Programmierung von CAx-Systemen

David Straub

Gliederung

1. Einführung
2. Topologie
3. Geometrie
4. Modellierungsstrategien
5. Datenaustausch
6. **Meshing & Simulation**
7. Parametrische Robustheit & Optimierung

Übersicht

1. Gitter (Mesh) – Grundbegriffe
2. STL – Oberflächentessellierung für 3D-Druck
3. FEM-Gitter & Gmsh – Vernetzung für Simulation (2D und 3D)
4. Finite-Elemente-Methode (FEM) – Struktursimulation

Lernziele

Nach dieser Einheit können Sie…

- erklären, was ein Gitter ist und warum es für Simulation gebraucht wird
- den Unterschied zwischen STL-Tessellierung und FEM-Vernetzung erklären
- ein CAD-Modell als STL exportieren und visualisieren
- mit Gmsh ein FEM-Gitter (2D oder 3D) aus einem Modell erzeugen
- eine einfache FEM-Simulation aufsetzen, ausführen und das Ergebnis plausibilisieren

Nicht erwartet: FEM-Theorie im Detail, komplexe Geometrien oder Randbedingungen.

Gitter (Mesh)

Was ist ein Gitter?

CAD-Modelle (B-Rep) beschreiben Geometrie **exakt** – durch mathematische Kurven und Flächen.

Ein **Gitter** (*Mesh*) ersetzt diese exakte Beschreibung durch eine endliche Menge einfacher geometrischer Grundelemente:

- **2D:** Dreiecke oder Vierecke
- **3D:** Tetraeder, Hexaeder, Prismen

Das Aufteilen einer Fläche in Dreiecke heißt **Triangulierung**. Den Prozess der Umwandlung von exakter Geometrie in ein Gitter nennt man **Tessellierung**.

> Das Gitter ist eine **Annäherung** – die exakte Form wird durch viele kleine, einfache Stücke approximiert.

Warum approximieren?

Viele Berechnungen lassen sich auf exakter B-Rep-Geometrie **nicht direkt durchführen**.

Ein Gitter macht die Geometrie **diskret** – und damit für numerische Methoden zugänglich:

- Differentialgleichungen lassen sich auf Gitterpunkten lösen
- Physikalische Größen (Spannung, Temperatur, Druck) werden an Knoten oder Elementen berechnet
- Jedes Element hat einfache, bekannte Eigenschaften

> **Faustregel:** Je feiner das Gitter, desto genauer die Annäherung – und desto mehr Rechenaufwand.

Zwei Gittertypen – zwei Zwecke

Nicht alle Gitter sind gleich – der Zweck bestimmt, was ein „gutes" Gitter ist:

	STL	FEM-Gitter
Beschreibt	Außenhülle des Körpers	Fläche (2D) oder Volumen (3D)

	STL	FEM-Gitter
Elemente	Dreiecke auf der Oberfläche	Dreiecke/Vierecke (2D), Tetraeder/Hexaeder (3D)
Qualitätsziel	geometrische Treue	gleichmäßige, gut geformte Elemente
Ebene Flächen	wenige große Dreiecke genügen	gleichmäßige Elementgröße nötig
Anwendung	3D-Druck, Visualisierung	Struktursimulation, Strömungssimulation

STL fragt: *Stimmt die Form?* FEM fragt: *Sind die Elemente gut genug für die Numerik?*

