RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

OpenSCAD for 3D Printing

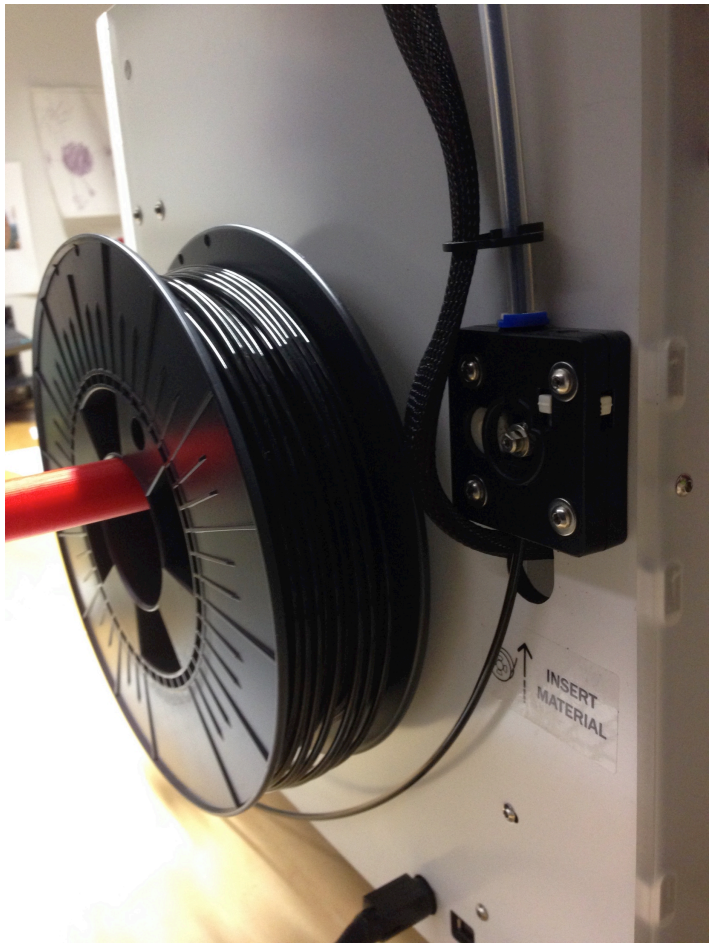
Peter Fischer
ZITI, Universität Heidelberg

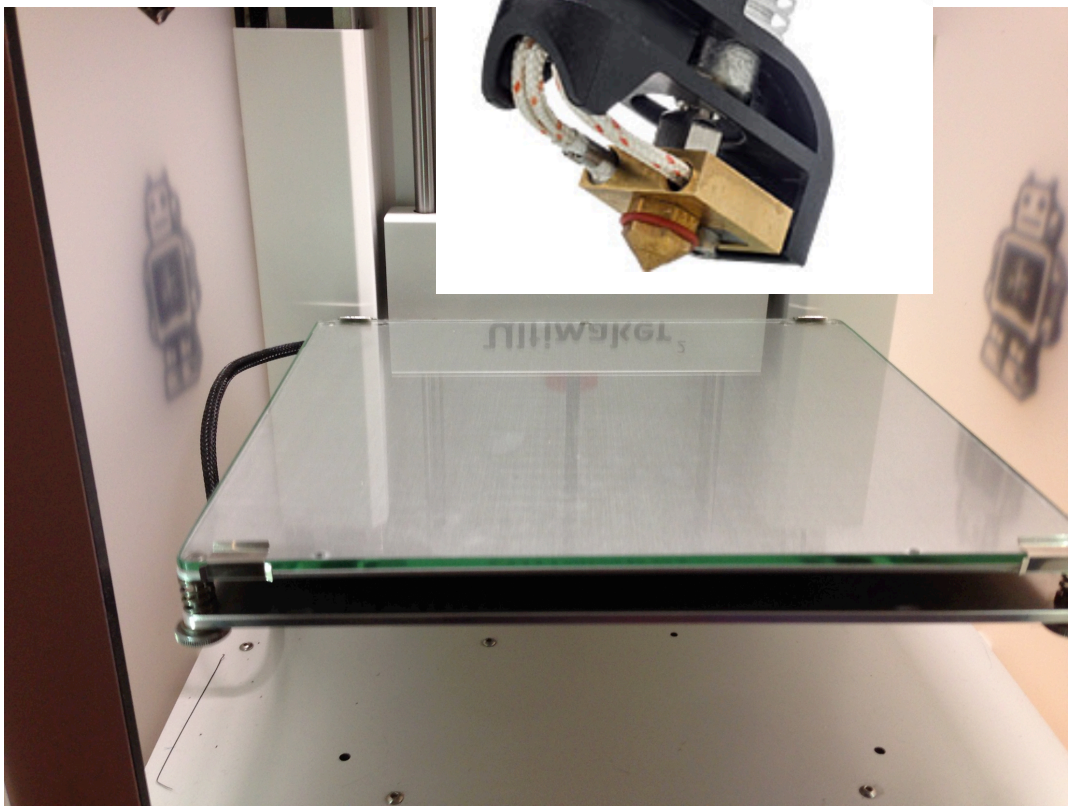
Wie funktioniert ein 3D Drucker ?

- z.B. so ähnlich wie eine Spritztüte:

Details

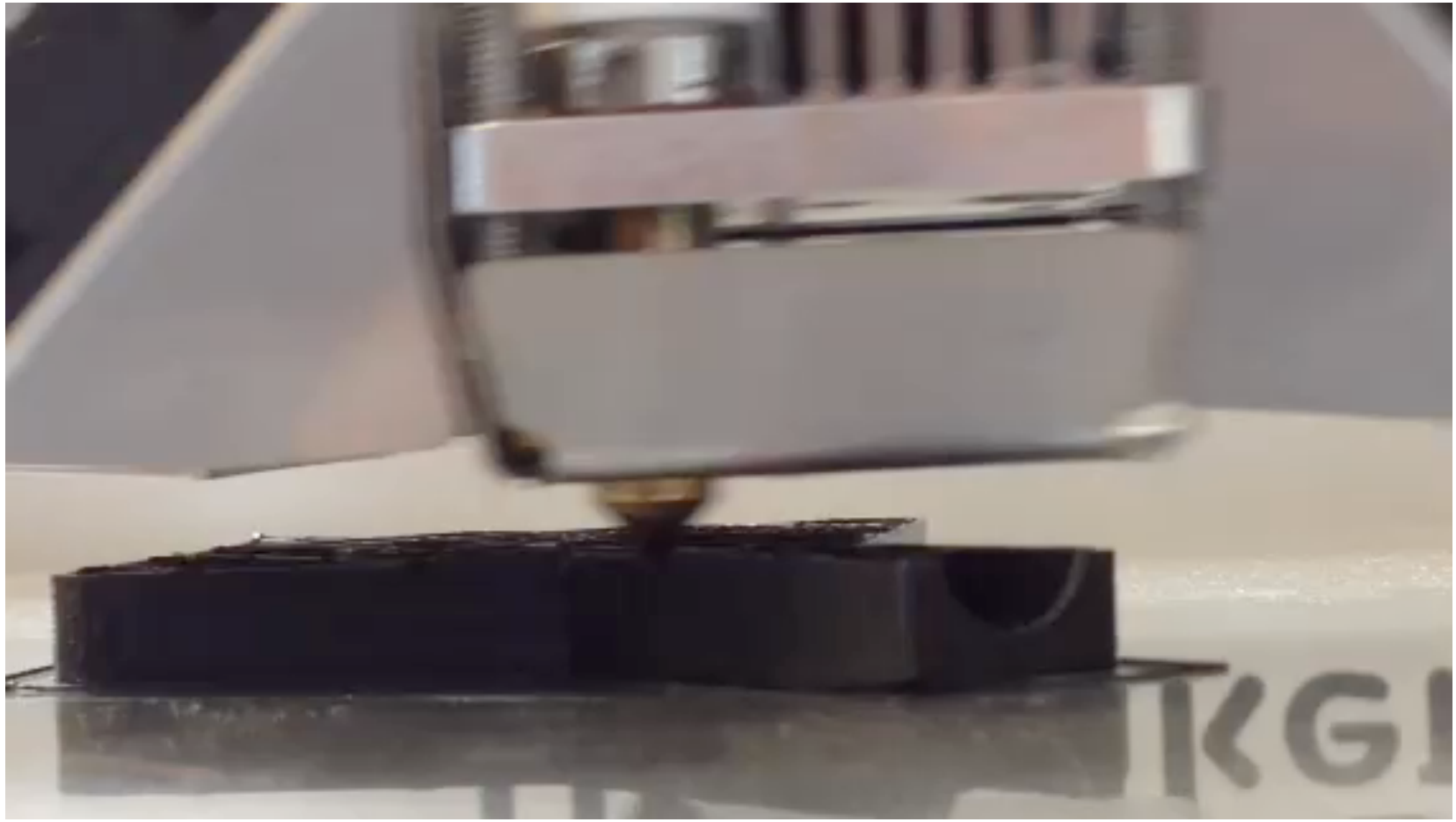
- Kunststoff-Faden (‘filament’) wird durch eine heiße Düse gepresst
- Verschiedene Materialien möglich





Filament 3D Printer

- Typische Dicke einer Lage ist 0.1-0.2 mm
- Inneres wird meist nicht ganz ausgefüllt



