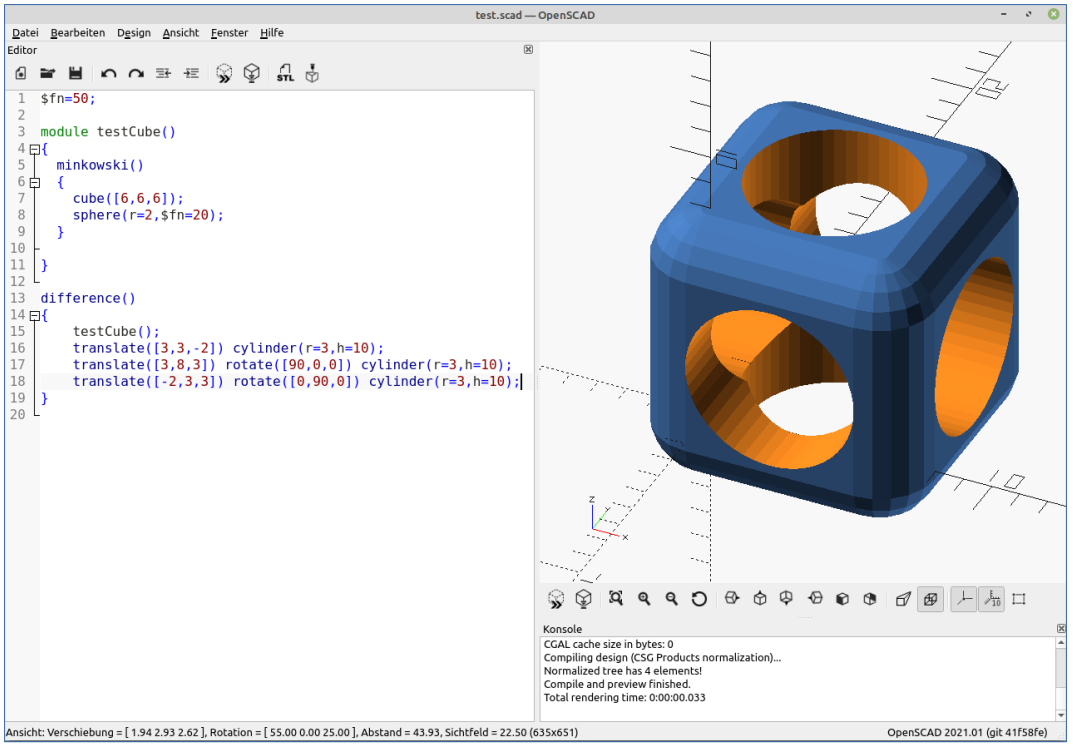
Erstellung eines 3D Objekts mit OpenSCAD



Ersteller: andimoto (www.github.com/andimoto)

Bevor es los geht...

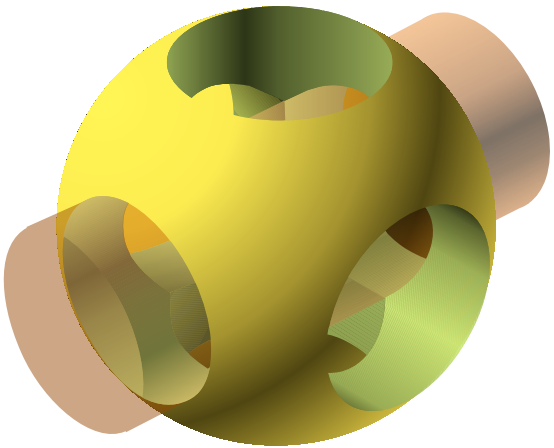
- OpenSCAD installiert?
- (optional) Lieblingseditor gestartet?

Was lernt man in diesem Workshop?

