Programmierung von CAx-Systemen

David Straub

Gliederung

1. Einführung
2. Topologie
3. **Geometrie**
4. Modellierungsstrategien
5. Datenaustausch
6. Simulation
7. Optimierung
8. Fertigung

Geometrie II: Freiformkurven und Flächen

- Motivation: Beispiel Wendelstein 7-X
- Freiformkurven (1D): Bézier, B-Spline, NURBS (inkl. Stetigkeitsbedingungen)
- Flächen (2D): analytisch, Sweep-Flächen, NURBS-Flächen

Beispiel: Plasmagefäß für Wendelstein 7-X

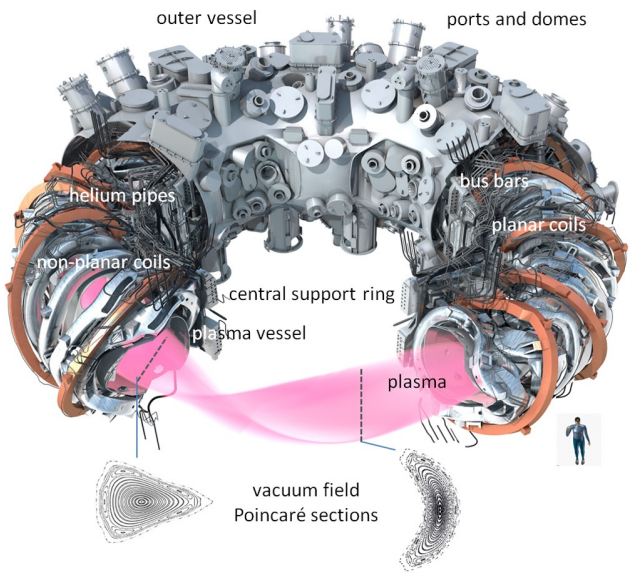
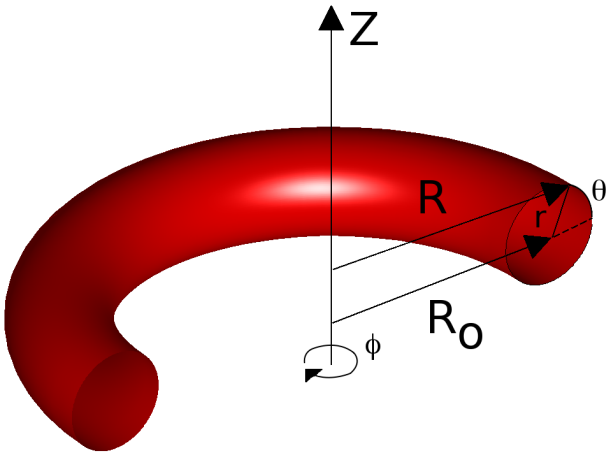


Figure 1: width:12cm

Mathematische Beschreibung der Plasmagrenzfläche

$$R(\theta,\varphi) = \sum_{m,n} R_{m,n} \cos(m\theta - n\varphi), \qquad Z(\theta,\varphi) = \sum_{m,n} Z_{m,n} \sin(m\theta - n\varphi)$$




```python
import numpy as np

R_mn = np.array([[0, 0, 5.559], [1, 0, 0.491],
    [0, 5, 0.264], [1, 5, -0.251], [2, 10, 0.064]])
Z_mn = np.array([[1, 0, 0.620], [0, 5, -0.238],
    [1, 5, 0.179], [2, 10, -0.056]])

def R(theta, phi):
    return sum(c * np.cos(m*theta - n*phi) for m, n, c in R_mn)

def Z(theta, phi):
    return sum(c * np.sin(m*theta - n*phi) for m, n, c in Z_mn)
```

Darstellung der Grenzfläche

Erzeugung der kartesischen Koordinaten

$$x = R(\theta, \varphi)\cos\varphi, \qquad y = R(\theta, \varphi)\sin\varphi, \qquad z = Z(\theta, \varphi)$$

```
n_t = n_p = 100
# n_t x n_p Arrays
theta, phi = np.meshgrid(np.linspace(0, 2*np.pi, n_t), np.linspace(0, 2*np.pi, n_p))
r = R(theta, phi)
z = Z(theta, phi)
x, y = r * np.cos(phi), r * np.sin(phi)
```

Plot mit Plotly

```
import plotly.graph_objects as go

fig = go.Figure(go.Surface(x=x, y=y, z=z, colorscale="Viridis", showscale=False))
fig.update_layout(scene=dict(aspectmode="data"))
```

