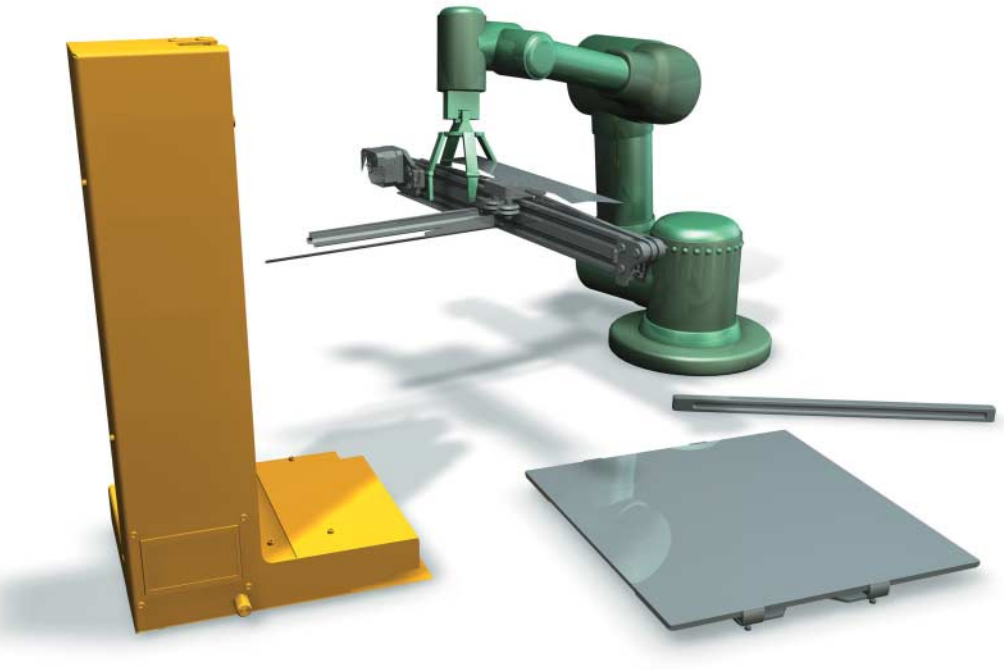
3D-Druck-Tutorial, Teil 2:
Konstruktion mit Blender, OpenSCAD und Co.

Aufgezeichnet



Klaus Knopper

Für eine 3D-Konstruktion kommen neben der Technik des maßgenauen Konstruierens auch künstlerische und ergonomische Aspekte zum Tragen. Freie Software kann dabei helfen, fehlendes zeichnerisches Geschick zu kompensieren.

Letztlich ist 3D-Design ein Thema, das aus mehreren Teilgebieten besteht: dem technischen für die Konstruktion der Teile gemäß den Maßvorgaben, dem künstlerischen – es soll gut aussehen, was für das Marketing wichtig ist – und für beide Richtungen relevanten Aspekten der Ergonomie. Hier gilt es, Gesichtspunkte wie leichtes Handling, das Reduzieren von Verletzungsgefahr oder Fehlbedienung zu berücksichtigen. Dieser zweite Tutorialteil konzentriert sich auf die technische Konstruktion, legt aber auch Wert auf Ergonomie und die Gewährleistung von Druckbarkeit auf den verfügbaren 3D-Drucker-Arten. Die künstlerisch-handwerkliche Komponente ist teils Übungssache, aber hierfür gibt es auch diverse Studiengänge, für die dieser Artikel natürlich kein Ersatz sein kann.

Die meisten hier vorgestellten Programme bieten Hilfen, die fehlende „Zielgenauigkeit“ oder mangelndes zeichnerisches Geschick durch die maßgenaue Eingabe von Daten kompensieren. OpenSCAD verzichtet sogar komplett auf das Zeichnen oder Positionieren mit Maus oder Grafiktablett. Doch dazu später mehr.

iX-TRACT

- Auch wenn Blender kein ideales 3D-Konstruktionspaket ist, lassen sich damit selbst ohne große Zeichenkünste 3D-Modelle erstellen.
- Völlig ohne Zeichnen kommt das auf Automatisierbarkeit und Wiederverwendbarkeit ausgerichtete Open-Source-Programm OpenSCAD aus.
- Anwender erstellen ihre 3D-Objekte per parametrischer Programmierung, eine Methode, die sich dank Modularisierung auch geskriptet nutzen lässt.