Ablauf

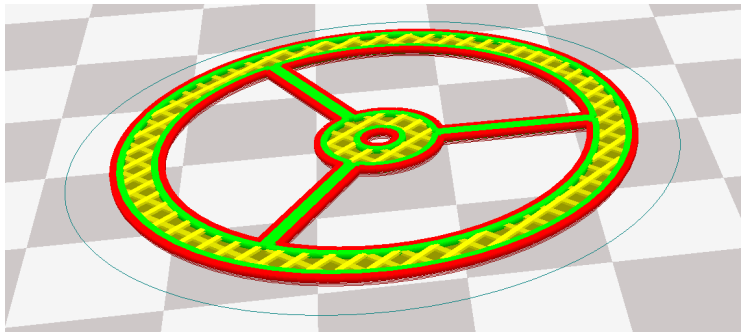
1. Beschreibung der Geometrie
   - 3D Konstruktionsprogramm
   - Blender
   - Textfile (z.B. OpenSCAD)
   - Download aus Internet (z.B. Thingsiverse)

WIR

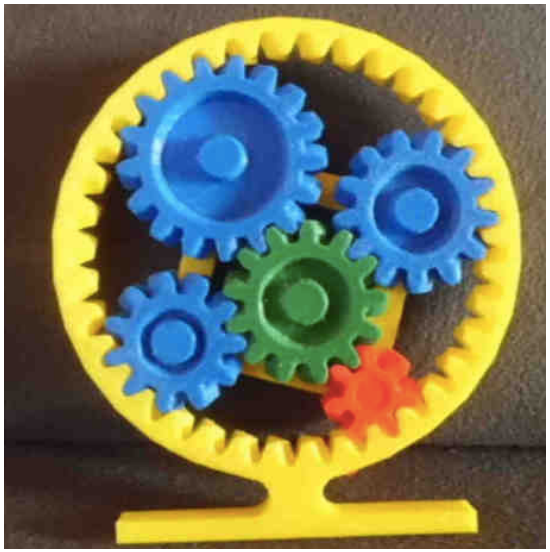
   - Ergebnis: file *.stl

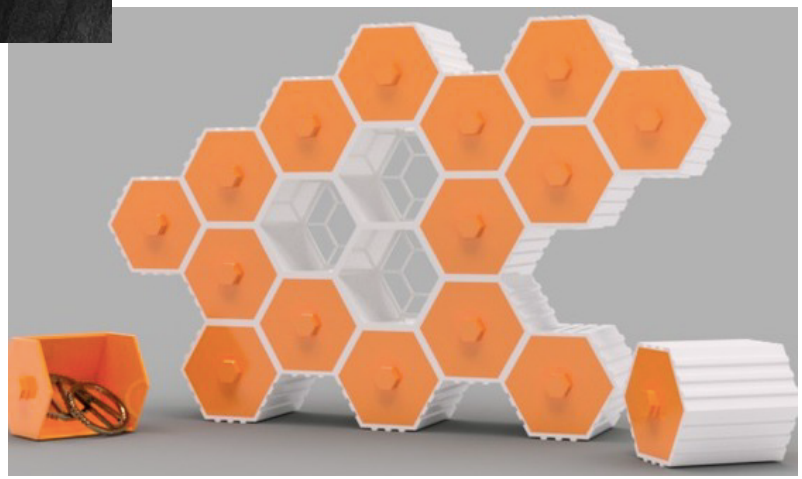
```
solid OpenSCAD_Model
  facet normal 0.629321 0.777146 -0
    outer loop
      vertex 13.3826 14.8629 0
      vertex 11.7557 16.1803 2
      vertex 13.3826 14.8629 2
    endloop
  endfacet
  facet normal 0.629321 0.777146 0
    outer loop
      vertex 11.7557 16.1803 2
      vertex 13.3826 14.8629 0
      vertex 11.7557 16.1803 0
```

1. Umwandlung der Geometriedaten in Befehle für den Drucker
   - ‘Slicer’, z.B. Cura
   - Ergebnis: file *.gcode



Thingsiverse

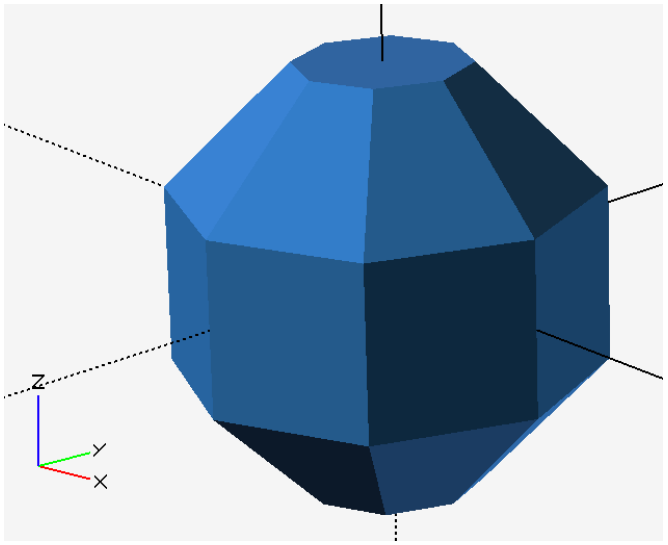




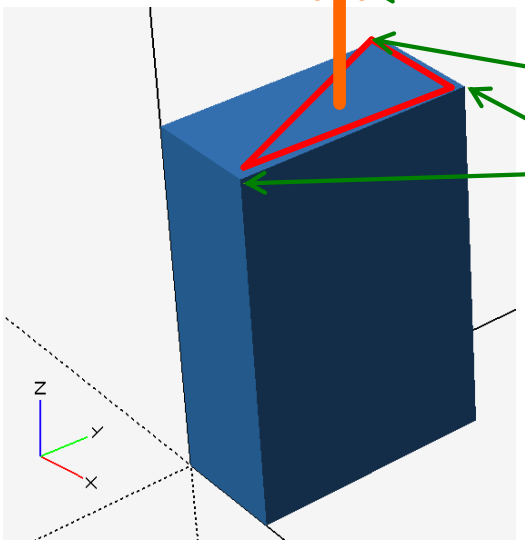
- Modelle z.T. parametrisierbar

.stl files

- ‚**S**tandard **T**esselation (or **T**riangle) **L**anguage‘
  (oder **ST**ereo**L**ithographie)
- Beschreibt Oberflächen durch **Dreiecke** (also oft nur eine Annäherung an die exakte Fläche)



‘Kugel’




```
solid OpenSCAD_Model
  facet normal -0 0 1
    outer loop
      vertex 0 2 3
      vertex 1 0 3
      vertex 1 2 3
    endloop
  endfacet
  facet normal 0 0 1
    outer loop
      vertex 1 0 3
      ...
      ...
```

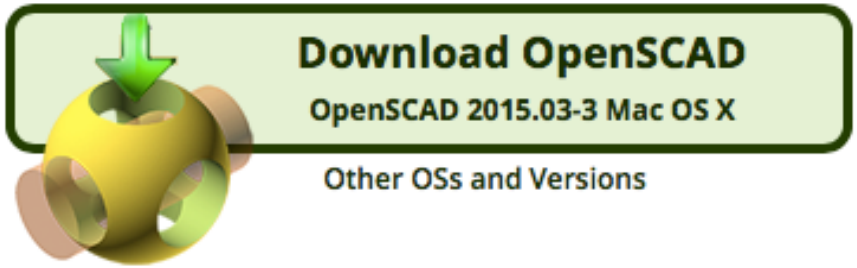
RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

OpenSCAD

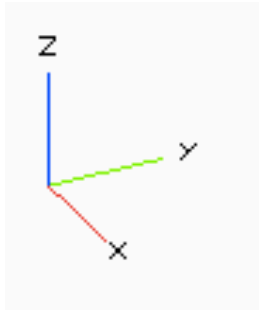
Unser 'Werkzeug' zum Erzeugen der Formen

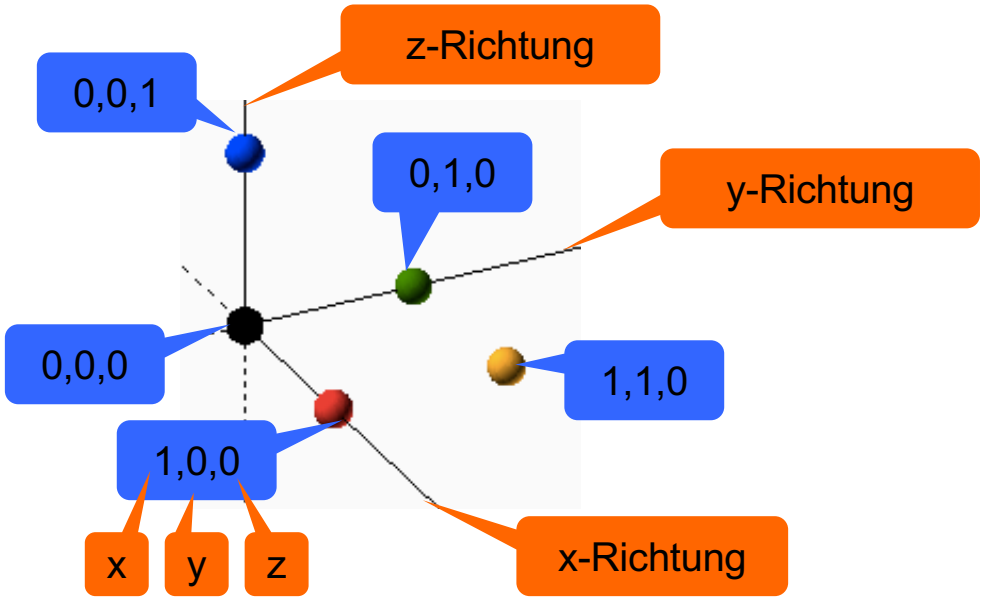
- Wir benutzen das (kostenlose) Programm 'OpenSCAD'