STL – Oberflächentessellierung

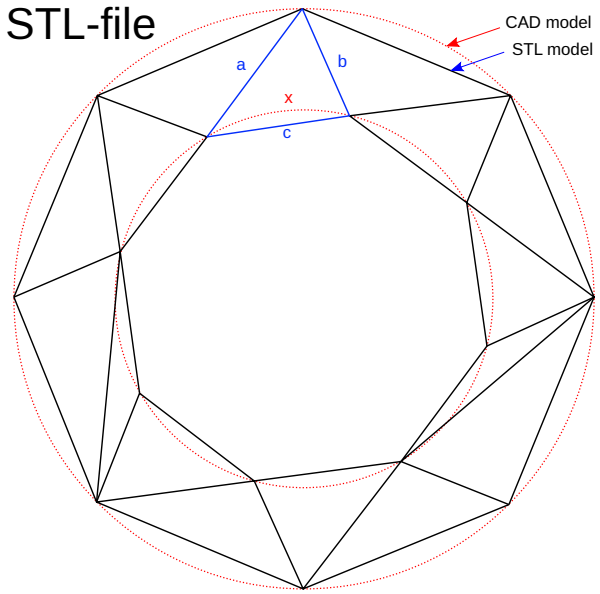
Das STL-Format

STL (*Stereolithography*, auch *Standard Tessellation Language*) ist das einfachste und verbreitetste Oberflächengitter-Format.

Gespeichert wird ausschließlich: - eine Liste von **Dreiecken** (je drei Eckpunkte + Normalenvektor)

Nicht gespeichert: Topologie, Maßeinheiten, Material, Farbe.

> STL beschreibt nur die Form der Hülle – sonst nichts.



STL: Dateiaufbau

Zwei Varianten: **ASCII** (lesbar) und **Binär** (5–10× kompakter, Standard in der Praxis).

ASCII-Beispiel:

```
facet normal 0 0 1
  outer loop
    vertex 0 0 0
    vertex 1 0 0
    vertex 0 1 0
```

```
  endloop
endfacet
```

Ein Dreieck = 1 `facet`-Block mit Normalenvektor und 3 Eckpunkten.

STL: Approximationsqualität

Die Oberfläche wird durch Dreiecke **angenähert** – je nach Einstellung grob oder fein:

```
from build123d import Box, export_stl

part = Box(10, 10, 10)
export_stl(part, "bauteil.stl", tolerance=0.1, angular_tolerance=0.1)
```

- **`tolerance`:** maximale Abweichung von der exakten Fläche (in mm), Standard: 0.001
- **`angular_tolerance`:** maximale Winkelabweichung zwischen benachbarten Segmenten (in Radiant), Standard: 0.1 rad 5.7°
- **`ascii_format`:** `True` = ASCII (lesbar), `False` = Binär (Standard, 5–10× kleiner)

Kleiner Toleranzwert → mehr Dreiecke → **größere Datei**

STL: Gültigkeitsbedingungen

STL ist nur eine Liste von Dreiecken – einzelne Flächen und offene Shells sind erlaubt.

Für den **3D-Druck** muss der Körper geschlossen sein: der Slicer braucht ein definiertes Innen/Außen.

Typische Fehler – entstehen durch fehlerhafte CAD-Geometrie, nicht beim Export: - Nicht-mannigfaltige Kanten (non-manifold edges, mehr als 2 Dreiecke an einer Kante) - Offene Kanten (Lücken im Netz) - Falsch orientierte Normalen

Häufige Ursache: Körper nur positioniert statt verschmolzen → `fuse()` / `+` verwenden.

> build123d erzeugt bei gültiger Geometrie in der Regel fehlerfreie STL-Dateien.

STL im 3D-Druck

Workflow: CAD-Modell → STL → Slicer → Druckanweisungen (G-Code)

Der Slicer berechnet Schichten, Füllmuster und Stützstrukturen **aus dem STL**.

Toleranz-Entscheidung:

`tolerance`	Dreiecke	Dateigröße	sichtbare Qualität
0.5 mm	wenige	klein	grob
0.1 mm	mittel	mittel	gut
0.01 mm	viele	groß	sehr fein

Für die meisten FDM-Drucker reicht `0.1 mm` – die Druckauflösung ist ohnehin der limitierende Faktor.

