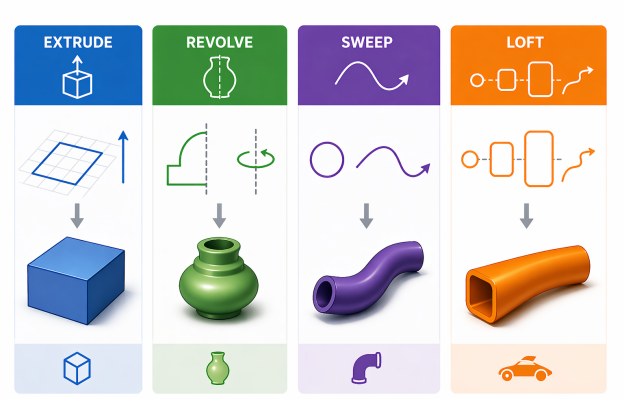
Programmierung von CAx-Systemen

David Straub

Gliederung

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



Modellierungsstrategien II

- **Sweep:** Querschnitt entlang Pfadkurve
- **Loft:** Interpolation zwischen mehreren Profilen
- **Körperoperationen:** Boolean, fillet/chamfer, mirror

Sweep – Querschnitt entlang Pfad

Sweep-Operation

Sweep = Querschnitt-Profil wird entlang einer Pfadkurve bewegt

```
bd.sweep(sections, path, multisection=False) → bd.Part
```

Parameter	Bedeutung
`sections`	Ein oder mehrere Querschnittsprofile (`Face` oder `Wire`)
`path`	Pfadkurve (`Edge` oder `Wire`), entlang der gesweept wird
`multisection`	`True` → interpoliert zwischen mehreren Profilen entlang des Pfads

Typischer Anwendungsfall: Rohre, Kabel, Kühlkanäle, Profile mit konstantem Querschnitt entlang gekrümmter Bahn

Pfadkurven – Wire und Spline

Jede **Kantenfolge** (Wire) kann als Sweep-Pfad dienen:

```
# Gerader Pfad
pfad = bd.Polyline((0, 0, 0), (100, 0, 0))

# Gekrümmter Pfad aus Spline
punkte = [(0, 0, 0), (30, 20, 10), (60, 15, 25), (100, 0, 30)]
pfad = bd.Spline(*punkte)

# Kreisbogen
pfad = bd.RadiusArc((0, 0, 0), (50, 50, 0), 60)
```

Spline: glatte Kurve durch gegebene Punkte – ideale Wahl für organische Bahnen

Querschnitt auf Pfad ausrichten

Der Querschnitt muss senkrecht zur Pfadtangente am Start liegen – `location_at(0)` liefert genau dieses Koordinatensystem:

```
pfad = bd.Spline((0, 0, 0), (50, 30, 20), (100, 0, 40))
querschnitt = pfad.location_at(0) * bd.Circle(radius=5)
rohr = bd.sweep(querschnitt, pfad)
```

Wandstärke bei Rohren: - **Option 1:** Zwei Sweeps (außen – innen) - **Option 2:** Ein Sweep + `offset()` der Innenfläche - **Option 3:** Bool'sche Subtraktion eines dünneren Rohrs

Sweep – Übergänge an Knicken (transition)

Bei Pfaden mit scharfen Ecken bestimmt `transition`, wie der Querschnitt an Knickstellen behandelt wird:

```
# Pfad mit zwei scharfen Knicken
pfad = bd.Polyline((0,0,0), (40,0,0), (50,50,0), (100,50,0))
q = pfad.location_at(0) * bd.Rectangle(10, 6)

rohr_t = bd.sweep(q, pfad, transition=bd.Transition.TRANSFORMED)  # Standard
rohr_r = bd.sweep(q, pfad, transition=bd.Transition.RIGHT)         # scharf rechtwinklig
rohr_o = bd.sweep(q, pfad, transition=bd.Transition.ROUND)         # abgerundet
```

Wert	Verhalten
`TRANSFORMED`	Querschnitt schräg geschnitten (Gehrung)
`RIGHT`	Scharfer rechtwinkliger Übergang
`ROUND`	Abgerundeter Übergang

location_at() – Punkt auf Pfad

`pfad.location_at(t)` liefert ein **lokales Koordinatensystem (Location)** an der Position `t` (0 = Start, 1 = Ende) des Pfads – inklusive Tangentenrichtung:

```
pfad = bd.Spline((0, 0, 0), (50, 30, 20), (100, 0, 40))

pfad.location_at(0)    # Startpunkt
pfad.location_at(0.5)  # Mittelpunkt
```

```
pfad.location_at(1)    # Endpunkt
```

Nützlich zum **Platzieren von Objekten** entlang des Pfads:

```
# Markierungen alle 25% entlang des Pfads
marker = [pfad.location_at(t) * bd.Sphere(2) for t in [0, 0.25, 0.5, 0.75, 1.0]]
```

Hinweis: In build123d findet sich auch die Kurzschreibweise `pfad ^ t` als Alias für `location_at(t)`.

