrierten Einheitskreises darstellt:

$$x^2(u) + y^2(u) - 1 \;=\; \frac{1}{(f+g+h)^2}[(f+g)^2 + (g+h)^2 - (f+g+h)^2]$$

$$= \frac{g^2 - 2fh}{(f+g+h)^2} = \frac{32u^2(1-4u)^2 - 2(1-4u)^2 16u^2}{(f+g+h)^2} = 0$$

Neben der oben gegebenen Parametrisierung *Gl. 1.23* besitzt der in Rede stehende Einheitskreis c auch noch die Standardparametrisierung:

$$\mathbf{c}(\psi) \;=\; \begin{bmatrix} x(\psi) \\ y(\psi) \end{bmatrix} \;=\; \begin{bmatrix} \cos\psi \\ \sin\psi \end{bmatrix} \tag{1.24}$$

wobei $\psi \in [0, 2\pi]$ die Bogenlänge auf c ist. Der Winkel ψ ist auch der Winkel zwischen der x-Achse des Koordinatensystems und dem Radiusvektor $\mathbf{c}(\psi)$.

1.3.4 Berechnung des Winkels ψ aus dem Parameter u

Wenn u gegeben ist, kann der entsprechende Winkel ψ wie folgt berechnet werden:

$$\psi \;=\; \arctan\,\frac{y(u)}{x(u)} \;=\; \arctan\,\frac{g+h}{f+g} \tag{1.25}$$

1.3.5 Berechnung des Parameters u aus dem Winkel ψ

Ist im umgekehrten Fall der Winkel ψ gegeben, dann kann der zugehörige Parameter u so berechnet werden:

$$u \;=\; \frac{(\sqrt{2}-1)(1-\cos\psi)+\sin\psi}{4(\sqrt{2}-1)\left[\sqrt{2}(\cos\psi+\sin\psi)+2\right]} \tag{1.26}$$

Diese Umrechnungsformel leitet man etwa wie folgt her:
Zunächst erhalten wir aus *Gl. 1.23*:

$$\frac{f+g}{f+g+h} = \cos\psi \qquad \frac{g+h}{f+g+h} = \sin\psi \tag{1.27}$$

Durch Umformen erhält man das homogene lineare Gleichungssystem

$$\left.\begin{aligned} f(1-\cos\psi)+g(1-\cos\psi)-h\cos\psi &= 0 \\ -f\sin\psi+g(1-\sin\psi)+h(1-\sin\psi) &= 0 \end{aligned}\right\} \tag{1.28}$$

in den Variablen f, g und h.

Dessen Lösung ist:

$$\begin{bmatrix} f \\ g \\ h \end{bmatrix} \;=\; \lambda\left(\begin{bmatrix} 1-\cos\psi \\ 1-\cos\psi \\ -\cos\psi \end{bmatrix} \times \begin{bmatrix} -\sin\psi \\ 1-\sin\psi \\ 1-\sin\psi \end{bmatrix}\right) = \lambda \begin{pmatrix} 1-\sin\psi \\ \cos\psi+\sin\psi-1 \\ 1-\cos\psi \end{pmatrix} \tag{1.29}$$

Durch Einsetzen von *Gl. 1.22* erhält man das inhomogene lineare Gleichungssystem in λ und u:

$$\left.\begin{aligned} \lambda(\sin\psi+\cos\psi)+8t &= 1 \\ \lambda(\cos\psi+\sin\psi-1+\sqrt{2}-\sqrt{2}\cos\psi)-4\sqrt{2}t &= 0 \end{aligned}\right\} \tag{1.30}$$

Anhand der *Cramerschen Regel* kann daraus die Variable u berechnet werden (siehe *Gl. 1.26*).

1.3.6 Ableitungen einer NURBS Kurve

Bei den Ableitungen einer rationalen Funktion hat der gebildete Nenner einen großen Einfluss auf die Berechnung. Die dargestellten Formeln zeigen die Ableitungen von $\mathbf{C}(u)$ im Bezug auf $\mathbf{C}^w(u)$.

Zunächst wird die Quotientenfunktion

$$C(u) = \frac{A(u)}{w(u)} \tag{1.31}$$

der reellwertigen und l-fach stetig differenzierbaren Funktion $A(u)$ (Zählerfunktion) und $w(u)$ (Nennerfunktion), wobei wir $w(u) = 0$ im betrachteten Intervall $[u_0, u_1]$ voraussetzen.

Wegen

$$A(u) = w(u)\, C(u)$$

folgt daher mit der *"Leibnitz Regel"*:

$$\frac{d^k}{(du)^k}\, A(u) = \sum_{i=0}^{k} \binom{k}{i} \frac{d^{(k-i)}}{(du)^{(k-i)}}\, w(u) \frac{d^i}{(du)^i}\, C(u), \qquad k = 0, \ldots, l \tag{1.32}$$

In weiterer Folge wird die Schreibweise des Funktionsparameters u unterdrückt. Außerdem werden Ableitungen in folgender Form notiert:

$$A_{(k)} := \frac{d^k\, A}{(du)^k}$$

Gl. 1.32 liest sich in diese Notation so:

$$A_{(k)} = \sum_{i=0}^{k} \binom{k}{i} w_{(k-i)} C_{(i)}, \qquad k = 0, \ldots, l \tag{1.33}$$

Somit sind die Ableitungen

$$C_{(k)} = \frac{d^k A}{(du)^k}$$

der Quotienten-Funktion $C = C(u)$ die Lösungen des linearen Gleichungssystems:

$$\underbrace{\begin{bmatrix} w_{(0)} & 0 & 0 & \dots & 0 \\ w_{(1)} & w_{(0)} & 0 & \dots & 0 \\ w_{(2)} & 2\,w_{(1)} & w_{(0)} & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ w_{(l)} & l\,w_{(l-1)} & \binom{l}{2} w_{(l-2)} & \dots & w_{(0)} \end{bmatrix}}_{=:\mathbf{G}} \begin{bmatrix} C_{(0)} \\ C_{(1)} \\ C_{(2)} \\ \vdots \\ C_{(l)} \end{bmatrix} = \begin{bmatrix} A_{(0)} \\ A_{(1)} \\ A_{(2)} \\ \vdots \\ A_{(l)} \end{bmatrix} \tag{1.34}$$

Die daraus erhaltene $(l+1) \times (l+1)$ Koeffizientenmatrix $\mathbf{G}$ ist ein untere Dreiecksmatrix und ihre inverse Matrix $\mathbf{G}^{-1}$ kann daher einfach berechnet werden. Die Ableitungen $C_{(0)}, \dots, C_{(l)}$ der Quotienten Funktion $C(u)$ sind dann wie folgt gegeben:

$$\begin{bmatrix} C_{(0)} \\ \vdots \\ C_{(l)} \end{bmatrix} = \mathbf{G}^{-1} \begin{bmatrix} A_{(0)} \\ \vdots \\ A_{(l)} \end{bmatrix} \tag{1.35}$$

Ist etwa $l = 3$ dann haben wir

$$\mathbf{G}^{-1} = \begin{bmatrix} \frac{1}{w_{(0)}} & 0 & 0 & 0 \\ -\frac{w_{(1)}}{w_{(0)}^2} & \frac{1}{w_{(0)}} & 0 & 0 \\ \frac{2\,w_{(1)}^2 - w_{(0)}w_{(2)}}{w_{(0)}^3} & -\frac{2\,w_{(1)}}{w_{(0)}^2} & \frac{1}{w_{(0)}} & 0 \\ \frac{-6\,w_{(1)}^3 + 6\,w_{(0)}w_{(1)}w_{(2)} - w_{(0)}^2 w_{(3)}}{w_{(0)}^4} & \frac{6\,w_{(1)}^2 - 3\,w_{(0)}w_{(2)}}{w_{(0)}^3} & -\frac{3\,w_{(1)}}{w_{(0)}^2} & \frac{1}{w_{(0)}} \end{bmatrix}. \tag{1.36}$$

Bemerkung 1.3

Ersetzt man die reellwertige Funktion $A(u)$ durch eine vektorwertige Funktion

$$\mathbf{A}(u) = \begin{bmatrix} A_1(u) \\ \vdots \\ A_d(u) \end{bmatrix},$$

deren Werte in $\mathbb{R}^d$ liegen, dann ist der Quotient

$$\mathbf{C}(u) := \begin{bmatrix} C_1(u) \\ \vdots \\ C_d(u) \end{bmatrix} = \frac{\mathbf{A}(u)}{w(u)} := \begin{bmatrix} \frac{A_1(u)}{w(u)} \\ \vdots \\ \frac{A_d(u)}{w(u)} \end{bmatrix}$$

ebenfalls eine solche Funktion. Auch *Gl. 1.34* , *Gl. 1.35* und *Gl. 1.36* sind dann für jede Komponente von $\mathbf{A}(u)$ und $\mathbf{C}(u)$ anzuwenden. In diesem Fall ist die k-te Ableitung der Funktion A_i, C_i:

$$A_{i,(k)} = \frac{d^k A_i}{(du)^k}, \qquad C_{i,(k)} = \frac{d^k C_i}{(du)^k}$$

Die *Gl. 1.35* muss in diesem Fall durch folgende Gleichung ersetzt werden:

$$\begin{bmatrix} C_{1,(0)} & \dots & C_{d,(0)} \\ \vdots & & \vdots \\ C_{1,(l)} & \dots & C_{d,(l)} \end{bmatrix} = \mathbf{G^{-1}} \begin{bmatrix} A_{1,(0)} & \dots & A_{d,(0)} \\ \vdots & & \vdots \\ A_{1,(l)} & \dots & A_{d,(l)} \end{bmatrix} \tag{1.37}$$

Gl. 1.37 wird dann zu:

$$\begin{bmatrix} \mathbf{C}^\top_{(0)} \\ \vdots \\ \mathbf{C}^\top_{(l)} \end{bmatrix} = \mathbf{G}^{-1} \begin{bmatrix} \mathbf{A}^\top_{(0)} \\ \vdots \\ \mathbf{A}^\top_{(l)} \end{bmatrix}. \tag{1.38}$$

Damit können etwa auch die Ableitungen einer NURBS Kurve

$$\mathbf{C}(u) = \frac{\mathbf{A}(u)}{w(u)} := \frac{\sum_{i=0}^{n} N_{i,p}(u)\, w_i\, \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u)\, w_i} \tag{1.39}$$

berechnet werden.

1.3.7 NURBS Flächen

Die Herleitungen und weitere Ausführungen der nachfolgend angeführten Definitionen und Eigenschaften von NURBS Flächen findet man z. B. in [4, Kap.4].

Definition 1.8 *NURBS Fläche*

Gegeben seien die natürlichen Zahlen p und q, die Knotenvektoren $\mathbf{U} = \{u_0, \ldots, u_r\}$ und $\mathbf{V} = \{v_0, \ldots, v_s\}$, ein Kontrollnetz $\mathbf{P}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j}) \in \mathbb{R}^3$ mit den Gewichten $w_{i,j} \in \mathbb{R}^+$ wobei $i = 0, \ldots, n$ und $j = 0, \ldots, m$. Dann ist

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)\; N_{j,q}(v)\; w_{i,j}\; \mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)\; N_{j,q}(v)\; w_{i,j}}, \qquad (u,v) = [u_{p-1}, u_{n+1}] \times [v_{q-1}, v_{m+1}] \quad (1.40)$$

die Parameterdarstellung einer *NURBS Fläche* vom Grad p, q zum Kontrollnetz $P_{i,j}$ mit den Gewichten $w_{i,j}$ und den Knotenvektoren $\mathbf{U}$ und $\mathbf{V}$. Die Herleitung der nachfolgend angeführten Eigenschaften von NURBS Flächen findet man z. B. in [4, S. 128ff].

Eigenschaften 1.5 *NURBS Fläche*

Die NURBS Flächen besitzen die gleichen Eigenschaften wie die gewöhnlichen B-Spline Flächen (siehe Eigenschaften 1.3 *B-Spline Flächen*, S. 9). Aufgrund der Gewichtung der Kontrollpunkte können die Eigenschaften wie folgt erweitert werden:

- Eine Änderung des Kontrollpunktes $\mathbf{P}_{i,j}$ und bzw. oder des Gewichtes $w_{i,j}$ hat nur im Intervall $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$ einen Einfluss auf die NURBS Fläche. Durch die Gewichte gibt es bei den NURBS Flächen eine weitere Möglichkeit die lokale Form der Fläche zu beeinflussen. Je größer das Gewicht $w_{i,j}$ eines Kontrollpunktes $\mathbf{P}_{i,j}$ gewählt wird, desto mehr nähert sich die Fläche an den Kontrollpunkt $\mathbf{P}_{i,j}$ an.
- Wenn alle Gewichte den gleichen Wert besitzen, ist die NURBS Fläche ident mit der gewöhnlichen B-Spline Fläche mit dem gleichen Kontrollnetz.

Die in *Kap. 1.3.1* , S. 11 eingeführten homogenen Koordinaten bieten eine effiziente Methode um NURBS Flächen darzustellen. Setzt man $\mathbf{P}^w_{i,j} := (w_{i,j}x_{i,j}, w_{i,j}y_{i,j}, w_{i,j}z_{i,j}, w_{i,j})$

dann ist durch

$$\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)\; N_{j,q}(v)\; \mathbf{P}^w_{i,j} \tag{1.41}$$

eine gewöhnliche B-Spline Fläche des $\mathbb{R}^4$ definiert. Die Zentralprojektion H aus dem Abschnitt 1.3.1 bildet diese gewöhnliche B-Spline Fläche $\mathbf{S}^w(u,v)$ auf die NURBS Fläche (*Gl. 1.40*) ab. Alternativ kann man im Sinne von Definition 1.6, S. 13, die *Gl. 1.41* auch als Beschreibung von $\mathbf{S}(u,v)$ in homogenen Koordinaten ansehen [4, S.132].

1.3.8 Rationale Drehflächen als NURBS Flächen

Die folgende Beschreibung von rationalen Drehflächen als NURBS Flächen wird an [4, S. 340 ff] und [5, 6] angelehnt.

Um eine NURBS Drehfläche $\mathbf{S}(u,v)$ zu generieren benötigen wir folgenden Input:

- die Drehachse a, die etwa durch einen ihrer Punkte $\mathbf{a}_1$ und einen normierten Richtungsvektor $\mathbf{d}_1$ festgelegt sei und
- eine erzeugende NURBS Kurve

$$\mathbf{C}(u) \;=\; \frac{\mathbf{A}(u)}{w(u)} \;:=\; \frac{\sum_{i=0}^{n} N_{i,p}(u)\; \tilde{w}_i\, \tilde{\mathbf{P}}_i}{\sum_{i=0}^{n} N_{i,p}(u)\, \tilde{w}_i} \tag{1.42}$$

($\tilde{\mathbf{P}}_0, \ldots, \tilde{\mathbf{P}}_n$ ist das Kontrollpolygon von $\mathbf{C}(u)$ und $\tilde{w}_0, \ldots, \tilde{w}_n$ ist die Folge der Gewichte; der Knotenvektor sei $\mathbf{U} = \{u_0, \ldots, u_r\}, r = m + p + 1$).

Bemerkung 1.4

In der Praxis wird als erzeugende Kurve $\mathbf{C}(u)$ meistens eine Meridiankurve gewählt – das ist eine in einer Ebene durch a liegende Kurve.

Das Kontrollnetz $\tilde{\mathbf{P}}_{i,j}$ der zu erstellenden NURBS Drehfläche $\mathbf{S}(u,v)$ entsteht nun aus dem Kontrollpolygon $\tilde{\mathbf{P}}_i$ der erzeugenden NURBS Kurve $\mathbf{C}$(u), indem man aus jedem Kontrollpunkt $\tilde{\mathbf{P}}_i$ das Kontrollpolygon $\tilde{\mathbf{P}}_i = \tilde{\mathbf{P}}_{i0}, \tilde{\mathbf{P}}_{i1}, \ldots, \tilde{\mathbf{P}}_{i8} = \tilde{\mathbf{P}}_{i0} = \tilde{\mathbf{P}}_i$ jenes Vollkreises erzeugt, der $\tilde{\mathbf{P}}_i$ enthält und dessen Achse a ist. (Die Punkte $\tilde{\mathbf{P}}_{i0}, \ldots, \tilde{\mathbf{P}}_{i8} = \tilde{\mathbf{P}}_{i0}$ sind die Ecken und Seitenhalbierungspunkte des diesem Kreis umschriebenen Quadrats (vgl. mit Abschnitt 1.3.3 und siehe *Abb. 1.13* und *Abb. 1.14*).

Um die Punkte $\tilde{\mathbf{P}}_{i0}, \ldots, \tilde{\mathbf{P}}_{i8} = \tilde{\mathbf{P}}_{i0}$ zu berechnen, muss zuerst der Fußpunkt $\mathbf{Q}_i$ des Lotes aus $\tilde{\mathbf{P}}_i$ auf a bestimmt werden (siehe *Abb. 1.12*):

$$\mathbf{Q}_i = \mathbf{a}_1 + \langle \mathbf{d}_1, \tilde{\mathbf{P}}_i - \mathbf{a}_1 \rangle \, \mathbf{d}_1 \tag{1.43}$$

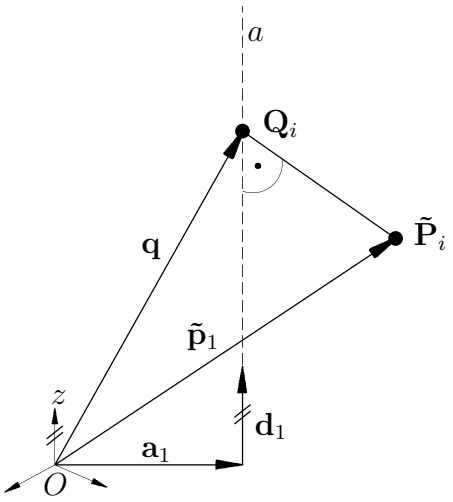

Abbildung 1.12: Lotfußpunkt $\mathbf{Q}_i$ zu einem Punkt $\tilde{\mathbf{P}}_i$.