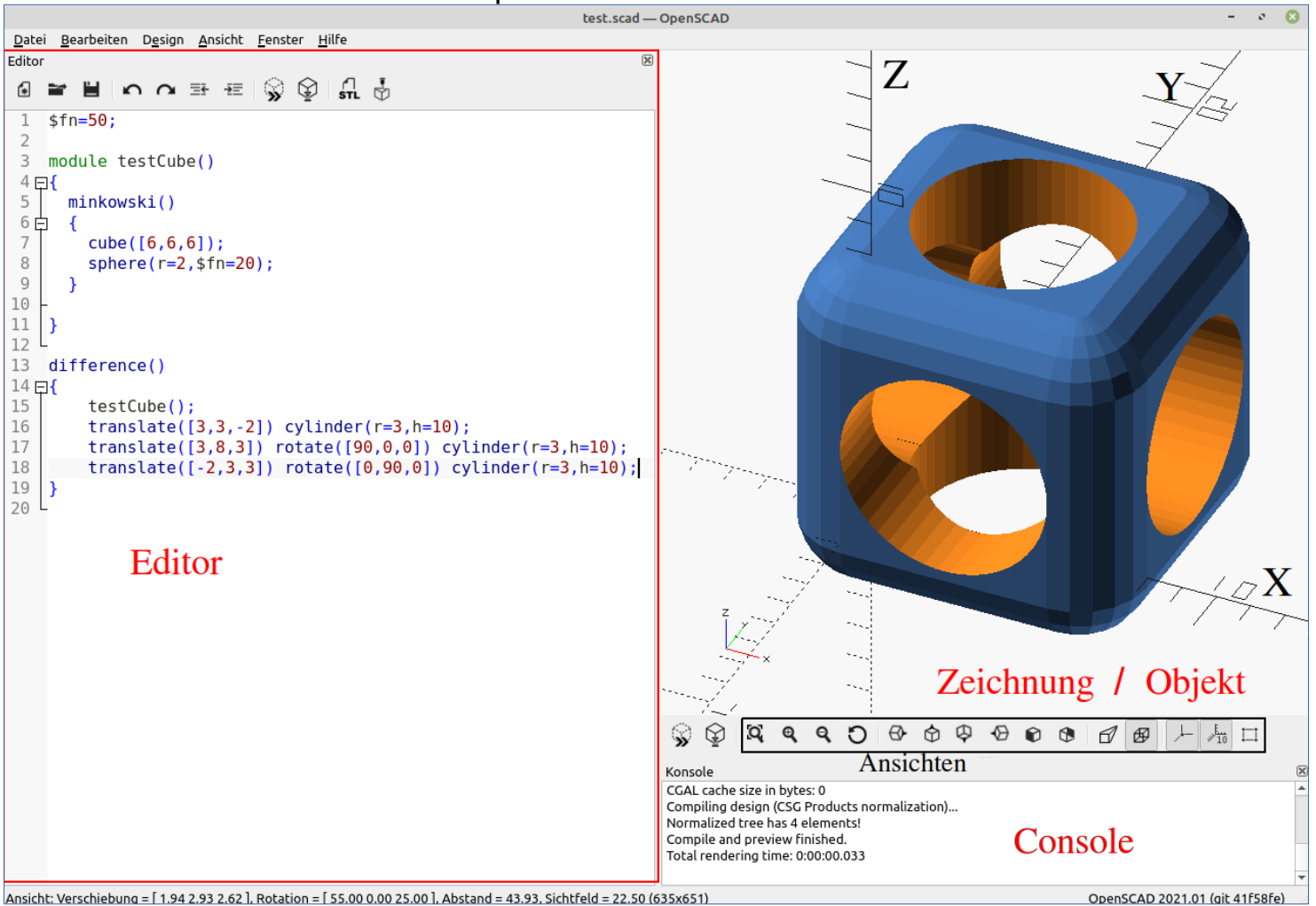
- Was ist OpenSCAD?
- Blick auf OpenSCAD – Wo ist was?
- Vorstellung grundlegender Formen, Transformationen und Funktionen durch Erstellung eines Modells

OpenSCAD

- CAD Programm zum Erstellen von (komplexen) 2D & 3D Modellen als auch zur Animation & Simulation
- Skriptbasiert → non-interactive
- Modell wird durch „Code“ beschrieben
- Modelle sind leicht parametrisierbar
- Sehr einfache Oberfläche
- Export von stl & 3mf Dateien für weitere Bearbeitung (3D Druck, etc.)



OpenSCAD – Was ist wo?