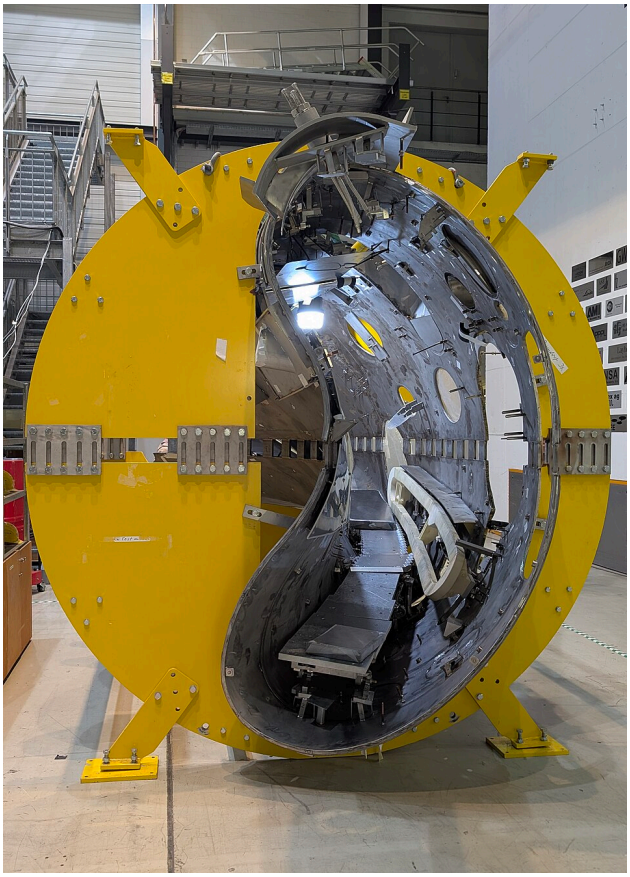
Grobes Gitter, $n_\theta = n_\varphi = 25$

Feines Gitter, $n_\theta = n_\varphi = 100$

W7-X: CAD-Geometrie des Plasmagefäßes

Vorgehen:

1. **Plasma-Querschnitt** $\varphi = \text{const}$: (R, Z)-Punkte $\rightarrow$ geschlossener Kurve (*Spline*)
2. ***Loft*** einer halben Feldperiode $\varphi \in [0, \pi/5] \rightarrow$ `Solid`
3. **Versatz** der Mantelfläche um d nach außen $\rightarrow$ Gefäßwand



Schritt 1: Querschnitt als Spline

```
import build123d as bd
from ocp_vscode import show

def make_section(phi_val, n=25):
    theta = np.linspace(0, 2*np.pi, n, endpoint=False)
    r = R(theta, phi_val);  z = Z(theta, phi_val)
    x, y = r * np.cos(phi_val), r * np.sin(phi_val)
    points = [bd.Vertex(1e3 * xi, 1e3 * yi, 1e3 * zi) for xi, yi, zi in zip(x, y, z)]
    spline = bd.Spline(*points, periodic=True)
    return points, spline

phi_values  = np.linspace(0, 2*np.pi/10, 10)
show([make_section(phi) for phi in phi_values])
```

- `periodic=True` → geschlossene Kurve ohne Knick am Anfangspunkt
- Rückgabe: `Edge` (ein einzelner Spline-Bogen)

Schritt 2: Loft einer halben Feldperiode

W7-X hat **5-fache toroidale Symmetrie** mit Spiegelsymmetrie pro Periode

→ Halbe Periode = $2\pi/10 = 36°$

```
phi_values  = np.linspace(0, 2*np.pi/10, 20)
sections    = [bd.Wire([make_section(phi)[1]]) for phi in phi_values]
plasma = bd.Solid.make_loft(sections)
show(plasma)
```

- `sections`: Liste von `Wire`-Objekten (hier: je ein Spline)
- `make_loft` → erzeugt eine durch die `sections` verlaufende Fläche (*Loft*)
- `[e.geom_type for e in plasma.faces()]` → `[BSPLINE, PLANE, PLANE]`

Schritt 3: Gefäßwand mit `offset_3d`

```
# Planare Deckelflächen als "Öffnungen" → offset_3d erzeugt eine Schale
caps = plasma.faces().filter_by(bd.GeomType.PLANE)
wall = plasma.offset_3d(caps, 200)

# Nur die äußere Versatzfläche behalten
vessel_face = wall.faces().filter_by(bd.GeomType.OFFSET)
show(plasma, vessel_face, names=["Plasmaoberfläche", "Gefäßwand"])
```

- `offset_3d(openings, amount)` → hält die `openings`-Flächen offen, versetzt alle anderen um `amount`
- `[e.geom_type for e in wall.faces()]` → `[BSPLINE, PLANE, OFFSET, PLANE]`
- `.filter_by(GeomType.OFFSET)` → selektiert die neu entstandene Außenfläche

Freiformkurven

Motivation: Grenzen analytischer Kurven

Wie beschreibt man eine freie Kurve mathematisch?