Anschließend ermittelt man das normal zu a liegende orthogonale Vektorpaar:

$$\mathbf{e}_{x,i} := \tilde{\mathbf{P}}_i - \mathbf{Q}_i = \mathbf{Q}_i\tilde{\mathbf{P}}_i \qquad \mathbf{e}_{y,i} := \mathbf{d}_1 \times \mathbf{e}_{x,i} \tag{1.44}$$

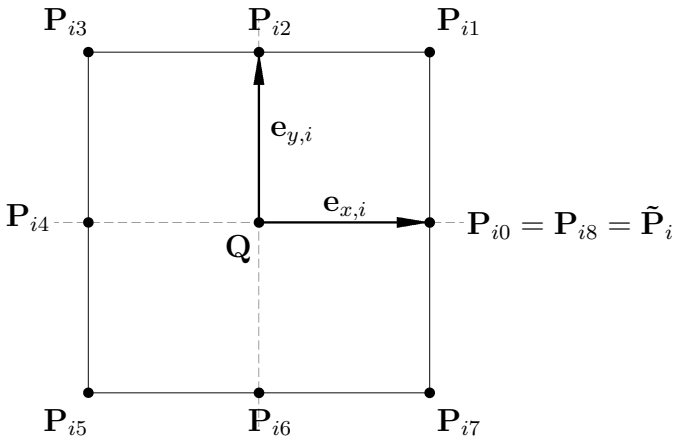

Abbildung 1.13: Vektorpaar $\mathbf{e}_{x,i}$ und $\mathbf{e}_{y,i}$ ausgehend vom Lotfußpunkt $\mathbf{Q}$.

Damit lassen sich die in Rede stehenden Punkte $\tilde{\mathbf{P}}_{i0}, \ldots, \tilde{\mathbf{P}}_{i8} = \tilde{\mathbf{P}}_{i0}$ etwa wie folgt berechnen (siehe *Abb. 1.13*):

$\mathbf{P}_{i0} = \tilde{\mathbf{P}}_i,\ \mathbf{P}_{i1} = \mathbf{P}_{i0} + \mathbf{e}_{yi},\ \mathbf{P}_{i2} = \mathbf{Q}_i + \mathbf{e}_{yi},\ \mathbf{P}_{i3} = \mathbf{P}_{i2} - \mathbf{e}_{xi},\ \mathbf{P}_{i4} = \mathbf{Q}_i - \mathbf{e}_{xi},$
$\mathbf{P}_{i5} = \mathbf{P}_{i4} - \mathbf{e}_{yi},\ \mathbf{P}_{i6} = \mathbf{Q}_i - \mathbf{e}_{yi},\ \mathbf{P}_{i7} = \mathbf{P}_{i0} - \mathbf{e}_{yi},\ \mathbf{P}_{i8} = \mathbf{P}_{i0} = \tilde{\mathbf{P}}_i.$

Abb. 1.14 zeigt ein Viertel eines so konstruierten $((n+1) \times 9)$-Kontrollnetzes $\mathbf{P}_{i,j}$, wobei $n = 3$ gewählt wurde. Die Gewichte der NURBS Drehfläche sind wie folgt zu berechnen:

$$w_{i,j} = \tilde{w}_i \text{ für } j = 0, 2, 4, 6, 8 \text{ und } w_{i,j} = \frac{\tilde{w}_i}{\sqrt{2}} \text{ für } j = 1, 3, 5, 7 \tag{1.45}$$

Als Knotenvektor in u-Richtung wird jener der erzeugenden NURBS Kurve $\mathbf{C}(u)$ also $\mathbf{U}$ genommen, während als Knotenvektor in v-Richtung – diese Richtung repräsentiert die Parallelkreise der Drehfläche – der Vektor $\mathbf{V} = \{0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1\}$ gewählt werden muss (vgl. Abschnitt 1.3.3).

Die resultierende Parameterdarstellung der erzeugenden NURBS Drehfläche lautet somit:

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{8} N_{i,p}(u)\ N_{j,2}(v)\ w_{i,j}\ \mathbf{P}_{i,j}}{\sum_{i=0}^{n} \sum_{j=0}^{8} N_{i,p}(u)\ N_{j,2}(v)\ w_{i,j}}, \qquad (u,v) \in [u_p, u_{n+1}] \times [0,1] \tag{1.46}$$

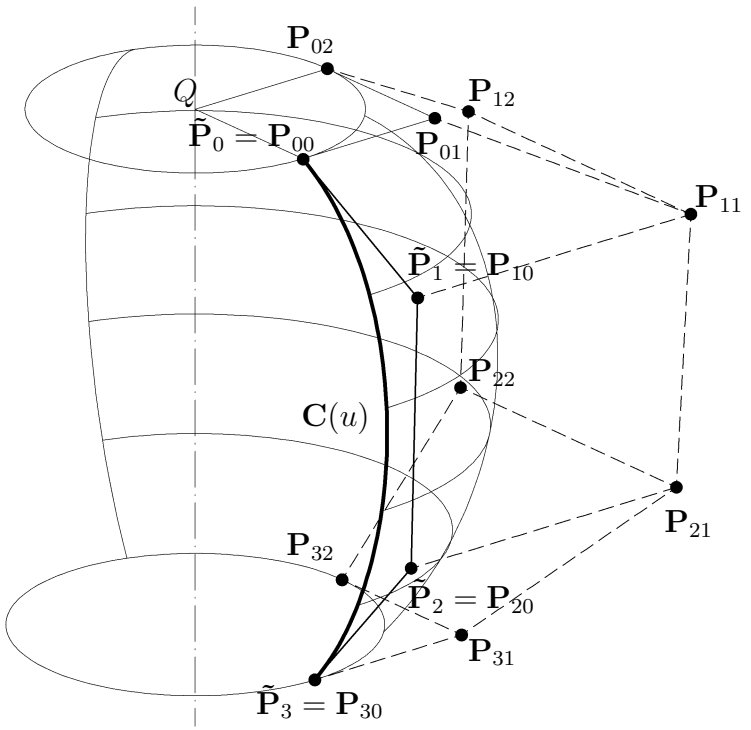

Abbildung 1.14: Rationale Drehfläche als NURBS Fläche.

Anmerkungen:

- Die obige Methode kann leicht adaptiert werden, um etwa nur ein Viertel der gesamten Drehfläche zu erzeugen (vgl. mit Abschnitt 1.3.3).
- in der MATLAB-Implementierung (Abschnitt 2.7, S. 60) wurde stets, die z-Achse als Drehachse a gewählt ($\mathbf{a}_1 = (0,0,0), \mathbf{d}_1 = (0,0,1)$). Dort wird eine etwaige andere Lage der Drehfläche durch die anschließende Anwendung einer Kongruenztransformation (Transformationsmatrix $\mathbf{T}$) erreicht.

1.3.9 Ableitungen einer NURBS Fläche

Es sei $S(u,v)$ eine bivariate reellwertige und l-fach stetig differenzierbare Funktion. Dann werden die Ableitungen im folgenden so bezeichnet:

$$S_{(k_1),(k_2)} \quad := \quad \frac{\partial^k S(u,v)}{(\partial u)^{k_1}(\partial v)^{k_2}}, \qquad k_1 + k_2 = k, \qquad k = 0, \ldots, l$$

Bemerkung 1.5

1. Es gibt genau $k+1$ Ableitungen der Ordnung k einer bivariaten Funktion $S(u,v)$:
   - $k = 1$: $S_{(1),(0)}$, $S_{(0),(1)}$
   - $k = 2$: $S_{(2),(0)}$, $S_{(1),(1)}$, $S_{(0),(2)}$
   - $k = 3$: $S_{(3),(0)}$, $S_{(2),(1)}$, $S_{(1),(2)}$, $S_{(0),(3)}$
     usw.
2. Üblicherweise werden die partiellen Ableitungen erster Ordnung einer bivariaten Funktion $S(u,v)$ in der sogenannten Jacobi-Matrix $\mathbf{J}$ zusammengefasst:
   $$\mathbf{J}(u,v) = \begin{bmatrix} \partial S/\partial u & \partial S/\partial v \end{bmatrix} = \begin{bmatrix} S_{(1),(0)}, S_{(0),(1)} \end{bmatrix} \tag{1.47}$$
   $\mathbf{J}$ ist eine (1×2)-Matrix.
3. Die partiellen Ableitungen zweiter Ordnung von $S(u,v)$ können in der sogenannten Hesse-Matrix $\mathbf{H}$ zusammengefasst werden:
   $$\mathbf{H}(u,v) = \begin{bmatrix} \partial^2 S/\partial u^2 & \partial^2 S/\partial u \partial v \\ \partial^2 S/\partial u \partial v & \partial^2 S/\partial v^2 \end{bmatrix} = \begin{bmatrix} S_{(2),(0)}, S_{(1),(1)} \\ S_{(1),(1)}, S_{(0),(2)} \end{bmatrix} \tag{1.48}$$
   $\mathbf{H}$ ist eine symmetrische (2×2)-Matrix.

Es wird nun eine bivariate reellwertige Quotienten-Funktion $S(u,v)$ mit der l-fach stetig differenzierbaren Zähler-Funktionen $A(u,v)$, und der ebenfalls l-fach stetig differenzierbaren Nenner-Funktion $w(u,v)$ betrachtet. Es ist also

$$S(u,v) \;=\; \frac{A(u,v)}{w(u,v)}. \tag{1.49}$$

Hierbei setzen wir im betrachteten Parametergebiet $w(u,v) \neq 0$ voraus.
Aus *Gl. 1.49* folgt

$$A(u,v) \;=\; w(u,v)\; S(u,v). \tag{1.50}$$

Mit der *"Leibnitz Regel"* erhalten wir daraus

$$\begin{aligned} A_{(i),(k-i)} \;&=\; \sum_{\alpha=0}^{i} \sum_{\beta=0}^{k-i} \binom{i}{\alpha} \binom{k-i}{\beta}\; w_{(i-\alpha),(k-i-\beta)}\; S_{(\alpha),(\beta)} \\ &=\; \sum_{\alpha=0}^{i} \sum_{\beta=0}^{k-i} c_{\alpha,\beta,i,k}\; S_{(\alpha),(\beta)}, \quad i = 0,\ldots,l, \end{aligned} \tag{1.51}$$

wobei

$$c_{\alpha,\beta,i,k} \;:=\; \binom{i}{\alpha} \binom{k-i}{\beta} w_{(i-\alpha),(k-i-\beta)}. \tag{1.52}$$

Aus dieser Formel kann leicht abgeleitet werden, dass die

$$1 + 2 + \cdots + l + 1 = \frac{(l+2)(l+1)}{2}$$

Ableitungen

$$S_{(0),(0)} = S;\;\; S_{(1),(0)}, S_{(0),(1)};\;\; S_{(2),(0)}, S_{(1),(1)}, S_{(0),(2)};\;\; \ldots;\;\; S_{(l),(0)}, S_{(l-1),(1)}, \ldots, S_{(0),(l)}$$

der Ordnung $\leq l$ der Quotienten-Funktion $S(u,v)$ einem linearen Gleichungssystem

$$\mathbf{G} \begin{bmatrix} S_{(0),(0)} \\ S_{(1),(0)} \\ S_{(0),(1)} \\ \vdots \\ S_{(l),(0)} \\ \vdots \\ S_{(0),(l)} \end{bmatrix} \;=\; \begin{bmatrix} A_{(0),(0)} \\ A_{(1),(0)} \\ A_{(0),(1)} \\ \vdots \\ A_{(l),(0)} \\ \vdots \\ A_{(0),(l)} \end{bmatrix} \tag{1.53}$$

mit einer unteren $\frac{(l+2)(l+1)}{2} \times \frac{(l+2)(l+1)}{2}$ Dreiecksmatrix $\mathbf{G}$ als Koeffizientenmatrix genügen müssen. Die Ableitungen der Quotienten-Funktion $S(u,v) = \frac{A(u,v)}{w(u,v)}$ erhält man daher so:

$$\begin{bmatrix} S_{(0),(0)} \\ S_{(1),(0)} \\ S_{(0),(1)} \\ \vdots \\ S_{(l),(0)} \\ \vdots \\ S_{(0),(l)} \end{bmatrix} = \mathbf{G}^{-1} \begin{bmatrix} A_{(0),(0)} \\ A_{(1),(0)} \\ A_{(0),(1)} \\ \vdots \\ A_{(l),(0)} \\ \vdots \\ A_{(0),(l)} \end{bmatrix} \tag{1.54}$$

Eine genauere Betrachtung von *Gl. 1.51* zeigt, dass die Einträge der Matrix $\mathbf{G}$, welche nicht null sind, wie folgt berechnet werden können:

$$\begin{aligned} \mathbf{G}_{\frac{k(k+3)}{2}-i+1,\frac{(\alpha+\beta)(\alpha+\beta+1)}{2}+\beta+1} &= c_{\alpha,\beta,i,k} = \binom{i}{\alpha}\binom{k-i}{\beta} w_{(i-\alpha),(k-i-\beta)} \\ &\text{für} \quad k = 0,\ldots,l;\ \ i = 0,\ldots,k; \\ &\text{und} \quad \alpha = 0,\ldots,i;\ \ \beta = 0,\ldots,k-i; \end{aligned} \tag{1.55}$$

Für unserer Zwecke sind Ableitungen bis zur Ordnung $l = 2$ ausreichend. Dann ist $\frac{(l+2)\,(l+1)}{2} = 6$ und

$$\mathbf{G} = \begin{bmatrix} w_{(0),(0)} & 0 & 0 & 0 & 0 & 0 \\ w_{(1),(0)} & w_{(0),(0)} & 0 & 0 & 0 & 0 \\ w_{(0),(1)} & 0 & w_{(0),(0)} & 0 & 0 & 0 \\ w_{(2),(0)} & 2\,w_{(1),(0)} & 0 & w_{(0),(0)} & 0 & 0 \\ w_{(1),(1)} & w_{(0),(1)} & w_{(1),(0)} & 0 & w_{(0),(0)} & 0 \\ w_{(0),(2)} & 0 & 2\,w_{(0),(1)} & 0 & 0 & w_{(0),(0)} \end{bmatrix}. \tag{1.56}$$

In diesem Fall berechnet sich die inverse Matrix von $\mathbf{G}$ wie folgt:

$$\mathbf{G}^{-1} = \begin{bmatrix} \frac{1}{w_{(0),(0)}} & 0 & 0 & 0 & 0 & 0 \\ -\frac{w_{(1),(0)}}{w_{(0),(0)}^2} & \frac{1}{w_{(0),(0)}} & 0 & 0 & 0 & 0 \\ -\frac{w_{(0),(1)}}{w_{(0),(0)}^2} & 0 & \frac{1}{w_{(0),(0)}} & 0 & 0 & 0 \\ \frac{2\,w_{(1),(0)}^2 - w_{(0),(0)}w_{(2),(0)}}{w_{(0),(0)}^3} & -2\,\frac{w_{(1),(0)}}{w_{(0),(0)}^2} & 0 & \frac{1}{w_{(0),(0)}} & 0 & 0 \\ \frac{2\,w_{(0),(1)}w_{(1),(0)} - w_{(0),(0)}w_{(1),(1)}}{w_{(0),(0)}^3} & -\frac{w_{(0),(1)}}{w_{(0),(0)}^2} & -\frac{w_{(1),(0)}}{w_{(0),(0)}^2} & 0 & \frac{1}{w_{(0),(0)}} & 0 \\ \frac{2\,w_{(0),(1)}^2 - w_{(0),(0)}w_{(0),(2)}}{w_{(0),(0)}^3} & 0 & -2\,\frac{w_{(0),(1)}}{w_{(0),(0)}^2} & 0 & 0 & \frac{1}{w_{(0),(0)}} \end{bmatrix} \tag{1.57}$$

Bemerkung 1.6

Ersetzt man die reellwertige Funktion $A(u, v)$ durch eine vektorwertige Funktion

$$\mathbf{A}(u, v) = \begin{bmatrix} A_1(u, v) \\ \vdots \\ A_d(u, v) \end{bmatrix}$$

dessen Werten in $\mathbb{R}^d$ liegen, dann ist der Quotient

$$\mathbf{S}(u, v) = \begin{bmatrix} S_1(u, v) \\ \vdots \\ S_d(u, v) \end{bmatrix} = \frac{\mathbf{A}(u, v)}{w(u, v)} = \begin{bmatrix} \frac{A_1(u,v)}{w(u,v)} \\ \vdots \\ \frac{A_d(u,v)}{w(u,v)} \end{bmatrix}$$

auch eine solche Funktion und die Gleichungen *Gl. 1.51* , *Gl. 1.53* und *Gl. 1.54* können auf jede Komponente von $\mathbf{A}$ and $\mathbf{S}$ angewendet werden. Nach Einführung der Schreibweise

$$A_{i,(k_1),(k-k_1)} := \frac{\partial^k A_i}{(\partial u)^{k_1}(\partial v)^{k-k_1}}, \qquad S_{i,(k_1),(k-k_1)} := \frac{\partial^k S_i}{(\partial u)^{k_1}(\partial v)^{k-k_1}}, \qquad i = 1, \ldots, d$$

wird dann (*Gl. 1.54*) zu

$$\begin{bmatrix} S_{1,(0),(0)} & \dots & S_{d,(0),(0)} \\ S_{1,(1),(0)} & \dots & S_{d,(1),(0)} \\ S_{1,(0),(1)} & \dots & S_{d,(0),(1)} \\ \vdots & & \vdots \\ S_{1,(l),(0)} & \dots & S_{d,(k),(0)} \\ \vdots & & \vdots \\ S_{1,(0),(l)} & \dots & S_{d,(0),(l)} \end{bmatrix} = \mathbf{G}^{-1} \begin{bmatrix} A_{1,(0),(0)} & \dots & A_{d,(0),(0)} \\ A_{1,(1),(0)} & \dots & A_{d,(1),(0)} \\ A_{1,(0),(1)} & \dots & A_{d,(0),(1)} \\ \vdots & & \vdots \\ A_{1,(l),(0)} & \dots & A_{d,(l),(0)} \\ \vdots & & \vdots \\ A_{1,(0),(l)} & \dots & A_{d,(0),(l)} \end{bmatrix}, \tag{1.58}$$

bzw. zu:

$$\begin{bmatrix} \mathbf{S}^{\top}_{(0),(0)} \\ \mathbf{S}^{\top}_{(1),(0)} \\ \mathbf{S}^{\top}_{(0),(1)} \\ \vdots \\ \mathbf{S}^{\top}_{(l),(0)} \\ \vdots \\ \mathbf{S}^{\top}_{(0),(l)} \end{bmatrix} = \mathbf{G}^{-1} \begin{bmatrix} \mathbf{A}_{(0),(0)} \\ \mathbf{A}^{\top}_{(1),(0)} \\ \mathbf{A}^{\top}_{(0),(1)} \\ \vdots \\ \mathbf{A}^{\top}_{(l),(0)} \\ \vdots \\ \mathbf{A}^{\top}_{(0),(l)} \end{bmatrix}. \tag{1.59}$$

Insbesondere können damit die partiellen Ableitungen der Parameterdarstellung (*Gl. 1.40*) einer NURBS Fläche berechnet werden. Hierbei ist $d = 3$ und

$$\mathbf{S}(u,v) = \frac{\mathbf{A}(u,v)}{w(u,v)} = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j} \mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j}} \tag{1.60}$$

zu setzen.