Editor oder Editor

- OpenSCAD hat eine AutoCompletion
  - Dateien werden in Tabs organisiert, keine Projektsicht möglich
- Dateien können bei Änderung automatisch neu geladen werden
  - „Design → Automatisch neu laden und Vorschau“ aktivieren
- Jeder beliebige Editor kann verwendet werden
- Viele Editoren haben Support für OpenSCAD Dateien
- Editor Plugins für OpenSCAD Dateien
  - https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Using_an_external_Editor_with_OpenSCAD

Aufbau der Syntax

- Objekte
  - 2D oder 3D Objekte
  - Objekte schließen immer mit ; ab
  - *Beispiele:*

Form	Beschreibung
`cube([20,20,20]);`	Kubus – alle Kanten 20mm
`sphere(50);`	Kugel mit Radius 50mm
`cylinder(r=2,h=50);`	Zylinder mit Radius 2mm & Höhe 50mm
`cylinder(r1=2,r2=4,h=50);`	Zylinder – unten 2mm, oben 4mm
`polygon(points=[[0,0],[0,1],[1,1]]);`	2D Polygon Fläche mit Koordinaten
`circle(d=30);`	2D Kreis mit Durchmesser 30mm

Aufbau der Syntax

- Aktionen (Actions) & Zuweisungen (Statements)
  - Aktionen schließen immer mit ; ab
  - Beispiel:

```
radius = 20;
U = 2*pi()*radius;
text="ABCDEFG";
points = [[0,0],
          [5,0],
          [0,6],
            …. ];
```

Aufbau der Syntax

Datentyp	Beschreibung
`Nummern (numbers)`	Jede Art von Nummer – 1,2,3, PI, 1.02, etc.
`Bool‘sche Werte`	Typ mit 2 möglichen Werten – Wahr oder Falsch (true/false). Werte für Falsch: **false, 0, -0, „“, [], undef**. Alle anderen Werte sind Wahr (sie existieren)
`Strings`	Das ist ein String `s=„OpenSCAD!“;`
`Bereiche (ranges)`	Bereiche geben Start und Ende als auch die Schrittweite an. Mit Nummern. Bsp: `[start:ende]` oder `[start:inkrement:ende]`
`undef`	Nicht definiert. z.B. eine fehlende Variable
`----------`	`----------`
`Variablen`	Variablen dürfen aus folgenden Zeichen bestehen: **[a-zA-Z0-9_]**

Aufbau der Syntax

- Transformationen (Operators)
  - Modifizieren Ort, Ausrichtung, Farbe, etc. eines oder mehrerer Objekte
  - Stehen immer vor einem Objekt (schließt NICHT mit ; ab)
  - Ist das „Adjektiv“ des Objekts
  - Beispiele:

Form und Transformation	Beschreibung
`translate([x,y,z]) cube([sx,sy,sz]);`	Kubus um x,y,z verschoben
`scale([2,1,1]) sphere(50);`	Kugel mit Radius 50mm um Faktor 2 in x Richtung skaliert
`rotate([0,45,0]) cylinder(r=2,h=50);`	Zylinder, gedreht um 45° um die y Achse

Aufbau der Syntax

- Module
  - module definiert ein Objekt
  - Abgeschlossene, wiederkehrende Objekte
  - analog zur Funktion/Methode in anderen Programmiersprachen
  - Kann mehrere Parameter (mit Standard Wert) haben
  - Bei Aufruf ohne Parameter werden die „default“-Werte verwendet
  - Syntax:

```
module teil001(parameter1=“default Wert“,    p2=33)
{
    cube(parameter1);
    translate([parameter1]) cylinder(r=parameter1,h=p2);
}
```

Gut zu Wissen

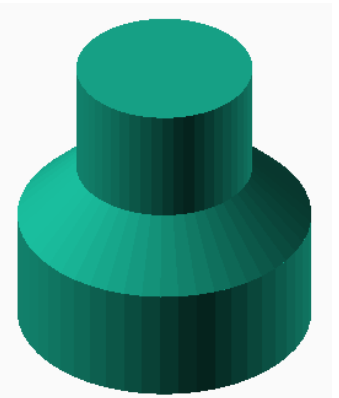
- Genereller Ablauf – *Vorschau (F5) -> Rendern (F6) -> Export (F7)*
  - Modell beschreiben
    - Das Speichern der Datei bewirkt eine Aktualisierung der Vorschau in OpenSCAD (Taste F5)
    - Das geht je nach Modell recht schnell, kann bei großen Modellen etwas dauern
  - Endgültige Prüfung und Berechnung des Modells – Rendern
    - Die Polygone des Modells werden berechnet
    - Muss mit der Taste F6 oder über das Menü angestoßen werden
    - Dauer noch länger als die Vorschau
  - Nach dem Rendern kann das Modell als STL- oder 3MF-Datei exportiert werden
    - Diese gängigen Dateiformate können dann in einen Slicer geladen werden
    - Weiter Dateiformate – DXF oder SVG für z.B. CNC, etc. (als „projection“)

