

Programmierung von CAx-Systemen

David Straub

Gliederung

1. Einführung
2. Topologie
3. **Grundformen**
4. Kurven
5. Freiformgeometrie
6. Profile
7. Codequalität
8. Datenaustausch
9. Robustheit
10. Simulation
11. Optimierung

Grundformen

- **Konstruktionsebenen:** aus Flächen ableiten
- **Extrude & Revolve:** Profil → Körper
- **Muster als Schleifen:** Raster, linearer Stapel, Spiegeln
- **Rotation:** Reihenfolge und Selektoren
- **Parametersätze:** als Dataclass

Durchgängiges Beispiel: die Pouch-Zelle, der Zellstapel und die Endplatten

Feature-basiertes Modellieren

Modell als Abfolge von Features

Feature = eine atomare Modellierungsoperation (Extrude, `+`, `-`, `fillet`, ...)

Konstruktionsstrategie: 1. **Grundform** – einfachster Körper als Ausgangsbasis 2. **Additive Features** – Bosse, Rippen, Zapfen 3. **Subtraktive Features** – Bohrungen, Nuten, Taschen 4. **Finishing** – Verrundungen und Fasen zuletzt

Konstruktionsebenen aus Flächen ableiten

Eine `Plane` ist ein **lokales Koordinatensystem** – Ursprung plus Ausrichtung. Aus einer Fläche abgeleitet sitzt sie genau auf dieser Fläche:

```
from cadquery import Plane, Location

box = cf.box(60, 60, 10)
top = box.faces(">Z")
plane = Plane(origin=top.Center())
```

Aus der Fläche ableiten ist die robuste Wahl: bleibt korrekt, auch wenn sich Parameter ändern. Eine fest verdrahtete Höhe (z. B. `z=10`) wäre nach einer Dickenänderung schlicht falsch.

Auf der Ebene platzieren: Locations verketteten

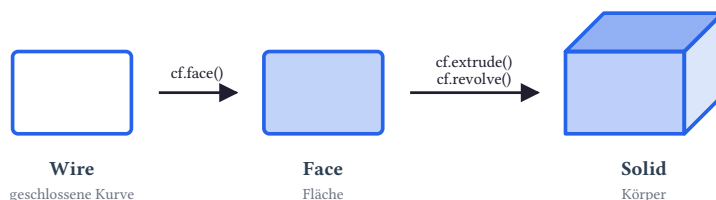
```
# richtig: Ebene, dann lokal darin versetzt
boss = cf.cylinder(d=8, h=8).moved(Location(plane) * Location((10, 0, 0)))
# Mittelpunkt (10, 0, 14) – sitzt auf der Oberseite bei z = 10

# falsch: der Versatz verwirft die Ebene still
boss = cf.cylinder(d=8, h=8).moved(Location(plane, (10, 0, 0)))
# Mittelpunkt (10, 0, 4) – wieder in globalen Koordinaten
```

`Location(plane)` ist die Lage der Ebene; die **Multiplikation** hängt einen lokal in dieser Ebene gemessenen Versatz an. Beide Zeilen laufen fehlerfrei durch – das Ergebnis unterscheidet sich um die volle Plattendicke.

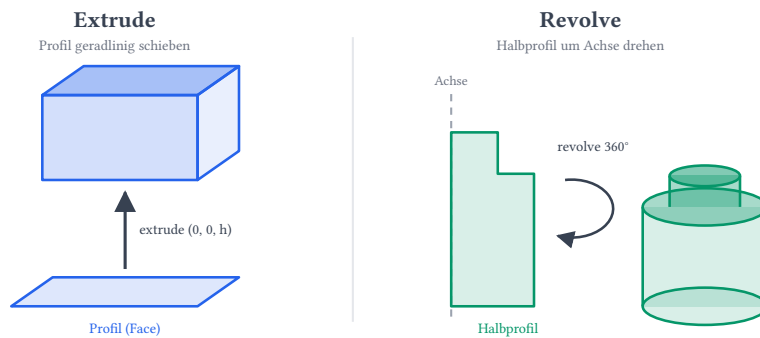
Von der Kurve zur Fläche

`extrude()` und `revolve()` erwarten eine **Fläche** als Eingang:



Der Wire muss **geschlossen und eben** sein – sonst bricht `cf.face()` ab oder liefert eine ungültige Fläche.