Anmerkung:

Im vektorwertigen Fall ist die (1×2)-Jacobi-Matrix (*Gl. 1.47*, S. 27) durch die $(d \times 2)$-Matrix

$$\mathbf{J}(u,v) = \begin{bmatrix} S_{1,(1),(0)} & S_{1,(0),(1)} \\ \vdots & \vdots \\ S_{d,(1),(0)} & S_{d,(0),(1)} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial S_1}{\partial u} & \dfrac{\partial S_1}{\partial v} \\ \vdots & \vdots \\ \dfrac{\partial S_d}{\partial u} & \dfrac{\partial S_d}{\partial v} \end{bmatrix} \tag{1.61}$$

zu ersetzen. Ist speziell $d = 3$ – also etwa im Fall von NURBS Flächen – erhalten wir daher eine (3×2)-Matrix.

Aus der (2×2)-Hesse-Matrix $\mathbf{H}$ des reellwertigen Falls werden beim Übergang zum vektorwertigen Fall d solche (2×2)-Matrizen:

$$\mathbf{H}_i(u,v) = \begin{bmatrix} S_{i,(2),(0)} & S_{i,(1),(1)} \\ S_{i,(1),(1)} & S_{i,(0),(2)} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial^2 S_i}{\partial u^2} & \dfrac{\partial^2 S_i}{\partial v \partial v} \\ \dfrac{\partial^2 S_i}{\partial u \partial v} & \dfrac{\partial^2 S_i}{\partial v^2} \end{bmatrix}, \qquad i = 1, \ldots, d \tag{1.62}$$

Insbesondere erhalten wir im Fall $d = 3$ – also etwa auch im Fall von NURBS Flächen – drei Hesse-Matrizen, die etwa in einer $(2 \times 2 \times 3)$-Hypermatrix zusammengefasst werden können (vgl. Abschnitt 2.6.5, S. 58).

Kapitel 2

Dokumentation der erstellten Software-Bibliothek

Dieses Kapitel stellt den Hauptteil der vorliegenden Arbeit dar. Zuerst wird das grundlegende Klassenmodell und die Vorgehensweise bei der Erstellung des Codes erläutert. Weiters werden die Eigenschaften, der Konstruktor und die Methoden der jeweiligen Klassen bzw. Super-Klassen sowie deren Unter-Klassen beschrieben. Für die Umsetzung der gestellten Aufgabe wurde das Softwarepaket MATLAB R2014b bzw. R2015a verwendet. Der erstellte MATLAB Code selbst ist in der schriftlichen Dokumentation der Arbeit nicht enthalten, er wird jedoch im jeweiligen Kapitel detailliert erläutert.

2.1 Klassenmodell - Übersicht

Die erstellte Software-Bibliothek beinhaltet folgende Klassen:
Die Klasse `BasisFunMat` dient der numerischen Auswertung der Basisfunktionen und deren Ableitungen. Die Klasse `NURBS` wird zur Berechnung des Tensorproduktes aus den Matrizen der Basisfunktionen und dem Array der Kontrollstruktur verwendet. Weiters wurde die Super-Klasse `Surface` zur Beschreibung von Flächen erstellt. Von der Super-Klasse `Surface` wurde die Super-Klasse `Revolved Surface` für Drehflächen abgeleitet. Von dieser wurden weitere Spezialisierungen für technisch bedeutsame Drehflächen wie Drehzylinder(`Cylinder`), Drehkegel (`Cone`) und Torus (`Torus`) implementiert. Zwei weitere Klassen `HyperMatrix` und `csys` wurden vom Auftraggeber zur Verfügung gestellt.

Übersicht der Gliederung:

1. Klasse `BasisFunMat`
2. Klasse `NURBS`
3. Klasse `HyperMatrix`
4. Klasse `csys`
5. Super-Klasse `Surface`

   ⟶ Super-Klasse `Revolved Surface`

   - Klasse `Cone`
   - Klasse `Cylinder`
   - Klasse `Torus`

Der Code der Klasse `BasFunMat` und der Klasse `NURBS` wurde in Anlehnung an die Vorgehensweise in [4, Kapitel 2] erstellt. Die implementierten Algorithmen dieser Klassen basieren auf dem *Inverted Triangular Scheme* (ITS) (siehe [3, S.68]). Da die Auswertung von Tensorprodukten ein wesentlicher Bestandteil dieser Arbeit ist, bietet das ITS eine effiziente Möglichkeit für die numerischen Auswertung. Das ITS bietet einen numerisch - mathematischen Zugang. Im Gegensatz dazu wird beim Algorithmus nach Cox de Boor ein Punkt einer NURBS Kurve oder NURBS Fläche durch fortgesetztes Streckenteilen ermittelt. Bezüglich der Effizienz der numerischen Auswertung mittels verschiedener Algorithmen zeigt [3, S. 64-69] eine deutliche Überlegenheit des ITS gegenüber der rekursiven Berechnung der Basisfunktionen mittels des Cox de Boor-Algorithmus.

Für die Darstellung von Flächen benötigt der Algorithmus nach [4, S.134, A 4.3] für jedes Paar von Parameterwerten Funktionsaufrufe zur Berechnung der Basisfunktionen. Dadurch werden Basisfunktionen für gleiche Parameterwerte mehrfach ausgewertet. Um diese Mehrfachberechnung zu vermeiden, wird die Berechnung anhand der Tensorprodukte durchgeführt. Für die numerische Auswertung des Tensorproduktes wird eine eigene Klasse `HyperMatrix` verwendet, welche durch den Auftraggeber zur Verfügung gestellt wurde. Eine weitere Klasse, welche vom Auftraggeber zur Verfügung gestellt wurde, ist die Klasse `csys`, mit welcher Koordinatentransformationen durchgeführt werden können. Die Darstellung der Flächen benötigt für jede Richtung (u_1 und u_2) einen Satz von Ba-

sisfunktionen. Dies kann durch das objektorientierte Konzept einfach umgesetzt werden und erfordert somit keine mehrfache Durchführung der Berechnung.

Eine numerisch effiziente Umsetzung von Tensorprodukten würde auch die Berücksichtigung der schwachen Besetzung der Matrizen der Basisfunktionen bieten. Da MATLAB keine schwach besetzten mehrdimensionalen Arrays zur Verfügung stellt, wird die Klasse `HyperMatrix` für Tensorprodukte mit vollständiger Besetzung der einzelnen Faktoren verwendet. Die objekt-orientierte Umsetzung bietet die Möglichkeit, dass diese Klasse zu einem späteren Zeitpunkt durch eine Klasse ersetzt werden kann, in der auch schwach besetzte Hypermatrizen realisierbar sind.

Für Drehflächen werden Interfaces erstellt, die dem Anwender die Berechnung der Kontrollpunkte und der Gewichte aus den Geometriedaten technisch wichtiger Flächen (Drehzylinder, Drehkegel und Torus) abnehmen.

Anmerkung: Es ist zu beachten, dass in MATLAB die Zählweise bei eins beginnt und daher in der erstellten Bibliothek der Zähler der auftretenden Arrays jeweils um eins erhöht wurde. Zum Beispiel wird im folgenden der Knotenvektor jeweils mit $\mathbf{U} = \{U_0, \ldots, U_m\}$ bezeichnet, während er im MATLAB-Code mit $\mathbf{U} = \{U_1, \ldots, U_{m+1}\}$ implementiert ist.

2.2 Klasse `BasisFunMat`

Die Klasse für die Basisfunktionen (`BasisFunMat`) dient der numerischen Bestimmung der B-Spline Basisfunktionen $N_{i,p}(u)$ und ihrer Ableitungen $N_{i,p}^{(k)}(u)$. Die Algorithmen für die numerische Auswertung werden an die Vorgehensweise in [4, S. 67ff] angelehnt.

Folgende Methoden, welche in *Kap. 2.2.3 - Kap. 2.2.9* detailliert beschrieben werden, wurden in der Klasse `BasisFunMat` erstellt:

- Methode `findSpan`
- Methode `calc`
- Methode `calc_deriv`
- Methode `calc_derivMult`
- Methode `calc_derivNum`

- Methode `plot`
- Methode `plot_ders`

2.2.1 Eigenschaften der Klasse `BasisFunMat`

- Knotenvektor: $\mathbf{U} = \{U_0, \ldots, U_m\}$
- Grad der Basisfunktionen: p
- Anzahl der Knoten minus eins: m
- Anzahl der Kontrollpunkte minus eins: n

Zwischen p, m, und n besteht der Zusammenhang $m = n + p + 1$.

2.2.2 Konstruktor der Klasse `BasisFunMat`

Input:

- Knotenvektor: $\mathbf{U} = \{U_0, \ldots, U_m\}$
- Anzahl der Kontrollpunkte minus eins: n

Output:

- Objekt der Klasse `BasisFunMat`

Der Konstruktor ist eine spezielle Methode, die eine Instanz der Klasse erstellt. Die Eingabeargumente werden den angeführten Eigenschaften zugewiesen. Der Konstruktor gibt das initialisierte Objekt zurück.

Demzufolge werden in der Klasse `BasisFunMat` im Konstruktor die oben beschriebenen Eigenschaften den Objekten zugewiesen, unter der Beachtung, dass m die Anzahl der Knoten minus eins ist und der Grad p durch $p = m - n - 1$ definiert ist.

2.2.3 Methode `findSpan`

Input: Parameterwert: u

Ouput: Index: i

Die Methode `findSpan` wurde in Anlehnung zu [4, S. 68, A2.1] erstellt und dient der Bestimmung jenes Index i, für den $U_i \leq u < U_{i+1}$ gilt. Um sicherzustellen, dass die Funktion `findSpan` jedenfalls einen Rückgabewert liefert, wurde der Algorithmus A2.1 für die beiden Fälle $u < U_p$ und $u > U_{m-p}$ wie folgt erweitert:

- $u < U_p \;\Rightarrow\; i = p$
- $u \geq U_n \;\Rightarrow\; i = m - p - 1$

2.2.4 Methode `calc`

Input: Parameterwert u oder Array von Parameterwerten $\mathbf{u} = \{u_0, \ldots, u_c\}$

Output: Matrix $\mathbf{N}$ der Werte der Basisfunktionen für diesen (diese) Parameterwert(e)

Die Methode `calc`, welche in Anlehnung zu [4, S. 70, A2.2] erstellt wurde, berechnet die nicht verschwindenden Funktionswerte $N_{i-p,p}(u), \ldots, N_{i,p}(u)$ der Basisfunktionen (siehe Eigenschaften 1.1, S. 2). Die Methode benötigt als Eingabewert entweder einen Parameterwert u oder ein Array von Parameterwerten $\mathbf{u} = \{u_0, \ldots, u_c\}$. Die Anzahl der Parameterwerte von $\mathbf{u}$ wird im MATLAB-Code mit `num_u` bezeichnet: $num\_u = c + 1$
Ausgegeben werden die Werte der Basisfunktionen $N_{j,p}$ in Form einer $(n+1) \times (c+1)$-Matrix

$$\mathbf{N} := \begin{bmatrix} N_{0,p}(u_0) & \ldots & N_{0,p}(u_c) \\ \vdots & & \vdots \\ N_{n,p}(u_0) & \ldots & N_{n,p}(u_c) \end{bmatrix}. \tag{2.1}$$

Aufgrund der Eigenschaften der B-Spline Basisfunktion (vgl. Eigenschaften 1.1. S. 2) besitzt die j-te Spalte von $\mathbf{N}$ höchstens $p + 1$ von Null verschiedenen Einträgen.

2.2.5 Methode `calc_deriv`

Input:

- Parameterwert: u
- maximaler Grad der Ableitung: $l \leq p$

Output:

- Matrix der abgeleiteten Basisfunktionen: **ders**

Die Methode `calc_deriv` welche in Anlehnung zu [4, S. 72, A2.3] erstellt wurde, berechnet die k-fachen Ableitungen der nicht verschwindenden B-Spline Basisfunktionen $N^{(k)}_{i-p,p}(u), \ldots, N^{(k)}_{i,p}(u)$, wobei $k = \{0, \ldots, l\}$ gilt.
Die Methode benötigt als Eingabewert einen Parameterwert u und den maximalen Grad l der zu berechnenden Ableitung, wobei $l \leq p$ gilt. Ausgegeben werden die k-fachen Ableitungen der Basisfunktionen $N^k_{j,p}$ in Form einer $(n+1) \times (l+1)$-Matrix

$$\mathbf{ders} = \begin{bmatrix} N^{(0)}_{0,p}(u_0) & \ldots & N^{(l)}_{0,p}(u_c) \\ \vdots & & \vdots \\ N^{(0)}_{n,p}(u_0) & \ldots & N^{(l)}_{n,p}(u_c) \end{bmatrix}. \tag{2.2}$$

Die $(k+1)$-te Spalte der Matrix beinhaltet die Ableitungen der Ordnung k an der Stelle u.

2.2.6 Methode `calc_derivMult`

Input:

- Array von Parameterwerten: $\mathbf{u} = \{u_0, \ldots, u_c\}$
- maximaler Grad der Ableitung: l

Output:

- Multidimensionales Array der abgeleiteten Basisfunktionen: **ders**

Die Methode `calc_derivMult` dient der Berechnung der k-fachen Ableitung der B-Spline Basisfunktionen für einen Vektor $\mathbf{u} = \{u_0, \ldots, u_c\}$ von Parameterwerten. Eingabewerte der Methode `calc_derivMult` ist der Vektor **u** und der maximale Grad der Ableitungen

l. Das ausgegebene drei-dimensionale MATLAB-Array der k-fach abgeleiteten B-Spline Basisfunktionen besitzt $n+1$ Zeilen, $c+1$ Spalten und $l+1$ Schichten. Letztere bilden die dritte Dimension dieses drei-dimensionalen MATLAB-Arrays.
In dieser Methode wird für jeden Parameterwert u die bereits beschriebene Methode `calc_deriv` aufgerufen und die k-fachen Ableitungen der B-Spline Basisfunktionen für jeden Wert des Vektors **u** berechnet. Dies erfolgt durch eine `for`-Schleife mit dem Zähler von $i = 1, \ldots, c+1$. (Im MATLAB-Code wurde `num_u` $= c+1$ gesetzt)

2.2.7 Methode `calc_derivNum`

Input:

- Parameterwert: u
- maximaler Grad der Ableitung: l

Output:

- Matrix der numerisch abgeleiteten Basisfunktionen: **dersNum**

Die Methode `calc_derivNum` dient zur numerischen Berechnung der k-fachen Ableitung der B-Spline Basisfunktionen für einen Parameter u. Diese Funktion basiert auf den Vorwärtsdifferenzenquotient und dient zur Kontrolle der Methode `calc_deriv`. Eingabewerte sind der Parameterwert u und die Ordnung l der maximalen Ableitung. Die ausgegebene Matrix besitzt die gleichen Dimensionen wie die Ausgabematrix der Methode `calc_deriv`. D.h. die Matrix besitzt $n+1$ Zeilen und $l+1$ Spalten und die (k+1)-te Spalte der Matrix beinhaltet die Ableitungen der Ordnung k.

In *Abb. 2.1* ist die Matrix der exakten Berechnung der Ableitungen mittels der Methode `calc_deriv`, und die Matrix der numerischen Berechnung mittels `calc_derivNum` dargestellt.
Gegeben sei hier der Knotenvektor $\mathbf{U} = [0,1,2,3,4,5,6,7,8,9,10]$, der Grad $p = 3$, der maximale Grad der Ableitungen mit $l = 3$, der Parameterwert $u = 5.3$, die Anzahl der Kontrollpunkte minus 1 mit $n = m - p - 1$.
Es ist ersichtlich, dass die Werte der numerischen Berechnung sehr nahe an den Werten der Methode `calc_deriv` liegen. Die Abweichung zwischen den beiden Methoden lässt sich aufgrund der numerischen Fehler des numerischen Verfahrens zurückführen. Für

die Kontrolle der Ableitungen einer niedrigen Ordnung ist eine ausreichende Genauigkeit vorhanden. Und es zeigt auch, dass die B-Spline Basisfunktionen eine hohe numerische Stabilität besitzen.

Exakte Berechnung der Ableitungen der B-Spline Basisfunktionen:

```
0         0         0         0
0         0         0         0
0.0572   -0.2450    0.7000   -1.0000
0.5902   -0.4650   -1.1000    3.0000
0.3482    0.6650    0.1000   -3.0000
0.0045    0.0450    0.3000    1.0000
0         0         0         0
```

Numerische Berechnung der Ableitungen der B-Spline Basisfunktionen:

```
0         0         0         0
0         0         0         0
0.0572   -0.2447    0.6990   -1.0000
0.5902   -0.4655   -1.0970    3.0000
0.3482    0.6650    0.0970   -3.0000
0.0045    0.0452    0.3010    1.0000
0         0         0         0
```

Abbildung 2.1: Exakte bzw. numerische Berechnung der B-Spline Basisfunktionen.