Online:
http://openscad.net



- Punkte im Raum werden durch 3 KOORDINATEN beschrieben ('x,y,z')



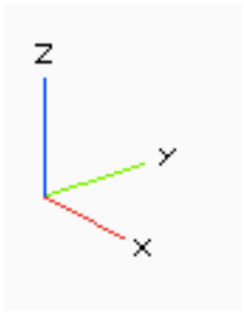


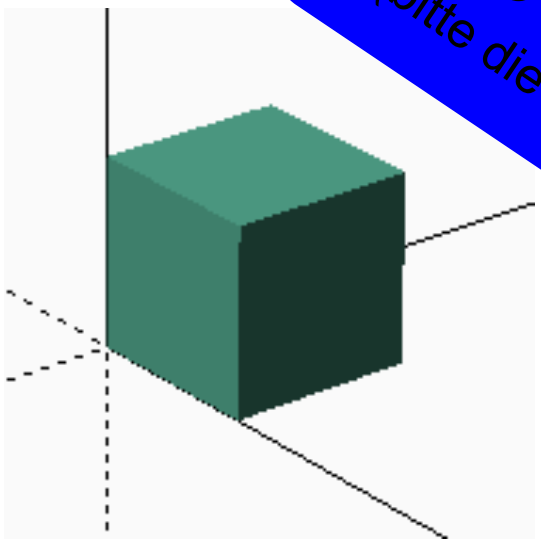
Würfel

- Einen Würfel definiert man mit

```
cube([10,10,10]);
```

- F5 drücken -> Anzeige:





AUSPROBIEREN!
(bitte die Zeile genau so abtippen!)

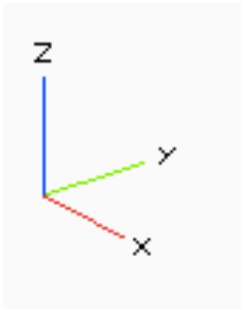
- Mit Maus drehen und Zoomen…

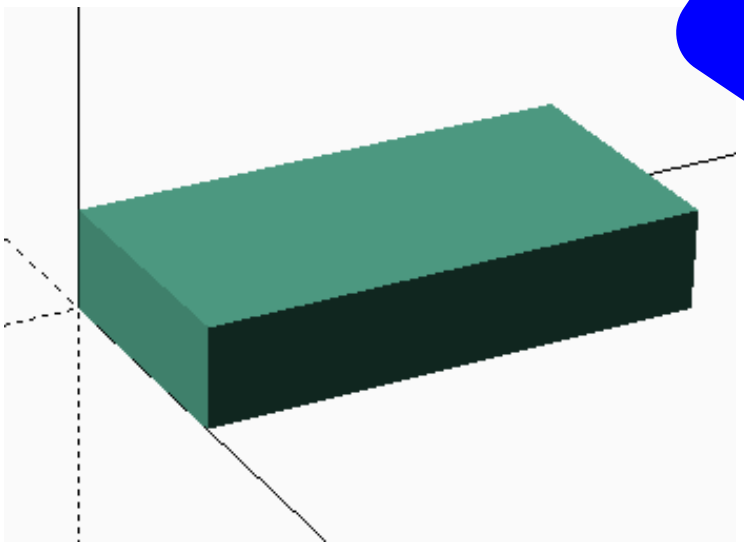
Ein Würfel…. Na ja.... Nicht so spannend...

Quader

- Die drei Werte im '**cube**' Befehl geben die Größe in x,y,z an:

```
cube([5,10,2]);
```





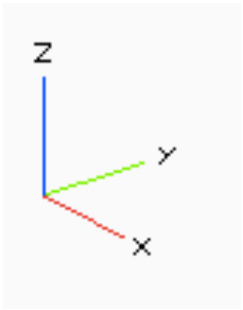
AUSPROBIEREN!
probiert auch andere Zahlen aus!

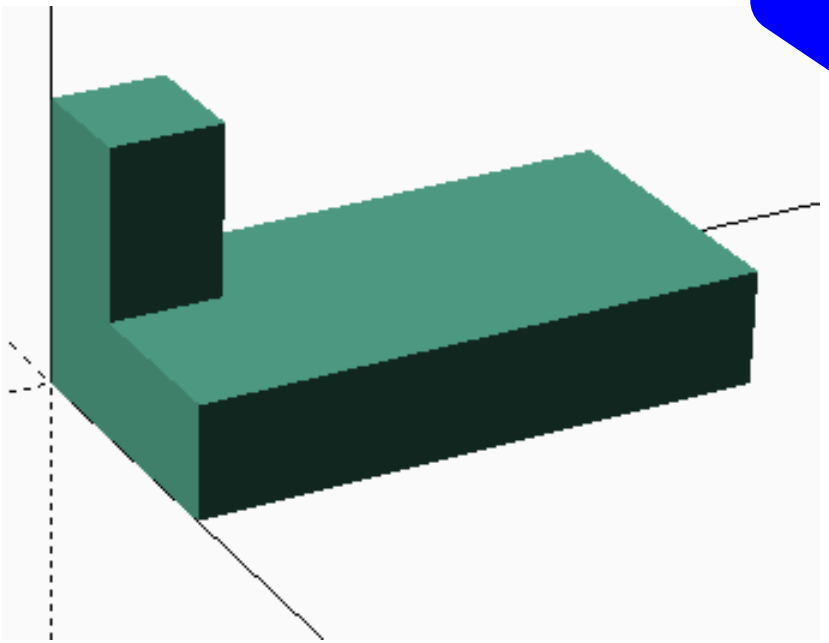
Ok gut…. schon besser.... Aber nur EINER ?

Zwei Quader

- Man kann mehrere Objekte definieren.
  - Jede Zeile muss mit einem '**;**' enden!!!!

```
cube([5,10,2]);
cube([2,2,5]);
```





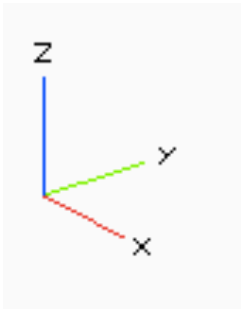
AUSPROBIEREN!

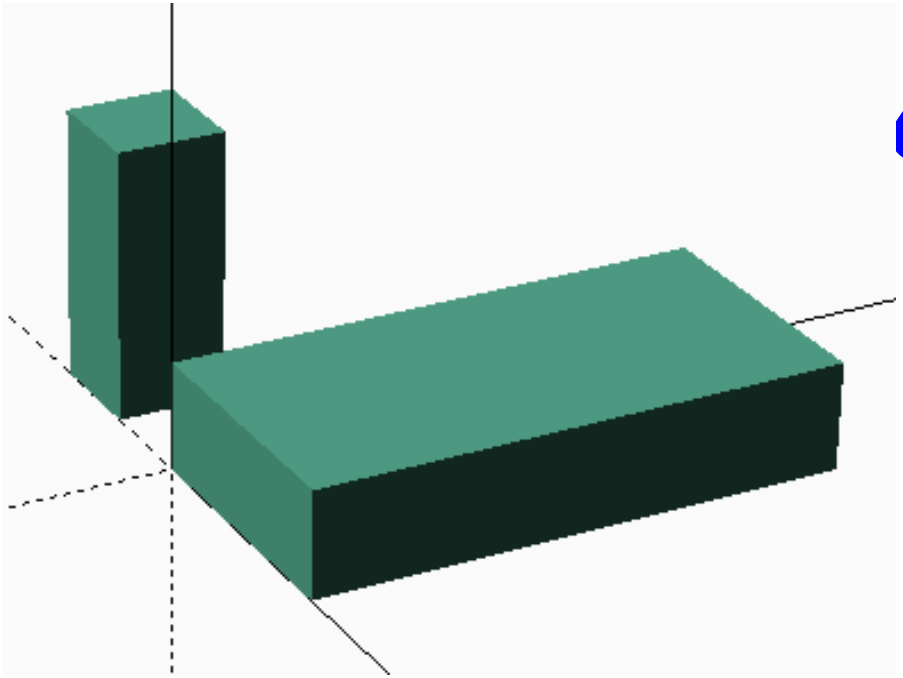
Nett. Aber die sind ja übereinander! Wie blöd!

Objekte bewegen

- Objekte kann man mit `translate([x,y,z])` verschieben:
  - Kein '`;`' hinter `translate()`, erst hinter dem Objekt!

```
cube([5,10,2]);
translate([-4,0,0]) cube([2,2,5]);
```



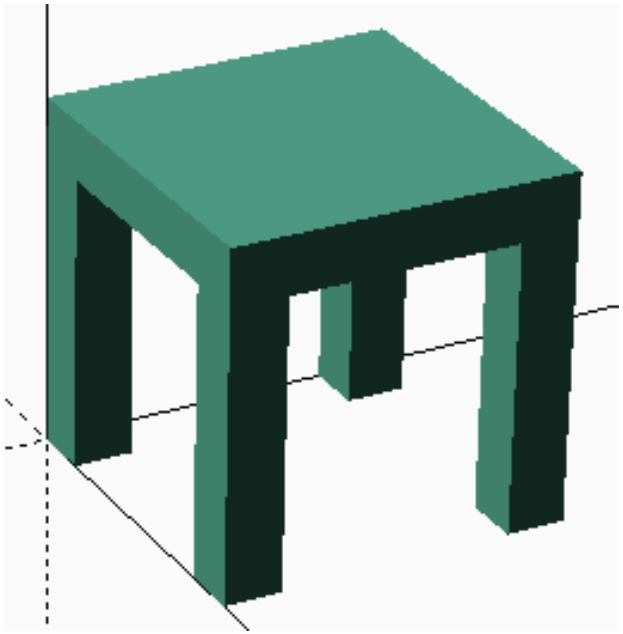


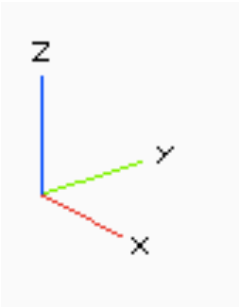
AUSPROBIEREN!