Analytische Kurven aus Teil 1 (Linie, Kreis, Ellipse, Kegelschnitte) haben eine **feste Form** – ihre Gleichung bestimmt den Kurvenverlauf vollständig, freie Formgebung ist nicht möglich.

Das W7-X-Beispiel vom Beginn dieser Vorlesung illustriert das: Der Querschnitt des Plasmabehälters lässt sich durch keinen analytischen Kurventyp ausdrücken.

→ Wir brauchen **Freiformkurven**: flexibel formbar, lokal steuerbar, für beliebige Geometrie geeignet.

Wiederholung: Parametrische Kurven

Eine **parametrische Kurve** bildet einen Parameter u auf einen Punkt im Raum ab:

$$C(u) = \begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix}, \qquad u \in [u_0,\, u_1]$$

Analytische Kurve	Parametrisierung
Linie	$C(u) = P_0 + u\,d$
Kreis (Radius r)	$x = r\cos u, \quad y = r\sin u$
Ellipse	$x = a\cos u, \quad y = b\sin u$

Vorteile der Parameterdarstellung: - Punkte, Tangenten, Krümmung durch einfache Ableitungen $C'(u)$, $C''(u)$ - Einfach zu verkürzen und einen Punkt auf der Kurve zu finden

Warum Polynome?

Eine Kurve ist eine Funktion $C(u) = (x(u),\, y(u),\, z(u))$ – aber welche Funktion wählt man?

Ansatz	Problem
Lineare Interpolation	**Knicke** an jedem Stützpunkt
Sinus, Kosinus	schwer zu verketten, teuer auszuwerten
Interpolationspolynome	Runge-Phänomen: Oszillationen bei vielen Punkten
Polynome niedrigen Grades	einfach, schnell, stabil – aber begrenzte Form
Stückweise Polynome	flexibel, stabil, lokal kontrollierbar

→ Alle Freiformkurven in CAD (Bézier, B-Spline, NURBS) basieren auf **stückweisen Polynomen niedrigen Grades**.

Kurven in Potenzbasis

Der naheliegende Ansatz: Koordinaten als Polynome in u,

$$C(u) = \sum_{i=0}^{n} a_i \, u^i = a_0 + a_1 u + a_2 u^2 + \cdots + a_n u^n$$

Probleme der Potenzbasis:

- **Kein geometrischer Bezug:** Die Koeffizienten a_i haben keine anschauliche Bedeutung – man kann den Kurvenverlauf nicht aus ihnen ablesen
- **Numerische Empfindlichkeit:** Bei hohen Graden reagiert die Kurvenform sehr sensibel auf kleine Änderungen einzelner Koeffizienten
- **Schlechte Interaktivität:** Um eine Kurve zu ändern, muss man ein Gleichungssystem lösen

→ Gesucht: eine Basis, deren Parameter **direkt geometrische Bedeutung** haben.

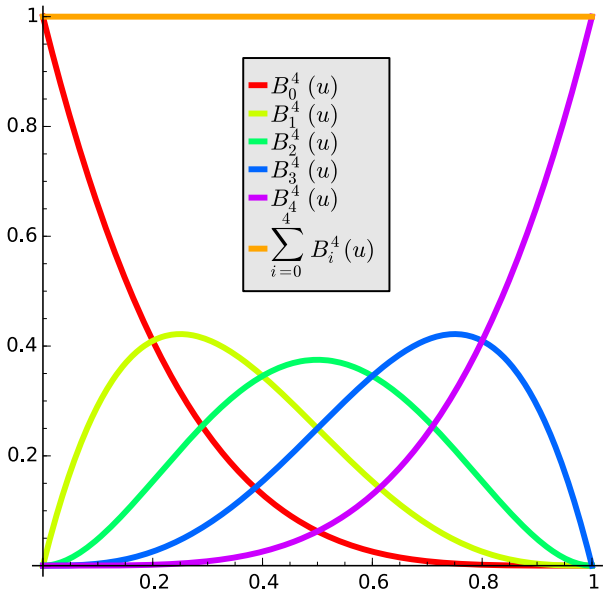
Pierre Bézier

Anfang der **1960er Jahre** bei **Renault**:

- Ziel: Karosserie-Design am Computer
- 1962 publiziert
- CAD-System: **UNISURF** (1968)

Idee: Kontrollpunkte, die die Kurve *anziehen* – geometrisch intuitiv, numerisch stabil.