STL visualisieren

Ein exportiertes STL lässt sich direkt zurückladen und im CAD-Viewer anzeigen:

```
import build123d as bd
import ocp_vscode

mesh = bd.import_stl("bauteil.stl")
ocp_vscode.show(mesh)
```

- Kein zusätzliches Paket nötig
- Zeigt das Gitter im gewohnten CAD-Viewer
- Gut zum Prüfen: Stimmt die Form? Dreiecke sichtbar?

  Für Gitterdaten (MSH, Simulationsergebnisse) braucht man **PyVista** – das lernen wir im nächsten Abschnitt.

? Rundzelle mit Crimp-Nut

Modellieren Sie eine zylindrische Rundzelle (18650: 18 mm, Höhe 65 mm) mit einer umlaufenden Crimp-Nut (ca. 3 mm vor der Oberkante, 1 mm tief, 2 mm breit).

Exportieren Sie die Zelle als STL bei zwei verschiedenen Toleranzen und vergleichen Sie:

```
import os, build123d as bd, ocp_vscode

mesh = bd.import_stl("rundzelle.stl")
print(f"Dateigröße: {os.path.getsize('rundzelle.stl') / 1024:.0f} KB")
ocp_vscode.show(mesh)
```

`tolerance`	Dateigröße	Sichtbare Qualität
0.5 mm	?	?
0.05 mm	?	?

Frage: Wo ist eine feine Toleranz besonders wichtig – und warum?

Tessellierung: im CAD-Kernel eingebaut

Ein B-Rep-Modell besteht aus mathematischen Kurven und Flächen – eine Grafikkarte kann damit **nichts anfangen**.

Jede Darstellung auf dem Bildschirm braucht Dreiecke. Ohne Tessellierung: kein Bild.

> **Der CAD-Kernel (OCCT) tesselliert selbst** – er wandelt das exakte B-Rep-Modell bei Bedarf in Dreiecke um.

`build123d.export_stl`, `ocp_vscode.show`, `pyvista_cad.plot_cad` – alle nutzen denselben Kern

FEM-Gitter & Gmsh

Warum ein eigenes Gitter für FEM?

STL beschreibt nur die **Hülle** – und hat ein anderes Qualitätsziel als FEM.

Für Simulationen braucht man:

- Elemente **im Inneren** (Fläche bei 2D, Volumen bei 3D) – nicht nur an der Oberfläche
- **Gleichmäßig geformte Elemente** – schlechte Dreiecke führen zu Rechenfehlern
- Kontrollierbare **Elementgröße** – fein dort, wo die Physik interessant ist
- Nachbarschaftsinformation: welche Elemente teilen einen Knoten?

  STL ist für FEM ungeeignet: falsches Qualitätsziel, kein Inneres, keine Topologie.

Gitter visualisieren mit PyVista

PyVista liest alle gängigen Gitterformate – STL, MSH, VTK – und zeigt sie mit Kanten:

```
pip install pyvista
```

```
import pyvista as pv

mesh = pv.read("bauteil.stl")   # oder .msh, .vtk, ...
mesh.plot(show_edges=True)
```

- Interaktives 3D-Fenster, drehbar mit der Maus
- Zeigt Gitterkanten – Elementgröße und -form werden sichtbar
- Funktioniert für STL (Hülle) **und** FEM-Gitter (Fläche oder Volumen)

  Damit können wir jetzt STL und FEM-Gitter direkt vergleichen.

Beispiel: Platte mit Bohrung

Gleiche Geometrie – zwei völlig verschiedene Gitter:

STL (Hülle für 3D-Druck) - Dreiecke nur auf der Außenfläche - Flachseiten: wenige, große Dreiecke - Am Bohrungsrand: kleinere Dreiecke (Kreisbogen) - Kein Inneres – der Drucker füllt selbst aus

FEM-Gitter (Fläche für Simulation) - Dreiecke füllen das **Innere** der Platte - Überall

gleichmäßige Elementgröße - Verfeinerung am Bohrungsrand möglich - 2D – kein Volumen nötig

> Ein STL-Gitter sieht auf einer ebenen Fläche spärlich aus – das ist **korrekt**. Ein FEM-Gitter auf derselben Fläche ist gleichmäßig dicht – das ist **nötig**.