Aber kann man damit denn was machen ?

Aufgabe

- Wer schafft so einen Hocker:






```
cube([1,1,5]);
translate([5,0,0]) cube([1,1,5]);
translate([0,5,0]) cube([1,1,5]);
translate([5,5,0]) cube([1,1,5]);
translate([0,0,5]) cube([6,6,1]);
```

Ok.... Nicht schlecht... Und wenn ich noch einen für meine Freundin brauche, muss ich dann alles nochmal tippen?

Objekte gruppieren

- Mehrere Objekte kann man in einem **`module name(){…}`** zusammenfassen:

```
module Hocker ()
{
  cube([1,1,5]);
  translate([5,0,0]) cube([1,1,5]);
  translate([0,5,0]) cube([1,1,5]);
  translate([5,5,0]) cube([1,1,5]);
  translate([0,0,5]) cube([6,6,1]);
}
```

- Dann kann man den **`Hocker`** wie einen **`cube`** benutzen:

```
Hocker();
translate ([0,-10,0]) Hocker();
```

Jetzt wird es langsam spannend.. Aber auch kompliziert...
Und alles ist so eckig!

Jetzt wird's rund!

```
$fn=60;
translate([0,0,7]) sphere(r=2);
cylinder(r=1,h=4);
```

AUSPROBIEREN!

sphere und **cylinder** sind klar.
Die Werte dahinter ('Parameter') verändern die Form.
Gar nicht so schwer...

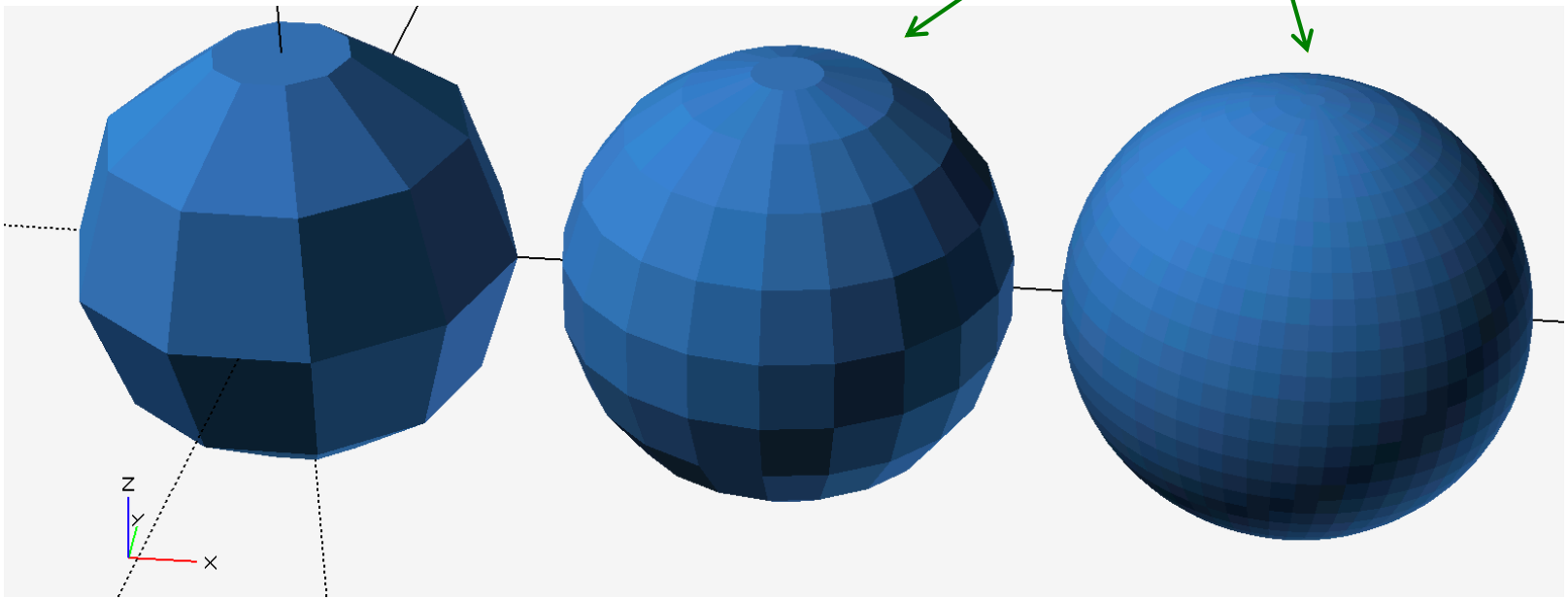
Aber was macht der komische Befehl in der ersten Zeile ?

Ändert doch mal die Zahl dort (z.B. auf 10)!

$fn = …

- Parameter, der Anzahl Flächenelemente vorgibt.
- Kann global definiert werden oder in einzelnen Shapes

```
R=1;
$fn = 10;
sphere(r=R);
translate([2.2*R,0,0]) sphere(r=R, $fn=20);
translate([4.4*R,0,0]) sphere(r=R, $fn=60);
```

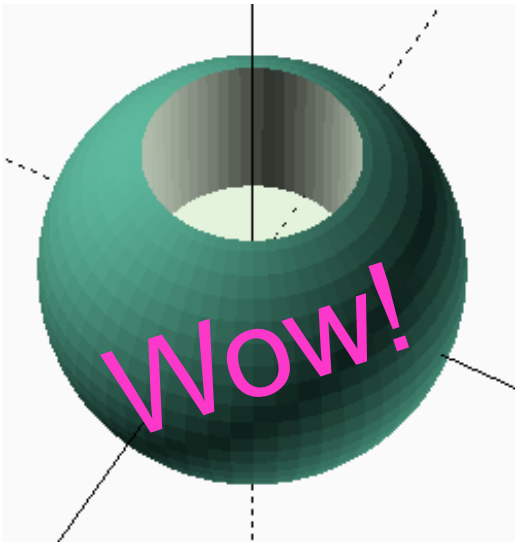


Schnipp Schnapp

- Man kann von Objekten etwas 'abschneiden':

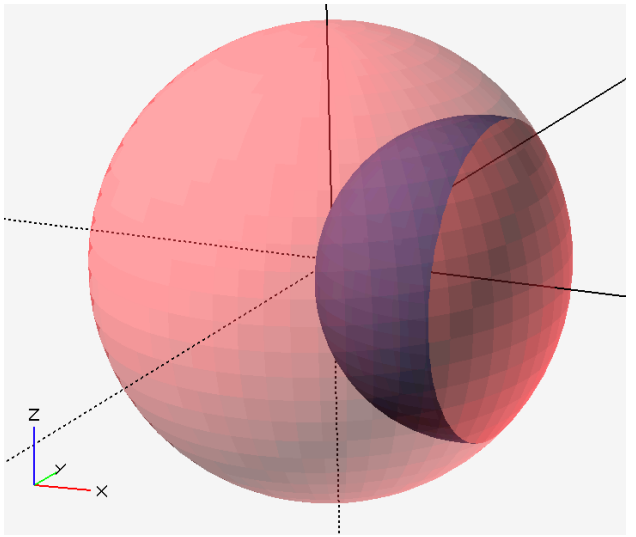
```
$fn=60;
difference () {
  sphere(r=4);
  cylinder(r=2,h=6);
}
```

AUSPROBIEREN!



- Ähnlich:

```
$fn=60;
intersection () {
#sphere(r=3);
translate([2,0,0])
  sphere(r=2);
}
```

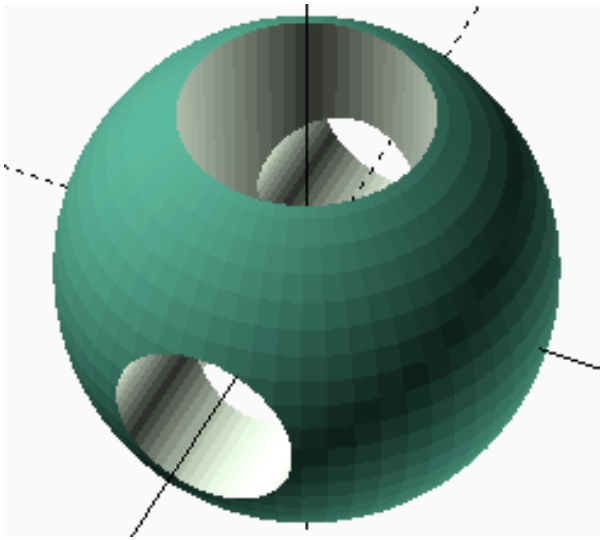


Zwei Löcher - Drehung

- Für ein Loch 2 muss man den Bohr- `cylinder` vorher drehen:

```
$fn=60;
difference () {
  sphere(r=4);
  cylinder(r=2,h=10,center=true);
  rotate([0,90,0]) cylinder(r=1.5,h=10,center=true);
}
```

- die drei Werte im `rotate` Befehl sind die Drehwinkel um die x-, y-, und z-Achsen.
- ('`center=true`' bewirkt, dass der `cylinder` nach oben und unten steht)



AUSPROBIEREN!

Noch ein paar Tipps

- Schreibt man ‘`!` vor eine Zeile, so wird nur diese Zeile ausgeführt. Nützlich um Shapes zu kontrollieren.
- Schreibt man '`#`' vor ein Objekt, so sieht man es, auch wenn es nur zum Schneiden benutzt wird.