Übung 1: Kühlkanal im Batteriepack

Nutzen Sie die zylindrischen Batteriezellen (Format **4680**) im **Sechseckraster** (HexLocations) aus der vorherigen Einheit (Übung 2.3).

Ihre Aufgabe: Modellieren Sie einen passenden **Serpentinen-Kühlkanal** (`Sweep`), der zwischen den Zellen hindurch verläuft, und integrieren Sie das Ganze in ein einfaches Gehäuse.

(Details und Parameter besprechen wir gemeinsam!)

Loft – Interpolation zwischen Profilen

Loft-Operation

Loft = Körper wird zwischen mehreren Querschnitten interpoliert

```
bd.loft(sections, ruled=False) → bd.Part
```

Parameter	Bedeutung
`sections`	Liste von Profilen (`Face` oder `Wire`) in verschiedenen Ebenen
`ruled`	`True` → lineare Interpolation (Regelfläche), `False` → glatt (Spline)

Anwendung: Übergang zwischen unterschiedlichen Querschnitten, aerodynamische Profile, Gehäuseformen

Profile auf verschiedenen Ebenen

Jedes Profil muss auf seiner eigenen Ebene definiert werden:

```
# Profil 1: Rechteck unten (z = 0)
p1 = bd.Plane.XY * bd.Rectangle(60, 40)

# Profil 2: Oval in der Mitte (z = 50)
p2 = bd.Plane.XY.offset(50) * bd.Ellipse(40, 25)

# Profil 3: Kreis oben (z = 100)
p3 = bd.Plane.XY.offset(100) * bd.Circle(20)

koerper = bd.loft([p1, p2, p3])
```

→ Glatter Übergang von Rechteck über Ellipse zu Kreis

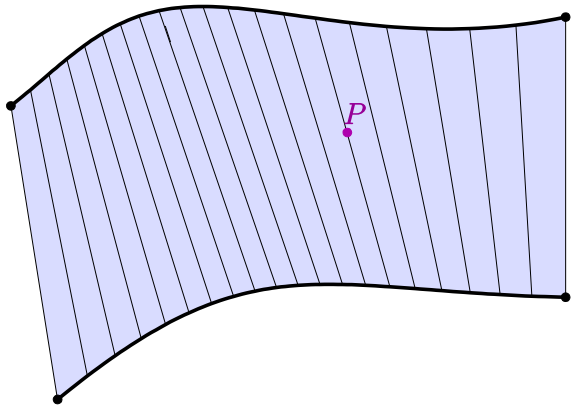
Regelflächen

Eine **Regelfläche** (*ruled surface*) entsteht, wenn jeder Punkt der Fläche auf einer **Geraden** zwischen zwei Kurven liegt.

Jede Linie auf der Fläche ist eine Gerade → die Fläche ist trotzdem im Allgemeinen **nicht eben**.

Typische Beispiele: - Zylinder, Kegel, Hyperboloid - Sattelähnliche Übergangsflächen zwischen zwei Kurven

Vorteil in der Fertigung: Regelflächen lassen sich durch **Abwickeln** aus Blech oder durch lineare Fräsbewegungen erzeugen.



Ruled vs. Smooth

```
bd.loft([p1, p2, p3], ruled=False)  # glatt (Spline-Interpolation)
bd.loft([p1, p2, p3], ruled=True)   # linear (Regelfläche)
```

Ruled (Regelfläche): - Gerade Linien zwischen entsprechenden Punkten der Profile - Gut für technische Teile mit klaren Kanten

Smooth (Spline): - Sanfte Übergänge, natürlicher für organische Formen - Standard (`ruled=False`)

Loft mit Vertex-Spitze

Ein Profil kann durch einen **einzelnen Punkt** (Vertex) ersetzt werden:

```
profil_basis = bd.Plane.XY * bd.Circle(30)
spitze = bd.Vertex(0, 0, 80)

kegel = bd.loft([profil_basis, spitze])
```

→ Loft endet in einem Punkt statt in einem zweiten Profil

Auch **asymmetrische Spitzen** möglich: Spitze außerhalb der Profilachse

Körperoperationen

Bool'sche Operationen

Körper lassen sich mit den Operatoren `+`, `-` und `&` kombinieren:

```
a = bd.Box(50, 50, 50)
b = bd.Cylinder(radius=15, height=60)

vereinigung  = a + b   # Vereinigung (Union)
differenz    = a - b   # Subtraktion (Cut)
schnitt      = a & b   # Schnittmenge (Intersection)
```

Operator	Bedeutung	Alias
`+`	Vereinigung	`bd.add()`
`-`	Subtraktion	`bd.cut()`
`&`	Schnittmenge	`bd.intersect()`

Verrundung und Fasen (fillet / chamfer)

