

Programmierung von CAx-Systemen

David Straub

Gliederung

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

Geometrie I: Kurven und Koordinatensysteme

- Geometrie im B-Rep
- Parametrische Darstellung
- Koordinatensysteme und Transformationen
- Analytische Kurven: Linie, Kreis, Ellipse
- Beispiel: Versatz einer Kurve

Rückblick: Geometrie im B-Rep

Was steckt hinter einer Kante?

Letzte Vorlesung: das B-Rep-Gerüst

Solid → Shell → Face → Wire → Edge → Vertex

Topologie beschreibt die **Struktur** – aber noch nicht die Form.

Geometrie beantwortet: *wo* und *wie* liegt ein Element im Raum?

Topologieelement	trägt als Geometrie
Vertex	Punkt (x, y, z)
Edge	parametrische Kurve $C(u)$ + Parameterintervall $[u_1, u_2]$
Face	parametrische Fläche $S(u, v)$ + Parameterbereich

Geometrie im B-Rep ablesen

geom_type verrät, welche Geometrie OCCT einem Element intern zuordnet:

Element	geom_type	charakteristische Größe
Kreiskante (Deckel/Boden)	CIRCLE	Radius, Mittelpunkt
Nahtkante (Mantellinie)	LINE	Richtungsvektor
Mantelfläche	CYLINDER	Achse, Radius
Deckel-/Bodenfläche	PLANE	Normalenvektor, Ursprung

```
zyl = bd.Cylinder(radius=10, height=20)
for e in zyl.edges():
    print(e.geom_type, round(e.length, 2))
# → CIRCLE 62.83 / CIRCLE 62.83 / LINE 20.0
```

Parametrische Darstellung

Drei Darstellungsformen

Wie lässt sich ein geometrisches Objekt mathematisch beschreiben?

Form	Kreis (Beispiel)	Problem
Explizit	$y = \pm\sqrt{R^2 - x^2}$	mehrwertig; versagt bei senkrechter Tangente
Implizit	$x^2 + y^2 - R^2 = 0$	schwer auszuwerten; kein natürlicher Startpunkt
Parametrisch	$x = R \cos u, \quad y = R \sin u$	universell, eindeutig auswertbar

CAD-Systeme verwenden ausschließlich die **parametrische** Darstellung – für Kurven und Flächen.

Was kann man damit anstellen?

Eine parametrische Kurve erlaubt geometrische **Anfragen**, die mit einer bloßen Formel nicht möglich wären:

- Punkt und Tangentenvektor bei beliebigem u : $\mathbf{C}(u)$, $\mathbf{C}'(u)$
- Bogenlänge zwischen zwei Parameterwerten
- Krümmung $\kappa(u)$
- Nächster Kurvenpunkt zu einem gegebenen Raumpunkt (Projektion)

```
e.position_at(0.5)  # Punkt bei t = 0.5
e.tangent_at(0.5)  # Tangentenvektor dort
e.length           # Gesamtlänge
```

Tangentenvektor

Eine Kurve im 3D-Raum als Funktion eines Parameters u :

$$\mathbf{C}(u) = \begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix}, \quad u \in [u_{\min}, u_{\max}]$$

Der **Tangentenvektor** gibt Richtung und Geschwindigkeit entlang der Kurve an:

$$\mathbf{T}(u) = \mathbf{C}'(u) = \frac{d\mathbf{C}}{du}$$

- **Richtung:** zeigt in die momentane Bewegungsrichtung
- **Länge $|\mathbf{T}|$:** „Geschwindigkeit“ im Parameterraum – hängt von der Parametrisierung ab

Bogenlänge

Die **Bogenlänge** s misst die tatsächlich zurückgelegte Weglänge entlang der Kurve:

$$s(u) = \int_{u_0}^u |\mathbf{C}'(\tilde{u})| d\tilde{u}, \quad \frac{ds}{du} = |\mathbf{C}'(u)|$$