2.2.8 Methode `plot`

Input: Array von Parameterwerten: $\mathbf{u} = \{u_0, \ldots, u_c\}$

Die Methode `plot` dient der grafische Darstellung der B-Spline Basisfunktionen $N_{i,p}(u)$. Eingabeparameter ist der Vektor $\mathbf{u}$. Für die Berechnung der B-Spline Basisfunktionen $N_{i,p}(u)$ wird die Methode `calc` aufgerufen. Verwendet wird die MATLAB Standard `plot`-Funktion mit dem Array von Parameterwerten $\mathbf{u}$ und einer Matrix $\mathbf{N}$, welche die Werte der Basisfunktion für die Stelle $u_0, \ldots, u_c$ enthält, als Eingabeargumente. Beispiele für solcherart erstellten Grafiken sind in Kapitel 1.1, ab S. 4 in *Abb. 1.2* sowie *Abb. 1.3* und auf der linken Seite in *Abb. 1.4* und *Abb. 1.5* ersichtlich.

2.2.9 Methode `plot_ders`

Input:

- Array von Parameterwerten: $\mathbf{u} = \{u_0, \ldots, u_c\}$
- Index: i
- maximaler Grad der Ableitungen: l

Die Methode `plot_ders` dient der grafischen Darstellung der abgeleiteten Basisfunktionen $N_{i,p}^{(k)}(u)$. Eingabeargumente sind der Vektor $\mathbf{u}$, der Index i und der Grad der höchsten Ableitung l ($k = \{0, \ldots, l\}$).
Zur Berechnung des drei-dimensionalen Arrays der abgeleiteten Basisfunktionen **ders** wird die Methode `calc_derivMult` (*Kap. 2.2.6*) aufgerufen.

In einer `for`-Schleife mit dem Zähler `k = 1:size(ders,3)` wird die MATLAB-Funktion `plot(u, ders(:, :, 1))` aufgerufen. Durch Anwendung des MATLAB-Befehls `hold on` können alle Ableitungen bis zum Grad l in einer gemeinsamen Grafik dargestellt werden.

Will man in einer Grafik nur Ableitungen eines Grades k bzw. Ableitungen für einen bestimmten Index i darstellen, so muss die MATLAB-Funktion `plot` auf `plot(u, ders(i, :, k))` geändert werden. Mit dieser Methode erstellte Grafiken sind z. B. in *Abb. 1.4*, S. 6 und in *Abb. 1.5*, S. 6 ersichtlich.

2.3 Klasse `HyperMatrix`

Diese Klasse wurde vom Auftraggeber bereitgestellt und im Rahmen dieser Arbeit mit Methoden erweitert. Die Klasse `HyperMatrix` dient der Berechnung eines Tensorproduktes. Diese Klasse wurde erstellt, da MATLAB selbst keine Funktion für eine Multiplikation von mehr-dimensionalen Arrays bietet. Die Multiplikation wird anhand der *Einsteinschen Summenkonvention* durchgeführt und beruht im wesentlichen auf den Quell-Code aus [1].

Die Klasse `HyperMatrix` beinhaltet folgende Methoden, wobei die Methoden `diag` und `mtimes` in *Kap. 2.3.2* und *Kap. 2.3.3* detailliert beschrieben werden.

- Methoden `disp`
- Methode `double`

- Methode `ndims`
- Methode `size`
- Methode `mtimes`
- Methode `diag`
- Methode `subsref`
- Methode `subsasgn`

Die Methoden `disp`, `ndims` und `size` werden benötigt, um die entsprechende MATLAB Funktionalität von gewöhnlichen Matrizen auch für Hypermatrizen verwenden zu können. Die Methode `double` dient der Konvertierung einer Hypermatrix in eine gewöhnliche Matrix. Die Methoden `subsref` und `subsasgn` werden benötigt, um auf Hypermatrizen mit der entsprechenden MATLAB Syntax von gewöhnlichen Matrizen zugreifen zu können. Die Method `diag` wurden im Rahmen dieser Arbeit erstellt und wird in *Kap. 2.3.3* detailliert beschrieben.
Die Klasse `HyperMatrix` besitzt die Eigenschaft `Array`.

2.3.1 Konstruktor der Klasse `HyperMatrix`

Input:

- Array

Output:

- Objekt der Klasse `HyperMatrix`

Im Konstruktor wird das Eingabeargument der Eigenschaft `Array` der Klasse `HyperMatrix` zugewiesen.

2.3.2 Methode `mtimes`

Es wurde die MATLAB Standard-Funktion `mtimes` durch eine neue Methode, die wiederum `mtimes` heißt, überladen. Mit `mtimes` können mehrdimensionale Arrays auf einfache Art und Weise multipliziert werden.

Wird in MATLAB die Methode `mtimes` aufgerufen, und einer der beiden zu multiplizierenden Faktoren ist als Hypermatrix definiert, verwendet MATLAB automatisch die neu implementierte Funktion `mtimes`. Eine schwache Besetzung einzelner Hypermatrizen wird noch nicht berücksichtigt.

Die Vorgehensweise der `mtimes` Multiplikation kann wie folgt beschrieben werden:

Gegeben sei:

Eine $(\alpha_1 \times \cdots \times \alpha_m)$-dimensionale Hypermatrix $\mathbf{A}$ mit den Elementen $a_{i_1,\ldots,i_m}$
mit $i_l \in \{1,\ldots,\alpha_l\}$
und eine $(\beta_1 \times \cdots \times \beta_m)$-dimensionale Hypermatrix $\mathbf{B}$ mit den Elementen $b_{j_1,\ldots,j_n}$
mit $j_l \in \{1,\ldots,\beta_l\}$
wobei für $\mu \in 1,\ldots,m$ und für $\nu \in 1,\ldots,n$ gelte: $\alpha_\mu = \beta_\nu =: \gamma$.
Dann ist das Produkt $\mathbf{C}$ von $\mathbf{A}$ und $\mathbf{B}$, in MATLAB-Schreibweise
$\mathtt{C} = \mathtt{mtimes}(\mathtt{A}, \mathtt{B}, \mu, \nu)$
jene $(\alpha_1 \times \cdots \times \alpha_{\mu-1} \times \alpha_{\mu+1} \times \cdots \times \alpha_m \times \beta_1 \times \cdots \times \beta_{\nu-1} \times \beta_{\nu+1} \times \cdots \times \beta_n)$-Hypermatrix
deren Elemente $c_{i_1,\ldots,i_{\mu-1},i_{\mu+1},\ldots,i_m,j_1,\ldots,j_{\nu-1},j_{\nu+1},\ldots,j_n}$ wie folgt berechnet werden:

$$c_{i_1,\ldots,i_{\mu-1},i_{\mu+1},\ldots,i_m,j_1,\ldots,j_{\nu-1},j_{\nu+1},\ldots,j_n} = \sum_{k=1}^{\gamma} a_{i_1,\ldots,i_{\mu-1},k,i_{\mu+1},\ldots,i_m,} \cdot b_{j_1,\ldots,j_{\nu-1},k,j_{\nu+1},\ldots,j_n}$$

Dann kann mit Hilfe der *Einsteinschen Summenkonvention* auch so geschrieben werden:

$$c_{i_1,\ldots,i_{\mu-1},i_{\mu+1},\ldots,i_m,j_1,\ldots,j_{\nu-1},j_{\nu+1},\ldots,j_n} = a_{i_1,\ldots,i_{\mu-1},k,i_{\mu+1},\ldots,i_m,} \cdot b_{j_1,\ldots,j_{\nu-1},k,j_{\nu+1},\ldots,j_n}$$

Beispiel:

Gegeben sei ein zwei-dimensionales Array $\mathbf{A}$ mit den Dimensionen (i, j) und ein ebenfalls zwei-dimensionales Array $\mathbf{B}$ mit den Dimensionen (j, k). Wird entlang der übereinstimmenden Dimension j multipliziert, muss in MATLAB-Schreibweise

```
C = mtimes(A,B,2,1)
```

geschrieben werden. Durch Anwenden der *Einsteinschen Summenkonvention* erhält man ein zwei-dimensionales Array $\mathbf{C}$ mit den Dimensionen (i,k).
D.h. es gilt $\mathbf{C}$(i, k) = $\mathbf{A}$(i, j) $\cdot$ $\mathbf{B}$(j, k).
Diese Vorgehensweise kann auch für Arrays mit höheren Dimensionen und entlang mehrerer Dimensionen gleichzeitig angewendet werden.

Als weiteres Beispiel sei ein drei-dimensionales Array **D** mit den Dimensionen (i, j, k) und ein vier-dimensionales Array **E** mit den Dimensionen (k, i, l, m) gegeben. Wird nun entlang der übereinstimmenden Dimensionen (i, k) multipliziert, muss in MATLAB-Schreibweise

```
F = mtimes(D,E,[1,3],[2,1])
```

geschrieben werden. Durch Anwenden der *Einsteinschen Summenkonvention* wird ein drei-dimensionales Array **F** mit den Dimensionen (j, l, m) erzeugt.

2.3.3 Methode `diag`

Diese Methode ist erforderlich um ein gewöhnliches Array in eine diagonale Hypermatrix konvertieren zu können.

Input: Objekt des Arrays aus dem eine diagonale Hypermatrix erzeugt werden soll

Output: Diagonale Hypermatrix

Zuerst wird mit dem Befehl `size`, die Dimension `d_size` des eingegebenen Arrays bestimmt. Danach werden die Werte des eingegebenen Arrays in einem Spaltenvektor gespeichert, und daraus mit der MATLAB Standard-Funktion `diag` eine diagonale Matrix **d** erzeugt. Als nächstes wird mit dem Befehl `reshape` ein Array **D** in MATLAB-Schreibweise wie folgt belegt:

```
D = reshape (d,cat(2,d_size,d_size))
```

Nun kann aus diesem Array mit der MATLAB-Schreibweise

```
D = HyperMatrix(D)
```

eine Hypermatrix gebildet werden.

2.4 Klasse `csys`

Mit der Klasse `csys` können Translationen und Drehungen realisiert werden. Diese Klasse wurde vom Auftraggeber bereitgestellt.

Die Anwendung der Methode `csys` erfolgt durch das Aufrufen der Methode und Eingabe der Translation (Tx, Ty, Tz) bzw. Rotation um eine Achse (Rx, Ry, Rz) mit einem Winkel

für die jeweilige Richtung. Die Eingabeargumente bezüglich Translation und Rotation können beliebig variiert werden. In dem unten angeführten Beispiel, ist eine Translation in z-Richtung mit dem Wert 10 und eine Rotation um die y-Achse mit dem Winkel $\pi/4$ gegeben. In der Klasse erfolgt die Belegung der zugehörigen (4×4)-Transformationsmatrix $\mathbf{T}$. Diese Matrix sieht wie folgt aus:

$$\mathbf{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} =: \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{o}^T & 1 \end{bmatrix} \tag{2.3}$$

Hierbei ist $\mathbf{R}$ eine (3×3)-Rotationsmatrix d.h.

$$\mathbf{R} \cdot \mathbf{R}^T = \mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \det \mathbf{R} = 1. \tag{2.4}$$

So generierte Transformationsmatrizen $\mathbf{T}$ werden für die weitere Berechnung in den Klassen `NURBS`, `Surface` und den nachfolgenden Klassen verwendet. Wird der Wert der Koordinatentransformation mit null eingegeben d.h. es soll keine Transformation ausgeführt werden, so wird $\mathbf{T}$ automatisch mit der Einheitsmatrix belegt.

Beispiel:

```
csys_1 = csys('Tz', 10, 'Ry', pi/4);
T = csys_1.T;
```

Da Drehungen nicht kommutativ sind, ist die Eingabereihenfolge zu beachten. Im obigen Beispiel wird zuerst die Translation und danach die Rotation durchgeführt.

2.5 Klasse `NURBS`

Die Klasse `NURBS` dient der numerischen Auswertung einer d-variaten rationalen NURBS-Funktion

$$\mathbf{x}(u_1, \ldots, u_d) = \frac{\sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} N_{i_1,p_1}(u_1) \cdots N_{i_d,p_d}(u_d)\, w_{i_1,\ldots,i_d}\, \mathbf{p}_{i_1,\ldots,i_d}}{\sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} N_{i_1,p_1}(u_1) \cdots N_{i_d,p_d}(u_d)\, w_{i_1,\ldots,i_d}}, \tag{2.5}$$

deren Werte im $\mathbb{R}^3$ liegen. Außerdem stellt diese Klasse für $d \leq 2$ eine Methode zur Berechnung der partiellen Ableitungen dieser Funktion zur Verfügung. Um die Funktion auszuwerten, wird ein Tensorprodukt aus den Basisfunktionen $N_{i_j,p_j(u_j)}$ mit dem Array der Kontrollpunkte gebildet.

Insbesondere erhalten wir

- für $d = 1$ NURBS Kurven
- für $d = 2$ NURBS Flächen
- und für $d = 3$ NURBS Volumen.

Die Klasse `NURBS` beinhaltet folgende Methoden, welche in *Kap. 2.5.3 - Kap. 2.5.8* detailliert beschrieben werden:

- `set`-Methoden
- Methode `homogeneous`
- Methode `inhomogeneous`
- Methode `calc_points`
- Methode `calc_derivative_curv`
- Methode `calc_derivative_surf`

2.5.1 Eigenschaften der Klasse `NURBS`

- Array der Kontrollstruktur: **P**
  Das Array der Kontrollstruktur **P** hat die Dimension von $3 \times (n_1 + 1) \times (n_2 + 1) \times \cdots \times (n_d + 1)$. Für den Index i_j der Dimension

$j = 1, 2, \ldots, d$ gilt $i_j = 0, 1, \ldots, n_j$. Für den Fall $d = 1$ ist die Kontrollstruktur ein Kontrollpolygon, für $d = 2$ ein Kontrollnetz und für $d = 3$ ein Kontrollgitter.

- Dimension der Kontrollstruktur: d
  Es gilt für Kurven $d = 1$, für Flächen $d = 2$ und für Volumen $d = 3$.

- Array der Gewichte: **w**
  Die Gewichte werden in einem $1 \times (n_1 + 1) \times (n_2 + 1) \times \cdots \times (n_d + 1)$ dimensionalen Array abgelegt. Zum Kontrollpunkt $\mathbf{P}_{i_1,\ldots,i_d}$ gehört das Gewicht $w_{i_1,\ldots,i_d}$ mit $i_j = 0, 1, \ldots, n_j$ und $j = 1, 2, \ldots, d$.

- Array des Knotenvektors: **U**
  U enthält die den d Dimensionen des NURBS zugehörigen Knotenvektoren $\mathbf{U}_1, \ldots, \mathbf{U}_d$. Die einzelnen Knotenvektoren $\mathbf{U}_j$ können eine unterschiedliche Anzahl von Elementen aufweisen, daher wird der Knotenvektor als Cell-Array initialisiert. Das Cell-Array hat die Form:

$$\begin{aligned} U\{1\} = &\quad [U1(1), &\ldots, &\quad U1(m(1)+1)] \\ U\{2\} = &\quad [U2(1), &\ldots, &\quad U2(m(2)+1)] \\ \vdots &\quad \vdots &\vdots &\quad \vdots \\ U\{d\} = &\quad [Ud(1), &\ldots, &\quad Ud(m(d)+1)] \end{aligned} \tag{2.6}$$

- Transformationsmatrix: **T**
  Die (4×4)-Transformationsmatrix **T** wird für eine Koordinatentransformation von einem lokalen Koordinatensystem in ein globales Koordinatensystem benötigt. Die Matrix wird durch Aufrufen der Funktion `csys` erhalten.

- Cell-Array der Objekte der Klasse `BasisFunMat`: `basFun`
  Die Anzahl der Objekte im Array entspricht der Dimension des Kontrollgitters.

- Kontrollstruktur: $\mathbf{P}^w$
  Die Kontrollstruktur $\mathbf{P}^w$ inkludiert die Gewichte und die Transformation vermöge **T** jedes einzelnen Punktes. Die ersten drei Zeilen von $\mathbf{P}^w_{i_1,\ldots,i_d}$ beinhalten die Koordinaten des Kontrollpunktes $\mathbf{P}_{i_1,\ldots,i_d}$ in x-, y- und z-Richtung multipliziert mit den Gewichten. Die letzte Komponente dieses $(d+1)$-dimensionalen Arrays beinhaltet

das Gewicht $w_{i_1,\ldots,i_d}$:

$$\mathbf{P}^w_{i_1,\ldots,i_d} = \begin{bmatrix} w_{i_1,\ldots,i_d} \cdot \mathbf{P}_{1,i_1,\ldots,i_d} \\ w_{i_1,\ldots,i_d} \cdot \mathbf{P}_{2,i_1,\ldots,i_d} \\ w_{i_1,\ldots,i_d} \cdot \mathbf{P}_{3,i_1,\ldots,i_d} \\ w_{i_1,\ldots,i_d} \end{bmatrix} \tag{2.7}$$

- Array: $[n(1), \ldots, n(d)]$.
- Array: $[m(1), \ldots, m(d)]$.
- Grade der Basisfunktionen: p
  Hierbei gilt: $p(i) = m(i) - n(i) - 1$.