Platte mit Bohrung – STL

```
import build123d as bd, pyvista as pv

plate = bd.Box(100, 50, 2)
hole = bd.Cylinder(radius=10, height=2)
part = plate - bd.Pos(50, 25) * hole
bd.export_stl(part, "platte.stl")

pv.read("platte.stl").plot(show_edges=True)
```

- Nur die **Hülle** – 6 Flächen des Quaders + Bohrungswand
- Flachseiten: wenige große Dreiecke (ebene Fläche → braucht keine feinen Elemente)
- Bohrungsrand: kleinere Dreiecke (Kreisbogen muss approximiert werden)

Platte mit Bohrung – FEM-Gitter (2D)

```
import cadgmsh, build123d as bd, pyvista as pv

plate = bd.Box(100, 50, 2)
hole = bd.Cylinder(radius=10, height=2)
part = plate - bd.Pos(50, 25) * hole

mesh = cadgmsh.mesh(part, dim=2, lc=5)
pv.from_meshio(mesh).plot(show_edges=True)
```

- `dim=2`: 2D-Flächengitter – füllt das Innere der Platte
- `lc=5`: maximale Elementgröße in mm
- `pv.from_meshio(mesh)`: direkt von cadgmsh zu PyVista – keine Datei nötig

Qualität von FEM-Gittern

Nicht jedes Gitter ist gleich gut – schlechte Elemente führen zu Rechenfehlern. Gilt für 2D und 3D:

Wichtige Qualitätskriterien:

- **Elementgröße:** passend zur erwarteten Detailgröße – auch auf ebenen Flächen!
- **Seitenverhältnis** (*Aspect Ratio*): Verhältnis der längsten zur kürzesten Kante – 1 ist ideal, > 10 ist problematisch
- **Skewness:** Winkelabweichung von der idealen Form – 0 ist ideal, $\to 1$ ist entartet

Konvergenz: > Wenn das Ergebnis sich bei Halbierung der Elementgröße kaum noch ändert, ist das Gitter fein genug.

Zielkonflikt: feineres Gitter → genauere Ergebnisse → mehr Rechenzeit

Gmsh: Werkzeug und Workflow

Gmsh ist das Standard-Werkzeug zur FEM-Vernetzung – frei, leistungsfähig, skriptbar:

`Geometrie → Gmsh → .msh → Solver`

- Vernetzt beliebige 2D- und 3D-Geometrien
- Steuerbar über GUI, Kommandozeile oder **Python-API**
- Ausgabe als `.msh` – lesbar von allen gängigen Solvern

Die native Python-API ist mächtig, aber ausführlich: Geometrie aufbauen, synchronisieren, Optionen setzen, Datei schreiben, finalize – viele Schritte für ein einfaches Ergebnis.

cadgmsh: Gmsh für build123d

cadgmsh ist ein schlanker Wrapper um die Gmsh-API, der build123d-Modelle direkt entgegennimmt:

```
pip install cadgmsh   # installiert gmsh mit
```

```
import cadgmsh, build123d as bd, pyvista as pv

part = bd.Cylinder(radius=10, height=40)
mesh = cadgmsh.mesh(part, dim=3, lc=4)   # meshio.Mesh
```

```
pv.from_meshio(mesh).plot(show_edges=True)
```

Parameter	Bedeutung
`dim=2` / `dim=3`	2D-Flächen- oder 3D-Volumengitter
`lc`	maximale Elementgröße

Ein Funktionsaufruf – kein Zwischenschritt, keine Hilfsdatei.

Finite-Elemente-Methode (FEM)

Von Gmsh zum Solver

Das Volumengitter ist fertig – jetzt kommt die Physik.

