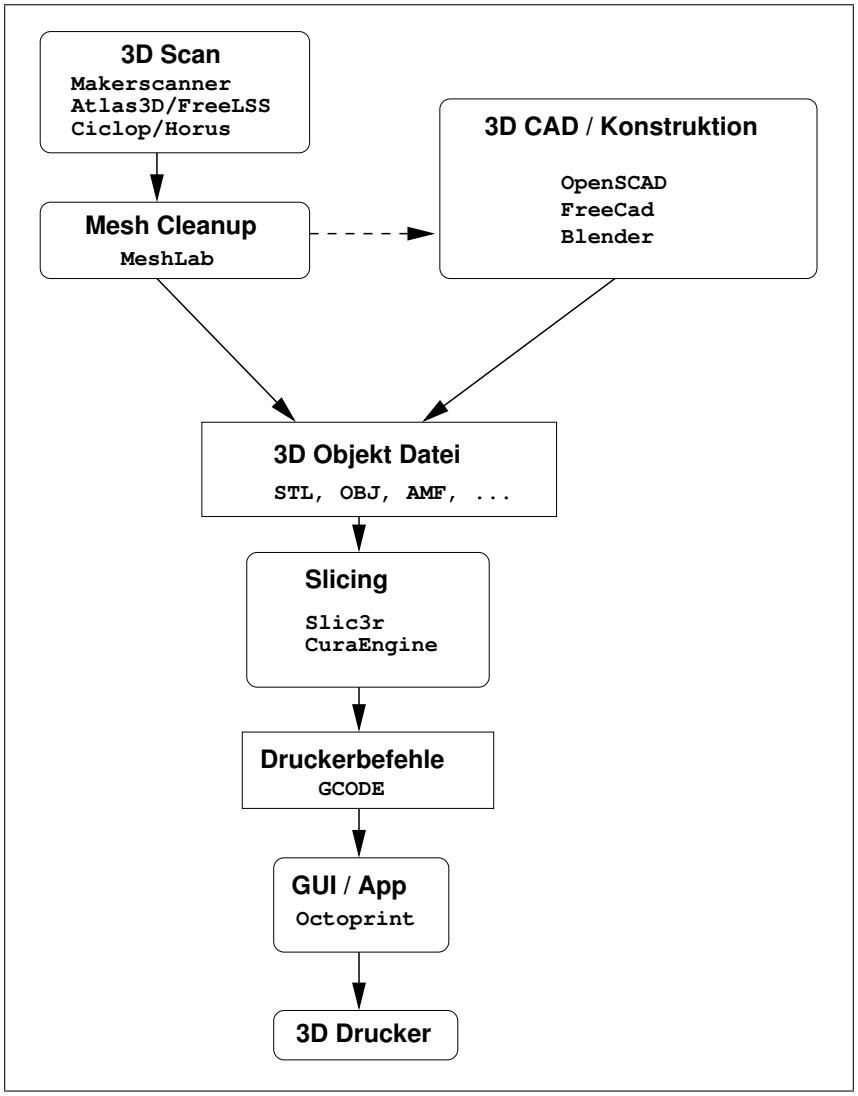
OpenSCAD

Einführung, Tutorial und Übungen

Syntax und Semantik einer sehr einfach gehaltenen, imperativen, prozeduralen Programmiersprache

Prof. Dipl.-Ing. Klaus Knopper <klaus.knopper@hs-kl.de>



Arbeitsprozesse bei der Verarbeitung von 3D-Daten (Beispiel)

1 Über OpenSCAD

OpenSCAD ist eine IDE (Integrated Development Environment = Integrierte Programmierumgebung) und ein CAD-Programm (Computer Aided Design), das die Konstruktion von druckbaren 3D-Objekten, z.B. Bauteilen, in einer einfachen, geometrieorientierten Programmiersprache erlaubt.

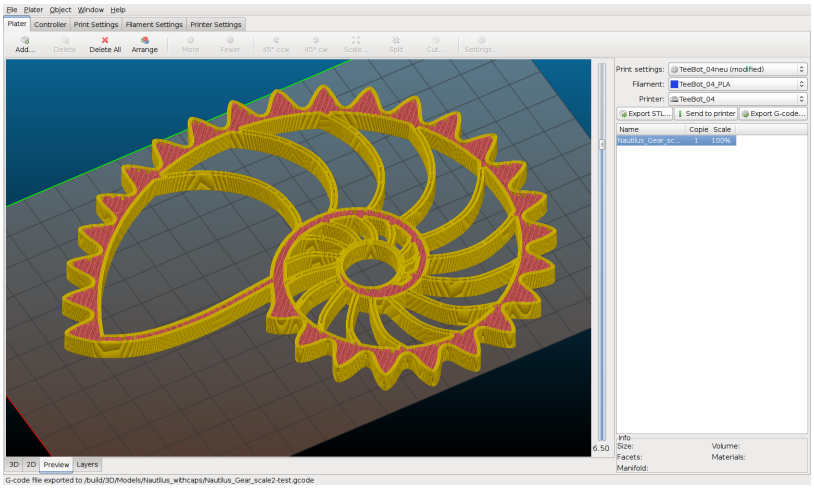
`http://www.openscad.org`

Bereits mit wenigen Befehlen können komplexe Objekte erstellt werden. Maus bzw. Touchscreen werden dabei lediglich im Betrachtungsmodus verwendet, um die Objekte zu drehen (linke Maustaste), zu verschieben (mittlere Maustaste oder linke+rechte Maustaste) und zu zoomen (Mausrad).