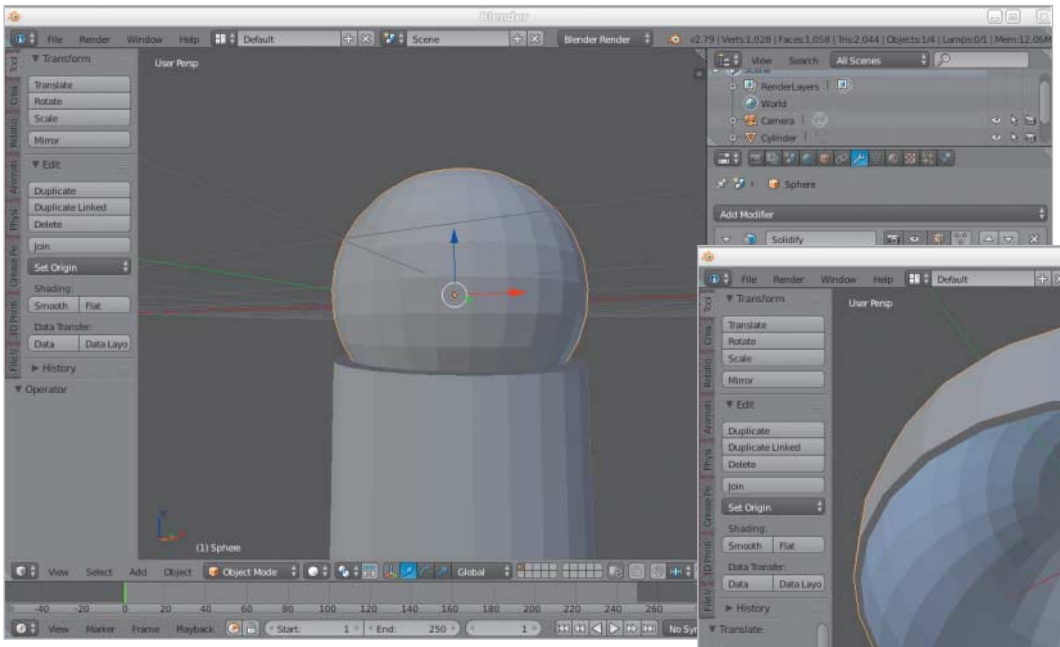
Kein(e) Zeichenkünstler(in), kein Problem!

Das wohl bekannteste Open-Source-Programm zum Erstellen von 3D-Objekten und -Szenen dürfte Blender sein. Es bietet neben dem Zeichnen beweglicher Modelle, 3D-Rendering und dem Editieren von 3D-Objekten auch die Option, animierte Filme zu generieren.

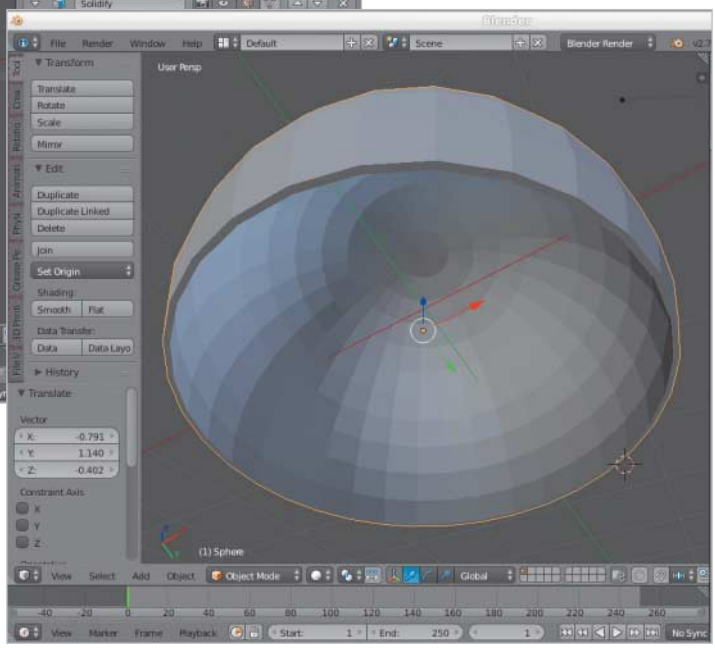
Zuerst wirkt Blender auf den 3D-Konstrukteur schockierend überladen: Es gibt viele Bedienelemente, die dazu mit oft nicht intuitiven Fachbegriffen beschriftet sind. Tatsächlich arbeitet man meist mit Tastaturkombinationen, die nicht immer ein grafisches Bedienelement als Äquivalent besitzen. Die Komplexität ist also noch höher, als sich aufgrund des GUI zunächst vermuten lässt. Es bedarf eines großen Monitors mit hoher Auflösung, um die benötigten Untermenüs und Reiter öffnen zu können. Für die erforderliche Bedienbarkeit empfiehlt sich auch ein Trackball oder Grafiktablett. Ähnlich wie bei anderen umfangreichen Programmsuiten verwenden Nutzer je nach Anwendungszweck aber nur einen Bruchteil der verfügbaren Funktionen.

Erste Konstruktionsschritte mit Blender

Auch wenn Blender für reine Konstruktionsaufgaben möglicherweise nicht das ideale Werkzeug ist, soll das Beispiel einer kugelförmigen Schale die Vorgehensweise beim Erzeugen einer für einen SLS-3D-Laser druckbaren Vorlage verdeutlichen. Blender zeigt zu Beginn als Standardobjekt einen Würfel, der sich durch Drücken der Taste X oder per „Entfernen“ entsorgen lässt. Anschließend wählt man als ersten Schritt im linken Menü unter „Create“ die Form „UV



Mit einem Zylinder lässt sich in die Kugel ein großes Loch bohren (Abb. 1).