2.5.2 Konstruktor der Klasse `NURBS`

Input:

- Array der Kontrollstruktur: **P**
- Cell-Array des Knotenvektors: **U**
- Transformationsmatrix: **T**
  Anmerkung: Die übergebene (4×4)-Matrix **T** kann etwa durch die Funktion `csys` erzeugt werden (siehe *Kap. 2.4* , S. 44).
- Array der Gewichte: **w**

Output:

- Objekt der Klasse `NURBS`

Die Eigenschaften **P**, **U**, **T** und **w** werden ihrem Objekt zugewiesen. Der Eigenschaft der Dimension d des Kontrollgitters wird die Anzahl der Zeilen des Knotenvektors **U** mit dem Befehl `size` zugewiesen. Die Zahl der Kontrollpunkte - 1 (n) einer Dimension i werden durch

```
n(i) = size (P,i+1) - 1
```

definiert. Die Eigenschaft `basFun` wird auf ein ein-dimensionales Cell-Array mit d Einträgen initialisiert. Der Eigenschaft der gewichteten Kontrollpunkte $\mathbf{P}^w$, wird das Ergebnis der Funktion `homogeneous` (siehe *Kap. 2.5.4* , S. 50) zugewiesen .
Mittels einer `for`-Schleife mit dem Zähler `1,...,d` werden für jede Dimension $j = 1, \ldots, d$ der Kontrollstruktur $\mathbf{P}$, die m_j, p_j und die Eigenschaft `basfun(j)` der Klasse `BasisFunMat` berechnet. Der Grad p_j errechnet sich gemäß *Gl. 1.7* , S. 8 durch $p(j) = m(j) - n(j) - 1$. Die Eigenschaft `basFun(f)` wird durch Aufrufen des Konstruktors der Klasse `BasisFunMat` mit den Input Argumenten $\mathbf{U}_j$ und n_j für jede Dimension j erzeugt.

2.5.3 `set`-Methoden

Es gibt drei `set`-Methoden, die der Festlegung

- des Arrays $\mathbf{P}$ des Kontrollgitters,
- des Arrays $\mathbf{w}$ der Gewichte
- und des Cell-Arrays $\mathbf{U}_1, \ldots, \mathbf{U}_d$ der Knotenvektoren dienen.

Diese Methoden ermöglichen es dem Anwender, ein erstelltes Objekt zu ändern, ohne dies neu zu erzeugen bzw. ohne eine übergeordnete Benutzerschnittstelle verwenden zu müssen. Wird eine neue Eigenschaft eingegeben, so wird das bestehende Objekt bezüglich dieser Eigenschaft geändert.
Eingabeargumente der jeweiligen Methoden sind die neuen Werte `new_P, new_w` und `new_U` des Arrays der Kontrollpunkte $\mathbf{P}$, des Arrays der Gewichte $\mathbf{w}$ bzw. des Knotenvektors $\mathbf{U}$.

In der jeweiligen Methode wird mit einer `if`-Bedingung und dem Befehl `isempty` abgefragt, ob ein neues Eingabeargument eingeben wurde. War dies nicht der Fall, wird die Standard MATLAB `set`-Methode zur Erzeugung des Objektes verwendet.

Falls mit der hier definierten `set`-Methode eine Änderung der genannten Arrays vorgenommen wurde, wird diese neue Eigenschaft dem Objekt zugewiesen.
In den `set`-Methoden für $\mathbf{w}$ und $\mathbf{P}$ wird das gewichtete Array der Kontrollstruktur $\mathbf{P}^w$ wie im Konstruktor durch Aufrufen der Methode `homogeneous` neu berechnet.

In der `set`-Methode von **U** wird nach einer Änderung von **U**, das Eigenschafts-Objekt `basFun` neu berechnet. Dies erfolgt gleich wie im Konstruktor durch Aufrufen der Konstruktoren der Klasse `BasisFunMat` für die jeweilige Dimension d der Kontrollstruktur mittels einer `for`-Schleife.

2.5.4 Methode `homogeneous`

Input: Objekt der Klasse NURBS

Output: Gewichtete und transformierte Hypermatrix $\mathbf{P}^w$ in homogenen Koordinaten

Die Methode `homogeneous` dient sowohl der Umrechnung der Koordinaten des euklidischen Raumes $\mathbb{R}^3$ in homogene Koordinaten (Definition von homogenen Koordinaten siehe Kapitel 1.3.1, S. 11), wie auch der Anwendung einer euklidischen Kongruenztransformation.

Die einzelnen Schritte der Methode können folgendermaßen dargestellt werden:

- Zuerst wird dem Objekt der Kontrollstruktur **P** eine vierte Zeile mit den Werten eins hinzugefügt.

$$\mathbf{P}_{i_1,\ldots,i_d} = \begin{bmatrix} x_{i_1,\ldots,i_d} \\ y_{i_1,\ldots,i_d} \\ z_{i_1,\ldots,i_d} \end{bmatrix} \rightarrow \begin{bmatrix} x_{i_1,\ldots,i_d} \\ y_{i_1,\ldots,i_d} \\ z_{i_1,\ldots,i_d} \\ 1 \end{bmatrix} \tag{2.8}$$

  Dies wird mit der Methoden `subsref` bzw. `subsasgn` umgesetzt. In MATLAB-Schreibweise wird dies wie folgt durchgeführt:

```
S.type = '()'
S.subs = cat(2,4,repmat({':'},1,d
P = subsasgn(P,S,ones(cat(2,1,n+1)))
```

- Danach erfolgt die Anwendung der Transformation $\mathbf{T}$ auf die Kontrollstruktur $\mathbf{P}$.

$$\mathbf{P}'_{i_1,\ldots,i_d} = \begin{bmatrix} x'_{i_1,\ldots,i_d} \\ y'_{i_1,\ldots,i_d} \\ z'_{i_1,\ldots,i_d} \\ 1 \end{bmatrix} = \mathbf{T} \cdot \mathbf{P}_{i_1,\ldots,i_d} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{i_1,\ldots,i_d} \\ y_{i_1,\ldots,i_d} \\ z_{i_1,\ldots,i_d} \\ 1 \end{bmatrix} \tag{2.9}$$

  Für die Multiplikation der (4×4)-Transformationsmatrix $\mathbf{T}$ mit der Kontrollstruktur $\mathbf{P}$ wird die Methode `mtimes` der Klasse `HyperMatrix` verwendet. Es wird entlang der übereinstimmenden Dimensionen multipliziert. In MATLAB-Schreibweise: $\mathtt{P'} = \mathtt{mtimes}(\mathtt{T}, \mathtt{P}, 2, 1)$

- Anschließend erfolgt die Berücksichtigung der Gewichtung der nun bereits transformierten Kontrollstruktur $\mathbf{P}'$.

$$\mathbf{P}^w_{i_1,\ldots,i_d} = \mathbf{W} \cdot \mathbf{P}'_{i_1,\ldots,i_d} = \begin{bmatrix} w_{i_1,\ldots,i_d} \cdot x'_{i_1,\ldots,i_d} \\ w_{i_1,\ldots,i_d} \cdot y'_{i_1,\ldots,i_d} \\ w_{i_1,\ldots,i_d} \cdot z'_{i_1,\ldots,i_d} \\ w_{i_1,\ldots,i_d} \end{bmatrix} \tag{2.10}$$

  Um diese Multiplikation durchführen zu können, wird das MATLAB Array der Gewichte $\mathbf{w}$ in eine diagonale Hypermatrix $\mathbf{W}$ konvertiert. Dieser Vorgang erfolgt mit der Methode `diag` aus der Klasse `Hypermatrix` (siehe *Kap. 2.3.3* , S. 44).
  Danach kann die Multiplikation der Hypermatrix $\mathbf{W}$ mit dem Array der transformierten Kontrollstruktur $\mathbf{P'}$, mittels der Methode `mtimes` aus der Klasse `Hypermatrix` durchgeführt werden. Dies kann in MATLAB-Schreibweise unter Berücksichtigung der Dimension d der Kontrollstruktur wie folgt dargestellt werden: $\mathtt{P^w} = \mathtt{mtimes}(\mathtt{P'}, \mathtt{W}, 2 : \mathtt{d} + 1, 1 : \mathtt{d})$

2.5.5 Methode `inhomogeneous`

Input: Hypermatrix $\mathbf{X}^w$ aus der Methode `calc_points`

Output: Hypermatrix $\mathbf{X}$

Die Methode `inhomogeneous` dient der Transformation der homogenen Koordinatenspalten (Vektoren in $\mathbb{R}^4$) in die affinen Koordinatenspalten des $\mathbb{R}^3$. Input-Argument ist eine Hypermatrix $\mathbf{X}^w$. Eine Darstellung der Vorgehensweise ist in *Gl. 2.11* ersichtlich.

Zuerst wird aus der Hypermatrix $\mathbf{X}^w$ mittels der Methoden `subsref` bzw. `subsasgn` die letzte Zeile der Matrix, welche die Gewichte w beinhaltet, extrahiert. Aus den Werten der letzten Zeile wird der Kehrwert gebildet. Danach wird aus der Hypermatrix des Kehrwerts, eine diagonale Hypermatrix $\mathbf{W}_{rec}$ mittels der Methode `diag` aus der Klasse `HyperMatrix` erstellt.

Anschließend erfolgt mittels `mtimes` eine Multiplikation dieser diagonalen Hypermatrix $\mathbf{W}_{rec}$ mit der Hypermatrix $\mathbf{X}^w$. Dies wird in MATLAB Schreibweise durch

```
X = mtimes(X_w,W_rec,2:d+1,1:d)
```

umgesetzt. Die Methode `mtimes` wird für diese Multiplikation nur verwendet wenn gilt

```
ndims(W_rec) > 2
```

wenn dies nicht der Fall ist, wird die Multiplikation der Matrix $\mathbf{X}^w$ mit dem diagonalen Kehrwert $\mathbf{W}_{rec}$, mit der MATLAB-Standard Multiplikation durchgeführt. Dies wird im MATLAB-Code durch eine `if - else` Verzweigung umgesetzt.

$$\mathbf{X}^w_{i_1,\dots,i_d} = \begin{bmatrix} w_{i_1,\dots,i_d} \cdot x'_{i_1,\dots,i_d} \\ w_{i_1,\dots,i_d} \cdot y'_{i_1,\dots,i_d} \\ w_{i_1,\dots,i_d} \cdot z'_{i_1,\dots,i_d} \\ w_{i_1,\dots,i_d} \end{bmatrix} \rightarrow \mathbf{X}_{i_1,\dots,i_d} = \begin{bmatrix} \frac{w_{i_1,\dots,i_d} \cdot x'_{i_1,\dots,i_d}}{w_{i_1,\dots,i_d}} \\ \frac{w_{i_1,\dots,i_d} \cdot y'_{i_1,\dots,i_d}}{w_{i_1,\dots,i_d}} \\ \frac{w_{i_1,\dots,i_d} \cdot z'_{i_1,\dots,i_d}}{w_{i_1,\dots,i_d}} \end{bmatrix} = \begin{bmatrix} x'_{i_1,\dots,i_d} \\ y'_{i_1,\dots,i_d} \\ z'_{i_1,\dots,i_d} \end{bmatrix} \tag{2.11}$$

2.5.6 Methode `calc_points`

Input: Array von Parameterwerten: $\mathbf{u} = \{u_{i_1,\dots,i_d}\}$

Output: Hypermatrix $\mathbf{X}$ des NURBS zu den gegebenen Parameterwerten: $\mathbf{X} = \{X_{i_1,\dots,i_d}\}$

Die Methode `calc_points` dient zur Auswertung von Punkten eines NURBS $\mathbf{X}(u)$, wobei $\mathbf{u} = (u_{i_1,\dots,i_d})$ das Array von Parameterwerten beschreibt. Für den Index i_j mit der Dimension $j = 1, \dots, d$ gilt $i_j = 0, \dots, r_j$ wobei in der Regel $r_j \gg n_j$ gilt. Das Eingabeargument fasst die Parameterwerte in einem Array $\mathbf{u}$ zusammen.

Mit einer `for`-Schleife, die den Zähler `1`, ..., `d` besitzt, werden die Matrizen der Basisfunktionen $\mathbf{N}_{i_j,p_j}$ berechnet, wobei $j = 1, \ldots, d$ gilt. Dies wird durch Aufrufen der Methode `calc` des Objekts `basFun` aus der Klasse `BasisFunMat` durchgeführt.
Anschließend erfolgt die Berechnung der Punkte eines NURBS in homogenen Koordinaten (siehe *Kap. 2.5.4* , S. 50). Dies erfolgt mittels einer `mtimes` Multiplikation der berechneten Basisfunktionen $\mathbf{N}$ mit der gewichteten und transformierten Hypermatrix $\mathbf{P}^w$. Diese Berechnung kann durch die MATLAB Schreibweise

```
X = mtimes(P_w,N,2,1)
```

umgesetzt werden. Die aus dieser Multiplikation erhaltene homogene Hypermatrix $\mathbf{X}$ wird danach in den euklidischen Raum $\mathbb{R}^3$ mittels der zuvor beschriebenen Methode `inhomogeneous` transformiert. Ergebnis ist die überschriebene inhomogene Hypermatrix $\mathbf{X}$.

2.5.7 Methode `calc_derivative_curv`

Die Methode `calc_derivative_curv` dient zur Berechnung der Ableitungsvektoren von NURBS Kurven.

Input:

- Parameterwert: u
- Ordnung der Ableitung: l

Output:

- $(3 \times (l+1))$-Matrix $\mathbf{X}_c$ der Ableitungsvektoren der Ordnung $0, \ldots, l$ einer NURBS Kurve

Die Berechnung der Ableitungsvektoren erfolgt anhand der in *Kap. 1.3.6* , S. 20 beschriebenen Methode.
Input Argumente sind der Parameterwert u und die maximale Ordnung l der Ableitungen.

Zuerst wird die Matrix der abgeleiteten Basisfunktionen **ders** durch Aufrufen der Methode `calc_deriv` des Objekts `basFun` aus der Klasse `BasisFunMat` berechnet.

Für die Berechnung der Ableitungsvektoren wird die Eigenschaft des Tensorproduktes und somit die Methode `mtimes` der Klasse `HyperMatrix` verwendet. Es wird die Hypermatrix

$\mathbf{P}^w$, mit der zuvor berechneten Matrix der abgeleiteten Basisfunktionen **ders** multipliziert. In MATLAB-Schreibweise:

```
Xc = mtimes(P_w,ders,2,1)
```

Die so berechneten Ableitungsvektoren können nun in einer gewöhnlichen $(4 \times (l+1))$-Matrix gespeichert werden. Die Umwandlung der Hypermatrix in diese gewöhnliche Matrix wird wieder mit der Funktion `double` erledigt (siehe *Kap. 2.3* , S. 41).

Danach wird eine transponierte Submatrix $\mathbf{F}$ gespeichert, welche aus der ersten bis dritten Zeile dieser Matrix besteht:

```
F = (Xc(1:3,:))'
```

Anschließend wird eine Matrix $\mathbf{G}$ wie in *Gl. 1.34* , S. 21 belegt. Dazu wird die letzte Zeile der oben berechneten homogenen Matrix verwendet, welche die Gewichte beinhaltet. Dazu werden zwei `for`-Schleifen mit den Schleifenzählern von `i = 0, ..., l` und `j = 0, ..., i` verwendet. Die Matrix $\mathbf{G}$ wird wie folgt belegt:

$$\mathbf{G}_{i+1,j+1} = \binom{i}{j} \cdot \mathbf{X}_c\,(4, i-j+1)$$

Die Berechnung der $(3 \times (l+1))$-Matrix $\mathbf{X}_c$, welche die Ableitungsvektoren beinhaltet, kann formal gemäß *Gl. 1.37* , S. 22 (an dieser Stelle wurden jedoch andere Bezeichnungen verwendet) durchgeführt werden.
In MATLAB wird für diese Berechnung die Funktion `linsolve` eingesetzt, diese Funktion löst ein Gleichungssystem. In dieser Arbeit wird bei der Verwendung von `linsolve` immer die Matrixeigenschaft `LT` für `Lower Triangular` angegeben, da in diesen Berechnungen stets untere Dreiecksmatrizen vorkommen. Durch diese Angabe verwendet MATLAB automatisch den speziellen Löser für untere Dreiecksmatrizen.

Die i-te Spalte der Matrix $\mathbf{X}_c$ enthält den $(i-1)$-ten Ableitungsvektor zum gegebenen Parameterwert u. In der ersten Spalte steht der zu u gehörende Kurvenpunkt.

2.5.8 Methode `calc_derivative_surf`

Die Methode `calc_derivative_surf` dient zur Berechnung der partiellen Ableitungsvektoren von NURBS Flächen.

Input:

- Parameterwertpaar u_1, u_2
- maximale Ordnung $l \leq 2$ der Ableitungen

Output:

- $(3 \times ((l+1)(l+2)/2))$-Matrix $\mathbf{X}_s$ der partiellen Ableitungen bis zur Ordnung l

Die Berechnung der partiellen Ableitungen erfolgt anhand der in *Kap. 1.3.9* , S. 27 beschriebenen Methode.

Eingabeargumente sind der Parameterwert u und die maximale Ordnung l der Ableitungen. Ausgegeben wird eine Matrix $\mathbf{X}_s$ mit den Ableitungsvektoren der abgeleiteten Fläche. Die Matrix entspricht der durch *Gl. 1.59* , S. 31 berechneten Matrix mit dem Unterschied dass die Matrix aus *Gl. 1.59* in MATLAB noch transponiert wurde.

Die Matrix $\mathbf{X}_s$besitzt drei Zeilen und $((l+1) \cdot (l+2))/2$ Spalten. Die Matrizen $\mathbf{D}_1$ bzw. $\mathbf{D}_2$ der abgeleiteten Basisfunktionen für die Richtungen u_1 und u_2 werden durch Aufrufen der Methode `calc_deriv` des Objekts `basFun` aus der Klasse `BasisFunMat` berechnet.
Danach wird die Matrix **Xs** belegt. Dazu wird, wie in *Gl. 1.8* , S. 9 gezeigt, das Produkt der zwei Arrays der Basisfunktionen $N_{i,p}(u_1)$ und $N_{j,q}(u_2)$ mit dem Kontrollnetz $\mathbf{P}_{i,j}$ benötigt. Diese Multiplikation wird Schrittweise durch die Methode `mtimes` durchgeführt.