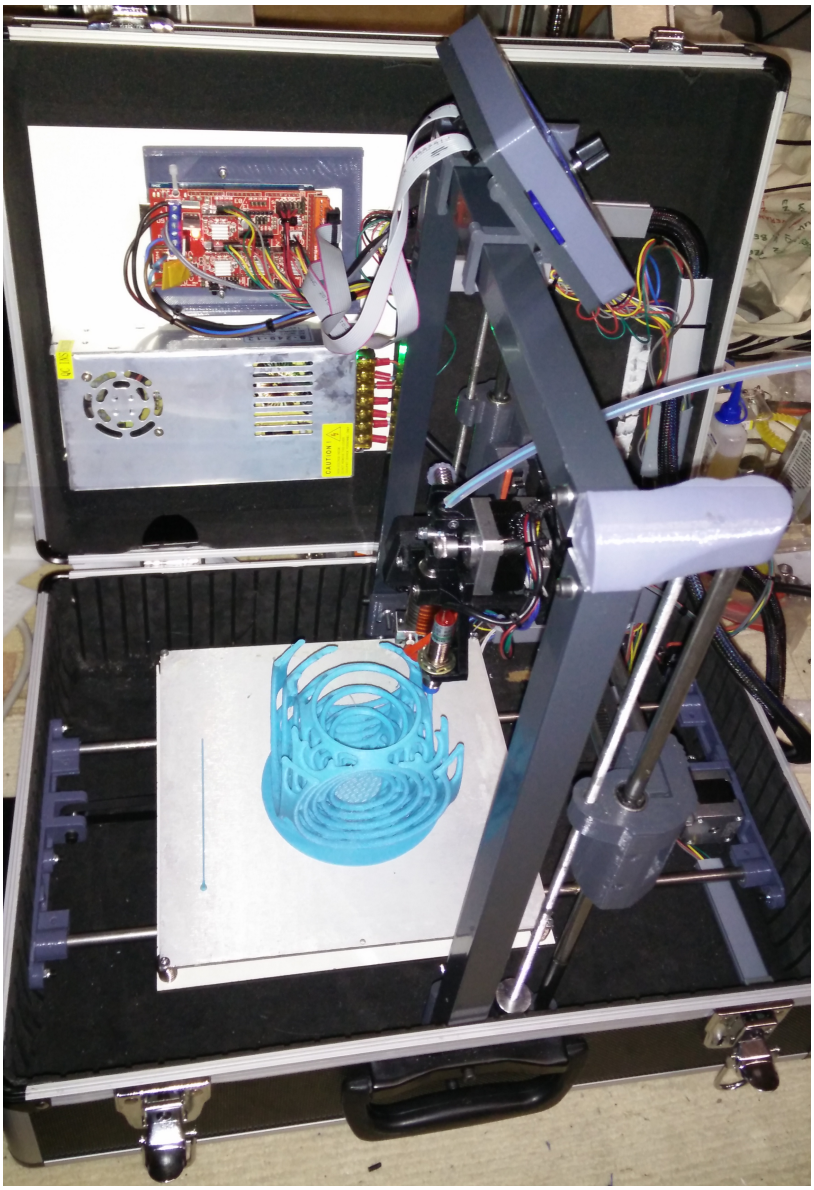
OpenSCAD verwendet eine ähnliche Syntax wie C oder JAVA, und eignet sich daher auch gut als erster Einstieg in die Programmierung, zumal hier mit deutlich weniger Programmieraufwand schnelle Erfolgserlebnisse (beinahe) garantiert sind.

Eine sehr ähnliche Sprache mit anderem Fokus ist x3dom, was als HTML+XML-Dialekt im Web-Browser mit WebGL visualisierte Grafik- und Audio-Objekte produziert.

Für die Ausgabe der in OpenSCAD entworfenen Modelle auf einen 3D-Drucker ist aktuell das Standard-Dateiformat STL üblich. Von OpenSCAD als STL exportierte Modelle können in Software für 3D-Drucker importiert werden, z.B. in Slicer-Programme wie slic3r, und somit in den zur Druckersteuerung verwendeten G-CODE weiterverarbeitet werden (s.a. nachfolgende Abbildungen).



STL-Objekte in Filamentbahnen (G-CODE) umrechnen mit Slic3r



G-CODE drucken auf Teebot

2 OpenSCAD Installation

OpenSCAD wird als Open Source Programm unter den Bedingungen der GNU General Public License Version 2 entwickelt und verbreitet, die Nutzung und Weitergabe des Programms in unveränderter oder abgewandelter Form sowie die integration in eigene Projekte sind hierdurch kostenlos möglich.

Fertig übersetzte, installierbare Versionen von OpenSCAD werden über die Webseite **`http://www.openscad.org/downloads.html`** angeboten.