Bézier-Kurven: Formel

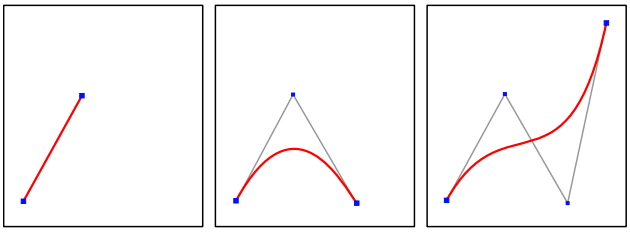


$$C(u) = \sum_{i=0}^{n} B_{i,n}(u)\, P_i, \qquad B_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}, \quad u \in [0,1]$$

- P_i: **Kontrollpunkte** (bilden das Kontrollpolygon)
- $B_{i,n}(u)$: **Bernstein-Basisfunktionen** – nicht-negativ, $\sum_i B_{i,n}(u) = 1$
- Grad n = Anzahl Kontrollpunkte -1

Geometrische Interpretation: $C(u)$ ist stets eine **Konvexkombination** der Kontrollpunkte → die Kurve liegt immer innerhalb der konvexen Hülle des Kontrollpolygons.

Bézier-Kurven: Grade 1, 2 und 3



Grad	Kontrollpunkte	Form
$n = 1$	P_0, P_1	gerade Strecke
$n = 2$	P_0, P_1, P_2	Parabel
$n = 3$	$P_0, \dots, P_3$	kubische Kurve

Eigenschaften: - Kurve läuft durch P_0 und P_n - Tangente in P_0: Richtung $P_1 - P_0$ - Kurve liegt in der **konvexen Hülle** der Kontrollpunkte

Interaktives Applet (GeoGebra)

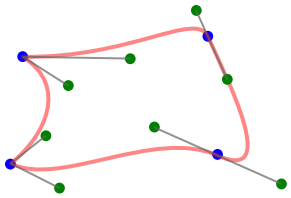
Bézier: Warum reicht das nicht für CAD?

> **Bézier ist ein konzeptioneller Zwischenschritt** – elegant und lehrreich, für reale CAD-Geometrie aber nicht ausreichend.

- Polynomgrad wächst mit Anzahl der Kontrollpunkte: 20 Punkte → Grad 19 → numerisch instabil
- Jeder Kontrollpunkt beeinflusst die **gesamte** Kurve – keine lokale Kontrolle

Erster Schritt zur Lösung: Kurve in **mehrere Segmente** aufteilen – jedes ein Bézier-Stück niedrigen Grades.

→ Aber wie glatt müssen diese Verbindungen sein?

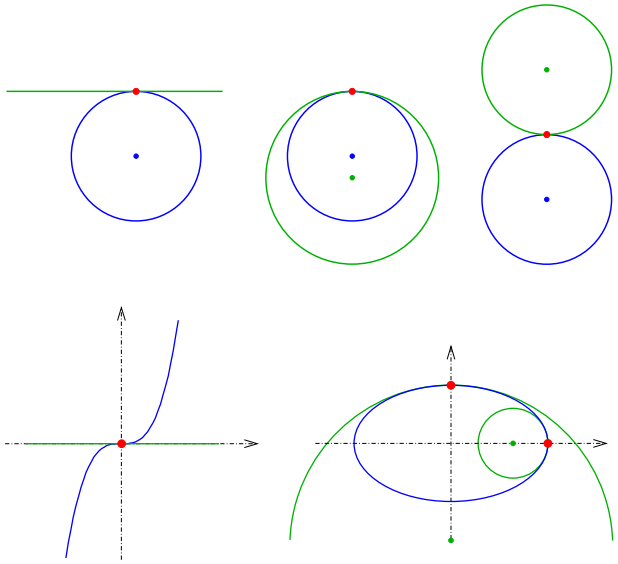


Stetigkeitsbedingungen

Beim Verbinden mehrerer Kurvenabschnitte bestimmt die **Anschlussbedingung** die wahrgenommene Qualität:

C^k: k-te Ableitung von $C(u)$ stetig (abh. von Parametrisierung) G^k: geometrisch stetig, unabh. von Parametrisierung

Grad	Bedingung	Effekt
C^0	kein Spalt, keine Überlappung	Knick sichtbar
C^1	gleiche Tangente (Richtung + Betrag)	glatter Übergang
G^1	gleiche Tangentenrichtung (Betrag egal)	glatt, flexibler als C^1
C^2 / G^2	gleiche Krümmung	reflexionsglatt



Stetigkeitsbedingungen: Warum C² wichtig ist

C^1 (glatte Tangente) klingt ausreichend – reicht aber oft nicht:

Anwendung	Warum C^2 / G^2 nötig?
Optik / Design	Lichtreflexion folgt der Krümmung – ein Sprung ist sichtbar
Fertigung	Fräserbahn folgt der Krümmung – Krümmungssprung → Maßabweichung
Fluiddynamik	Druckgradient und Grenzschicht reagieren auf Krümmungsänderungen

Typische Anforderungen in der Praxis: - C^0: Boolesche Operationen (scharfe Kante erwünscht) - G^1: `fillet` / Verrundung, Standard-Übergänge - G^2: Karosseriedesign, aerodynamische Flächen, Medizinprodukte

Stückweise Bézier – warum nicht?

Mehrere Bézier-Segmente mit C^2 verbinden: geht, kostet aber viele Bedingungen.

Bei zwei kubischen Bézier-Stücken $a(u)$ und $b(u)$ am Verbindungspunkt:

Bedingung	Gleichung	Eingeschränkte Freiheitsgrade
C^0	$a_3 = b_0$	1 Punkt fixiert
C^1	$a_3 - a_2 = b_1 - b_0$	weiterer Punkt fixiert
C^2	Krümmungsgleichheit	noch ein Punkt fixiert

Interaktives Applet (GeoGebra)

→ Bei C^2 sind an jedem Verbindungspunkt **3 von 4 Kontrollpunkten** abhängig – bei vielen Segmenten ineffizient.

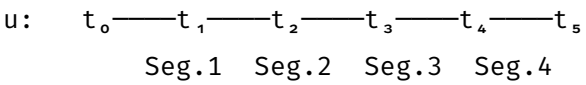
Gesucht: C^2-Stetigkeit **automatisch**, ohne Gleichungssystem.

B-Spline-Kurven: Grundidee

Eine einzige Formel für die gesamte Kurve – kein separates Stück pro Segment:

$$C(u) = \sum_{i=0}^{n} P_i \, N_{i,k}(u)$$

Der **Knotenvektor** $T = [t_0, \ldots, t_{n+k}]$ teilt u in Polynomstücke auf:



- $n + 1$ Kontrollpunkte, Ordnung k (= Grad + 1), z. B. $k = 4$ für kubisch
- Die Basisfunktionen $N_{i,k}(u)$ sorgen automatisch für Stetigkeit an den Knoten

B-Spline-Basisfunktionen: Grad 0 und 1

Grad 0 – Startpunkt der Rekursion, Rechteckfunktion:

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

Grad 1 – explizit durch Einsetzen in die Rekursion:

$$N_{i,1}(u) = \begin{cases} \dfrac{u - t_i}{t_{i+1} - t_i} & t_i \leq u < t_{i+1} \\ \dfrac{t_{i+2} - u}{t_{i+2} - t_{i+1}} & t_{i+1} \leq u < t_{i+2} \\ 0 & \text{sonst} \end{cases}$$

Linear ansteigend, dann linear abfallend – eine **Hutfunktion** über zwei Spannen, C^0 an den Knoten.

B-Spline-Basisfunktionen: höhere Grade (Cox–de Boor)

Höhere Grade entstehen durch **Rekursion** über den Grad:

$$N_{i,k}(u) = \frac{u - t_i}{t_{i+k-1} - t_i} \, N_{i,k-1}(u) \; + \; \frac{t_{i+k} - u}{t_{i+k} - t_{i+1}} \, N_{i+1,k-1}(u)$$

Jede Stufe: **eine Spannen breiterer Träger**, eine Stufe mehr Stetigkeit:

Grad	Ordnung k	Träger von $N_{i,k}$	Stetigkeit an Knoten
0 (konstant)	1	1 Spanne	unstetig (Rechteck)
1 (linear)	2	2 Spannen	C^0 – Knick
2 (quadratisch)	3	3 Spannen	C^1 – glatt
3 (kubisch)	4	4 Spannen	C^2 – krümmungsstetig

→ Kubische B-Splines ($k = 4$) sind der **CAD-Standard**: C^2 ist eingebaut, kein Gleichungssystem nötig.