Zwei Wege vom Profil zum Körper



extrude() – Profil zu Körper

```
profil = cf.face(cf.rect(20, 10)) # Rechteck, zentriert im Ursprung
block = cf.extrude(profil, (0, 0, 5))
```

Profil-Bausteine: `cf.rect(b, h)` · `cf.circle(r)` (Bohrungen!) · `cf.ellipse(r1, r2)` · `cf.polygon(*pts)` · `cf.spline(*pts)` (Freiformkurven → Einheit 5)

`cf.polygon` schließt die Kontur selbst; `cf.polyline` bleibt offen – dort wiederholt der letzte Punkt den ersten:

```
pts = [(0, 0, 0), (20, 0, 0), (20, 10, 0), (0, 10, 0), (0, 0, 0)]
profil = cf.face(cf.polyline(*pts))
```

revolve() – Rotationskörper (zum Kennenlernen)

Statt ein Profil geradlinig zu schieben, dreht `revolve` es um eine Achse – aus einem **Halbprofil** (x = Radius, z = Höhe) wird ein Rotationskörper:

```
# Halbprofil in der XZ-Ebene: x = Radius, z = Höhe
halbprofil = cf.face(cf.polyline((0, 0, 0), (15, 0, 0), (15, 0, 8), (0, 0, 8), (0, 0, 0)))
scheibe = cf.revolve(halbprofil, (0, 0, 0), (0, 0, 1), 360) # volle Umdrehung um Z
```

Das Batteriemodul hat kein Rotationsteil – Revolve zeigen wir diese Woche einmal; die Technik bleibt für später im Werkzeugkasten.

Muster: Wiederholung durch Schleifen

Rechteckraster als Schleife

```
platte = cf.box(60, 60, 8)
for ix in range(3):
    for iy in range(3):
        x, y = (ix - 1) * 16, (iy - 1) * 16
        loch = cf.cylinder(d=4, h=10).moved(cf.Location((x, y, 0)))
        platte = platte - loch
```

Kein spezielles „Pattern“-Objekt nötig – eine Schleife über `Location`-Werte reicht und bleibt lesbar.

Zellstapel als Schleife

```
zelle = cf.extrude(cf.face(cf.rect(148, 98)), (0, 0, 11))
zelle = zelle.fillet(6, zelle.edges("|Z"))

pitch = 11 + 1.5          # Zelldicke + Kompressionspad
stapel = zelle
for i in range(1, 12):
    stapel = stapel + zelle.moved(cf.Location((0, 0, i * pitch)))
```

Dasselbe Schleifenmuster wie beim Raster, linear entlang einer Achse. Genau so entsteht gleich der Zellstapel.

Praktikum A: Zelle und Zellstapel

Aufgabe 1: Pouch-Zelle extrudieren

Bauen Sie eine Pouch-Zelle – im Kern ein flacher Block mit gerundeten Ecken:

- **148 × 98 × 11 mm**, Ecken mit **r = 6 mm** gerundet

Hinweise: `cf.rect`, `cf.face`, `cf.extrude`, `.fillet(r, ...edges("|Z"))`

Prüfen: `zelle.isValid()`; Volumen ungefähr $148 \cdot 98 \cdot 11$

Aufgabe 2: Stapeln und auf der Grundplatte platzieren

1. Stapeln Sie **12 Zellen** mit einer Schleife, Abstand **12,5 mm** (Zelldicke + 1,5 mm Kompressionspad).
2. Setzen Sie den Stapel auf die **Oberseite Ihrer Grundplatte** – die Höhe aus der Fläche ableiten statt einzutippen.