Nach dem Entfernen des Zylinders bleibt bei entsprechender Skalierung eine kugelförmige Schale übrig (Abb. 2).

Sphere“ aus. Die Größe lässt sich bei Bedarf mit „Tools/Scale“ verändern, was für dieses Experiment aber nicht nötig ist.

Für das Generieren einer Hülle muss man das Objekt zunächst aushöhlen. Die Funktion, die sich nach Auswahl der Kugel (rechte Maustaste über der Kugel in Blender) im rechten Menü unter „Modifier/Solidify“ findet, arbeitet bei Blender, verglichen mit MeshLabs „Uniform Remeshing“-Filter aus dem ersten Tutorialteil [1], außerordentlich zuverlässig. Sie bietet mit dem Wert *Clamp* auch die Möglichkeit, Artefakte durch fehlende Eckpunkte zu reduzieren. Die in Abbildung 1 gezeigten Einstellungen mit *Offset 1*, *Thickness 5* und *Clamp 10* liefern ein brauchbares Ergebnis.

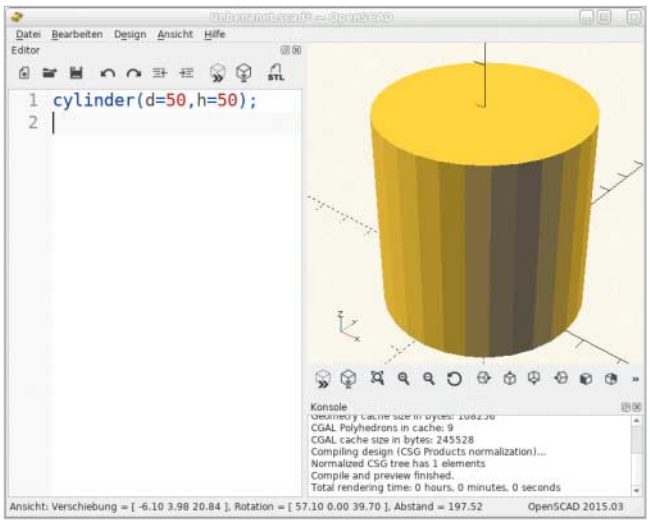
Das Aushöhlen von Objekten lässt sich in Blender auch dazu verwenden, importierte STL-Modelle zu bearbeiten, um das recht teure Harz Resin bei Laser-3D-Druckern zu sparen und die Objekte weniger wuchtig und schwer zu gestalten. Ein Objekt lediglich auszuhöhlen, führt im Falle des SLS-Drucks jedoch dazu, dass sich in den Hohlräumen noch flüssiges Resin ansammelt, was für die Haltbarkeit des Ausdrucks ungünstig ist. Zum Auswaschen müssen Hohlräume eine Verbindung nach außen besitzen, durch die die Flüssigkeit und das beim Waschen verwendete Lösungsmittel abfließen können.

Hierfür fügt man analog zum ersten Schritt zunächst einen Zylinder hinzu, skaliert diesen auf eine passende Größe und positioniert ihn so, dass er die ausgehöhlte Kugel auf einer Seite überlappt. Zum Bewegen des Zylinders an die richtige Stelle sind die farbigen Pfeile hilfreich, die bei einem Rechtsklick auf den Zylinder erscheinen. Diese sind die „Anfasser“, um den Zylinder entlang der gewählten Raumachse zu bewegen.

Nach der erneuten Auswahl der Kugel mit der rechten Maustaste wählt man, nun wieder im rechten Menü, unter den Werkzeugen den Modifikator „Boolean“ aus. Der bildet die Differenz des ausgewählten Objekts mit dem im Menü selektierten. Dadurch bohrt der Zylinder quasi ein Loch in die Kugel. Anschließend muss man den Zylinder wieder herausnehmen (Auswahl mit rechter Maustaste, dann „X“ oder „Entfernen“). Übrig bleibt eine angebohrte Hohlkugel beziehungsweise eine kugelförmige Schale wie in Abbildung 2, je nach zuvor eingestellter Skalierung der Bohrung. Das Objekt lässt sich nun per „Export“ als STL speichern. Es ist günstig für eine spätere Weiterbearbeitung, auch eine Kopie im Blender-eigenen Format zu speichern.

3D-Konstruktion ohne Zeichnen mit OpenSCAD

Zunehmender Beliebtheit erfreut sich das 3D-Konstruktionsprogramm OpenSCAD. Es ist kein CAD-Programm im klassischen Sinn, da das „Designen“ hier tatsächlich eher in maßgenauer, parametrisierter Konstruktion besteht und interaktives Zeichnen weder vorgesehen noch möglich ist. Lediglich in der Voransicht kann man Objekte zur Kontrolle drehen (linke Maustaste), verschieben (mittlere



OpenSCAD bietet Benutzern drei Teilbereiche: Editor, Vorschau- und Protokollbereich (Abb. 3).