Gut zu Wissen

- Die Variable **`$fn`**
  - Bestimmt die Anzahl der Polygone die für das Modell verwendet werden
    - Bsp: `$fn = 100;`
  - Je höher der Wert, desto detaillierter das Modell, desto länger benötigt das Rendern (oder auch die Vorschau)
  - Vorsichtig damit!! :)
- Debugging (Fehlersuche)
  - # vor ein Objekt markiert dieses rot
  - % vor ein Objekt deaktiviert dieses und markiert es grau
  - ! vor ein Objekt um nur dieses Objekt anzuzeigen

Übung Teil 1: Halbe Achse mit Rad

- **Gesamtlänge** der Achsenhälfte (inkl. Reifen) soll angegeben werden
- Reifenradius und Reifenhöhe sollen **veränderbar** sein
- Achsenradius und Achsenlänge sollen **veränderbar** sein
- **Zwischenteil** soll sich „**anpassen**“ (soll nicht zu steil/flach sein! Soll druckbar sein!)
- Teil soll als „module“ angelegt werden



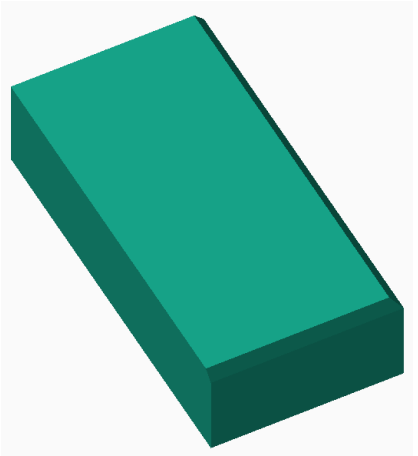
Formen und Transformationen		
`module ReifenMitAchse(...){ … }`		
`cylinder(r,h);`	`cylinder(r1,r2,h);`	`translate([x,y,z])`

Weitere Transformationen

Transformation	Beschreibung
`hull(){...};`	Legt eine Hülle um die angegebenen Objekte
`minkowski(){p1,p2};`	Rechnet Objekt 2 zu Objekt 1 hinzu. Bsp. Kanten eines Würfels abrunden
`resize([x,y,z]) object();`	Vergrößert das Objekt auf die angegebenen Absolutwerte
`mirror([x,y,z]) object();`	Spiegelt das Objekt entlang der ausgewählten Achsen. Mit ‚1‘ wird die Auswahl gesetzt

Übung Teil 2: Karosserie ;)

- Länge der Karosserie soll angegeben werden
- Höhe soll angegeben werden können (liegend!)
- Halbe Breite soll angegeben werden (liegend!)
- Die obere und hintere Kante soll abgeschrägt werden
- Teil soll als „module“ angelegt werden



Formen und Transformationen		
`module BusHinten(...){ … }`		
`hull(){...};`	`cube([x,y,z]);`	`translate([x,y,z])`

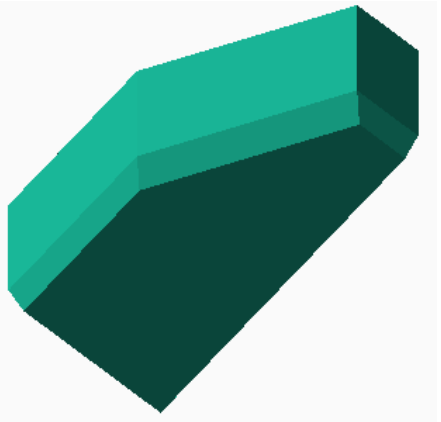
Weitere Funktionen

Objekt als Transformation (2D → 3D)	Beschreibung
`linear_extrude(height, center, convexity, twist) 2d_object();`	Macht aus einem 2D Objekt ein 3D Objekt mit einer Höhe. 2D Objekte liegen auf xy Ebene und werden in z Richtung extrudiert.
`rotate_extrude(angle, convexity) 2d_object();`	Extrudiert das 2D Objekt um die z Achse in einem angegebenen Winkel (oder 360°). 2D Objekt wird dazu auf xz Ebene „aufgestellt“.