Hinweise: `.moved(cf.Location(...))`, `grundplatte.faces(">Z")`, `Plane(origin=...)`,
`Location(plane) * Location(...)`

Prüfen: `(grundplatte + stapel_platziert).isValid()`

Reihenfolge und Selektoren

Warum „Finishing zuletzt“? Selektoren fragen den aktuellen Stand

Sie haben gerade die Trägerplatte verrundet. Wäre erst gebohrt worden, hätte das schiefgehen können – denn ein Selektor wie `"%CIRCLE"` beantwortet seine Frage **an dem Modell, wie es in der Zeile steht**, nicht am fertigen Teil:

```
platte = cf.box(40, 30, 6)
tasche = cf.cylinder(d=18.6, h=3).moved(cf.Location((0, 0, 3)))
loch = cf.cylinder(d=3.4, h=6).moved(cf.Location((16, 11, 0)))

basis = platte - tasche - loch                # Loch schon gebohrt
rand = basis.faces(">Z").edges("%CIRCLE")    # 2 Treffer: Tasche UND Loch!
zu_frueh = basis.fillet(1.0, rand.Edges())  # verrundet auch den Lochrand
```

Vor dem Bohren fände `"%CIRCLE"` nur **einen** Rand (die Tasche). Beide Varianten sind `isValid()` – der Unterschied (2,6 mm³) fällt nur auf, wenn man danach sucht. Deshalb: Finishing zuletzt, oder präziser selektieren (Radius statt „ist ein Kreis“).

Spiegeln und Rotation

Spiegeln statt zweimal bauen

Zwei symmetrische Features sind eine Entscheidung plus ihr Spiegelbild:

```
loch = cf.cylinder(d=4, h=10).moved(cf.Location((30, 0, 0)))
loecher = loch + loch.mirror("YZ", basePointVector=(0, 0, 0))
```

`mirror` spiegelt an einer Ebene (hier „YZ“ durch den Ursprung) und gibt die gespiegelte Kopie zurück – Original und Kopie zusammen ergeben beide Löcher aus einer einzigen Platzierung. Gleich bei den Endplatten angewendet.

Rotation: Drehen mit `moved`

```
gedreht = cf.box(20, 5, 5).moved(rz=45)
```

`moved` nimmt Verschiebung (`x`, `y`, `z`) und Drehung (`rx`, `ry`, `rz`, in Grad) gemeinsam entgegen. Eine Drehung erfolgt **immer um den Ursprung** – nicht um den Mittelpunkt des Bauteils, außer der liegt zufällig dort.

Rotation: Reihenfolge ändert das Ergebnis – Vorhersage-Check

```
a = cf.box(20, 5, 5).moved(rz=90).moved(x=30) # erst drehen, dann verschieben
b = cf.box(20, 5, 5).moved(x=30).moved(rz=90) # erst verschieben, dann drehen
```

Vorher raten: wo landet `b`? Dann live prüfen. `a` landet bei `x=30`, `y=0`, `b` bei `x=0`, `y=30` – gleiche zwei Zahlen, unterschiedliches Ergebnis. Bei `a` dreht sich der (im Ursprung sitzende) Quader nur um sich selbst, die Verschiebung trägt ihn danach hin. Bei `b` sitzt der Quader schon bei `x=30`, wenn die Drehung um den Ursprung ihn samt Mittelpunkt mitschwingen lässt.

→ Beim Stapeln der Zellen wird diese Reihenfolge zur echten Entscheidung. Heute sehen Sie das Prinzip einmal und sagen es selbst voraus.

Parameter als Dataclass

Warum die Maße bündeln?