Zuerst werden die Basisfunktionen $\mathbf{D}_1$ der u_1-Richtung, mit der gewichteten und transformierten Hypermatrix des Kontrollnetzes $\mathbf{X}^w$ multipliziert. Dadurch erhält man die Hypermatrix $\mathbf{M}$. Diese Hypermatrix $\mathbf{M}$ wird danach mittels einer weiteren `mtimes` Multiplikation mit den Basisfunktionen $\mathbf{D}_2$ der u_2-Richtung multipliziert. Das Ergebnis ist die Matrix $\mathbf{X}$.

Anschließend wird mit der nun berechneten Matrix $\mathbf{X}$ die Matrix $\mathbf{X}_s$ belegt. Diese Belegung erfolgt mittels einer doppelten `for` -Schleife mit den Schleifenzählern `i = 0,...,l` und `j = i,...,0` .
Für die Belegung der Spalten der Matrix $\mathbf{X}_s$ wird eine Variable α definiert. Diese Variable wird in Abhängigkeit der beiden Schleifen mit $\alpha = ((\mathtt{i} * (\mathtt{i}+1))/2) + (\mathtt{i} - \mathtt{j} + 1)$ festgelegt. Als nächstes erfolgt die Initialisierung der Matrix G mit $((l+1) \cdot (l+2))/2$ Zeilen und Spalten. Danach werden die ersten drei Zeilen der Matrix $\mathbf{X}_s$ transponiert und unter der Variablen F gespeichert.

Anschließend erfolgt die Belegung der Matrix $\mathbf{G}$ mit der letzten Zeile der Matrix $\mathbf{X}_s$, welche ja die Ableitungen der Nennerfunktion der NURBS-Fläche beinhalten, gemäß *Gl. 1.56* , S. 29.

Für die Durchführung werden drei `for`-Schleifen verwendet, die Schleifenzähler sind $\mathtt{i} = 0, \ldots, \mathtt{j}$, $\alpha = 0, \ldots, \mathtt{i}$ und $\beta = 0, \ldots, \mathtt{j} - \mathtt{i}$.
Es werden mehrere Zusatzvariablen eingeführt, um die Wiederholung von Berechnungsschritten zu vermeiden.

- `jj3 = j` $\cdot \frac{\mathtt{j}+3}{2}$
- `ji = j - i`
- `ab =` α `+` β
- `ab1 = ab + 1`

Die Belegung von $\mathbf{G}$ erfolgt gemäß *Gl. 1.55* , S.29 durch:

```
G(jj3 - i + 1, ab * ab1/2 + beta + 1) = nchoosek(i,alpha) *
nchoosek(ji,beta) * Xs(4,(j - ab) * (j - ab + 1)/2 + ji - beta + 1)
```

Anschließend wird mit der Funktion `linsolve` die Matrix $\mathbf{X}_s$ der Ableitungsvektoren berechnet, wobei wieder die Option `LT` verwendet wird. Die erste Spalte beinhaltet den Ableitungsvektor der Ordnung null, die zweite und dritte Spalte beinhaltet die Ableitungsvektoren der ersten Ableitung in Richtung u_1 bzw. u_2. Die Spalten vier bis sechs beinhalten die Vektoren der zweiten Ableitung. Hierbei beinhaltet die 4. bzw. 5. bzw. 6. Spalte die Ableitungen $\frac{\partial^2 \mathbf{S}}{\partial {u_1}^2}$ bzw. $\frac{\partial^2 \mathbf{S}}{\partial u_1 \partial u_2}$ bzw. $\frac{\partial^2 \mathbf{S}}{\partial {u_2}^2}$.

2.6 Super-Klasse `Surface`

Die Super-Klasse `Surface` stellt eine Benutzerschnittstelle für Flächenobjekte dar. Es werden Methoden zur Berechnung von Flächenpunkten, Methoden zur Berechnung der Jacobi-Matrix und des Hesse-Tensors sowie `plot`-Funktionen für die grafische Darstellung der Flächen Klassen implementiert. Von dieser Klasse werden in weiterer Folge Spezialfälle abgeleitet, welche die Eigenschaften dieser Super-Klasse nutzen. Es wird die Super-Klasse `Revolved Surface` für Drehflächen abgeleitet und von dieser Klasse weitere Klassen für technisch wichtige Drehflächen, wie Drehkegel, Drehzylinder und Torus. Die Konstrukto-

ren der abgeleiteten Klassen ermöglichen eine automatisierte Erstellung des Kontrollnetzes, der Gewichte und der Knotenvektoren für die genannten Flächen.

2.6.1 Eigenschaft der Klasse `Surface`

NURBS-Objekt für eine Fläche: `Surf`

2.6.2 Konstruktor der Klasse `Surface`

Input:

- Kontrollstruktur: $\mathbf{P}$
- Array $\mathbf{U}$ der beiden Knotenvektoren: $\mathbf{U} = \{\mathbf{U}_1, \mathbf{U}_2\}$
- Transformationsmatrix: $\mathbf{T}$
- Array der Gewichte: $\mathbf{w}$

Output:

- Objekt der Klasse `Surface`.

Nach Überprüfung der Anzahl der Inputargumente – diese muss vier sein – wird ein entsprechendes Objekt der Klasse `NURBS` erzeugt.

2.6.3 Methode `calc_points_surf`

Input: 2 Arrays für die u_1- und u_2-Parameterwerte

Output: Matrix der Flächenpunkte: $\mathbf{X}$

Diese Methode gibt die Inputdaten an die Methode `calc_points` der Klasse `NURBS` weiter und man erhält die Matrix $\mathbf{X}$ als Rückgabewert.

2.6.4 Methode `jacobian`

Input: Parameterwertepaar: $\mathbf{u} = [u_1, u_2]$

Output: (3×2)-Jacobi-Matrix: $\mathbf{J}$

Die Methode `jacobian` dient der Belegung der Jacobi-Matrix, siehe Anmerkung, S. 31:

$$\mathbf{J}(u_1, u_2) = \begin{bmatrix} \partial\mathbf{S}/\partial u_1\,(u_1, u_2) & \partial\mathbf{S}/\partial u_2\,(u_1, u_2) \end{bmatrix}$$

In der Methode `jacobian` wird die Funktion `calc_derivative_surf` aus der Klasse `NURBS` aufgerufen und in einer Matrix $\mathbf{X}_s$ gespeichert. Es kann nun mittels den Standard-MATLAB Befehlen auf die jeweiligen Spalten zugegriffen werden. Um die partielle Ableitung in u_1-Richtung zu erhalten wird die zweite Spalte benötigt. Die partielle Ableitung in u_2-Richtung ist in der dritten Spalte enthalten. Die Belegung der Jacobi-Matrix kann einfach mit der MATLAB-Schreibweise

```
J = [X_s(:,2),X_s(:,3)]
```

durchgeführt werden.

2.6.5 Methode `hesse`

Input: Parameterwertepaar: $\mathbf{u} = [u_1, u_2]$

Output: $(2 \times 2 \times 3)$-Hesse-Hypermatrix: $\mathbf{H}$

Die Belegung der Hesse-Hypermatrix erfolgt mit der Methode `hesse`. In der Hesse-Hypermatrix können die partiellen Ableitungen zweiter Ordnung einer bivariaten Funktion zusammengefasst werden (siehe Anmerkung, S. 31). Die Hesse-Hypermatrix kann wie folgt dargestellt werden:

$$\mathbf{H}(u_1, u_2) = \begin{bmatrix} \partial^2\mathbf{S}/\partial u_1^2\,(u_1, u_2) & \partial^2\mathbf{S}/\partial u_1 \partial u_2\,(u_1, u_2) \\ \partial^2\mathbf{S}/\partial u_2 \partial u_1\,(u_1, u_2) & \partial^2\mathbf{S}/\partial u_2^2\,(u_1, u_2) \end{bmatrix}$$

Für diese Methode wird wie schon in der Mehtode `jacobian` die Methode `calc_derivative_surf` aus der Klasse `NURBS` aufgerufen und in einer Matrix $\mathbf{X}_s$ gespeichert. Der nächste Schritt ist die Zusammensetzung der Hesse-Hypermatrix aus den entsprechenden Teilen der Matrix $\mathbf{X}_s$. Hierzu gibt es mehrere Möglichkeiten, es wurde die Variante umgesetzt, dass jeweils die x-, y- und z-Koordinaten in einer eigenen Schicht, also in der dritten Dimension, vorhanden sind. Daher hat die Hypermatrix $\mathbf{H}$ zwei Zeilen, zwei Spalten und drei Schichten.

2.6.6 Methode `plot_surface`

Diese Methode dient zur grafischen Darstellung einer allgemeinen NURBS Fläche. Geplottet wird ein approximiertes Netz der Fläche bestehend aus $(nop\_u_1) \times (nop\_u_2)$ Flächenpunkten.

Input:

- Anzahl der Punkte in u_1 -Richtung: nop_u_1
- Anzahl der Punkte in u_2 -Richtung: nop_u_2
- Variable Argumente der Eigenschaften des Plots (Farbe- und Transparenz-Eigenschaften)

Zuerst wird ein Cell-Array **u** initialisiert, damit für jede der beiden Richtungen u_1 und u_2 ein unterschiedliches Array zugewiesen werden kann. Danach werden die beiden Arrays für die jeweilige Richtung belegt. Dies erfolgt durch eine äquidistante Aufteilung des Parameterintervalls in nop_u_1 bzw. nop_u_2 Werte:

$$\left.\begin{aligned} u_{1,i} &= u_{1,p_1} + i\,\frac{u_{1,m-p_1+1} - u_{1,p_1}}{nop\_u_1 - 1} \qquad i = 0, \ldots, nop\_u_1 - 1 \\ u_{2,i} &= u_{2,p_2} + i\,\frac{u_{2,m-p_2+1} - u_{2,p_2}}{nop\_u_2 - 1} \qquad i = 0, \ldots, nop\_u_2 - 1 \end{aligned}\right\} \tag{2.12}$$

Durch das Aufrufen der Methode `calc_points` der Klasse `NURBS` wird das approximierende Netz in einer Hypermatrix **X** zurückgegeben. Für die weitere Verwendung wird diese Hypermatrix **X** mit der Funktion `double` in eine gewöhnliche Matrix konvertiert.

Anschließend wird zeilenweise die Funktion `squeeze` für die Komponenten der dreidimensionalen Matrix **X** durchgeführt, um alle Dimensionen mit nur einem Element zu entfernen. Danach wird die MATLAB-Funktion `surf` für die Erzeugung der grafischen Darstellung angewendet. Weiters werden die Plot-Eigenschaften für Oberflächenfarbe, Transparenz und Linienfarbe festgelegt, welche mittels der variablen Argumentenliste definiert wurden. Damit kann eine optionale benutzerdefinierte Eingabe der Plot-Eigenschaften erfolgen. In *Abb. 2.2* ist eine allgemeine NURBS Fläche dargestellt, welche auf diese Weise erzeugt wurde.

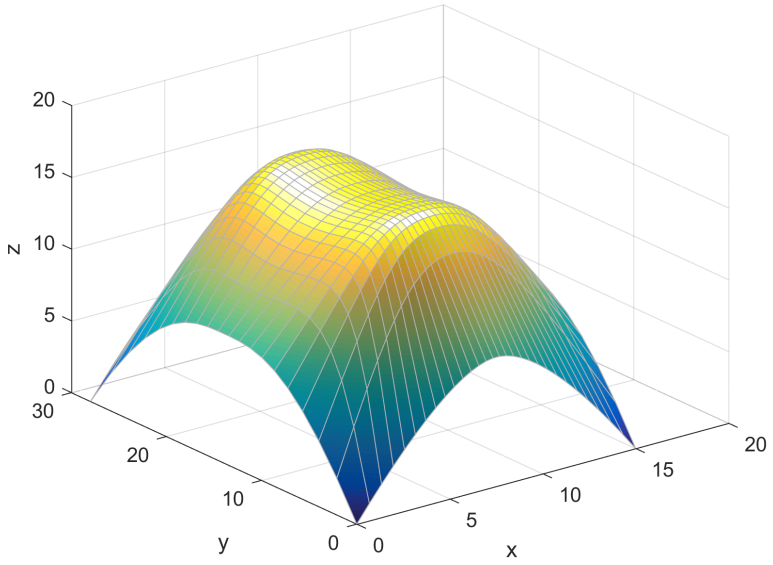

Abbildung 2.2: Allgemeine NURBS Fläche.

2.7 Klasse `Revolved Surface`

Die Klasse `Revolved Surface` wird von der Klasse `Surface` abgeleitet und erbt daher alle Eigenschaften dieser Klasse. Die Klasse wird um Eigenschaften und Methoden, welche speziell für Drehflächen gelten, ergänzt und erweitert.

2.7.1 Eigenschaft der Klasse `Revolved Surface`

- Punkt auf der Achse: $\mathbf{a}_1$
- normierter Richtungsvektor: $\mathbf{d}_0$

2.7.2 Konstruktor der Klasse `Revolved Surface`

Input:

- Kontrollpolygon der erzeugenden Kurve: $\mathbf{P}$
- Knotenvektor der erzeugenden Kurve: $\mathbf{U}$
- Transformationsmatrix: $\mathbf{T}$

- Array der Gewichte der erzeugenden NURBS Kurve (Profilkurve): **w**

Output:

- Objekt der Klasse `Revolved Surface`

Die Berechnung des Kontrollnetzes für die gesamte Drehfläche erfolgt aus dem Kontrollpolygon einer beliebigen NURBS Kurve (erzeugende Kurve), und wird direkt im Konstruktor durchgeführt. Die Vorgehensweise zur Erstellung einer rationalen Drehfläche als NURBS Fläche ist in *Kap. 1.3.8* , S. 24 ausgeführt.
Im Konstruktor der Klasse `Revolved Surface` wird das Kontrollnetz $\mathbf{P}_S$, die Gewichte $\mathbf{w}_S$ und der Knotenvektor $\mathbf{U}_S$ berechnet und danach der Konstruktor der Klasse `Surface` mit diesen Eingabeargumenten aufgerufen. Damit wird ein NURBS-Objekt für Drehflächen erzeugt.

Das Kontrollnetz der Drehfläche erhält man mittels der in *Kap. 1.3.8* , S. 24 beschriebenen Vorgangsweise.

Die Datenstruktur des drei-dimensionalen Arrays $\mathbf{P}_S$ (Kontrollnetz der Drehfläche) ist wie folgt aufgebaut: Die erste Dimension weist drei Elemente entsprechend der drei Koordinaten in x-, y- und z-Richtung eines Punktes auf. Die zweite Dimension entspricht der Anzahl der Punkte des Kontrollpolygons **P** der erzeugenden NURBS Kurve. Die dritte Dimension hat neun Einträge, entsprechend der Punktezahl entlang des Quadrats, welches dem Drehkreis umschrieben ist (Ecken und Seitenhalbierenden Punkte, siehe *Abb. 1.11* , S. 17).

Zuerst wird die erste Schicht mit dem Kontrollpolygon der NURBS Kurve belegt:

```
PS(:,:,1) = P
```

Danach erfolgt die Belegung der Schichten zwei bis neun in einer `for`-Schleife mit dem Zähler `i = 1, ..., size(P,2)`.

Für die Lotfußpunktmethode müssen ein Punkt auf der Drehachse a und ein normierter Richtungsvektor bekannt sein. Da a priori die z-Achse als Drehachse verwendet wird, wurde

```
a1 = [0.0; 0.0; 0.0]
d1 = [0.0; 0.0; 1.0]
```

in Richtung der z-Achse gesetzt.

Danach erfolgt die Berechnung des Lotfußpunktes $\mathbf{Q}$, vom Punkt $\mathbf{P}_i$ des Kontrollpolygons der NURBS Kurve auf die Drehachse a gemäß *Gl. 1.43* , S. 25. Dies kann in MATLAB-Schreibweise wie folgt dargestellt werden:

```
Q = a1 + dot (d1, P(:,i) - a1) * d1
```

Anschließend erfolgt die Berechnung der zur Drehachse a normalen Vektoren $\mathbf{e}_x$ und $\mathbf{e}_y$ wie in *Kap. 1.3.8* , S. 24 beschrieben. In MATLAB-Schreibweise:

```
ex = P(:,i) - Q
ey = cross(d1, ex)
```

Mit den nun berechneten Vektoren kann die Belegung der Kontrollstruktur $\mathbf{P}_S$ des Drehkreises ausgehend vom Lotfußpunkt $\mathbf{Q}$ durchgeführt werden. Die Berechnung der einzelnen Schichten des Arrays $\mathbf{P}_S$ liest sich als MATLAB-Code so:

```
PS(:,i,2) = Q + ex + ey;
PS(:,i,3) = Q + ey;
PS(:,i,4) = Q - ex + ey;
PS(:,i,5) = Q - ex;
PS(:,i,6) = Q - ex - ey;
PS(:,i,7) = Q - ey;
PS(:,i,8) = Q + ex - ey;
PS(:,i,9) = PS(:,i,1);
```

Danach erfolgt die Berechnung des Gewichtsvektors $\mathbf{w}_S$ des Kontrollnetzes der Drehfläche. Die Gewichte des Kontrollpolygons der erzeugenden Kurve sind gegeben, und diese werden mit den entsprechenden Gewichten des Drehkreises multipliziert. Die Erzeugung des Drehkreises erfolgt gemäß *Kap. 1.3.8* , S. 24.

Der Gewichtungsvektor $\mathbf{w}_S$ kann mit zwei `for`-Schleifen berechnet werden. Die erste besitzt den Zähler `j = 1:2:9` und belegt den Vektor $\mathbf{w}_S$ an den ungeraden Stellen, entsprechend der ungeraden Kontrollpunkte auf der Seitenhalbierenden des Quadrates mit dem Wert eins. Die zweite Schleife besitzt den Zähler `j = 2:2:8` und belegt den Gewichtsvektor $\mathbf{w}_S$ an den geraden Stellen, entsprechend der Kontrollpunkte in den Eckpunkten des Quadrates mit dem Faktor $1/\sqrt{2}$ (siehe *Gl. 1.45* , S. 26).