Listing 1: Quadratische Pyramide mit *polyhedron()*

```
polyhedron(
  points=[ [10,10,0],[10,-10,0],[-10,-10,0],[-10,10,0],[0,0,10] ],
  faces=[ [0,1,4],[1,2,4],[2,3,4],[3,0,4],[1,0,3],[2,1,3] ]
 );
```

oder linke und rechte Maustaste), vergrößern und verkleinern (Mausrad). Das Projekt stellt Binaries für Linux, macOS und Windows bereit. Darüber hinaus gibt es die Software auch in den Repositories diverser Linux- und xBSD-Derivate.

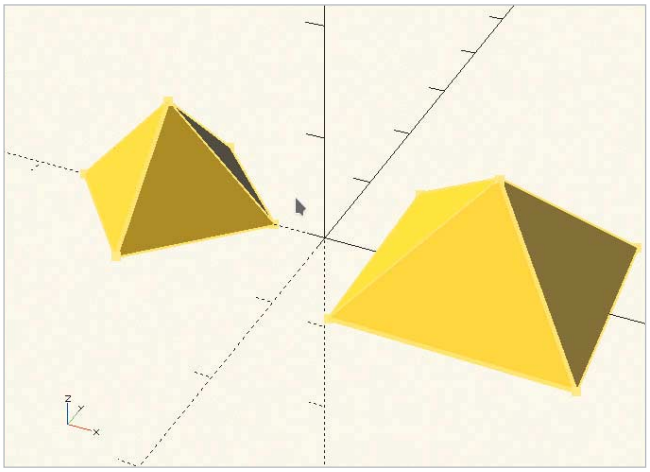
Schon mit wenigen Befehlen lassen sich komplexe Objekte erstellen. OpenSCAD verwendet eine ähnliche Syntax wie C, PHP oder Java, ist aber nicht objektorientiert. Es eignet sich auch als erste Programmiersprache für die Einführung ins Programmieren, zumal mit wenig Aufwand schnelle Erfolgserlebnisse (beinahe) garantiert sind.

Nach dem Start öffnet sich das in Abbildung 3 gezeigte dreigeteilte Fenster. Auf der linken Seite befindet sich der Programmeditor, rechts eine Voransicht und darunter das für IDEs typische Konsolenfenster, in dem Statusmeldungen und Fehlerhinweise erscheinen.

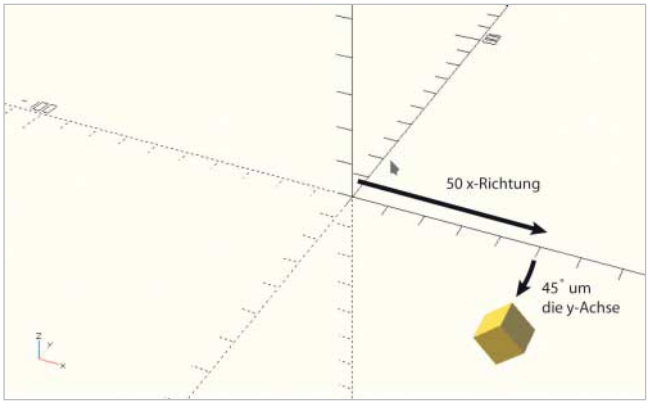
Nach Eingabe eines Kommandos im Editorfenster „kompiliert“ ein Druck auf die Taste F5 dieses und stellt das Objekt in der Voransicht dar. Per F6 wird das Objekt vollständig gerendert, sodass man es als STL-Datei exportieren kann. Hierzu noch ein Hinweis: In der schnellen F5-Voransicht werden bestimmte Darstellungen direkt von der Grafikkarte gezeichnet, was im Fall von Differenz- und Schnittmengenberechnungen bei einigen wenigen Grafikkarten (vor allem NVIDIA ist dafür bekannt) zu Darstellungsfehlern wie fehlenden Seiten oder Objektteilen führen kann. Diese verschwinden aber beim vollständigen Rendering per F6.

Drei Grundformen: Quader, Kugel, Kegelstumpf

Die meisten per OpenSCAD erzeugten Modelle sind aus drei Grundformen zusammengesetzt: Quader, Kugel und Kegelstumpf. Die Zahlen in den Übergabeparameter-Klammern geben die Längen der entsprechenden Seiten, Durchmesser und Höhe an. Die 3D-Objekte sind grundsätzlich frei skalierbar, standardmäßig nutzt die Software jedoch die Einheit mm: Die Anweisung *cube(50);* erzeugt einen Würfel mit 50 mm Kantenlänge.



Zwei Pyramiden: links per *$fn*-Trick und rechts über die Funktion *polyhedron* (Abb. 4)



Per *translate* und *rotate* lassen sich Objekte beliebig im Raum positionieren (Abb. 5).