Die **Finite-Elemente-Methode (FEM)** ist ein numerisches Verfahren zur Lösung von Differentialgleichungen auf einem Gitter. Ein **FEM-Solver** löst diese Gleichungen elementweise: - jedes Element liefert einen Beitrag zur Gesamtlösung - die Elemente sind über gemeinsame Knoten gekoppelt - das Ergebnis ist eine Näherungslösung im gesamten Volumen

Werkzeugkette:

```
build123d → cadgmsh → scikit-fem → Ergebnis → PyVista
```

```
pip install cadgmsh scikit-fem
```

Grundidee der FEM

Ein kontinuierliches Problem (Differentialgleichung im Volumen) wird auf endlich viele **Unbekannte an Knoten** reduziert.

Für lineare Elastizität: gesucht ist die **Verschiebung** u an jedem Knoten.

Das führt auf ein lineares Gleichungssystem:

$$K\,u = f$$

- K: **Steifigkeitsmatrix** – hängt von Geometrie und Material ab
- u: gesuchte Knotenverschiebungen
- f: äußere Kräfte (Lastvektor)

  Lösen = ein großes lineares Gleichungssystem auflösen.

Materialgesetz: lineare Elastizität

Das Hooksche Gesetz verknüpft Dehnung ε und Spannung σ:

$$\sigma = \lambda\,\mathrm{tr}(\varepsilon)\,I + 2\mu\,\varepsilon$$

Die **Lamé-Parameter** λ und μ folgen aus dem E-Modul E und der Querkontraktionszahl ν:

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}$$

In Python:

```
from skfem.models.elasticity import lame_parameters
lam, mu = lame_parameters(E=210e3, nu=0.3)  # Stahl, Einheiten MPa + mm
```

FEM-Setup: Zylinder unter Zug

An jedem Gitterknoten gibt es 3 **Freiheitsgrade** (DOFs): Verschiebung in x, y, z.

Randbedingungen legen fest, was bekannt ist:

Fläche	Bedingung	Bedeutung
Unterseite $z = -20$	$u_x = u_y = u_z = 0$	fest eingespannt
Oberseite $z = +20$	Kraft $F = 1000\,\mathrm{N}$ in z	Zugbelastung
Rest	–	gesucht

Ohne Randbedingungen ist das Gleichungssystem singulär – der Körper könnte sich beliebig verschieben.

Randbedingungen in scikit-fem

Physical Groups aus cadgmsh werden direkt als benannte Ränder übernommen:

```
f = basis.zeros()   # Lastvektor: 1 Eintrag pro DOF, initial 0

# Unterseite: alle 3 DOFs pro Knoten fixieren (x, y, z)
fixed = basis.get_dofs(skfem_mesh.boundaries["bottom"]).all()

# Oberseite: nur z-DOFs – Kraft in z-Richtung aufprägen
top = basis.get_dofs(skfem_mesh.boundaries["top"]).nodal["u^3"]
f[top] = 1000.0 / len(top)   # 1000 N gleichmäßig verteilt
```

Kein Koordinatenfilter, keine Gleitkomma-Toleranz – die Geometrie ist bereits bekannt.

Von-Mises-Vergleichsspannung

Aus dem Spannungstensor σ wird eine skalare Vergleichsgröße berechnet:

$$\sigma_{\mathrm{VM}} = \sqrt{\frac{1}{2}\left[(\sigma_{11}-\sigma_{22})^2 + (\sigma_{22}-\sigma_{33})^2 + (\sigma_{33}-\sigma_{11})^2 + 6(\sigma_{12}^2+\sigma_{23}^2+\sigma_{13}^2)\right]}$$

- **Fließbedingung:** $\sigma_{\mathrm{VM}} < \sigma_{\mathrm{yield}}$ → kein plastisches Versagen
- Für Stahl: $\sigma_{\mathrm{yield}} \approx 250\,\mathrm{MPa}$
- Die Von-Mises-Spannung ist die häufigste Ergebnisgröße in der FEM-Strukturanalyse