Anschließend erfolgt die Definition des Knotenvektors $\mathbf{U}_S$ für eine Drehfläche. Dieser wird als Cell-Array mit zwei Zeilen initialisiert. Der erste Knotenvektor `US{1}` für die Richtung der Erzeugenden (Profilkurve), ist einfach der Knotenvektor $\mathbf{U}$ der erzeugenden Kurve. Die zweite Richtung ist die Drehrichtung des Drehkreises und besitzt den Knotenvektor eines Vollkreises:

```
US{2} = {0,0,0,0.25,0.25,0.5,0.5,0.75,0.75,1,1,1}
```

Damit sind alle Eigenschaften zur Erzeugung eines NURBS Objektes für Drehflächen definiert. Mittels Aufruf des Konstruktors der Vorgängerklasse `Surface` mit den Übergabeparametern $\mathbf{P}_S$, $\mathbf{U}_S$, $\mathbf{w}_S$ und $\mathbf{T}$ kann ein solches erzeugt werden.
Schließlich werden noch der Aufpunkt und der Richtungsvektor festgelegt:

```
obj.a = T(1:3,1:3)*a1 + T(1:3,4);
obj.d0 = T(1:3,1:3)*d1;
```

2.7.3 Methode `phi_u2`

Input: Array des Winkels: $[\varphi_1, \ldots, \varphi_\delta]$

Output: Array $\mathbf{u}_2$ des Kurvenparameters: $\mathbf{u}_2 = [u_{2,1}, \ldots, u_{2,\delta}]$

Die Methode `phi_u2` dient der Umrechnung des Winkelarguments $\varphi_i \in [0, 2\pi]$ eines Drehkreises der Parameterdarstellung (siehe *Kap. 1.24* , S. 18) in den zugehörigen Kurvenparameter $\mathbf{u}_2 \in [0, 1]$ der NURBS Parametrisierung (*Gl. 1.26* , S. 19). Diese Umrechnung wird für die Drehrichtung (u_2) der NURBS Drehfläche umgesetzt. Die erstellte Methode basiert auf einem Vollkreis, der sich aus vier Viertelkreisen zusammensetzt. Die Berechnung erfolgt anhand der in *Kap. 1.3.5* , *S.*19 erläuterten Methode.

Zunächst wird, abhängig vom Quadranten in dem φ liegt, der Winkel $\psi \in [0, \frac{\pi}{2}]$ berechnet:

$$0 \leq \varphi_i < \frac{\pi}{2} \implies \psi = \varphi_i$$

$$\frac{\pi}{2} \leq \varphi_i < \pi \implies \psi = \varphi_i - \frac{\pi}{2}$$

$$\pi \leq \varphi_i < \frac{3\pi}{2} \implies \psi = \varphi_i - \pi$$

$$\frac{3\pi}{2} \leq \varphi_i < 2\pi \implies \psi = \varphi_i - \frac{3\pi}{2}$$

Die Berechnung von $\mathbf{u}_2(i)$ erfolgt dann über *Gl. 1.26* , S. 19:

```
u2(i) =  (s2_1*(1-cs) + sn)/(s2_14*(s2_*(cs+sn)+2));
```

Hierbei werden folgende Hilfsvariablen verwendet:

```
s2_ = sqrt(2);
s2_1 = s2_-1;
s2_14= s2_1*4;
cs = cos(psi);
sn = sin(psi);
```

Der Kurvenparameter $\mathbf{u}_2(i)$ wird dann mit der Kontrollfunktion `switch` für den Quadranten zwei bis vier berechnet, je nach Quadrant wird der Wert 0.25, 0.5 bzw. 0.75 zum Kurvenparameter $\mathbf{u}_2(i)$ addiert. Somit befindet sich der berechnete Parameter im richtigen Quadranten des Vollkreises.

2.7.4 Methode `plot`

Bemerkung: Die Klasse `Revolved Suface` benötigt eine eigene `plot`-Methode, da in Drehrichtung das eingegebene Winkelargument mit der Methode `phi_u2` umgerechnet werden soll. Dies ist erforderlich da man in Drehrichtung eine äquidistante Unterteilung des eingegebene Winkelargumentes haben möchte.

Input:

- Startwinkel der Drehung: *phi0*
- Endwinkel der Drehung: *phi1*
- Anzahl der Punkte der erzeugenden Kurve (Richtung u_1): *nop_u1*
- Anzahl der Punkte in Drehrichtung (Richtung u_2): *nop_u2*
- Variable Argumente der Eigenschaften des Plots (Farbe- und Transparenz-Eigenschaften)

Zuerst wird $\mathbf{u}$ als Cell-Array mit zwei Zeilen definiert: $\mathtt{u} = \{\mathtt{u\{1\}}; \mathtt{u\{2\}}\}$

Die Berechnung von `u{1}` in Richtung der Erzeugenden erfolgt mittels der Funktion `linspace`, hierbei wird eine äquidistante Aufteilung des Parameterintervalls

$(u_{1,p+1}, \ldots, u_{1,m-p-1})$ in nop_u_1 Werte erhalten:

$$u_{1,i} = u_{1,p_1+1} + i \, \frac{u_{1,m-p_1+1} - u_{1,p_1+1}}{nop\_u_1 - 1} \qquad i = 0, \ldots, nop\_u_1 - 1 \tag{2.13}$$

Anschließend wird `u{2}` mit der Methode `phi_u2` berechnet. Eingabeargument für diese Methode ist der Vektor $[\varphi_1, \ldots, \varphi_{nop_u 2}]$. Dieser Vektor wird mit der Funktion `linspace` im Intervall von *phi0* bis *phi1* mit der Anzahl von nop_u_2 äquidistanten Punkten in Drehrichtung berechnet.

Die weiteren Berechnungsschritte erfolgen gleich wie in der Methode `plot_surface` (siehe *Kap. 2.6.6* , S. 59). In *Abb. 2.3* ist die grafische Darstellung einer allgemeinen Drehfläche ersichtlich.

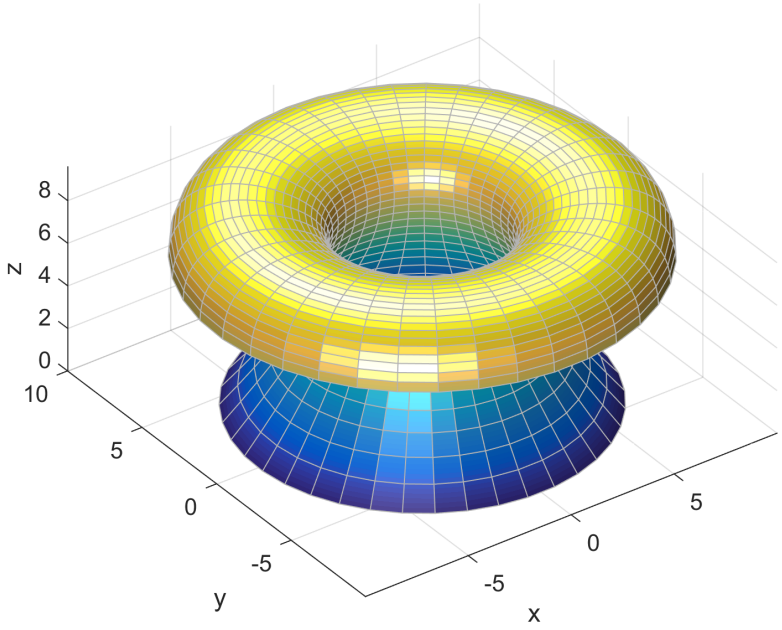

Abbildung 2.3: Allgemeinen NURBS Drehfläche.

2.8 Klassen für technisch bedeutsame Drehflächen

In diesem Kapitel werden Klassen für technisch bedeutsame Drehflächen wie

- Drehzylinder
- Drehkegel
- Torus

beschrieben. Diese Klassen werden von der Super-Klasse `Revolved Surface` für Drehflächen abgeleitet, welche selbst eine Unterklasse von `Surface` ist. Die Eigenschaften und Methoden der Klassen der technisch bedeutsamen Drehlächen, von welchen sich die Klassen ableiten, werden übernommen und um spezifische Eigenschaften und Methoden erweitert und ergänzt. Die Klassen für einen Drehzylinder (`Cylinder`), Drehkegel (`Cone`) und Torus (`Torus`) besitzen als Eigenschaften spezifische geometrische Daten, aus denen die Kontrollpunkte, Gewichte und der Knotenvektor für die Erzeugenden-Kurve bestimmt werden können. Die erzeugende Kurve eines Drehzylinders bzw. Drehkegels bzw. Torus ist eine Gerade parallel zur Drehachse bzw. eine die Drehachse schneidende Gerade bzw. ein Kreis in einer Ebene durch die Drehachse.

2.8.1 Klasse `Cylinder`

Eigenschaften der Klasse `Cylinder`

- Radius r des Drehzylinders
- z-Koordinate *h1* des Startpunktes der Erzeugenden
- z-Koordinate *h2* des Endpunktes der Erzeugenden

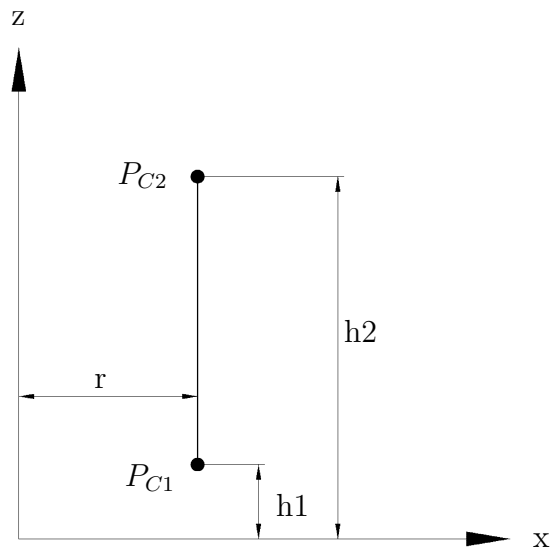

Abbildung 2.4: Bestimmungsmaße für einen Drehzylinder.

Konstruktor der Klasse `Cylinder`

Die Klasse `Cylinder` besitzt außer dem Konstruktor keine eigenen Methoden.

Input:

- Radius r des Zylinders
- z-Koordinate *h1* des Startpunktes der Erzeugenden
- z-Koordinate *h2* des Endpunktes der Erzeugenden
- Transformationsmatrix $\mathbf{T}$

Output:

- Objekt der Klasse `Cylinder`

Im Konstruktor erfolgt die Definition der geometrischen Eigenschaften der Erzeugendenstrecke des Drehzylinders (*Abb. 2.4*). Es werden die Erzeugendenstrecke $\mathbf{P}_C$, der Knotenvektor $\mathbf{U}_C$ und die Gewichte $\mathbf{w}_C$ definiert. A priori wird immer ein Drehzylinder, dessen Achse die z-Achse ist, erzeugt. Mittels einer Transformation kann jedoch der Zylinder in eine beliebige Lage gebracht werden.
Die MATLAB-Schreibweise liest sich wie folgt:

```
PC =[r, r;
     0, 0;
    h1, h2];
UC = [0,0,1,1];
wC = ones(2,1);
```

Danach erfolgt der Aufruf des Konstruktors der Vorgängerklasse `Revolved Surface`, und die nun bekannten geometrischen Daten werden als Eingabeargumente übergeben. Eine grafische Darstellung eines so erzeugten Drehzylinders ist in *Abb. 2.5* ersichtlich.

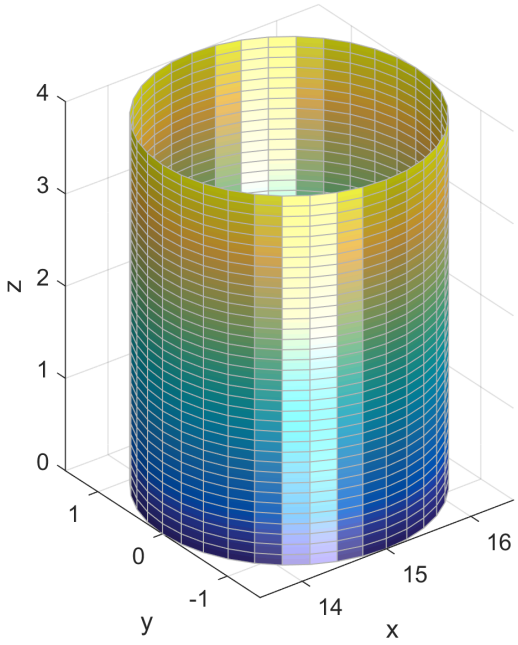

Abbildung 2.5: Drehzylinder.

2.8.2 Klasse `Cone`

Eigenschaften der Klasse `Cone`

- halber Öffnungswinkel α
- Abstand *s1* des Startpunktes der Erzeugenden vom Koordinatenursprung $O(0,0,0)$ (= Spitze des Kegels)
- Abstand *s2* des Endpunktes der Erzeugenden vom Koordinatenursprung $O(0,0,0)$

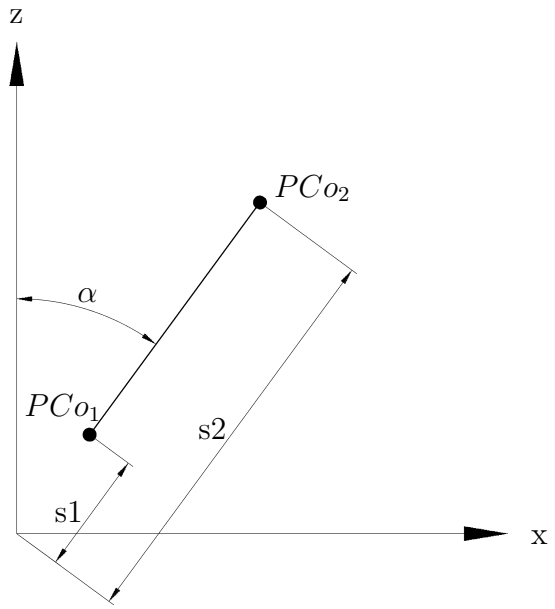

Abbildung 2.6: Bestimmungsmaße für einen Drehkegel.

Konstruktor der Klasse `Cone`

Die Klasse `Cone` besitzt außer dem Konstruktor keine eigenen Methoden.

Input:

- halber Öffnungswinkel α
- Abstand *s1* (siehe oben)
- Abstand *s2* (siehe oben)
- Transformationsmatrix $\mathbf{T}$

Output:

- Objekt der Klasse `Cone`

A priori wird immer ein Drehkegel, dessen Achse die z-Achse ist, erzeugt. Mittels einer Transformation kann jedoch der Drehkegel in eine beliebige Lage gebracht werden. Im Konstruktor erfolgt die Definition der Erzeugendenstrecke des Drehkegels (*Abb. 2.6*). Es wird die Erzeugendenstrecke $\mathbf{P}_{Co}$, der Knotenvektor $\mathbf{U}_{Co}$ und die Gewichte $\mathbf{w}_{Co}$ definiert.

Die MATLAB-Schreibweise liest sich wie folgt:

```
a1 = sin(alpha);
a2 = cos(alpha);
PCo = [a1*s1,  a1*s2;
          0,      0;
       a2*s1,  a2*s2];
UCo = [0,0,1,1];
wCo = ones(2,1);
```

Danach erfolgt der Aufruf der Vorgängerklasse `Revolved Surface`, und die nun bekannten geometrischen Daten werden als Eingabeargumente übergeben. Eine grafische Darstellung eines so erzeugten Drehkegels ist in *Abb. 2.7* ersichtlich.

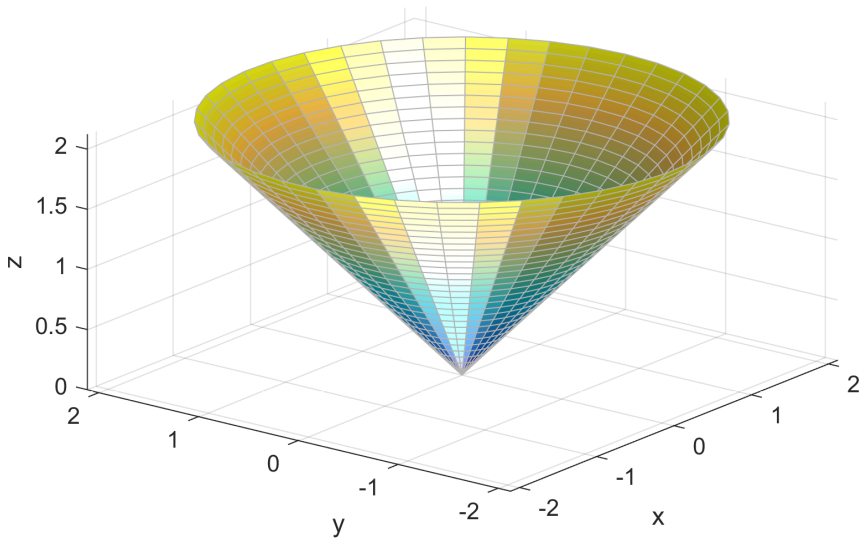

Abbildung 2.7: Drehkegel.

2.8.3 Klasse `Torus`

In der Klasse `Torus` erfolgt zusätzlich das Aufrufen der Methode `phi_u2` für den Kreis der Erzeugenden und für die Richtung der Drehung. Weiters wird in der Klasse `Torus` eine eigene `plot` -Methode implementiert.

Eigenschaften der Klasse `Torus`

- Radius *r1* des Mittenkreises
- Radius *r2* des Meridiankreises (= erzeugenden Kurve des Torus)

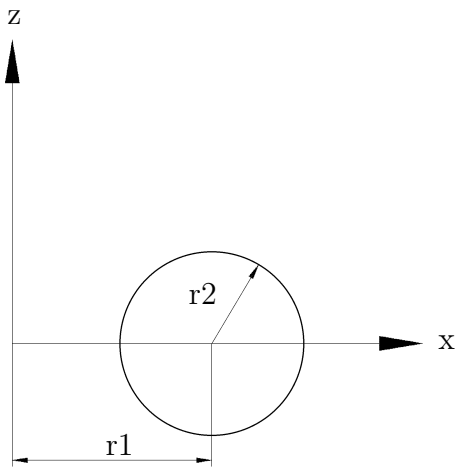

Abbildung 2.8: Bestimmungsmaße für einen Torus.