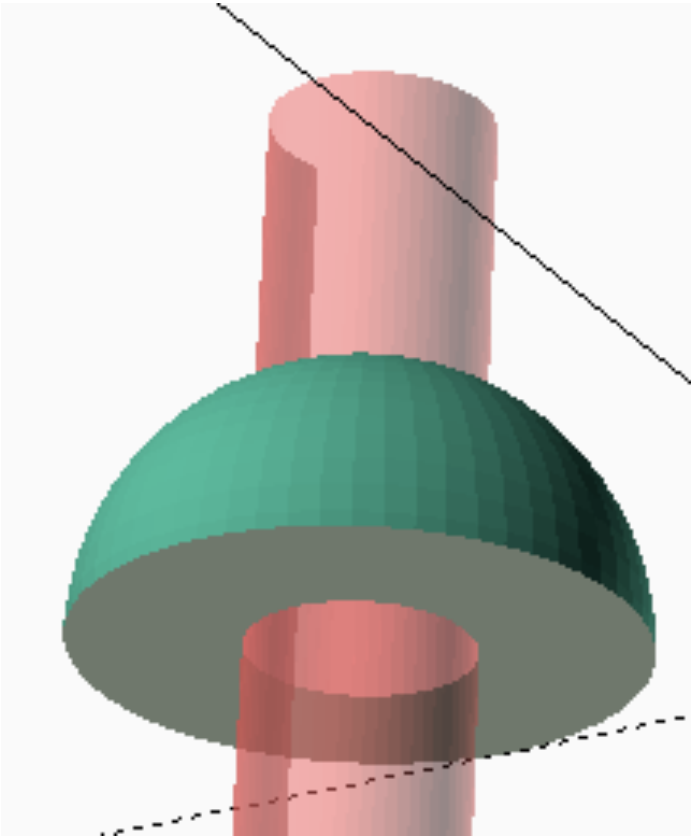
- Wenn man eine Größe oft braucht, kann man sich mit

```
v = 5;
```

eine ‘variable’ = Platzhalter `v` (oder anderer Name) definieren, den man dann überall benutzen kann, z.B.

```
cube([v,v,v]);
```

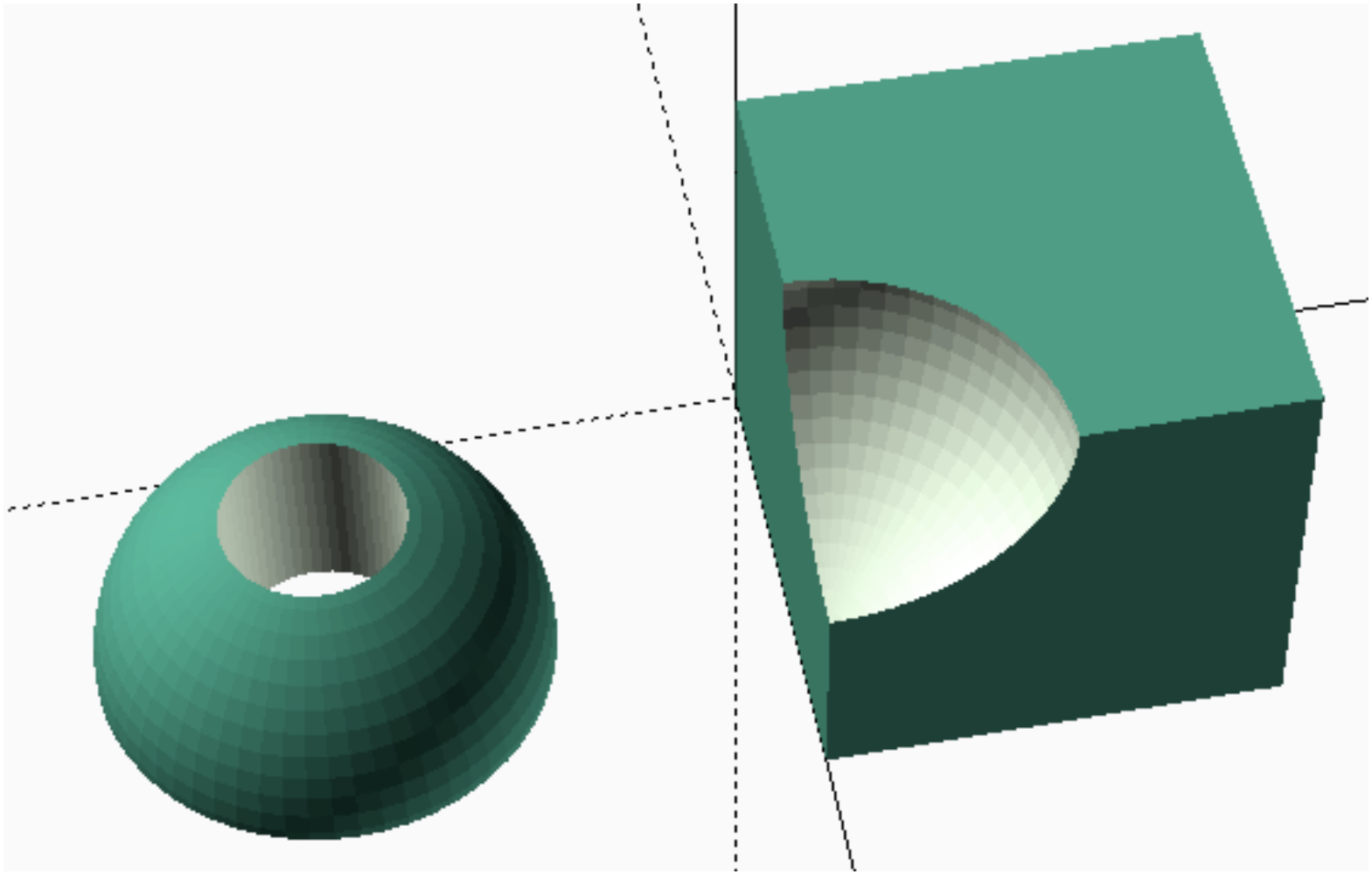
- Mathe ist möglich



Zusammenfassung Befehle bisher

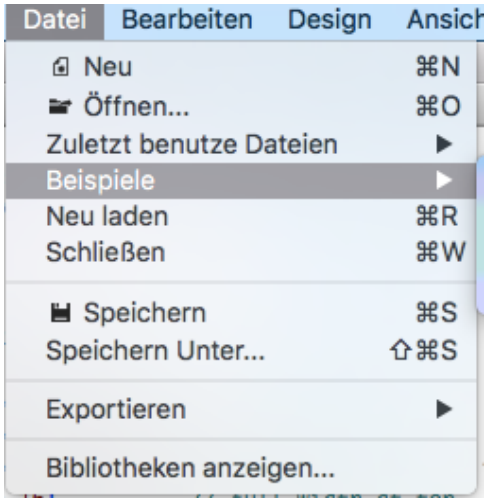
```
// Kommentar
/* mehrzeiliger Kommentar */

variable = Wert;                    // z.B. H = 3;

module (param=default,...) { ... }

cube([x,y,z]);
sphere(r=..);
cylinder(r=..,h=..);// entlang z-Achse

translate([x,y,z]) {...}
scale([x,y,z]) {..}
rotate([anglex,angley,anglez]) {...}

difference() {Grundform; Schnitt1; Schnitt2; ...}
```

Aufgabe



Noch mehr Tipps

- Unter



gibt es viele Beispiele

- https://www.openscad.org/documentation.html

https://www.openscad.org/cheatsheet/index.html

Syntax

```
var = value;
var = cond ? value_if_true : value_if_false;
module name(…) { … }
name();
function name(…) = …
name();
include <….scad>
use <….scad>
```

Constants

undef	undefined value
PI	mathematical constant π (~3.14159)

Special variables

$fa	minimum angle
$fs	minimum size
$fn	number of fragments
$t	animation step
$vpr	viewport rotation angles in degrees
$vpt	viewport translation
$vpd	viewport camera distance
$children	number of module children
$preview	true in F5 preview, false for F6

Modifier Characters

*	disable
!	show only
#	highlight / debug
%	transparent / background

2D

```
circle(radius | d=diameter)
square(size,center)
square([width,height],center)
polygon([points])
polygon([points],[paths])
text(t, size, font,
     halign, valign, spacing,
     direction, language, script)
import("….ext")
projection(cut)
```

3D

```
sphere(radius | d=diameter)
cube(size, center)
cube([width,depth,height], center)
cylinder(h,r|d,center)
cylinder(h,r1|d1,r2|d2,center)
polyhedron(points, faces, convexity)
import("….ext")
linear_extrude(height,center,convexity,twist,slices)
rotate_extrude(angle,convexity)
surface(file = "….ext",center,convexity)
```

Transformations

```
translate([x,y,z])
rotate([x,y,z])
rotate(a, [x,y,z])
scale([x,y,z])
resize([x,y,z],auto)
mirror([x,y,z])
multmatrix(m)
color("colorname",alpha)
color("#hexvalue")
color([r,g,b,a])
offset(r|delta,chamfer)
hull()
minkowski()
```

Boolean operations

```
union()
difference()
intersection()
```

List Comprehensions

```
Generate [ for (i = range|list) i ]
Generate [ for (init;condition;next) i ]
Flatten [ each i ]
Conditions [ for (i = …) if (condition(i)) i ]
Conditions [ for (i = …) if (condition(i)) x else y ]
Assignments [ for (i = …) let (assignments) a ]
```

Flow Control

```
for (i = [start:end]) { … }
for (i = [start:step:end]) { … }
for (i = […,…,…]) { … }
for (i = …, j = …, …) { … }
intersection_for(i = [start:end]) { … }
intersection_for(i = [start:step:end]) { … }
intersection_for(i = […,…,…]) { … }
if (…) { … }
let (…) { … }
```