Neben der schnellen, nativen Version von OpenSCAD, die direkt unter Linux, Windows oder Mac ausführbar ist, gibt es noch eine Browser/WebGL-basierte Variante mit leicht unterschiedlicher nativer Syntax: **`http://openjscad.org/`**

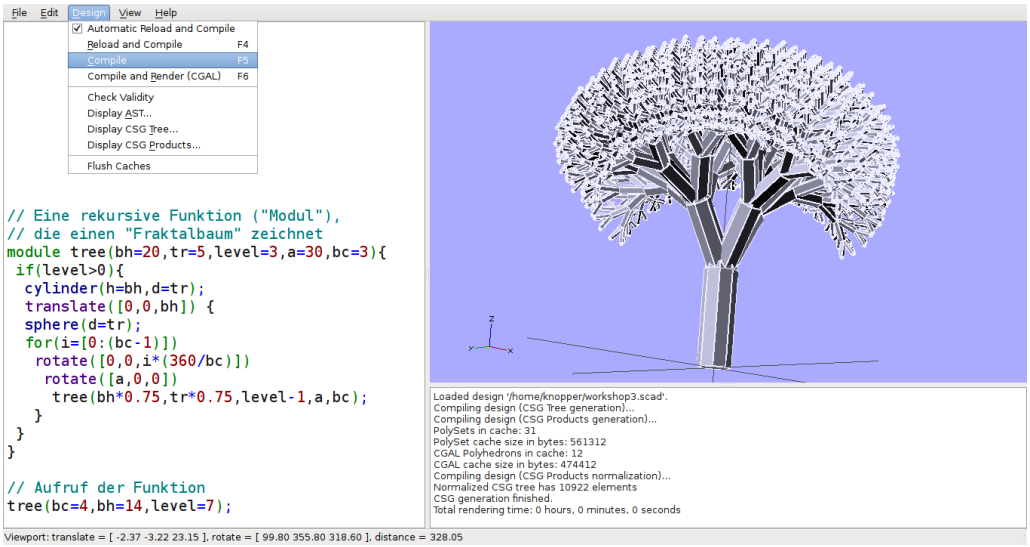
Mit OpenSCAD erstellte Dateien können per „Drag & Drop" in Open**J**SCAD im Browser dargestellt und getestet werden. Allerdings stehen nicht alle aktuellen OpenSCAD-Funktionen in OpenJSCAD zur Verfügung, daher sollte eher die native (Betriebssystemspezifische) Variante des Programms im Kurs verwendet werden.

Für Android auf Smartphones ist OpenSCAD mit etwas reduziertem Funktionsumfang als App unter dem Namen ScorchCAD verfügbar.

3 Bedienung

Nach dem Start von OpenSCAD wird typischerweise ein dreigeteiltes Fenster angezeigt:

1. Ein einfacher Texteditor für die Bearbeitung des Programms (Eingabe),
2. Ein Bereich für die graphische Ausgabe zum Betrachten des Ergebnisses (Ausgabe),
3. Ein Bereich, in dem Fehler- und Statusmeldungen angezeigt werden (Verarbeitung).



Neben der normalen Texteingabe im Editor stehen folgende Programm-Funktionen zur Verfügung:

Editor-Bereich	
Interaktion	**Wirkung**
F5	Übersetzung (Compile) und Gittermodell aktualisieren (schnelle Variante)
F6	Übersetzung (Compile) und aufwändige Version rendern (langsame Variante, für STL-Export notwendig)
F4	Dokument neu laden, sonst wie F5 (bei externem Quelltexteditor verwenden)

Grafik-Bereich	
Interaktion	**Wirkung**
Linke Maustaste	Modell drehen
Rechte Maustaste	Modell verschieben
Mausrad	Zoom

4 Syntax und Semantik der Programmiersprache OpenSCAD

Die Syntaxbeschreibung von OpenSCAD passt gut lesbar noch auf eine Din A4 Seite. Die kürzeste Variante zum Nachschlagen ist das aktuelle Online-Cheatsheet:

`http://www.openscad.org/cheatsheet/`

Achtung: Die auf der Seite angegebene Syntaxbeschreibung ist wörtlich (literal), die verwendeten eckigen Klammern für die Angabe von Punkten oder Vektoren werden also wirklich genau so im Programmcode verwendet, und sind KEIN EBNF!

Die ausführliche, englische Beschreibung der einzelnen Funktionen (Semantik) von OpenSCAD ist unter

`https://en.wikibooks.org/wiki/OpenSCAD_User_Manual`

als Wikibook verfügbar.

5 Tutorial

5.1 Kommentare und Kommentarblöcke

In OpenSCAD können im Quelltext, wie bei den meisten anderen Programmiersprachen (z.B. JAVA, C++) Hinweise und Erklärungen untergebracht werden, die vom Compiler ignoriert werden. Man bezeichnet dies als **Kommentar**.

Hierfür gibt es zwei Varianten:

```
// Kommentar
```

gilt nur bis zum Ende der Zeile (benötigt also keine weitere Anweisung zur „Aufhebung" des Kommentars), während

```
/* Kurzer Kommentar */

/* Hier sind
mehrere
Kommentarzeilen
vorhanden */
```

einen **Kommentarblock** darstellt, der mit der Zeichenfolge `/*` beginnt, und mit der Zeichenfolge `*/` endet.

Kommentare können auch verwendet werden, um Teile des Quelltextes zu deaktivieren, man bezeichnet dies als **Auskommentieren**.

```
intersection() {
 cube(60,center=true);
 sphere(40);
 /* cylinder(80,40); */ /* Später! */
}
```

5.2 Anweisungen und Anweisungsblöcke

Anweisungen *müssen* wie in JAVA mit einem Semikolon (`;`) beendet werden.

```
Anweisung1; Anweisung2;
```

Anweisungen können z.B. Aufrufe primitiver Grafikfunktionen wie `cube()` sein.