Die Variante mit einem in eckigen Klammern aufgeführten Vektor [x,y,z] anstelle einer einfachen Zahl erlaubt die Angabe unterschiedlicher Seitenlängen:

```
cube([20,40,60]);
```

Als Faustregel kann man festhalten: Ecken oder Ebenen setzen auf dem Mittelpunkt des Koordinatensystems auf, während bei runden Formen die Koordinatenachse(n) durch den Kreismittelpunkt gehen. Mit der zusätzlichen Angabe *center=true* lässt sich der Mittelpunkt eines Objekts in den Mittelpunkt des Koordinatensystems verschieben.

```
cube([20,40,60], center=true);
```

Bei einer Kugel gibt man entweder den Radius *(r=…)* oder den Durchmesser *(d=…)* an. Sie ist automatisch zentriert. Beispielsweise setzt

```
$fn=200;
sphere(d=50);
```

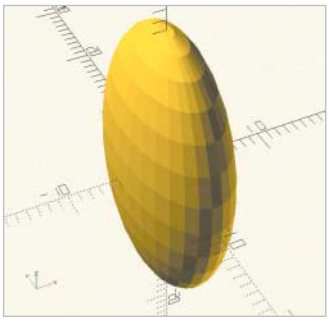
die Auflösung des Umfangs auf 200 Eckpunkte, was sich auf die Näherung der runden Form durch Dreiecke auswirkt. Details hierzu im nächsten Abschnitt.

Wie schon in Abbildung 3 gezeigt, lässt sich ein gerader Zylinder mit Durchmesser 50 und Höhe 50 per *cylinder(d=50, h=50);* erzeugen. Bei Zylindern kann man auch Varianten mit verschiedenen Durchmessern unten und oben definieren. So erzeugt etwa *cylinder(d1=50, d2=0, h=20);* einen spitz zulaufenden Kegel, während *cylinder (d1=50, d2=25, h=20);* einen Kegelstumpf ergibt.

Rund nach eckig oder „low poly“: weitere Formen

Für frei definierbare Körper gibt es auch die Funktion *polyhedron([array_eckpunkte], [array_flächen])*, wobei die Definition der Außenflächen durch Angabe der jeweiligen Indizes im Punkte-Array im Uhrzeigersinn erfolgt. Hält man sich nicht an die Reihenfolge, zeigen die Flächennormalen in die falsche Richtung und das Objekt hat „Löcher“. Außer bei automatisch in OpenSCAD-Syntax umgerechneten Formen findet *polyhedron()* eher selten Verwendung, viele damit erzeugte Körper lassen sich auch durch Kombination der Grundformen erstellen. Listing 1 zeigt exemplarisch die Definition einer quadratischen Pyramide mit Kantenlänge und Höhe von 10 mm. Eine alternative Methode zum Generieren ist der im nächsten Abschnitt gezeigte *$fn*-Trick.

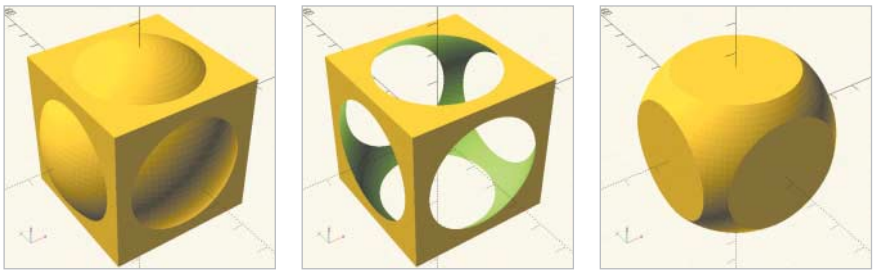
Über die Variable *$fn* kann man bei runden Körpern die Anzahl der Eckpunkte



Eine in einen Quader migrierte Kugel wird zum Ellipsoid (Abb. 6).

Listing 2: Mengenoperatoren

```
union(){
    cube(50,center=true);
    sphere(d=64, $fn=100);
}
difference(){
    cube(50,center=true);
    sphere(d=64, $fn=100);
}
intersection(){
    cube(50,center=true);
    sphere(d=64, $fn=100);
}
```



Drei Mengenoperatoren, angewandt auf eine Kugel und einen Würfel: Vereinigung (links), Differenz (Mitte) und Schnittmenge (rechts) (Abb. 7)

auf dem Umfang festlegen, mit denen OpenSCAD eine Näherung an die runde Form durch Dreiecksflächen vornimmt. Dies ist einerseits quasi ein Art Auflösung, andererseits lassen sich dadurch neue, von der Skalierung unabhängige Formen festlegen. Die folgenden Beispiele zeigen den Effekt.

```
cylinder(d=50, h=10, $fn=3);
```

erzeugt ein Dreieck,

```
cylinder(d=50, h=10, $fn=6);
```

eine sechseckige Wabenform und

```
cylinder(d1=50,d2=0,h=25, $fn=4);
```

eine Pyramide, wie beim Polyhedron-Beispiel oben, nur einfacher. Abbildung 4 zeigt beide Varianten parallel.

```
sphere(d=50, $fn=12);
```

hingegen ergibt eine sehr grob gerasterte Annäherung an eine Kugel.