Anmerkung (nicht Teil dieses Workshops):

- Die Transformation „rotate_extrude()“ wird nicht weiter verwendet
- „convexity“ dient nur zur korrekten Darstellung und hat keinen Einfluss auf das Modell
- „center“ bestimmt die Lage auf der xy Ebene. Wert „false“ legt Bbjekt **auf** die xy Ebene. Wert „true“ legt xy Ebene mittig durch das Objekt

Übung Teil 3: Front ;)



- Front soll mit Polygon angegeben werden
- Die obere und fordere Kante soll abgeschrägt werden
- Teil soll als „module“ angelegt werden

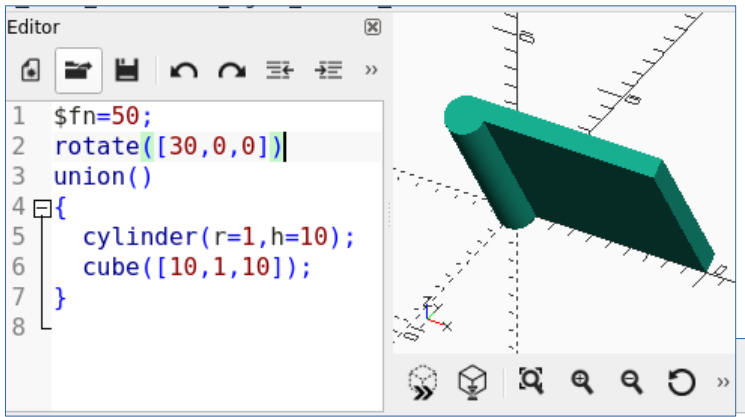
Formen und Transformationen		
`module BusVorne(...){...}`		
`hull(){...};`	`polygon(points);`	`translate([x,y,z])`
`linear_extrude(...)`		
`polyPoints = [[x0,y0],[x1,y1],[x2,y2],...];`		