Mehrere Anweisungen können (wie in JAVA) mit „geschweiften Klammern" `{...}` zu einem *Anweisungsblock* zusammengefasst werden.

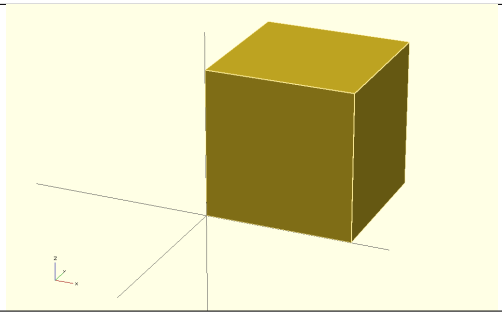
```
{
 Anweisung1;
 Anweisung2;
 Anweisung3;
}
```

Ein weiteres Semikolon hinter der schließenden Klammer ist (bis auf wenige Ausnahmen) nicht erforderlich, aber erlaubt.

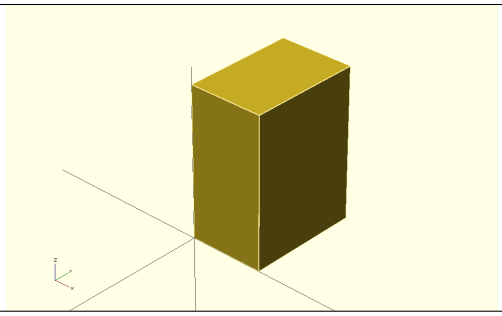
5.3 Grafik-Primitiven

Primitive Grafikfunktionen zeichnen geometrische Objekte, deren Eigenschaften optional in den runden Klammern angegeben werden können. Dabei sind die Standardeinstellungen je nach Funktion unterschiedlich.

Während

`cube();`	

einen Würfel mit Kantenlänge 1 zeichnet, erzeugt

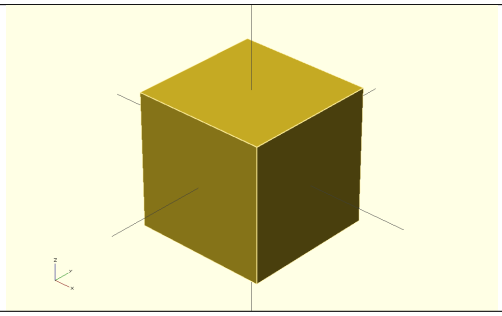
`cube([2,3,4]);`	

einen Quader mit den Seitenlängen $x = 2$, $y = 3$, $z = 4$.

Die **positionalen Parameter** in der eckigen Klammer haben hier also die Bedeutung der drei Seitenlängen.

Beim `cube()` befindet sich die „linke untere Ecke" im Mittelpunkt des Koordinatensystems. Der Würfel lässt sich mit

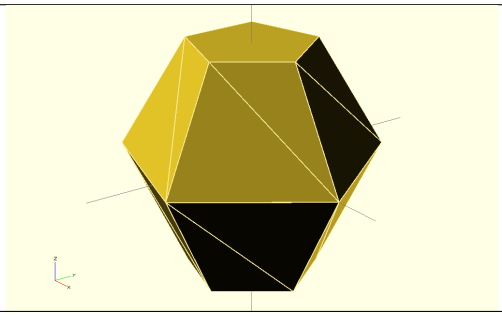
```
cube(center=true);
```



so verschieben, dass der Mittelpunkt des Koordinatensystems in seinem eigenen Mittelpunkt liegt, was dem Standard-Verhalten der Kugel entspricht (diese ist grundsätzlich an ihrem Mittelpunkt zentriert).

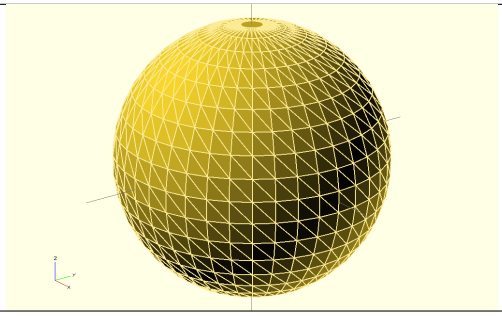
Die Funktion

```
sphere();
```



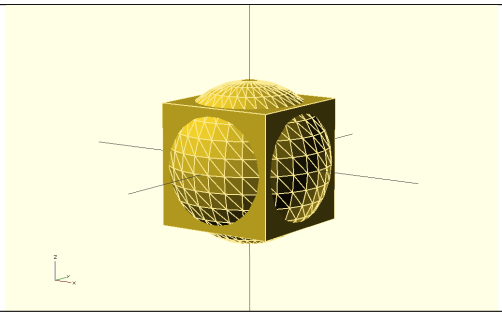
erzeugt eine Kugel mit Radius 1, die im Grafikfenster allerdings nicht wirklich wie eine Kugel aussieht. Dies liegt an der Standard-Auflösung, mit der Objekte durch Aneinanderlegen von Dreiecken dargestellt werden. Durch Setzen der Variable **`$fn=40;`** kann die Auflösung verbessert werden, hierdurch werden mehr Dreiecke pro Zeicheneinheit gerendert, was sich vorwiegend bei runden Elementen lohnt. Allerdings erhöht sich dadurch die Rechenzeit für die Berechnung der Modelle und auch für die Darstellung der Objekte teilweise dramatisch.

```
$fn=40;
sphere();
```