Type test functions

```
is_undef
is_bool
is_num
is_string
is_list
```

Other

```
echo(…)
render(convexity)
children([idx])
assert(condition, message)
assign (…) { … }
```

Functions

```
concat
lookup
str
chr
ord
search
version
version_num
parent_module(idx)
```

Mathematical

```
abs
sign
sin
cos
tan
acos
asin
atan
atan2
floor
round
ceil
ln
len
let
log
pow
sqrt
exp
rands
min
max
norm
cross
```

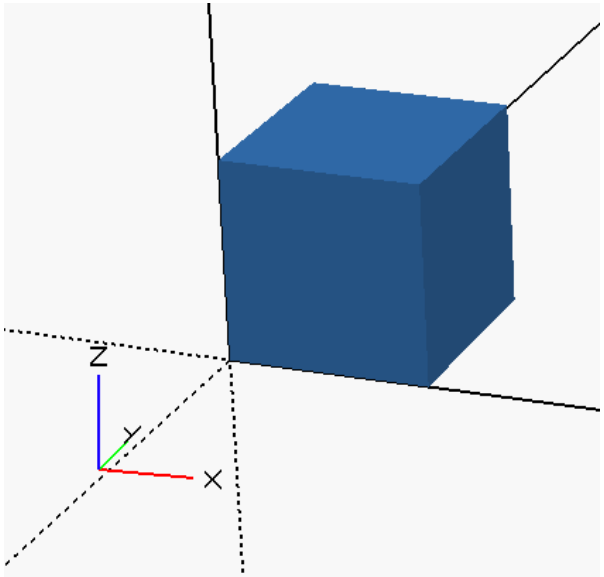
Ein paar fortgeschrittene Themen

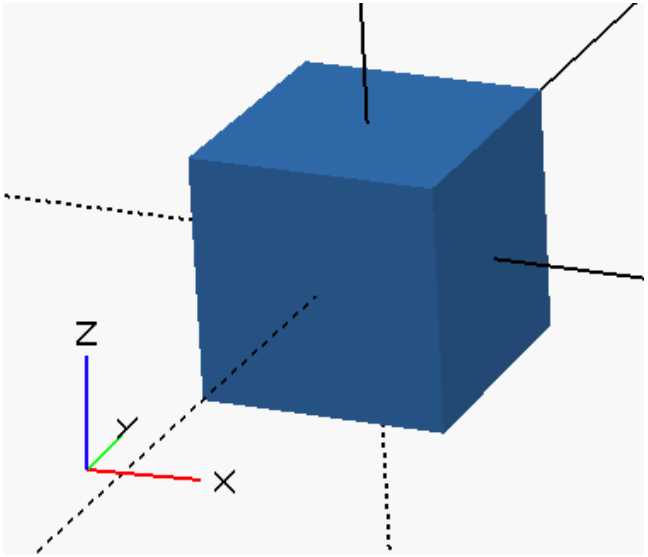
Option ‘center = true’

- Legt z.B. fest, ob cube() im ersten Quadranten liegt, oder um Ursprung zentriert ist.

```
cube([1,1,1]);
```

```
cube([1,1,1], center=true);
```

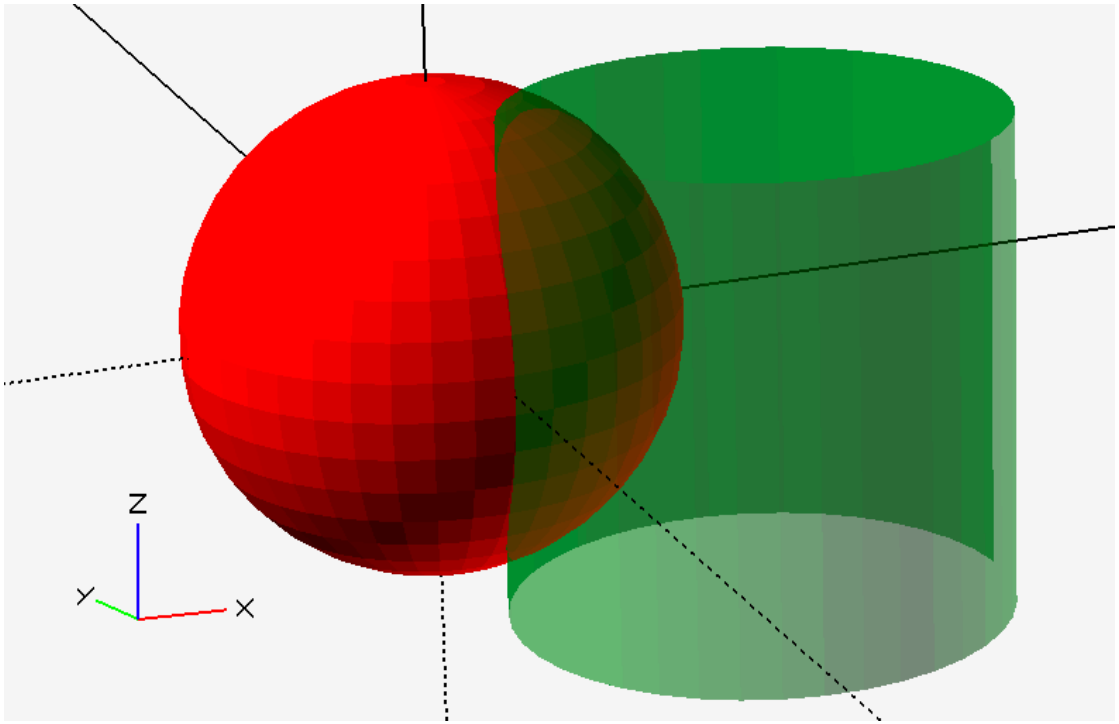




Farbe und Durchsichtigkeit

- Nur zur Darstellung. Es wird nur eine Form erzeugt.

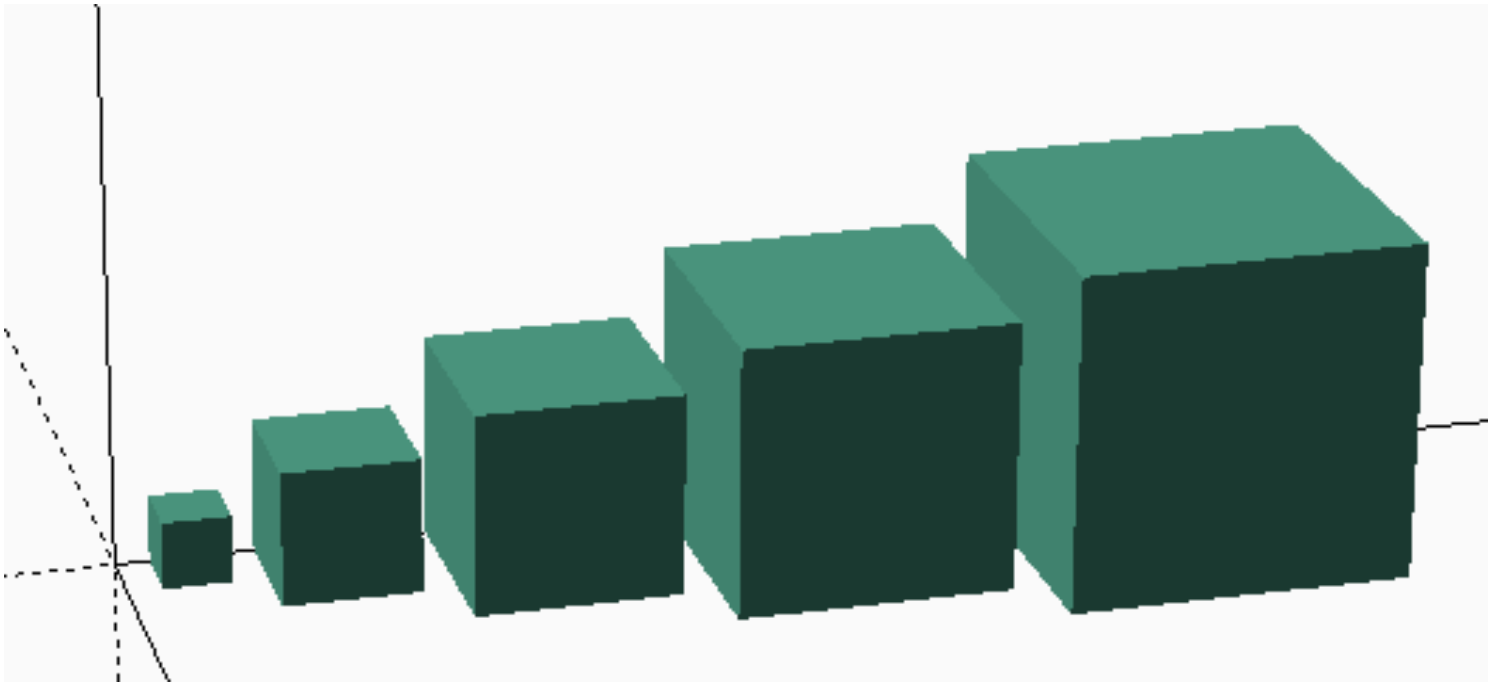
```
$fn = 40;
color("red") sphere(r=1);
color("green",0.5) translate([1,-1,-1]) cylinder(r=1, h=2);
```



Viele Objekte: for - Schleife

- Wenn man viele Objekte braucht, kann man deren 'Erzeugung' automatisieren:

```
for (i=[0:1:5]) translate([0,i*i/2,0]) cube([i,i,i]);
```