B-Spline: lokaler Träger = lokale Kontrolle

Da $N_{i,k}$ nur auf k Spannen ungleich null ist, beeinflusst P_i **nur diesen Bereich** der Kurve:

	Stückw. Bézier	B-Spline
Stetigkeit	explizit erzwungen	**eingebaut** in $N_{i,k}$
Modifikation	global pro Segment	**lokal** – nur k Spannen
Grad	fest (z. B. 3)	frei wählbar, unabh. von n

B-Spline: interaktive Visualisierung

Eine **einzelne kubische Basisfunktion** $N_{i,3}(u)$ (Grad 3, Ordnung $k = 4$) als Funktion von u:

- Träger: **5 Knoten** $a_0 < a_1 < a_2 < a_3 < a_4$ → **4 Spannen** (k Spannen, da Ordnung $k = 4$)
- Knotenabstände $\delta_i = a_{i+1} - a_i$ sind verschiebbar → Form der Basisfunktion ändert sich
- Bei gleichmäßigen Knoten ($\delta_i = \text{const}$): symmetrische Glockenform

  Analogie: $N_{i,3}(u)$ ist das B-Spline-Äquivalent zur Bernstein-Basisfunktion $B_{i,n}(u)$ bei Bézier – aber mit **lokalem Träger** statt globalem.

Interaktives Applet (GeoGebra)

Knotenvielfachheit

Ein Knoten darf im Knotenvektor **mehrfach** vorkommen. Die Vielfachheit k steuert die Stetigkeit an diesem Knoten:

Vielfachheit k	Stetigkeit	Effekt
1	C^{p-1}	Standard – glatt (z. B. C^2 für kubisch)
$p-1$	C^1	nur noch tangensstetig
p	C^0	Knick – wie Bézier-Verbindungspunkt
$p+1$	–	Kurve läuft durch Kontrollpunkt

Anwendungen: - **Anfangs-/Endknoten:** Vielfachheit $p+1 \rightarrow$ Kurve startet/endet exakt im ersten/letzten Kontrollpunkt (*clamped* B-Spline) - **Scharfe Kante:** lokale Vielfachheit $p \rightarrow$ gezielter Knick in der Kurve - **Bézier-Extraktion:** Vielfachheit p an allen inneren Knoten $\rightarrow$ stückweise Bézier-Darstellung

NURBS: Warum?

Kann ein B-Spline einen perfekten Kreis darstellen?

Antwort: **Nein** – B-Splines sind stückweise Polynome, Kreise sind trigonometrisch. Ein B-Spline kann einen Kreis nur **annähern**, nie exakt darstellen.

Das zeigt: B-Splines sind **nicht allgemein genug** für alle Kurvenformen.

Hinweis: OCCT speichert Kreise intern trotzdem als `CIRCLE` (analytisch, exakt) – nicht als B-Spline. Aber: CAD-Systeme, die intern *nur* eine Kurvenart verwenden, brauchen eine Darstellung, die auch Kreise exakt abdeckt.

$\rightarrow$ Lösung: **Gewichte** hinzufügen $\rightarrow$ NURBS kann Kreise, Kegelschnitte und Freiformkurven **in einem Format** darstellen.

Warum kein Polynom für den Kreis?

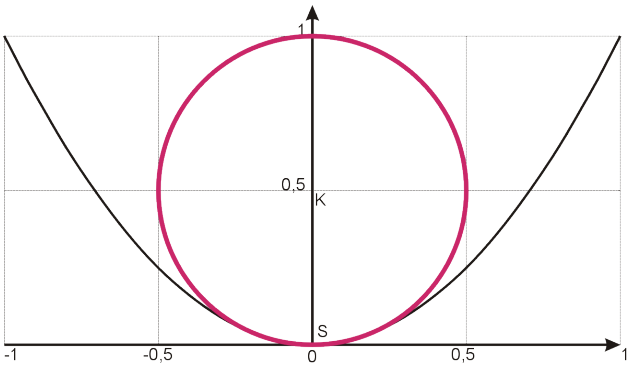
Ein B-Spline ist stückweise **polynomial**:

$$x(t)^2 + y(t)^2 = r^2 \quad \forall t$$

Sind $x(t)$, $y(t)$ Polynome vom Grad p, so ist $x(t)^2 + y(t)^2$ ein Polynom vom Grad $2p$.