Objekte können zu einer neuen Form addiert werden, indem mehrere Anweisungen mit der Funktion
`union(){ anweisung1; anweisung2; ... }`
zusammengefasst, oder einfach nur hintereinander aufgeschrieben werden:

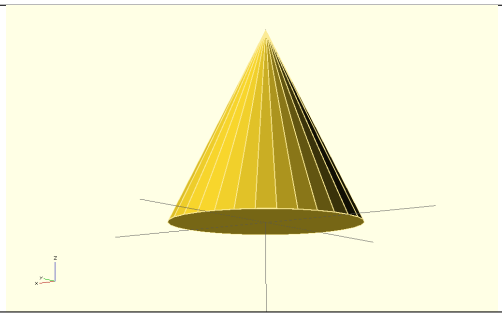
```
sphere(40);
cube(60,center=true);
```



In diesem Beispiel hat die Kugel den **Radius 40** und der zentrierte Würfel die **Seitenlänge 60**. Das gleiche Bild entsteht, wenn bei der Kugel der **Durchmesser 80** angegeben wird: **`sphere(d=80);`**.

Zylinder:

```
cylinder(r1=25,r2=0,h=50);
```



Weitere primitive Grafikobjekte und die zugehörigen Parameter im „Live-Test" (s.a. **`http://www.openscad.org/cheatsheet/`**).

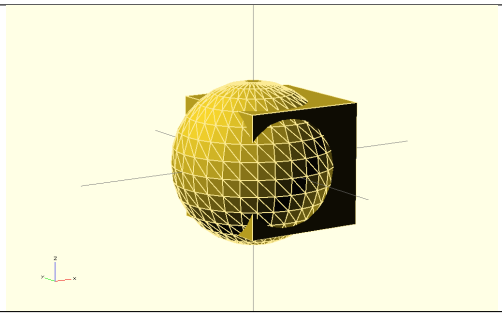
5.4 Einfache Transformationen

Alleine mit der Addition von Objekten lassen sich noch keine beliebigen 3D-Formen herstellen, hierzu sind Verknüpfungs- sowie Positionierungsfunktionen notwendig, sogenannte **Transformationen**.

In OpenSCAD bezieht sich eine Transformation immer auf die direkt dahinter stehende Anweisungsfolge, die Reihenfolge der Anweisungen ist also eigentlich „von rechts nach links" zu lesen.

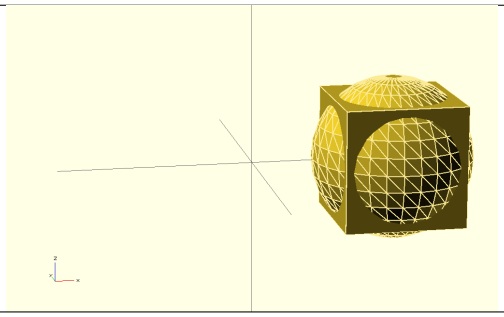
Im folgenden Beispiel wird der Würfel aus dem vorigen Beispiel um 10 Einheiten in x-Richtung (nach rechts) verschoben.

```
sphere(40);
translate([10,0,0])
  cube(60,center=true);
```



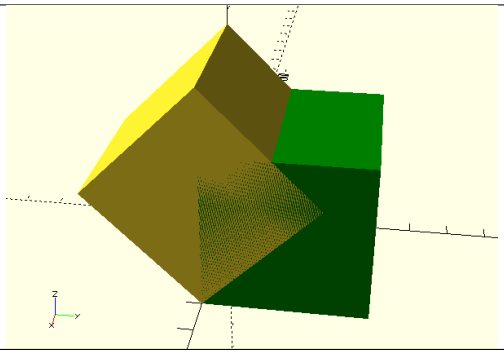
Hier werden hingegen **beide** Objekte gemeinsam um 70 Einheiten nach rechts verschoben, was durch die „Klammerung" erreicht wird:

```
translate([70,0,0]) {
 sphere(40);
 cube(60,center=true);
}
```



Hier wird ein zweiter Würfel an der X-Achse um 45 Grad (positiv = links) gekippt:

```
color("green") cube(40);
rotate([45,0,0]) cube(40);
```



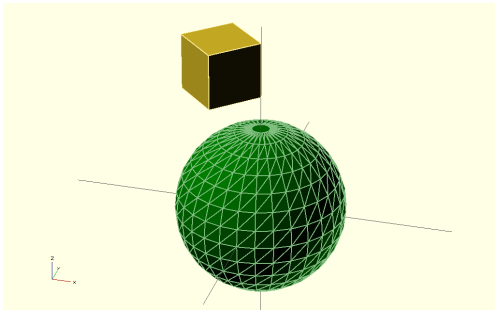
Vorsicht: Eine Translation oder Rotation wird immer vom Mittelpunkt des Koordinatensystems aus durchgeführt, und *nicht* vom Mittelpunkt des Objektes aus, wie man vielleicht annehmen würde!

Bei einer Anweisungsfolge ist entscheidend, in welcher Reihenfolge welches Kommando ausgeführt wird. Wird die Reihenfolge verändert, entstehen völlig andere Szenarien oder Objekte, auch bei ansonsten gleichen Parametern.