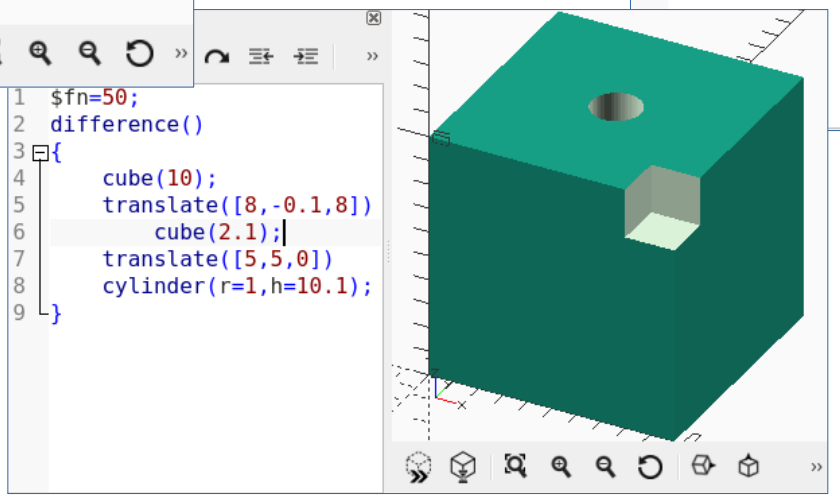
Bool'sche Operationen

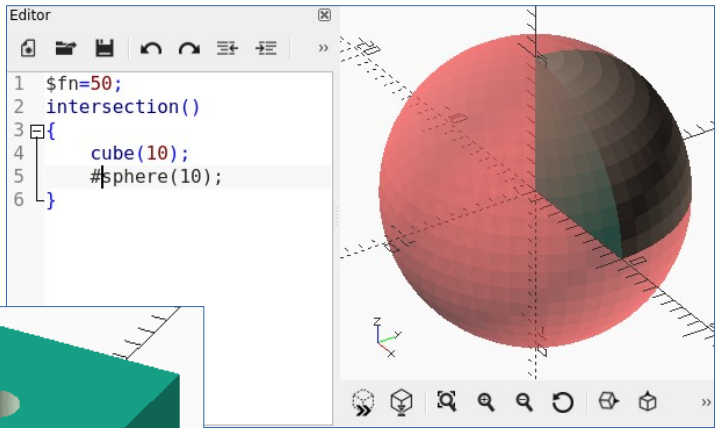
Bool'sche Operation	Beschreibung
`union() { p1(); p2(); ...}`	**ODER** Operation. Gruppiert alle inneren Objekte zu einer Einheit.
`difference() { p1(); p2(); ...}`	**UND NICHT** Operation. Zieht vom 1. Objekt alle weiteren Objekte ab, welche die Form des 1. Objekts kreuzen.
`intersection() { p1(); p2(); ...}`	**UND** Operation. Die kreuzenden Volumen aller inneren Objekt bilden eine Einheit → neues Objekt.

Anmerkung: Alle weiteren Transformationen und Operationen werden auf die gesamte Einheit angewendet. Jede bool'sche Operation bildet, anders gesagt, ein Objekt (welches aber nicht aufgerufen werden kann).

Bool'sche Operationen







Übung Teil 4: halbes Bus‘le

- Alle 3 Teile sollen so zusammen gesetzt werden, dass ein halbes Bus‘le entsteht
- Die Achsen und Räder sollen vom Fahrzeug „entkoppelt“ sein - sie sollen drehbar sein
- Fahrzeug soll parametrisierbar sein
- Fahrzeug soll als „module“ angelegt werden

Formen und Transformationen		
`module van(...){...}`		
`ReifenMitAchse(...);`	`BusHinten(...);`	`BusVorne(...);`
`hull(){...};`	`difference(){...};`	`translate([x,y,z])`

Übung Teil 4.1: ganzes Bus‘le

- Es soll ein komplettes Fahrzeug zusammen gesetzt und gerendert werden
- Fertiges Model soll als STL Datei exportiert werden

Formen und Transformationen	
`module halbesBusle(){...}`	
`mirror([x,y,z]){...};`	`translate([x,y,z])`

Extra: Bus‘le slicen

- Das fertige Model (STL/3MF) kann in einen Slicer geladen werden
  - Slicer „schneiden“ ein 3D Model in Schichten, welche ein 3D Drucker nacheinander drucken kann
  - Slicer erstellen G-Code, ein Format für CNC Maschinen (inkl. 3D Drucker)
  - 3D Drucker lesen G-Code und führen die „Bewegungen und Befehle“ aus
  - Es gibt verschiedene Slicer – PrusaSlicer, SuperSlicer, Cura, Simplify3D, etc.

  Als Extra kann das Bus‘le im Online-Slicer Kiri geladen und gesliced werden

  https://grid.space/kiri/

Extra: Grafische SCAD Programme

- Es gibt weitere Programme mit diesem Ansatz bzw. graphische UIs für OpenSCAD
  - TinkerCAD (Autodesk) – www.tinkercad.com
    - Basisformen mit verschiedenen Transformationen & Operationen
    - kostenlos, cloud-basiert, Anmeldung nötig (z.B. google, apple)
    - Export zu STL
  - BlocksCAD - www.blockscad3d.com (ähnlich dazu: MakeCode)
    - Basisformen mit verschiedenen Transformationen & Operationen
    - blockbasierte Programmiersprache – ähnlich zu Skratch (Raspberry Pi, etc)
    - kostenlos, Anmeldung nur zum Speichern nötig, Export zu STL möglich
  - OpenSCAD Graph Editor - https://github.com/derkork/openscad-graph-editor
    - Design mit Nodes
    - momentan in am Anfang der Entwicklung, noch fehlerhaft
    - OpenSCAD muss installiert sein
- Weitere: RapCAD, Antimony, ImplicitCAD, OpenJSCAD, CADQuery, AngelCAD, etc.