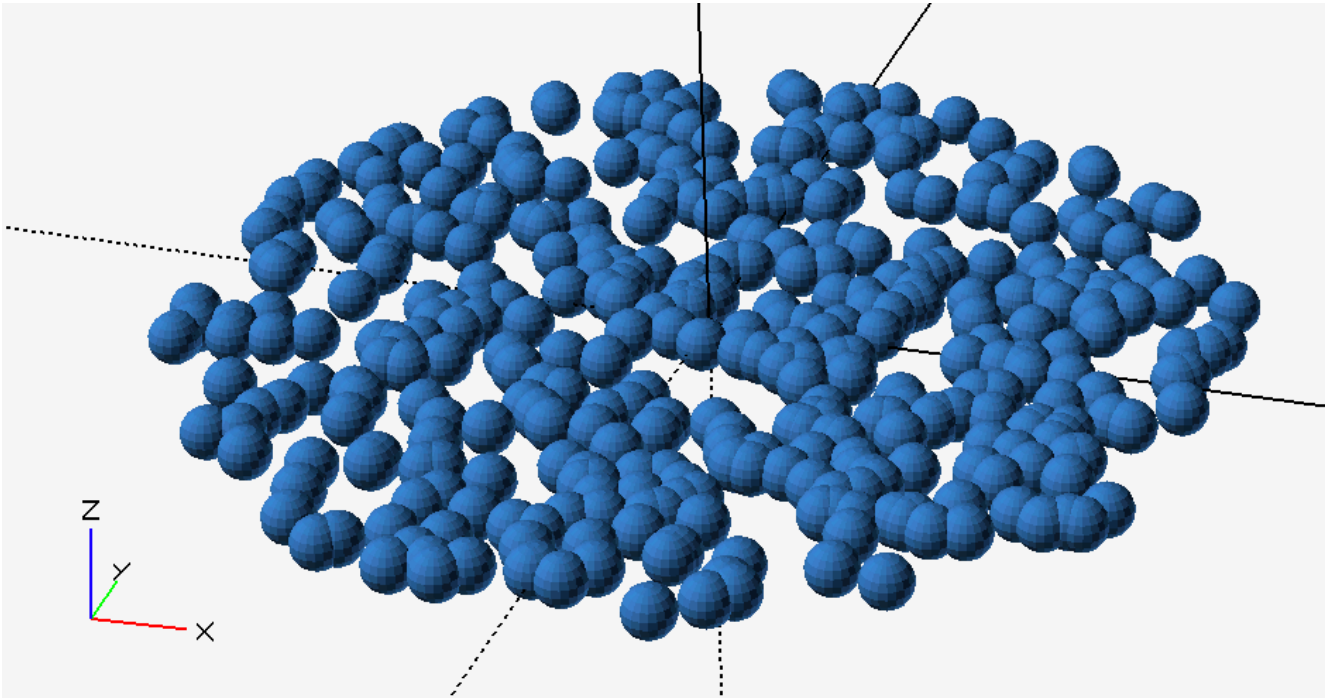


if - else

- **if (condition) {…} else {...}**

```
$fn = 20;
for (i=[0:1:500]) {
  x = rands(-2,2,1)[0];
  y = rands(-2,2,1)[0];
  if (x*x+y*y<4) translate([x,y,0]) sphere(r=0.1);
}
```

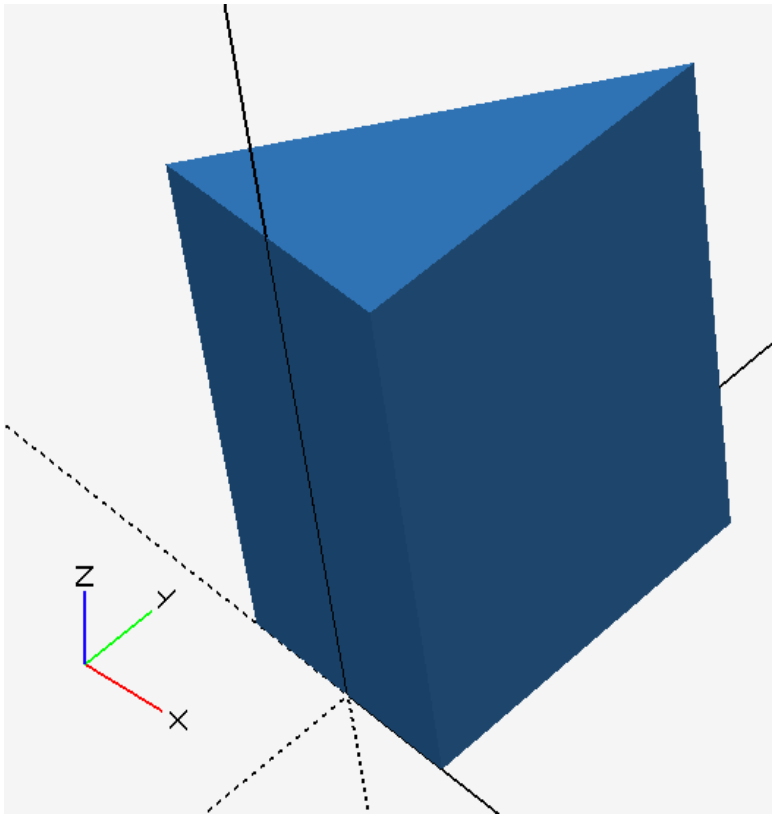
rands(min,max,len)
(gibt Vektor zurück)

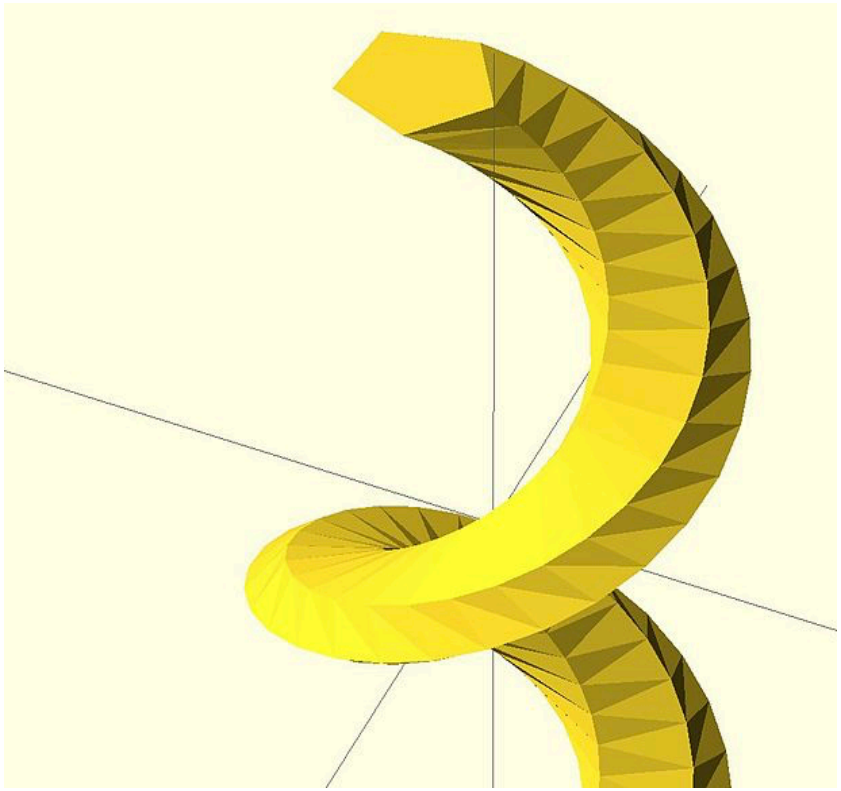


Extrudieren

- 2D Form (hier ein **`polygon()`**) wird ‘in die Länge gezogen’

```
linear_extrude(height = 5, convexity = 10)
polygon(points=[[1,0],[1,3],[-1,0]]);
```