Weitere Beispiele (Demo): Rotieren, Skalieren, Einfärben, Spiegeln

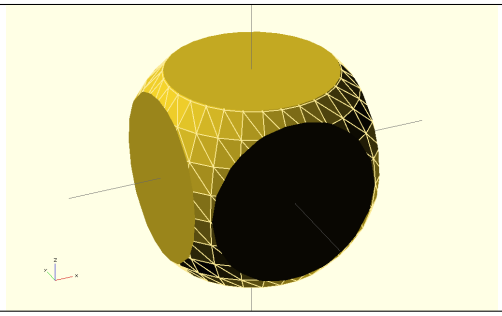
Eine verschachtelte Anweisungsfolge (die Werte in eckigen Klammern sind jeweils Längen in x,y,z-Richtung, bei der Rotation Winkel um x,y,z-Achse):

```
mirror([1,1,0])
 translate([0,0,52]) {
  translate([0,0,-52])
   // relativ skalieren
   scale([1.5,1.5,1.5])
   // absolut skalieren
   // resize([50,50,100])
    color("green") sphere(25);
    rotate(45,[0,0,1]) cube(25);
}
```



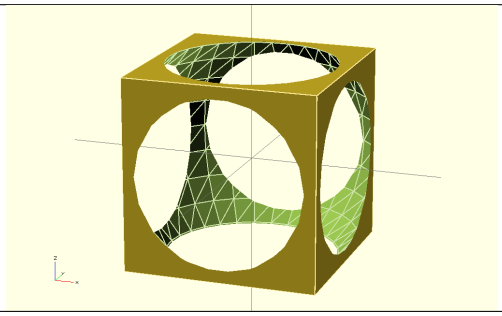
Nur die **Schnittmenge** von Objekten darstellen:

```
intersection(){
 sphere(20);
 cube(30,center=true);
}
```



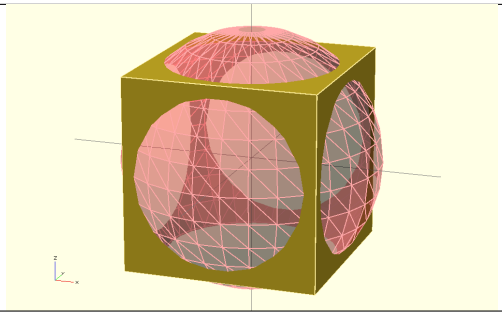
Nur die **Differenz** von Objekten darstellen, d.h. vom ersten Objekt alle weiteren Objekte „abziehen".

```
difference() {
 cube(30,center=true);
 sphere(20);
}
```



Tipp: Mit Präfix **#** vor einer Anweisung kann deren Inhalt „halbdurchsichtig" dargestellt werden, also z.B. so:

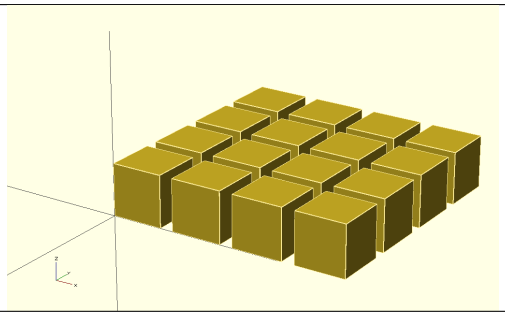
```
difference() {
 cube(30,center=true);
 #sphere(20);
}
```



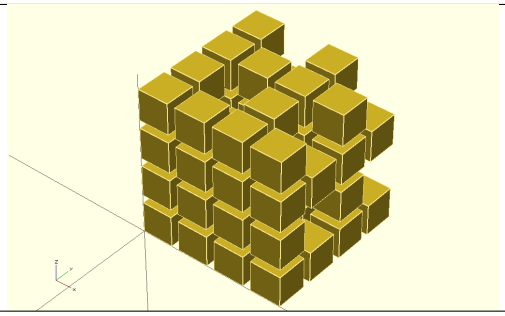
5.5 Mehrfache Ausführung (Schleifen)

Bei diesem Konstrukt wird eine Anweisungsfolge mehrfach, mit sich verändernden „Laufvariablen" ausgeführt. Die Syntax ist hier bei OpenSCAD kompakter, aber leider ausnahmsweise etwas weniger eingängig als in der aus PHP, JAVA oder C bekannten **`for()`**-Schleife.

```
for(x=[0:3]){
 for(y=[0:3]){
  translate([x*25,y*25,0])
   cube(20);
 }
}
```



```
for(x=[0:3], y=[0:3],z=[0:3]) {
 translate([x*25,y*25,z*25]) {
  if( (x*y*z)%2 == 0) {
   cube(20);
  }
 }
}
```

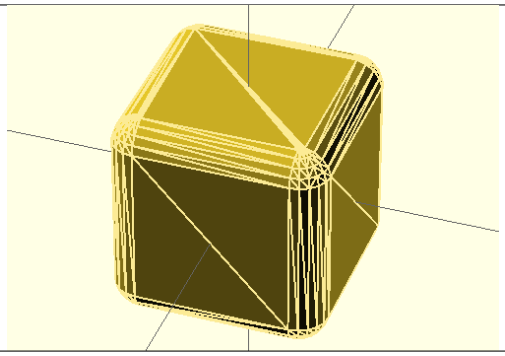


5.6 Komplexe Transformationen

Einige Transformationen sind sehr rechenaufwändig.