Bogenlängenparametrisierung (s als Parameter):

$$\left| \frac{d\mathbf{C}}{ds} \right| = 1$$

→ Tangentenvektor hat immer Länge 1

→ Geometrisch „natürliche“ Parametrisierung; rechnerisch aufwändig

Krümmung

Die **Krümmung** κ misst, wie stark sich die Tangentenrichtung mit der zurückgelegten Weglänge ändert:

$$\kappa = \left| \frac{d^2\mathbf{C}}{ds^2} \right|$$

→ Bogenlängenparametrisierung

Für beliebige Parametrisierung (via Kettenregel + Lagrange-Identität):

Der **Normalenvektor** ergibt sich aus den Tangenten in u - und v -Richtung:

$$\mathbf{n}(u, v) = \frac{\partial \mathbf{S}}{\partial u} \times \frac{\partial \mathbf{S}}{\partial v}$$

→ Vorzeichen bestimmt, welche Seite „außen“ ist (erinnert an Orientierung aus VL2)

Die **Flächentypen** (Ebene, Zylinder, NURBS-Flächen, ...) sind Thema von **Vorlesung 4**.

Koordinatensysteme und Transformationen

Vektoren, Achsen, Ebenen

Drei Grundobjekte für die Arbeit im Raum:

Klasse	Bedeutung	Beispiel
<code>Vector</code>	Punkt oder freier Vektor	<code>Vector(1, 2, 3)</code>
<code>Axis</code>	Ursprung + Richtung (Achse)	<code>Axis((0,0,0), (0,0,1))</code>
<code>Plane</code>	Ebene: Ursprung + Orientierung	<code>Plane.XY</code>

Auf `Vector` sind die üblichen Operationen definiert: Addition, Skalierung, Länge (`v.length`), Normierung, Skalarprodukt (`v.dot(w)`), Kreuzprodukt (`v.cross(w)`).

Standardachsen und Standardebenen

Vordefinierte Achsen: `Axis.X`, `Axis.Y`, `Axis.Z` (Weltkoordinatensystem)

Vordefinierte Ebenen:

Ebene	Normalenrichtung	Verwendung
<code>Plane.XY</code>	z -Achse	Standard-Skizzierebene
<code>Plane.XZ</code>	y -Achse	Frontansicht
<code>Plane.YZ</code>	x -Achse	Seitenansicht

Benutzerdefiniert: - `Axis((5, 3, 0), (0, 0, 1))` – Achse durch beliebigen Punkt -
`Plane(origin=(0,0,10), z_dir=(1,0,0))` – Ebene mit beliebiger Normalen

Location: Lage eines Objekts im Raum

Jedes Objekt hat eine **Location** – seine Position und Orientierung im Weltkoordinatensystem.

Eine Location kodiert: - **Translation:** Verschiebung (dx, dy, dz) - **Rotation:** Orientierung (als Quaternion / Rotationsmatrix gespeichert)

Location ist das lokale **Koordinatensystem** eines Objekts relativ zur Welt.

Kann aus einem Punkt (`Location((20, 10, 5))`) oder aus einer Ebene (`Location(Plane.XZ)`) erzeugt werden.

Transformationen

Operation	Methode
Verschiebung	<code>obj.move(Location((dx, dy, dz)))</code>
Drehung	<code>obj.rotate(Axis.Z, winkel_grad)</code>
Spiegelung	<code>obj.mirror(Plane.XZ)</code>
Skalierung	<code>obj.scale(faktor)</code>

Wichtig: Jede Transformation liefert ein **neues Objekt** – das Original bleibt unverändert.