Einfache Transformationen

Zum Transferieren von Objekten an die richtige Stelle dient die Funktion *translate ([x,y,z])*. Vor das jeweilige Objekt gestellt verschiebt sie es um die in den eckigen Klammern angegebenen Koordinaten. *rotate([rx,ry,rz])* rotiert in der angegebenen Reihenfolge ein Objekt um die x-, y- und z-Achse gemäß der „Rechte-Faust-Regel“: Zeigt der Daumen in Richtung der jeweiligen Achse, so zeigen die gekrümmten Finger in die positive Rotationsrichtung. Die Winkelangaben *rx, ry* und *rz* sind in Grad.

Setzt man *translate* und *rotate* in Kombination ein, so ist zu beachten, dass jedes Kommando auf das unmittelbar folgende wirkt. Die Ausführungsreihenfolge ist also von rechts nach links. Abbildung 5 zeigt das Ergebnis folgender Kombination:

```
rotate([0,45,0])
    translate([50,0,0])
        cube(10);
```

Mit *scale()* lässt sich ein Objekt in der Größe verändern, stauchen oder ziehen. Beispielsweise modifiziert

```
scale([1, 1, 1.4]) sphere(d=100);
```

eine Kugel zu einem Ei, indem es sie in z-Richtung um den Faktor 1,4 streckt. Um ein Objekt auf eine definierte Größe zu skalieren, lässt sich in der Funktion *resize* eine „Bounding Box“ angeben, in die OpenSCAD das skalierte Objekt integriert. Das Beispiel

```
resize([10,20,30]) sphere(d=100);
```

in Abbildung 6 passt eine Kugel in eine deutlich kleinere quaderförmige Abgrenzung ein.

Um Objekte zu kombinieren, genügt es, sie einfach hintereinanderzuschreiben. Überlappende Teile werden dabei nicht doppelt gezählt, das heißt, beim späteren Drucken ist nur die äußere Hülle relevant. Zum sichtbaren Zusammenfassen zweier Objekte auch im Quellcode kann das optionale Kommando *union()* dienen, das

Tutorialinhalt

Listing 3: *openscad_schleife1.scad*

```
$fn=60;

difference(){
    cylinder(d=100, h=10, center=true);
    for(rz=[0:360/8:360])
        rotate([0,0,rz])
            translate([32,0,0])
                cylinder(d=20, h=10,center=true);
}
```

Listing 4: *openscad_schleife2.scad*

```
for(x=[0:10:100],y=[0:10:100],z=[0:10:100])
    if(x == 0 || x==100
       || y==0 || y == 100
       || z == 0 || z == 100)
        translate([x,y,z])
            sphere(d=12, $fn=8);
```

Listing 5: *openscad_minkowski.scad*

```
md = 10;
translate([md/2,md/2,md/2])
  minkowski(){
    cube([50-md, 50-md, 50-md]);
    sphere(d=md);
  }
```

Listing 6: Modul *dp* für variable Doppelpyramiden

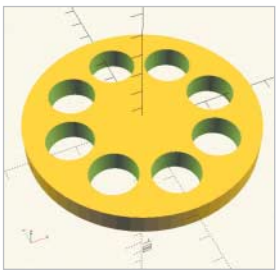
```
// Doppelpyramide
module dp(d=5){
    for(m=[0,1])
        mirror([0,0,m])
            cylinder(d1=d,d2=0,h=d/2,$fn=4);
}

// Modul aufrufen in zwei Größen
dp();
translate([10,0,0]) dp(d=10);
```

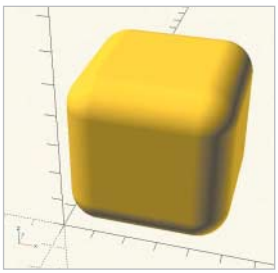
Listing 7: *mcube()* – Würfel mit gefasten Kanten

```
module mcube(v=[1,1,1],center=false,d=5){
    translate(center?[0,0,0]:[d/2,d/2,d/2])
        minkowski(){
            cube([v[0]-d,v[1]-d,v[2]-d], ↵
                center=center);
            dp(d=d);
        }
}
```

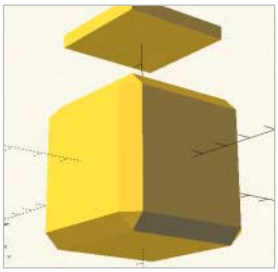
man vor die in geschweiften Klammern stehenden Objekte schreibt. Mit *difference()* lassen sich vom ersten Objekt in den geschweiften Klammern alle weiteren subtrahieren, dies ist die einfachste Variante, um Hohlräume und Löcher in Objekte zu schneiden. Die Schnittmenge aus mehreren Objekten *(intersection)* lässt nur das übrig, was in allen Komponenten enthalten ist (siehe Listing 2 und Abbildung 7).



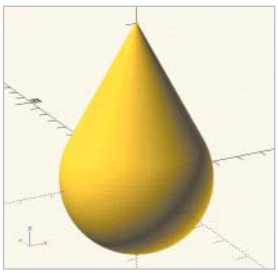
Per For-Schleife gebohrte Löcher (Abb. 8)



***minkowski()* rundet Objektkanten (Abb. 9).**



Druckvorlage einer Box mit passendem Deckel (Abb. 10)



Zum Tropfen verzerrte Kugel (Abb. 11)

Ein äußerst praktisches Konstrukt, das in jeder Programmiersprache enthalten ist, ist die Möglichkeit, über Schleifen Kommandos mehrfach und mit veränderlichen Parametern auszuführen.