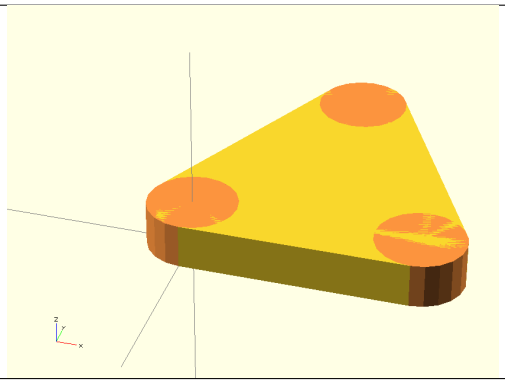
Minkowski: Kanten abrunden (bzw. zweites Objekt an den Kanten des ersten anwenden). Achtung: Hierdurch vergrößert sich das Objekt um den Radius der für die Abrundung verwendeten Kugel auf jeder Seite!

```
$fn=20;
minkowski(){
 cube(25);
 sphere(5);
}
```



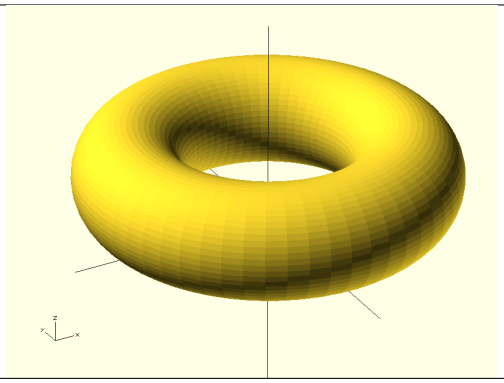
Hull: Eine „minimale" Hülle (Folieneffekt) um Objekte zeichnen.

```
$fn = 20;
hull() {
 #translate([0,0,0])
cylinder(r=2,h=2);
 #translate([10,0,0])
cylinder(r=2,h=2);
 #translate([5,10,0])
cylinder(r=2,h=2);
}
```



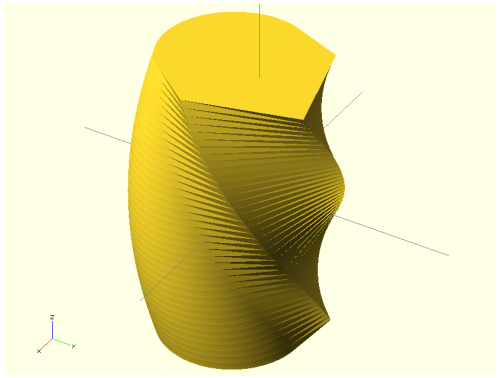
Rotate_Extrude: Rotationskörper definieren.

```
$fn=60;
rotate_extrude() {
 translate([50,20]) circle(20);
}
```



Linear_Extrude: Flächen in eine Richtung „herausziehen“ und (optional) dabei verdrehen.

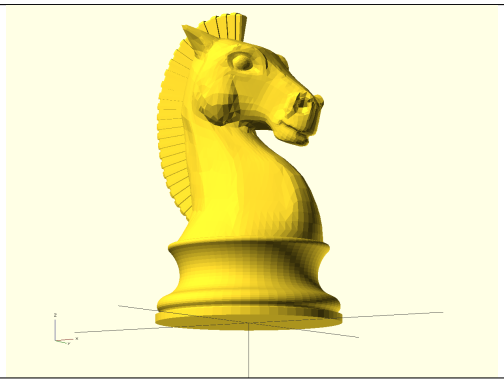
```
$fn=80;
linear_extrude(height=50,
center=true, twist=120,
slices=40){
 hull() {
  translate([10,0])
   circle(15);
  rotate(45)
    square(23,center=true);
 }
}
```



5.7 3D-Dateien importieren

OpenSCAD unterstützt, abhängig von aktivierten Erweiterungen, eine Anzahl von „Fremdformaten“ wie STL oder DXF, die als Vektorgrafik importiert und durch eigene Konstrukte erweitert werden können.

```
// Sockel-Profil aus DXF-Datei
scale(0.2)
 translate([0, 0, 30])
  rotate_extrude(convexity = 10)
   import_dxf("sockel_profil.dxf");
// Kopf aus STL-Datei
translate([-8, -12, 28])
 scale(3.2)
  import("horse3.stl");
```



Auf diese Weise lässt sich beispielsweise auch Stützmaterial für Objekte mit Überhang einfügen, die nicht direkt in OpenSCAD erstellt wurden.

5.8 Eigene Funktionen (Module) erstellen

Module (analog JAVA: Methoden) erweitern den Sprachumfang von OpenSCAD durch Implementierung von Algorithmen, die sich aus bereits bekannten Anweisungen zusammensetzen.