Transformationen lassen sich verketteten: `zyl.rotate(Axis.Z, 30).move(Location((20, 0, 0)))`

Drehungen im Raum

Eine Drehung im 3D-Raum ist durch **Achse** + **Winkel** vollständig beschrieben:

$$\text{Drehung} = (\hat{\mathbf{e}}, \varphi)$$

- $\hat{\mathbf{e}}$: Einheitsvektor der Drehachse (beliebig im Raum)
- φ : Drehwinkel (im Uhrzeigersinn von oben, Rechte-Hand-Regel)

Intern wird daraus eine **Rotationsmatrix** $\mathbf{R} \in \mathbb{R}^{3 \times 3}$ berechnet, die auf jeden Punkt angewendet wird: $\mathbf{p}' = \mathbf{R} \mathbf{p}$

```
# Drehung um die Z-Achse, 45°
zyl.rotate(Axis.Z, 45)

# Drehung um eine beliebige Achse durch Punkt (5, 0, 0)
zyl.rotate(Axis((5, 0, 0), (0, 0, 1)), 30)
```

Reihenfolge von Transformationen

Translationen **kommutieren** – die Reihenfolge ist egal:

$$\mathbf{T}_1 \mathbf{T}_2 = \mathbf{T}_2 \mathbf{T}_1$$

Rotationen **kommutieren nicht** – die Reihenfolge ist entscheidend:

$$\mathbf{R}_x \mathbf{R}_z \neq \mathbf{R}_z \mathbf{R}_x$$

Rotation + Translation **kommutieren ebenfalls nicht**:

$$\mathbf{T} \mathbf{R} \neq \mathbf{R} \mathbf{T}$$

→ Bei verketteten Transformationen immer auf die **Reihenfolge** achten.

Ebene aus einer Fläche

Ein besonders nützliches Muster: Konstruktionsebene direkt aus einer vorhandenen Fläche ableiten.

```
ebene = Plane(oberseite) # Ursprung + Orientierung der Fläche
```

Die Ebene übernimmt automatisch den Flächenmittelpunkt als Ursprung und die Flächennormale als z -Richtung.

Anwendung: Bohrung senkrecht auf einer schrägen Fläche – die Skizze wird auf der Fläche platziert, unabhängig von deren Lage im Raum.

Kurventypen

Analytische Kurven

Exakt durch eine geschlossene Formel beschreibbar:

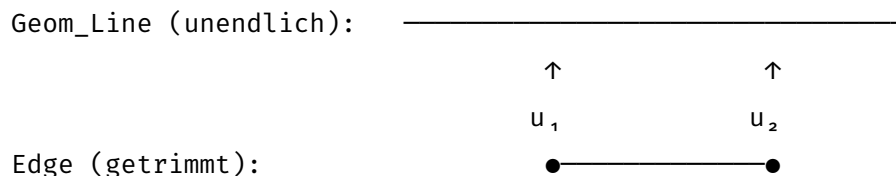
Typ	$\mathbf{C}(u)$	geom_type	Vorkommen
Linie	$\mathbf{A} + u \mathbf{d}$	LINE	Kanten eines Quaders
Kreis	$\mathbf{M} + R(\cos u \mathbf{e}_1 + \sin u \mathbf{e}_2)$	CIRCLE	Kanten eines Zylinders
Ellipse	wie Kreis, aber $R_x \neq R_y$	ELLIPSE	Kegelschnitte

Für Standardkörper (Box, Cylinder, Sphere, ...) reichen analytische Kurven vollständig aus.

Trimming: wie wird eine Kurve endlich?

Analytische Kurven sind mathematisch **unbegrenzt** – eine Linie reicht von $-\infty$ bis $+\infty$.

Eine Edge ist immer ein **endliches Stück**: Kurve + Parameterintervall $[u_1, u_2]$



→ Dasselbe Kreisobjekt steckt hinter einem Halbkreis-Edge wie hinter einem Viertelkreis – nur das Intervall $[u_1, u_2]$ unterscheidet sie.