Schleifen elegant einsetzen

Auch OpenSCAD bietet dieses Feature über eine syntaktisch besonders kompakt gestaltete *for()*-Schleife. Sie gibt in der eckigen Klammer Startwert, Schrittweite und Endwert vor. Spätestens hier zeigt sich der Vorteil parametrisierter Programmiersprachen gegenüber reinem Zeichnen: Es spart viel Zeit.

Das erste Schleifen-Beispiel in Listing 3 bohrt acht Löcher in den Zylinder und rotiert nach jedem Durchlauf um 360/8°. Dass genau genommen wegen des Starts bei 0 und des Endes bei 360° eins der Löcher zweimal gebohrt wird, sei hier großzügig ignoriert, da es kaum Rechenzeit kostet und zum gleichen Ergebnis führt (Listing 3 und Abbildung 8).

Im zweiten Schleifen-Beispiel wird ein Würfel aus Kugeln zusammengesetzt, die das Programm aber nur auf der äußeren Hülle generiert. Der Code fasst die eigentlich drei verschachtelten Schleifen für jede Raumrichtung in einer einzigen zusammen (Listing 4).

Komplexe Transformationen und eigene Module

Schon komplexer ist die Funktion *minkowski()*. Sie trägt auf das erste in der geschweiften Klammer angegebene Objekt Material auf, das vom Mittelpunkt des zweiten Objekts aus nach außen definiert ist.

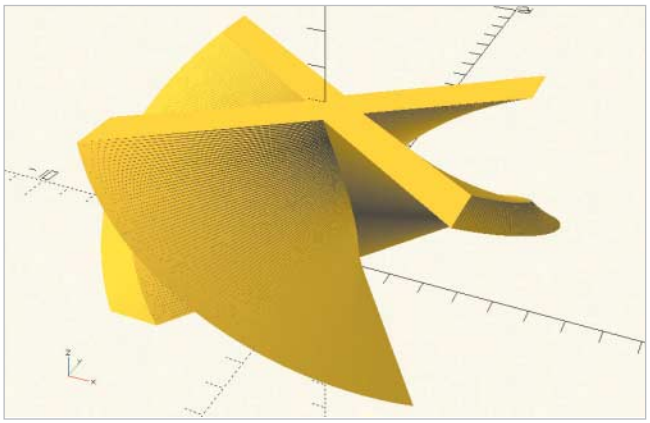
Der häufigste Anwendungsfall ist das Ersetzen scharfer Kanten durch abgerundete. Um die Abmessungen des ursprünglichen Objektes nach der Transformation zu erhalten, muss OpenSCAD dieses um den Durchmesser des aufgetragenen Körpers verkleinern und um den halben Durchmesser des aufgetragenen Körpers in x-, y- und z-Richtung verschieben. Listing 5 und Abbildung 9 zeigen dies am Beispiel eines 50×50×50 großen Würfels.

Da *minkowski()* gerade mit einer (durchaus naheliegenden) Kugel als Objekt zur Kantenglättung furchtbar langsam ist und sich die abgerundeten Ecken im FDM-Druckverfahren auch nicht aus jedem Winkel optimal drucken lassen, ist die weiter unten vorgestellte Doppelpyramide eine gute Variante, gefaste Kanten (im 45°-Winkel) zu erzeugen, die in der Berechnung schnell und im Druck relativ problemlos sind.

Module sind namentlich aufrufbare Bausteine, die sich mit Übergabeparametern auch parametrisieren lassen. Das ist vor allem dann praktisch, wenn ein Bauteil mehrfach verwendet wird, erhöht aber auch die Übersichtlichkeit, da sich so zu tief verschachtelte Schleifen oder Transformationskonstrukte vermeiden lassen.

Listing 8: Box mit passendem Deckel

```
// Box ohne Deckel
module box(){
    difference(){
        mcube([50,50,50],d=10,center=true);
        translate([0,0,5]) mcube([40,40,55],d=10,center=true);
        translate([0,0,45]) cube([40,40,55],center=true);
    }
}
// Deckel
module deckel(){
    translate([0,0,50]) cube([39.5,39.5,5],center=true);
}
```



Durch Ziehen und Drehen des Buchstabens X entsteht ein 3D-Objekt (Abb. 12).

Ein aus einer Zeichnung entstandenes Relief (Abb. 13)

Listing 6 zeigt das Modul *dp()*, das eine Doppelpyramide mit variabler Größe erzeugt (Standard: 5). Die darin genutzte OpenSCAD-Funktion *mirror()* spiegelt ein Objekt entlang einer Flächennormalen. Das *dp()*-Modul lässt sich nun in einer selbst definierten Funktion namens *mcube()* verwenden, die einen „gefasten" Würfel zeichnet (Listing 7). Da hierbei nur wenige Polygone im Spiel sind, ist die Funktion *minkowski()* auf einmal richtig schnell.