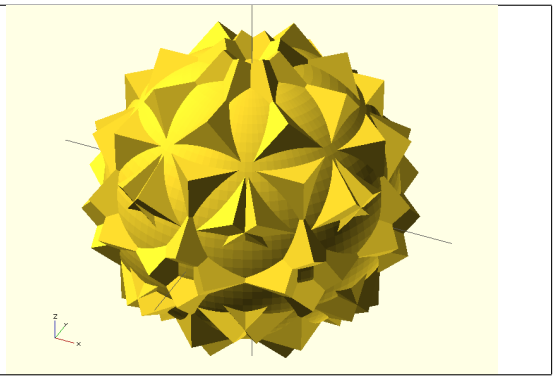
Syntax: **`module name_des_moduls(parameter1=standardwert1, ...){`**

```
  Anweisungen ...
}
```

Später wird das Modul wie die vordefinierten Funktionen verwendet:
`name_des_moduls(10, 20, ...);`

```
module spaceball( radius=15,
                  seite=20 ){
 sphere(radius);
 for(rx=[0:7],ry=[0:7],rz=[0:7])
  rotate([rx*45,ry*45,rz*45])
   cube(seite,center=true);
}

spaceball();
```



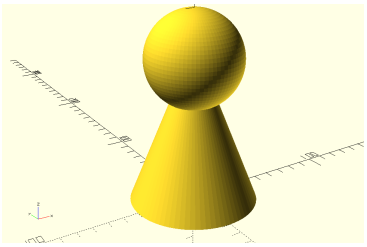
Module, die analog **`translate()`** auf die nachfolgenden Anweisungen wirken, können mit der Funktion **`child();`** auf die zuvor produzierten Objekte zugreifen.

```
module mirror_and_show(vec=[1,0,0]){
 mirror(vec) child();
 // # = Objekt transparent anzeigen
 // % = Objekt transparent nur in der Ausgabe anzeigen, nicht drucken
 %child();
}
```

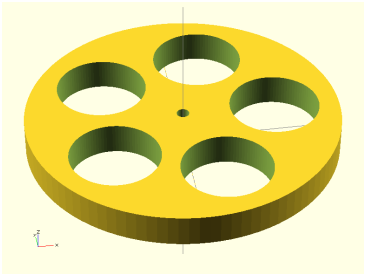
6 Übungen

6.1 Stellen Sie die abgebildeten Objekt in OpenSCAD dar. (8 Punkte)

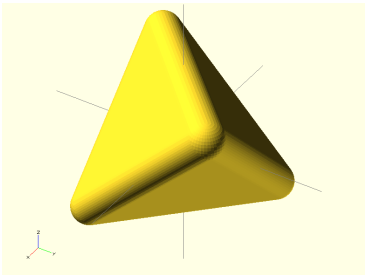
6.1.1 Spielfigur

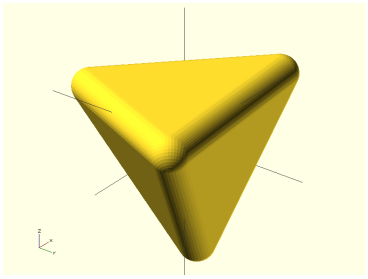


6.1.2 Rad mit Aussparungen



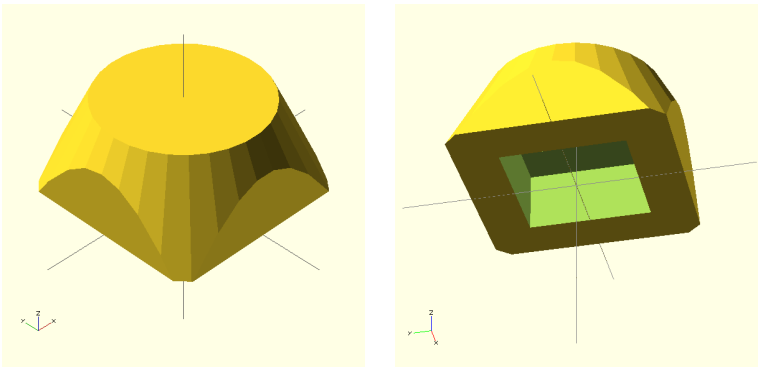
6.1.3 Tetraeder mit abgerundeten Kanten





Hinweis: In diesem Beispiel war **`minkowski()`** nicht erforderlich.

6.1.4 Podest, innen hohl (um Material zu sparen)



Hinweise zur Abgabe:

- Abgabe der Lösungen für diese Übung per E-Mail an Marc Beck <marc.beck@hs-kl.de>
  bis **Mittwoch, 08.06.2016**.
- Bitte geben Sie als „Betreff" an: **Abgabe GDI Übung 8 SS2016 *Ihr Name***.

7 Links

`https://www.hs-kl.de/betriebswirtschaft/studiengaenge/bachelor/`
Information Management an der HS-KL

`http://www.openscad.org/` OpenSCAD Homepage

`http://www.openscad.org/cheatsheet/` OpenSCAD „Cheat Sheet" - die ultrakurze Kompakt-Beschreibung aller OpenSCAD-Funktionen (mit Links zu Details)

`http://www.thingiverse.com/` Druckbare 3D-Objekte, Beispiele und Experimente zum herunterladen, teilweise auch mit Quelltexten für OpenSCAD

`http://www.repetier.com/` RepetierHost, ein Darstellungs-, Positionierungss- und Druckprogramm für 3D-Formate

`http://www.octoprint.org/` OctoPrint, interaktive web-basierte Steuerung und Überwachung von 3D-Drucern auf Basis eines Raspberry Pi2 Minicomputers

`http://slic3r.org/` Slic3r, ein Slicer (Ebenenzerlegung, Perimeter und Füllung berechnen und G-CODE für Drucker erzeugen) für 3D-Formate