Erwartetes Ergebnis: einfacher Zugstab

Zylinder $r = 10\,\mathrm{mm}$, $l = 40\,\mathrm{mm}$, Stahl, $F = 1000\,\mathrm{N}$ Zug:

Analytische Lösung:

$$\Delta l = \frac{F \cdot l}{E \cdot A} = \frac{1000 \cdot 40}{210000 \cdot \pi \cdot 10^2} \approx 0{,}0006\,\mathrm{mm}$$

$$\sigma_z = \frac{F}{A} = \frac{1000}{\pi \cdot 10^2} \approx 3{,}18\,\mathrm{MPa}$$

Plausibilitätscheck: - Verschiebung = 0 an Einspannung, linear zunehmend nach oben - Von-Mises 3 MPa im Kern, erhöht an Einspannung (Spannungskonzentration)

scikit-fem: Überblick

```
import cadgmsh, build123d as bd
from skfem.io.meshio import from_meshio

# Gitter erzeugen mit benannten Rändern
part = bd.Cylinder(radius=10, height=40)
cadmesh = cadgmsh.mesh(part, dim=3, lc=4, physical={
    "top":    part.faces().sort_by(bd.Axis.Z).last,
    "bottom": part.faces().sort_by(bd.Axis.Z).first,
})
skfem_mesh = from_meshio(cadmesh)
```

```python
# Steifigkeitsmatrix (Stahl: E=210 GPa, v=0.3)
lam, mu = lame_parameters(E=210e3, nu=0.3)
basis = Basis(skfem_mesh, ElementVector(ElementTetP1()))
K = linear_elasticity(lam, mu).assemble(basis)

# Lastvektor + Randbedingungen
f = basis.zeros()
top_dofs = basis.get_dofs(skfem_mesh.boundaries["top"]).nodal["u^3"]
f[top_dofs] = 1000.0 / len(top_dofs)
fixed_dofs = basis.get_dofs(skfem_mesh.boundaries["bottom"]).all()

# Lösen
u = solve(*condense(K, f, D=fixed_dofs))
```

FEM-Ergebnisse visualisieren

PyVista unterscheidet zwei Arten von Ergebnisdaten:

```python
# Knotenbasiert – 1 Wert pro Knoten (z.B. Verschiebung)
grid.point_data["Verschiebung z [mm]"] = u[basis.nodal_dofs[2]]

# Elementbasiert – 1 Wert pro Tetraeder (z.B. Spannung)
grid.cell_data["Von-Mises [MPa]"] = vm
```

Danach wie gewohnt: `grid.plot(scalars="...")` oder `pv.Plotter` für mehrere Ansichten.

> Vollständiger Code (inkl. Grid-Konvertierung): `fem_zug_zylinder.py`

Grenzen der linearen statischen FEM

Diese Einheit behandelt den einfachsten Fall: **statisch, linear, isotropes Material**.

Nicht abgedeckt:

Phänomen	Erweiterung
Große Verformungen	Nichtlineare Geometrie
Plastizität, Bruch	Nichtlineares Material

Phänomen	Erweiterung
Kontakt, Reibung	Kontaktmechanik
Schwingungen, Stoß	Dynamik / Modalanalyse
Wärme, Strömung	Multiphysik
Dünnwandige Strukturen	Schalenelemente

Für reale Bauteile ist die Wahl des richtigen Modells mindestens so wichtig wie die Berechnung selbst.

? FEM: Zug-Simulation

Führen Sie `fem_zug_zylinder.py` aus und überprüfen Sie:

1. Stimmt die maximale Verschiebung mit der analytischen Lösung überein?
2. Wie sieht die Von-Mises-Spannung im Kern aus – gleichmäßig?
3. Wo ist die Spannung erhöht, und warum?

Variationen: - Elementgröße in Gmsh halbieren (`-clmax 2.0`) – ändert sich das Ergebnis? - E-Modul auf Aluminium ändern: $E = 70\,000\,\text{MPa}$ - Kraft verdoppeln – ist die Reaktion linear?