Input:

- Radius *r1* (siehe oben)
- Radius *r2* (siehe oben)
- Transformationsmatrix **T**

Output:

- Objekt der Klasse `Torus`

Konstruktor der Klasse `Torus`

Im Konstruktor erfolgt die Definition der geometrischen Eigenschaften des Meridiankreises *Abb. 2.8* . Es werden das Kontrollpolygon $\mathbf{P}_T$ dieses Kreises, der Knotenvektor $\mathbf{U}_T$ und die Gewichte $\mathbf{w}_T$ definiert. A priori wird immer ein Torus, dessen Achse die z-Achse ist, erzeugt. Mittels einer Transformation kann jedoch die Drehachse beliebig gewählt werden.

Die MATLAB-Schreibweise liest sich wie folgt:

```
r12 = r1 + r2;
r1_2r2 = r1 + 2*r2;
f = 1/sqrt(2);

PT = [ 0,      0,      0,     0,    0,     0,  0,  0,   0;
     r12, r1_2r2, r1_2r2, r1_2r2,  r12,   r1, r1, r1, r12;
       0,      0,     r2,   2*r2, 2*r2, 2*r2, r2,  0,  0];

UT = [0,0,0,0.25,0.25,0.5,0.5,0.75,0.75,1,1,1];

wT = [1, f, 1, f, 1, f, 1, f, 1];
```

Danach erfolgt der Aufruf der Vorgängerklasse `Revolved Surface`, und die nun bekannten geometrischen Daten werden als Eingabeargumente übergeben.

Methode `plot` der Klasse `Torus`

Input:

- Startwinkel der Drehung: *phi0*
- Endwinkel der Drehung: *phi1*
- Startwinkel für den erzeugenden Kreis (Meridiankreis): *phi0_lk*
- Endwinkel für den erzeugenden Kreis (Meridiankreis): *phi1_lk*
- Anzahl der Punkte in Richtung der Erzeugenden: nop_u_1
- Anzahl der Punkte in Richtung der Drehung : nop_u_2
- Variable Argumente der Eigenschaften des Plots (Farbe- und Transparenz-Eigenschaften)

Die Klasse `Torus` benötigt eine eigene `plot`-Methode, da hier in Richtung der Erzeugenden und in Richtung der Drehung die Methode `phi_u2`, für die Umrechnung des Vektors des Winkels φ in den Vektor $\mathbf{u_2}$ des Parameters angewendet werden muss. Dies ist erforderlich da man eine äquidistante Unterteilung des Meridiankreises haben möchte.

Zuerst erfolgt die Berechnung des Vektors φ in Richtung der Drehung, mit der Funktion `linspace` im Intervall von *phi0* und *phi1* mit der Anzahl von nop_u_2 Punkten. Dieser Vektor φ wird dann der Methode `phi_u2` als Eingabeargument übergeben. Das Ergebnis dieser Umrechnung mittels der Methode `phi_u2` wird das Cell-Array $u\{1\}$ gespeichert.

Danach erfolgt die Berechnung des Vektors *phit* in Richtung der Erzeugenden (entspricht dem Meridiankreis). Dies erfolgt mittels der Funktion `linspace` im Intervall von *phi0_lk* und *phi1_lk* mit der Anzahl von nop_u_1 Punkten. Dieser Vektor *phit* wird dann der Methode `phi_u2` als Eingabeargument übergeben. Das Ergebnis dieser Umrechnung wird im Cell-Array $u\{2\}$ gespeichert.

Anschließend wird mit der Methode `calc_points` die Matrix $\mathbf{X}$ berechnet. Diese Matrix wird mit der Funktion `double` in eine gewöhnliche Matrix umgewandelt. Danach wird mit der MATLAB Funktion `surf` und unter Anwendung der Funktion `squeeze`, wie auch schon in der Klasse `Revolved Surface`, die Grafik erzeugt (*Abb. 2.9*).

In *Abb. 2.10* ist als Beispiel ein Teil einer Torusfläche mit unterschiedlichen Start- und Endwinkeln für Meridian- und Drehkreis dargestellt. Beim Drehkreis wurde der Startwinkel mit π und der Endwinkel mit 2π gewählt, in Richtung des erzeugenden Kreises wurde der Startwinkel mit 0 und der Endwinkel mit $\pi/2$ gewählt.

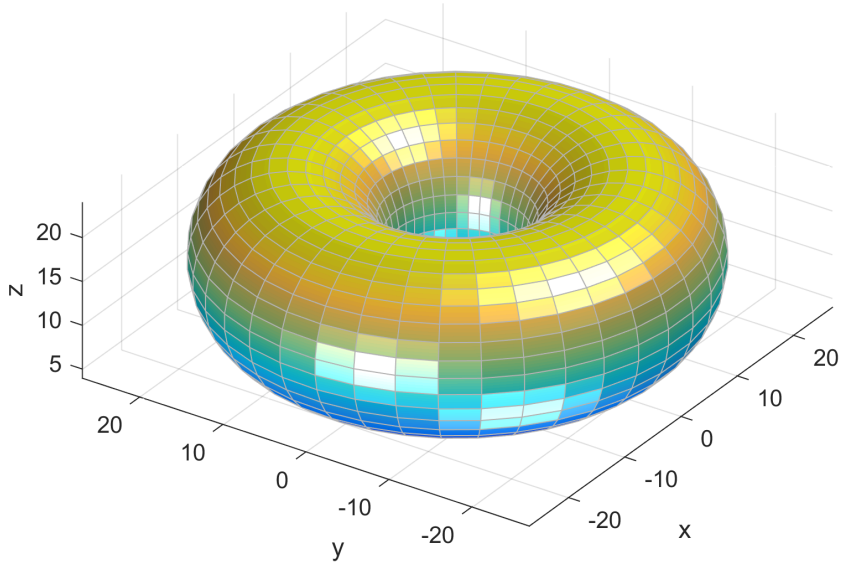

Abbildung 2.9: Torus.

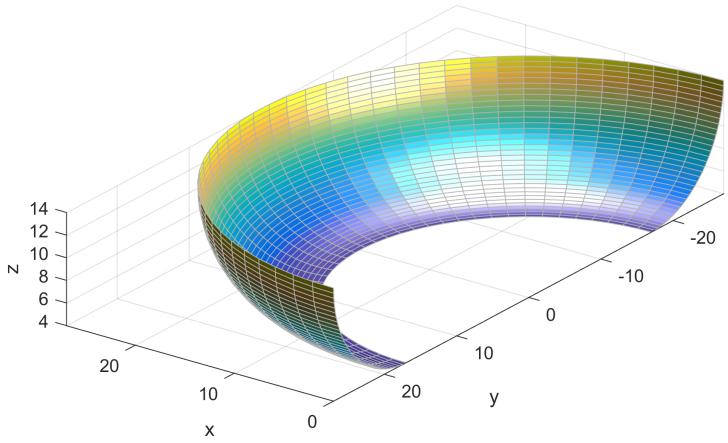

Abbildung 2.10: Teilfläche eins Kreisringtorus.

2.9 Beispiele

2.9.1 Einschaliges Drehhyperboloid

Ein einschaliges Drehhyperboloid ist jene Drehfläche, die entsteht, wenn man eine Strecke $\mathbf{P}_0\mathbf{P}_1$ um eine dazu windschiefe Drehachse rotieren lässt (*Abb. 2.11*).

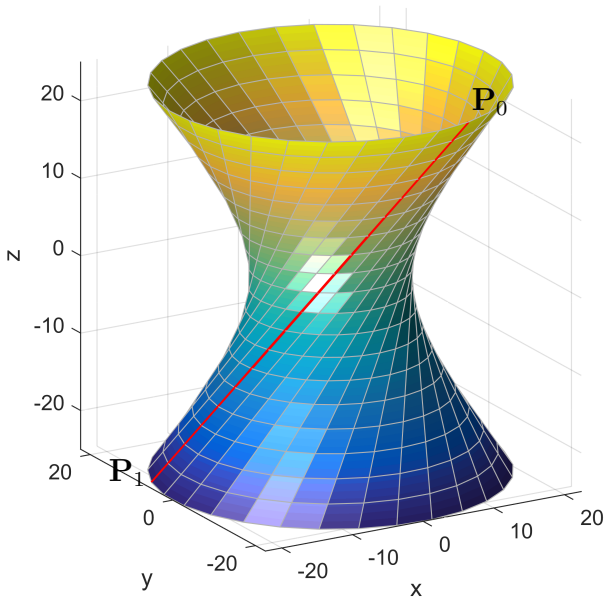

Abbildung 2.11: Einschaliges Drehhyperboloid.

2.9.2 Berechnung der Tangentialebene einer NURBS Fläche mittels der Jacobi-Matrix

Gegeben sei eine allgemeine NURBS Fläche gemäß *Gl. 1.40* , S. 23.

Zuerst wird ein Vektor $[u_1, u_2]$ definiert. Die Parameter u_1 und u_2 dieses Vektors müssen im Intervall $[u_{1_i}, u_{1_{i+p+1}})$ und $[u_{2_j}, u_{2_{j+q+1}})$ liegen (siehe *Kap. 1.3.7* , S. 23). Diese Parameter legen den Flächenpunkt fest, für den die Tangentialebene zu bestimmen ist.

Mit dem Vektor $[u_1, u_2]$ als Eingabeargument, kann nun die Methode `jacobian` aufgerufen werden. Als Output dieser Methode wird ein (3×2)-Matrix $\mathbf{J}$ erhalten, welche in den Spalten die beiden Richtungsvektoren für die u_1- und u_2- Richtung beinhaltet. Die Normierung der gesamten Matrix erfolgt spaltenweise. D.h. die erste Spalte wird durch ihre Vektornorm dividiert und analog dazu wird die zweite Spalte durch ihre Vektornorm dividiert. In MATLAB-Schreibweise kann für diesen Vorgang

```
J_n = [J(:,1)/norm(J(:,1)),J(:,2)/norm(J(:,2))]
```

geschrieben werden. Um den zugehörigen Normalenvektor $\mathbf{n}$ zu erhalten, wird das Kreuzprodukt der Spalten des normierten Vektors gebildet, in MATLAB Schreibweise:

```
n = cross(J_n(:,1),J_n(:,2))
```

Für eine grafische Darstellung wird der zugehörige Punkt auf der NURBS Fläche benötigt. Dieser Punkt wird mit der Methode `calc_points_surface` und dem zuvor definierten Vektor $[u_1, u_2]$ als Eingabeargument berechnet. Zur Erstellung einer Grafik der Ableitungsvektoren wird die MATLAB-Funktion `quiver3` verwendet. Die x-, y- und z-Koordinaten werden aus dem errechneten Punkt entnommen und die dazugehörigen Richtungsvektoren werden aus den beiden Spalten der Matrix $\mathbf{J}_n$ bzw. aus dem Vektor $\mathbf{n}$ des Kreuzproduktes entnommen.

Diese Vorgehensweise lässt sich analog auf eine Drehfläche anwenden. Bei einer Drehfläche wird jedoch der Parameter u_2 (Drehrichtung) mittels der Methode `phi_u2` mit dem Drehwinkel φ als Eingabeargument berechnet. Dieser Drehwinkel φ gibt die Position der Tangentialebene in Drehrichtung an. Die weitere Berechnung erfolgt wie oben beschrieben.

In *Abb. 2.12* ist die Darstellung der drei Vektoren am Beispiel eines Drehzylinders ersichtlich. Die Vektoren $\mathbf{j}_{u_1}$ und $\mathbf{j}_{u_2}$ der Jacobi-Matrix spannen die Tangentialebene des

Drehzylinders auf. Der Vektor des Kreuzproduktes **n**, in rot dargestellt, zeigt normal auf diese Ebene.

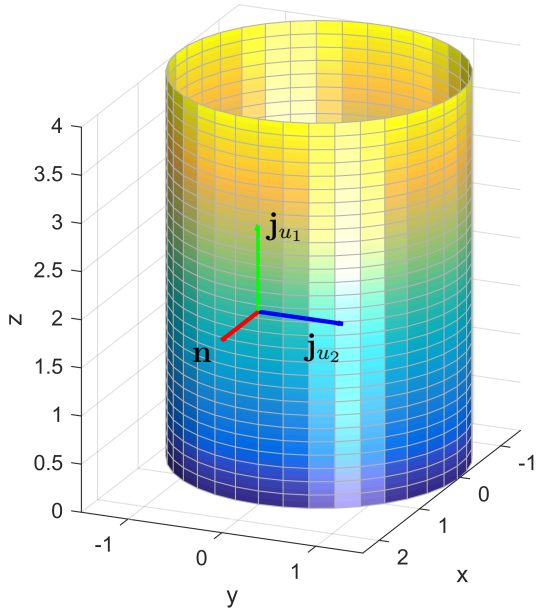

Abbildung 2.12: Jacobi-Matrix am Beispiel eines Drehzylinders.

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde eine objekt-orientierte Software-Bibliothek für Non-Uniform Rational B-Splines (NURBS) erstellt. Die Implementierung erfolgte mit dem Softwareprogramm MATLAB unter Zuhilfenahme der Schnittstelle zu C aus den External Interfaces von MATLAB.

Die Bibliothek bildet die Grundlage eines Geometriemodells, welches in Zukunft als Basis eines Finite-Elemente Modells zur Untersuchung des Schwingungsverhalten einer Stator-Wicklung von Synchrongeneratoren dienen soll.

Um für verschiedene Geometrievarianten eine möglichst übersichtliche und leicht zu wartende Basis für die Modellbildung zu erstellen, ist eine objekt-orientierte Bibliothek für Computer Aided Geometric Design (CAGD) Voraussetzung. Für die mathematische Beschreibung dieser Objekte haben sich NURBS als Standard etabliert. Daher ist ein Datenaustausch zwischen der erstellten Bibliothek und einer CAD-Software einfach realisierbar.

Der erste Teil der Arbeit behandelt die theoretischen Grundlagen, welche als Basis für die Implementierung der Software-Bibliothek dienen.
Es wurden die Eigenschaften und Definitionen von den B-Spline Basisfunktionen und deren Ableitungen beschrieben. Die grundlegenden Eigenschften von gewöhnlichen B-Spline Kurven und Flächen wurden dargestellt.
Der überwiegende Teil dieses Abschnittes beschäftigt sich mit Rationalen B-Splines (NURBS). Hier wurde die Verwendung von homogenen Koordinaten erläutert. Die Eigenschaften und Definitionen von NURBS Kurven sowie die Vorgehensweise zur Darstellung von NURBS Kurven als Kreise wurden dargestellt. Zudem erfolgte die Beschreibung der Vorgehensweise zur Erzeugung der Ableitungen von NURBS Kurven, sowie die Herleitungen der Berechnung des Winkels ψ aus dem Parameter t sowie umgekehrt die Berechnung des Parameters t aus dem Winkel ψ. Weiters wurden die Eigenschaften und Definitionen

von NURBS Flächen sowie die Erstellung einer rationalen Drehfläche als NURBS Fläche beschrieben. Außerdem erfolgte eine Herleitung der Vorgehensweise zu Erstellung der Ableitungen von NURBS Kurven.

Der zweite Teil der vorliegenden Arbeit beinhaltet die Dokumentation der erstellten Software-Bibliothek. Es wurden alle Eigenschaften, Konstruktoren und Methoden der jeweiligen Klassen detailliert erläutert. Die Bibliothek gliedert sich in eine Klasse B-Spline Basisfunktionen zur numerischen Auswertung der Basisfunktionen und deren Ableitungen und in eine Klasse NURBS zur numerischen Berechnung des Tensorproduktes aus den Matrizen der Basisfunktionen und dem Array der Kontrollstruktur. Weiters wurde eine Super-Klasse zur grafischen Beschreibung von Flächen implementiert. Von dieser wurde eine weitere Super-Klasse für Drehflächen abgeleitet und davon wiederum Spezialisierungen für technisch bedeutsame Drehflächen.

Um in weiterer Folge eine noch effizientere Implementierung von Tensor-Produkten zu erhalten, wäre die Berücksichtigung der schwachen Besetzung der Matrizen der Basisfunktionen von Vorteil. Aufgrund der objekt-orientierten Umsetzung der Software-Bibliothek, kann ein Austausch von einzelnen Klassen einfach durchgeführt werden.

Literaturverzeichnis

[1] Bader, B. W. und T. G. Kolda: *Efficient MATLAB computations with sparse and factored tensors*. SIAM Journal on Scientific Computing, S. 205–231, 2008. 41

[2] Hirz, M., A. Gfrerrer, W. Dietrich und J. Lang: *Integrated Computer-Aided Design in Automotive Development*. Springer-Verlag, 2013. viii, 3, 10, 14, 15

[3] Jankauskas, K.: *Time-Efficient NURBS Curve Evaluation Algorithms*. In: *Proceedings of the 16th International Conference on Information and Software Technologies IT2010*, S. 60–69, Kaunas, Lithuania, 21.–23. Apr. 2010. 34

[4] Piegl, L. und W. Tiller: *The NURBS Book*. Monographs in Visual Communication. Springer-Verlag, 2. Aufl., 1997. 1, 3, 6, 7, 9, 10, 11, 13, 15, 16, 23, 24, 34, 35, 37, 38

[5] Röschel, O.: *Eine Charakterisierung kinematischer rationaler Bézierflächenstücke*. In: Ludwig, M. (Hrsg.): *Beiträge aus dem Symposium Computergeometrie*, Bd. 109/90 d. Reihe *Studientexte*, S. 146–155. Technische Universität Dresden, Sektion Mathematik, Weiterbildungszentrum Computermathematik, 1990. 24

[6] Röschel, O.: *Kinematic rational Bézier patches I*. Rad HAZU, [456] 10:95–108, 1991. `http://web.math.pmf.unizg.hr/~duje/radhazumz/vol10.html` (Zugriff am 10. Sep. 2015). 24