Dieselben Maße tauchen in jeder Funktion wieder auf – die Zelle brauchte `zell_b`, `zell_h`, `zell_t`, der Stapel `n_zellen`, `spacer`, die Endplatten gleich `plattenstaerke` ... Reicht man sie einzeln durch, gerät leicht ein Wert in Vergessenheit oder zwei passen nicht zusammen:

```
pitch = zell_t + spacer          # im Stapel
aussen = cf.box(zell_b + 12, ...) # in der Endplatte
# zell_t, n_zellen, plattenstaerke ... überall lose herumgereicht
```

Besser: alle Maße des Moduls in **einem** Objekt.

@dataclass : Maße in einem Objekt

```
from dataclasses import dataclass

@dataclass
class ModulParam:
    zell_b: float = 148.0    # mm, Zellbreite
    zell_h: float = 98.0     # mm, Zellhöhe
    zell_t: float = 11.0     # mm, Zelldicke
    n_zellen: int = 12
    spacer: float = 1.5      # mm, Kompressionspad
    plattenstaerke: float = 8.0 # mm, Endplatte
```

- Das `@dataclass` davor ist ein **Dekorator** – er erzeugt Konstruktor und Attribute automatisch, ohne Boilerplate.
- Die `: float` sind **Typ-Hinweise** (dokumentieren die erwartete Art des Werts) – für eine Dataclass nötig.

ModulParam: der Stapel, jetzt parametrisch

```
def zelle_bauen(p: ModulParam) -> cf.Shape:
    z = cf.extrude(cf.face(cf.rect(p.zell_b, p.zell_h)), (0, 0, p.zell_t))
    return z.fillet(6, z.edges("|Z"))

def stapel_bauen(p: ModulParam) -> cf.Shape:
    pitch = p.zell_t + p.spacer
    zelle = zelle_bauen(p)
    stapel = zelle
    for i in range(1, p.n_zellen):
        stapel = stapel + zelle.moved(cf.Location((0, 0, i * pitch)))
    return stapel
```

Derselbe Stapel wie in Praktikum A – aber die Maße kommen jetzt aus `p`.

Varianten mit `replace()`

```
from dataclasses import replace

p_standard = ModulParam()
p_gross = replace(p_standard, n_zellen=16, zell_t=14.0)
```

Ein Parametersatz, viele Varianten – der Rest der Werte bleibt unverändert.

Praktikum B: Endplatten und Zusammenführung

Aufgabe 3: Endplatten mit Mirror

Schreiben Sie `endplatten(p: ModulParam)`. Der Stapel wird von zwei **gleichen** Platten verspannt – eine bauen, die andere spiegeln:

- Platte **12 mm größer** als der Zellquerschnitt (Breite und Höhe), Dicke aus `ModulParam`
- Die untere sitzt direkt **unter** dem Stapel; die obere entsteht durch **Spiegelung** an der Ebene auf halber Stapelhöhe

Hinweise: `cf.box`, `.moved(cf.Location(...))`, `.mirror("XY", basePointVector=(0, 0, ...))`

Ergänzen Sie die dafür nötigen Felder in `ModulParam`.

Aufgabe 4 (Zusatz): alles an einem Parametersatz

Erweitern Sie `ModulParam` um die Maße aus Einheit 1 (Grundplatte) und dieser Einheit (Zellstapel, Endplatten). Ziel: ein einziger `ModulParam()` -Aufruf parametrisiert das bisherige Modul – Grundplatte, Stapel, Endplatten.

Abschluss

Leseauftrag & Ausblick

- **Leseauftrag:** Buch Kapitel 3 und 4
- **Wer mehr will:** ein gekerbtes Profil extrudieren (statt Rechteck) oder ein Revolve-Teil (Buchse, Rundstab) frei bauen
- **Nächste Woche:** Kurven – die Mathematik hinter den `Location` -, Plane- und Rotations-Aufrufen von heute
- Bis dahin: `w03/` committet und gepusht