Das *mcube()*-Modul kann man nun wiederum nutzen, um eine innen hohle Box zu erzeugen. Diese soll einen passenden „versenkbaren" Deckel erhalten, der für ausreichend Spiel beim Zusammensetzen etwas schmaler sein darf (Listing 8). Wie in Abbildung 10 zu sehen, rendert der Aufruf

```
box();
deckel();
```

die Kombination aus Box und Deckel.

Eine weitere Methode, mehrere Objekte zu einem einzigen Volumenkörper zu verbinden, ist das Ziehen einer Hülle mit der Funktion *hull()*. Das Kommando

```
hull(){
     sphere(d=50);
     translate([0,0,50]) sphere(d=1); }
```

verbindet beispielsweise zwei unterschiedlich große Kugeln zu einem Tropfen (Abbildung 11).

Von 2D nach 3D: Extrusion

Mit *rotate_extrude()* lassen sich 2D-Körper um die z-Achse rotieren und bilden dabei rotationssymmetrische Objekte. Der Code in Listing 9 erzeugt einen Ring aus einem gestauchten Kreis (Ellipse).

Per *linear_extrude()* lässt sich ein 3D-Objekt vom Boden nach oben „ziehen". Optional kann man über den Parameter *twist* dabei eine Drehung durchführen. Das folgende Beispiel

Listing 9: Ring durch Rotation erzeugen

```
module ring(d=25, $fn=100){
  rotate_extrude()
    translate([d/2,0])
      scale([0.5,1])
        circle(d=5);
}
ring();
```

```
linear_extrude(height=5, twist=90, slices=100)
    text("X", valign="center", halign="center");
```

erzeugt aus dem Buchstaben X ein dreidimensionales gedrehtes Konstrukt (Abbildung 12).

Auch zum Umwandeln von Graustufenbildern in 3D-Objekte bietet OpenSCAD mit *surface()* eine Funktion. Der Grauwert eines Bildpunktes entspricht der Höhe. Da OpenSCAD alle Pixel in Eckpunkte eines Polygons umwandelt, sollte das Bild nicht allzu hoch aufgelöst sein (beispielsweise 200×200 Pixel). Aus Graustufenfotos lassen sich mit dieser Funktion sogenannte Litophane generieren, die, von der Rückseite beleuchtet, dem Foto einen beinahe holografischen Effekt verleihen. Ganz unkompliziert lassen sich so selbst Kinderzeichnungen in ein Relief oder einen Keksausstecher umwandeln (Listing 10, Abbildung 13). Der Parameter *invert* gibt dabei an, ob helle Flächen erhaben oder eingeprägt dargestellt werden sollen.

Darüber hinaus existieren weitere Varianten zur Konvertierung zwischen 3D-Bildern und OpenSCAD-Objekten. Das Programm png23d wandelt beispielsweise Logos und Ähnliches direkt in OpenSCAD-Polygone um [2]. Auch lassen sich per *import()* schon vorhandene 3D-Objekte als STL-Dateien importieren. Ob man auf diese aber auch die Funktionen *difference()* und *intersection()* anwenden kann, hängt allerdings von der Qualität der STL-Vorlage ab: Diese muss „manifold" sein, das heißt, das Modell darf keine Löcher oder unvollständige Polygone enthalten. Das ist, wie schon der erste Tutorialteil thematisiert hat, vor allem bei eingescannten Vorlagen nicht immer gewährleistet.

Listing 10: Graustufenbilder in Reliefs umwandeln

```
// cat.png 200x200 Pixel mit Gimp gezeichnet
translate([0,0,4])
  scale([1,1,0.05])
     surface(file="cat.png", invert=true);

// Grundplatte
cube([250,250,2]);
```

Fazit

Für zeichnerisch begabte Personenkreise sind interaktive 3D-Zeichenprogramme wie Blender und FreeCAD hilfreich. OpenSCAD ist ein maßgetreues Konstruktionsprogramm für die parametrische Programmierung von 3D-Objekten. Es ist vor allem auf Automatisierbarkeit und Wiederverwendbarkeit optimiert und erlaubt auch das „Skripten" von 3D-Vorlagen.

Auf Plattformen wie Thingiverse finden sich im Abschnitt „Customizable" Beispiele für anpassbare Vorlagen in OpenSCAD-Syntax. Mit diesen kann man auch ohne Programmierkenntnisse Vorlagen anpassen und die STL-Dateien neu generieren lassen. (avr@ix.de)

Prof. Dipl.-Ing. Klaus Knopper

lehrt Softwareengineering am Fachbereich Betriebswirtschaft der Hochschule Kaiserslautern und forscht unter anderem im Bereich Open-Source-Systeme sowie räumlich-akustischer Repräsentation von Softwaremodellen.

Quellen

[1] Klaus Knopper; 3D-Workflow; Abgetastet; 3D-Druck-Tutorial, Teil 1: 3D-Scan mit Open-Source-Software; *iX* 2/2019, S. 118

[2] Codebeispiele und weitere Tools: ix.de/ix1903142