Dieses kann nur dann für **alle** t konstant gleich r^2 sein, wenn alle nicht-konstanten Terme verschwinden – d. h. x und y wären konstant. **Widerspruch.**

Ein B-Spline kann einen Kreis nur **annähern**, niemals exakt darstellen.



NURBS: rationale Parametrisierung des Kreises

Rationale Funktionen (Quotient zweier Polynome) umgehen dieses Problem:

$$x(t) = \frac{1-t^2}{1+t^2}, \qquad y(t) = \frac{2t}{1+t^2} \quad \Rightarrow \quad x^2 + y^2 = 1 \checkmark$$

Viertelkreis als quadratische NURBS (exakt):

Kontrollpunkt	$P_0 = (1,0)$	$P_1 = (1,1)$	$P_2 = (0,1)$
Gewicht h_i	1	$\frac{\sqrt{2}}{2} \approx 0{,}707$	1

Das reduzierte Gewicht $h_1 < 1$ „zieht" die Kurve vom Kontrollpunkt weg – genau so, dass der Kreisbogen entsteht.

NURBS – Non-Uniform Rational B-Splines

Erweiterung des B-Spline um **Gewichte** $h_i > 0$:

$$C(u) = \frac{\sum_{i=0}^{n} h_i \, P_i \, N_{i,k}(u)}{\sum_{i=0}^{n} h_i \, N_{i,k}(u)}$$

- $h_i = 1$ für alle $i \rightarrow$ normaler B-Spline (Spezialfall)
- Größeres $h_i \rightarrow$ Kurve nähert sich stärker dem Kontrollpunkt P_i

Entscheidender Vorteil:

B-Spline kann Kreise und Kegelschnitte nur **annähern**. NURBS stellt sie **exakt** dar.

NURBS: die universelle Darstellung

Linie / Kreis Bézier B-Spline NURBS

Alle analytischen Kurven sind Spezialfälle von NURBS – daher ist NURBS der **einheitliche Standard** in industriellen CAD-Systemen (CATIA, SolidWorks, NX, OCCT).

`geom_type` in OCCT	Bedeutung
`LINE`, `CIRCLE`, `ELLIPSE`	analytisch gespeichert (exakt)
`BSPLINE`	NURBS-Kurve (Freiformkurve oder konvertierte Analytik)

Beim Import aus STEP: Freiformkurven erscheinen als `BSPLINE`, auch wenn sie ursprünglich Kreise waren – je nach exportierendem System.

Übung: Spline vs. Polyline

Datei: `praktikum_stetigkeit.py` – Aufgabe 1

Erzeuge dieselben 5 Punkte einmal mit `Spline` und einmal mit `Polyline`. Gib `geom_type` aller Kanten aus:

```
punkte = [(0,0,0),(10,15,0),(25,5,0),(40,20,0),(50,0,0)]
with bd.BuildLine() as bl_spline:
    bd.Spline(*punkte)
with bd.BuildLine() as bl_poly:
    bd.Polyline(*punkte)
print([e.geom_type for e in bl_spline.edges()])
print([e.geom_type for e in bl_poly.edges()])
```

Füge danach `tangents=[(1,0,0),(0,-1,0)]` hinzu – ändert sich der `geom_type`?

Übung: `fillet` als Flächenerzeuger

Datei: `praktikum_stetigkeit.py` – Aufgabe 2

```python
box = bd.Box(50, 30, 20)
box_f = bd.fillet(box.edges(), 3)
print(set(f.geom_type for f in box.faces()))
print(set(f.geom_type for f in box_f.faces()))
```

Welche `geom_type`-Werte erscheinen neu? Warum auch `SPHERE`?

Flächen

Parametrische Flächen: Wiederholung

Jede Fläche ist eine Funktion von zwei Parametern u und v:

$$S(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}, \qquad u \in [u_0, u_1], \quad v \in [v_0, v_1]$$

Jede `Face` in einem B-Rep-Modell trägt eine solche parametrische Fläche.