Verrundung (`fillet`) rundet Kanten ab, **Fasen** (`chamfer`) fast sie an:

```
koerper = bd.Box(60, 40, 30)

# Alle Kanten verrunden
gerundet = bd.fillet(koerper.edges(), radius=5)

# Nur obere Kanten anfasen
obere_kanten = koerper.edges().filter_by(bd.Axis.Z)
angefast = bd.chamfer(obere_kanten, length=3)
```

Tipp: `.edges()` liefert alle Kanten – mit `.filter_by()` oder `.sort_by()` gezielt auswählen.

mirror

`mirror` spiegelt einen Körper an einer Ebene:

```python
halbkoerper = bd.Box(30, 50, 20)

# An der YZ-Ebene spiegeln und vereinigen
voll = halbkoerper + bd.mirror(halbkoerper, about=bd.Plane.YZ)
```

Typischer Workflow: **halbes Modell bauen → spiegeln** spart Aufwand und erzwingt Symmetrie.

Übung 2: Batteriepack-Gehäusedeckel

Gehäusedeckel für ein Batteriepack: rechteckige Basis, verjüngte Oberseite.

Parameter: - `l_basis, b_basis = 150, 100` mm (Länge und Breite der Basis) - `h_gesamt = 30` mm (Gesamthöhe des Deckels) - `radius_verj = 20` mm (Verjüngungsradius oben) - `wandstaerke = 3` mm

Aufgabe 2.1 – Basis und Oberseite

1. Basisprofil auf `Plane.XY`: `Rectangle` mit abgerundeten Ecken (`fillet` auf Vertices)
2. Oberes Profil auf `Plane.XY.offset(h_gesamt)`: kleineres `Rectangle` (um `2 * radius_verj` kleiner)
3. Beide Profile mit `loft()` verbinden → Außendeckel

Prüfen: Welche Flächentypen hat der geloftete Körper?

Aufgabe 2.2 – Aushöhlen mit offset()

`bd.offset(part, amount, openings)` erzeugt eine Schale mit konstanter Wandstärke: - `amount` negativ → nach innen (Aushöhlung) - `openings` → welche Fläche soll offen bleiben? (Tipp: unterste Fläche mit `sort_by`)

Prüfen: Volumen vorher/nachher – ist die Wandstärke plausibel?

Aufgabe 2.3 – Befestigungspunkte *(Zusatz)*

Vier zylindrische Bosse (Befestigungspunkte) an den Ecken der Basis: - Positionen mit `GridLocations` (2×2 Raster) - Bosse addieren (`Cylinder`), Löcher subtrahieren - Tipp: `align` am `Cylinder` beachten, damit der Boss auf der Unterseite sitzt

Aufgabe 2.4 – Parametrische Variation *(Zusatz)*

Ändern Sie die Parameter: - `l_basis = 200, b_basis = 120` → größerer Deckel - `wandstaerke = 4` → dickere Wand

Passt sich das Modell korrekt an? Welche Teile müssten ggf. manuell angepasst werden?

Sweep vs. Loft – Wann was?

Vergleich der Strategien

Kriterium	Sweep	Loft
Pfad	Explizite Kurve	Implizit (senkrecht zwischen Profilen)
Querschnitt	Konstant oder variabel	Variabel (interpoliert)
Typische Form	Rohr, Schlauch, Schiene	Übergangsformen, Verkleidungen
Kontrolle	Über Pfadgeometrie	Über Anzahl und Form der Profile

Faustregel: - **Konstanter Querschnitt** entlang gekrümmter Bahn → **Sweep** - **Variierende Querschnitte** übereinander → **Loft** - Beides kombinierbar: Sweep mit mehreren Profilen (`multisection=True`)

Multisection Sweep

```
pfad = bd.Spline((0, 0, 0), (50, 30, 20), (100, 0, 40))

profil_start = pfad.location_at(0)   * bd.Circle(10)
profil_mitte = pfad.location_at(0.5) * bd.Ellipse(12, 8)
profil_ende  = pfad.location_at(1)   * bd.Circle(6)

rohr_variabel = bd.sweep([profil_start, profil_mitte, profil_ende], pfad, multisection=True)
```

→ Querschnitt ändert sich entlang des Pfads (Hybrid aus Sweep und Loft)

Zusammenfassung

Zusammenfassung

	Sweep	Loft
Idee	Profil entlang Pfad	Interpolation zwischen Profilen
Pfad	explizit	implizit (senkrecht)
Querschnitt	konstant / variabel	variiert zwischen Profilen

- `location_at(t)` → Location auf Pfad (Platzierung, Multisection)
- `ruled=True` → Regelfläche; `ruled=False` → glatt (Standard)
- `+` / `-` / `&` → Boolean: Vereinigung, Subtraktion, Schnitt
- `fillet` / `chamfer` → Kanten abrunden / anfasen
- `mirror` → Symmetrie aus halbem Modell