Ellipse: variable Krümmung

$$\mathbf{C}(u) = \begin{pmatrix} a \cos u \\ b \sin u \end{pmatrix}, \quad \kappa(u) = \frac{ab}{(a^2 \sin^2 u + b^2 \cos^2 u)^{3/2}}$$

An den vier Scheitelpunkten:

Stelle	κ	$R_K = 1/\kappa$
Ende der großen Halbachse ($u = 0^\circ$)	b/a^2	a^2/b (flach)
Ende der kleinen Halbachse ($u = 90^\circ$)	a/b^2	b^2/a (stark gebogen)

→ Die Ellipse hat **keine** konstante Krümmung – im Gegensatz zum Kreis, und das entscheidet, wie Operationen wie Versatz oder Abrundung auf sie wirken.

Beispiel: Versatz einer Kurve

Was ist ein Versatz (Offset)?

Eine **Versatzkurve** entsteht, indem jeder Punkt der Originalkurve um eine konstante Distanz d entlang des Normalenvektors verschoben wird:

$$\mathbf{C}_{\text{offset}}(u) = \mathbf{C}(u) + d \cdot \hat{\mathbf{n}}(u)$$

Typische CAD-Anwendungen: - Wanddicke bei Dünnwandteilen - Freiraum um ein Bauteil (Kollisionsprüfung) - Werkzeugpfad beim Fräsen

Versatz eines Kreises

Kreis (Radius R): alle Normalen zeigen durch den **Mittelpunkt** → der Versatz verschiebt den Radius gleichmäßig.

$$R_{\text{offset}} = R + d$$

→ wieder ein Kreis!

Grund: Konstante Krümmung → Normalen drehen gleichmäßig → Versatz ändert nur den Radius.

Versatz einer Ellipse

Ellipse ($a = 30, b = 15$): Normalen drehen **ungleichmäßig** – die Krümmung variiert.

Die Versatzkurve lässt sich nicht als $a' \cos u, b' \sin u$ schreiben:

```
ellipse = bd.Edge.make_ellipse(x_radius=30, y_radius=15)
offset = ellipse.offset_2d(5)
for e in offset.edges():
    print(e.geom_type)    # → OFFSET
```

OCCT berechnet den Versatz numerisch als **Offset-Kurve** – keine analytische Ellipse mehr.

Warum ist die Versatzkurve keine Ellipse mehr?

Bei konstanter Krümmung (Kreis) wirkt der Versatz überall gleich. Bei **variabler Krümmung** (Ellipse) wirkt er ungleichmäßig:

- Stark gebogene Stellen (kleiner R_K): Versatz „staucht“ die Kurve
- Flache Stellen (großer R_K): Versatz ändert die Form kaum

Geometrische Operationen erzeugen häufig Kurven **höherer Komplexität** als die Eingabe — selbst bei einfachen Ausgangskurven.

Daher brauchen CAD-Systeme eine Darstellung, die **beliebige** glatte Kurven beschreiben kann.

Zusammenfassung

- Edge trägt eine Kurve $\mathbf{C}(u)$, Face trägt eine Fläche $\mathbf{S}(u, v)$
- Parametrische Darstellung: universell, eindeutig auswertbar
- Tangente $\mathbf{C}'(u)$ und Krümmung $\kappa = 1/R_K$ aus Ableitungen
- Vector, Axis, Plane, Location – räumliches Handwerkszeug; Transformationen erzeugen neue Objekte
- Analytische Kurven (Linie, Kreis, Ellipse): exakt, aber begrenzt
- Geometrische Operationen erzeugen komplexere Geometrie: Versatz einer Ellipse → keine Ellipse mehr

Ausblick: Geometrie II

- **Freiformkurven:** Bézier, B-Spline, NURBS
- **Stetigkeitsbedingungen:** $C^0/C^1/C^2$ beim Verbinden von Kurven
- **Flächen:** analytisch, Sweep-Flächen, NURBS-Flächen
- **Geometrische Anfragen:** Projektion, Abstand, Schnittpunkte