Analytische Flächen

Exakt durch eine geschlossene Formel beschreibbar:

Typ	Charakteristik	`geom_type`	build123d
Ebene	konstante Normale	`PLANE`	`Box`, `Rectangle`
Zylinder	$S(u,v) = C + R\cos u\, e_1 + R\sin u\, e_2 + v\, e_3$	`CYLINDER`	`Cylinder`
Kegel	wie Zylinder, $R = R(v)$	`CONE`	`Cone`
Kugel	$R(\cos u \cos v, \sin u \cos v, \sin v)$	`SPHERE`	`Sphere`
Torus	Kreisring um Achse	`TORUS`	`Torus`

Für Standardkörper reichen analytische Flächen vollständig aus.

Sweep-Flächen

Entstehen durch Bewegung eines **Profils** entlang einer **Leitkurve**:

Typ	Beispiel	`geom_type` (Kreis-Profil)	`geom_type` (Freiform)
Extrusion	Wandfläche	`CYLINDER`	`BSPLINE`
Rotation	Kegelfläche	`SPHERE` / `TORUS`	`BSPLINE`
Sweep	Rohr, Schiene	`CYLINDER`	`BSPLINE`

Typ	Beispiel	`geom_type` (Kreis-Profil)	`geom_type` (Freiform)
Loft	Tragflügel, Rumpf	`BSPLINE`	`BSPLINE`

Das Eingangsprofil bestimmt die Komplexität der Ausgabefläche.

NURBS-Flächen

Verallgemeinerung der NURBS-Kurve auf zwei Parameter:

$$S(u,v) = \frac{\sum_i \sum_j h_{ij}\, P_{ij}\, N_{i,p}(u)\, N_{j,q}(v)}{\sum_i \sum_j h_{ij}\, N_{i,p}(u)\, N_{j,q}(v)}$$

- **Kontrollpunktnetz** P_{ij}: 2D-Gitter statt 1D-Liste
- Spezialfall $h_{ij} = 1$: Nenner $= 1$ (da $\sum_i N_{i,p}(u) = 1$) $\rightarrow$ **B-Spline-Fläche**
- In OCCT: `geom_type == 'BSPLINE'` gilt für beide Fälle – B-Spline und NURBS sind dieselbe Klasse

Anwendungen: Karosserie, Strömungsflächen, Tragflügel, Medizinprodukte, …

Stetigkeitsbedingungen bei Flächen

Dieselben Bedingungen wie bei Kurven, jetzt für aneinanderstoßende Flächen:

Grad	Bedingung	Sichtbar als
G^0	kein Spalt	scharfe Kante
G^1	Normalenvektoren stetig (gleiche Tangentenebene)	Kante verschwindet im Licht
G^2	Krümmung stetig	Reflexion läuft nahtlos durch

In build123d: - `fillet(edges, radius)` $\rightarrow$ G^1 an der Kante - Verbindungsflächen zwischen Loft-Profilen $\rightarrow$ je nach Einstellung G^0 bis G^2

W7-X: Die `BSPLINE`-Mantelfläche und die `OFFSET`-Gefäßwand stoßen an den Deckeln auf `PLANE`-Flächen – dort ist nur G^0 garantiert (scharfe Kante).

Übung: Loft → NURBS-Fläche

Datei: `praktikum_stetigkeit.py` – Aufgabe 3

Lofte von einem Kreis (unten) auf ein Rechteck (oben):

```
kreis = bd.Wire([bd.Edge.make_circle(10)])
# ... Rechteck in 40 mm Höhe ...
loft = bd.Solid.make_loft([kreis, rechteck])
print(set(f.geom_type for f in loft.faces()))
```

Warum erscheint kein `CIRCLE` in den Flächen?

Zusammenfassung

Kernkonzepte dieser Vorlesung

Freiformkurven:

Bézier (global) → **B-Spline** (lokal) → **NURBS** (+ Gewichte, universell)

- NURBS ist der Standard: kann Linien, Kreise und Freiformkurven **in einem Format** darstellen
- Lokale Modifikation: Kontrollpunkt beeinflusst nur benachbarte Segmente

Stetigkeitsbedingungen: C^0 (kein Spalt) → G^1 (glatt) → G^2 (reflexionsglatt)

Flächen: - Analytisch: Ebene, Zylinder, Kegel, Kugel, Torus - Sweep-Flächen: Extrusion, Rotation, Sweep, Loft - NURBS-Flächen: universelle Freiformflächen

Ausblick: Modellierungsstrategien

In der nächsten Einheit: **Wie baut man komplexe Modelle systematisch auf?**

- Parametrische Konstruktionsstrategie (Feature-Baum)
- Boolesche Operationen und ihre Reihenfolge
- Robuste Selektoren und parametrische Abhängigkeiten
- Wiederverwendbare Komponenten und Baugruppen