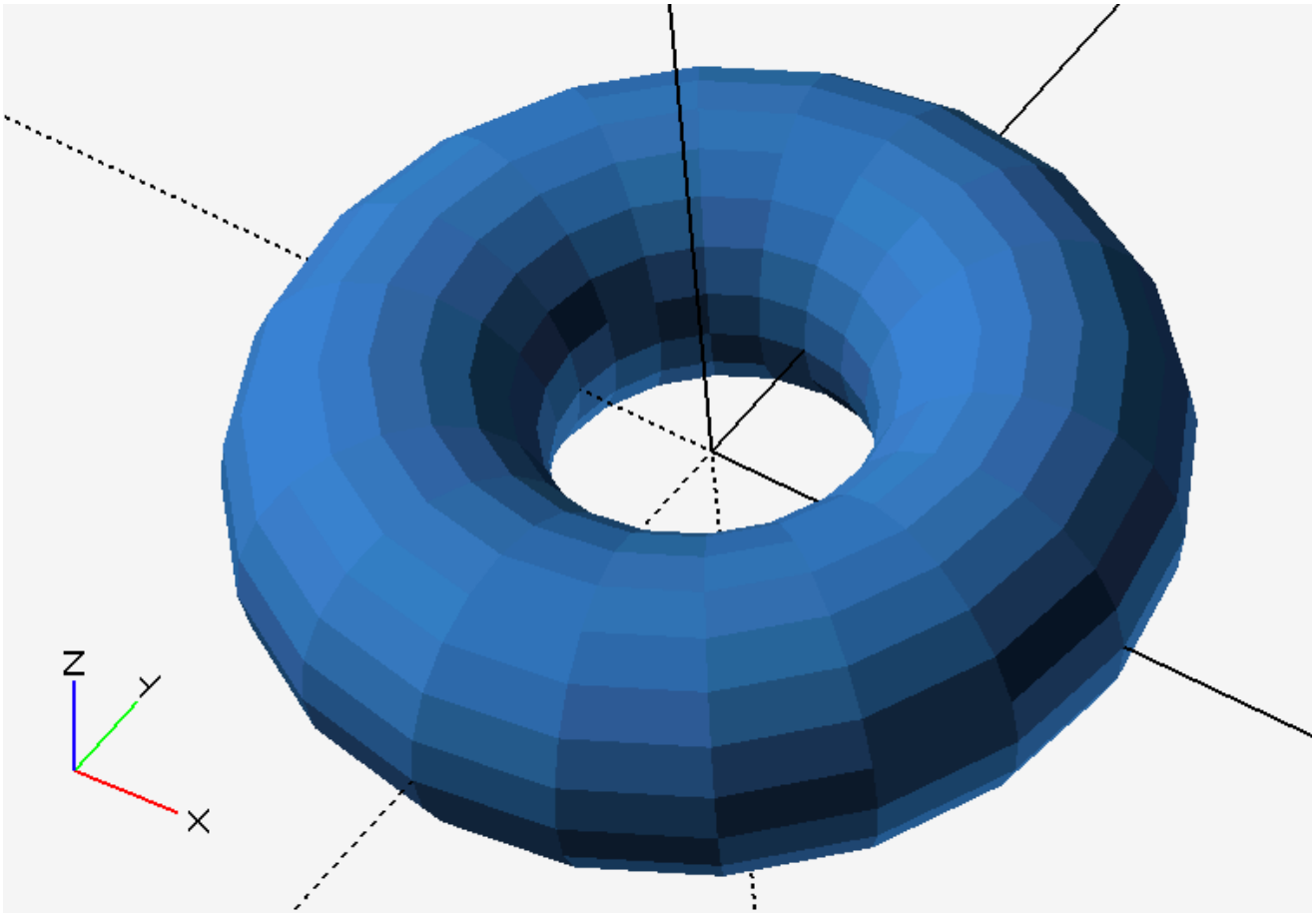


- Weitere Parameter erlauben ‘twist’ (für Schrauben)

rotate_extrude

- wie linear_extrude, aber entlange Kreis:

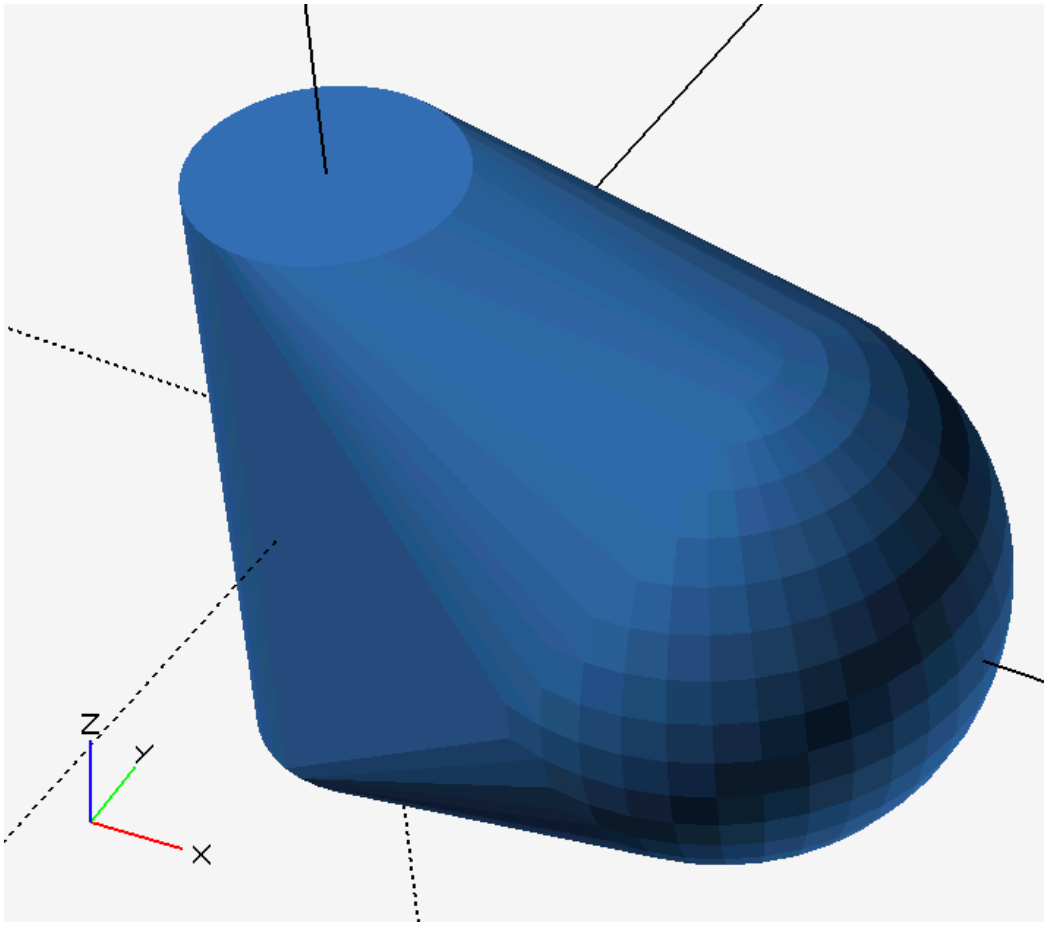
```
rotate_extrude(convexity = 10)
translate([2, 0, 0])
circle(r = 1);
```



hull()

- Volumen, das mehrere Körper ‘umspannt’ (wie mit einer Folie). Kann sehr praktisch sein!

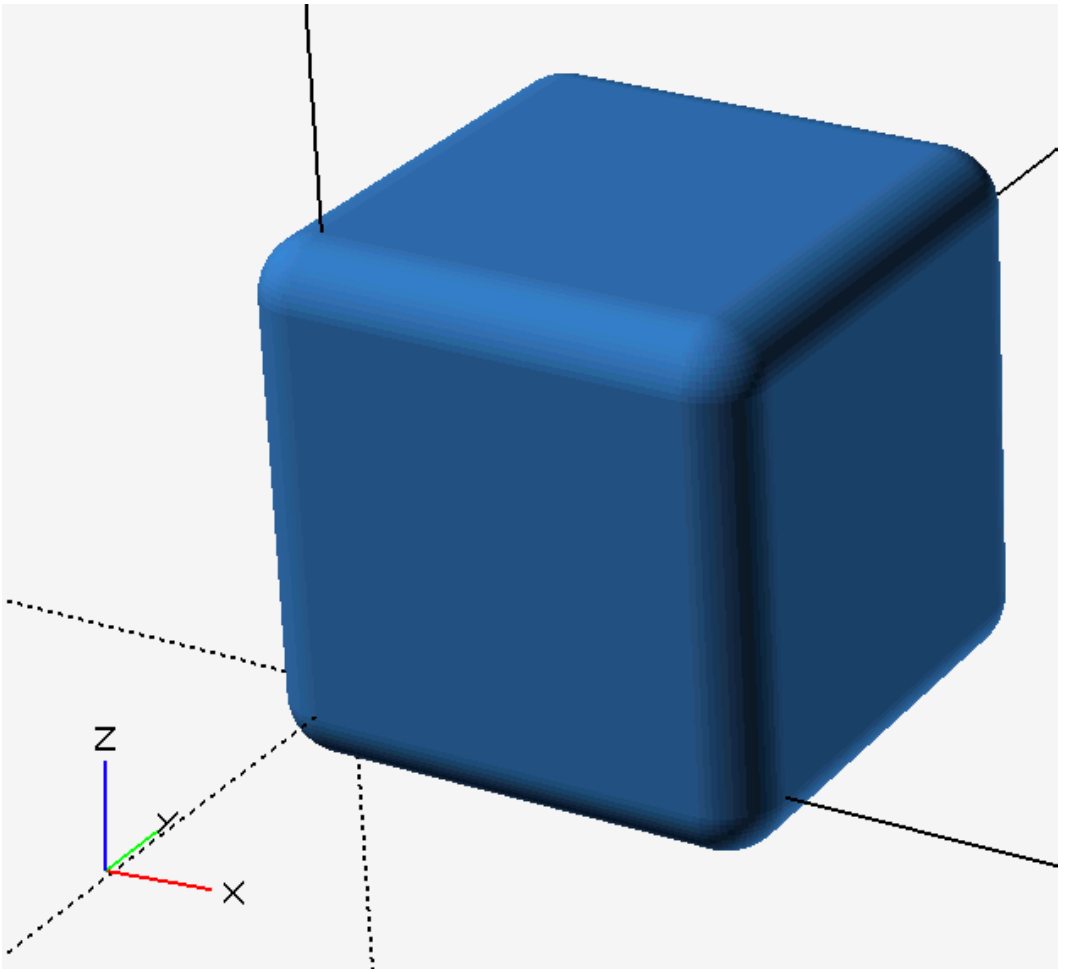
```
hull() {
   cylinder(r=1,h=5,center=true);
   translate([3,0,0]) sphere(r=2);
}
```



minkowski ()

- Object B wird über alle Oberflächenpunkte on A bewegt. Gut zum Abrunden von Kanten und Ecken

```
minkowski () {
   cube([2,2,2]);
   sphere(r=0.3);
}
```



Häufige Fehler

- Semikolon vergessen
- Klammern verwechselt oder vertauscht

- Objekte der Größe 0 (cube([5,5,0]);

- Objekte weit weg oder nicht im Sichtbereich

- Zu viele Semikola (z.B. hinter translate())

Hausaufgabe

- Erzeugen Sie ein nicht ganz triviales 3D Objekt, z.B.