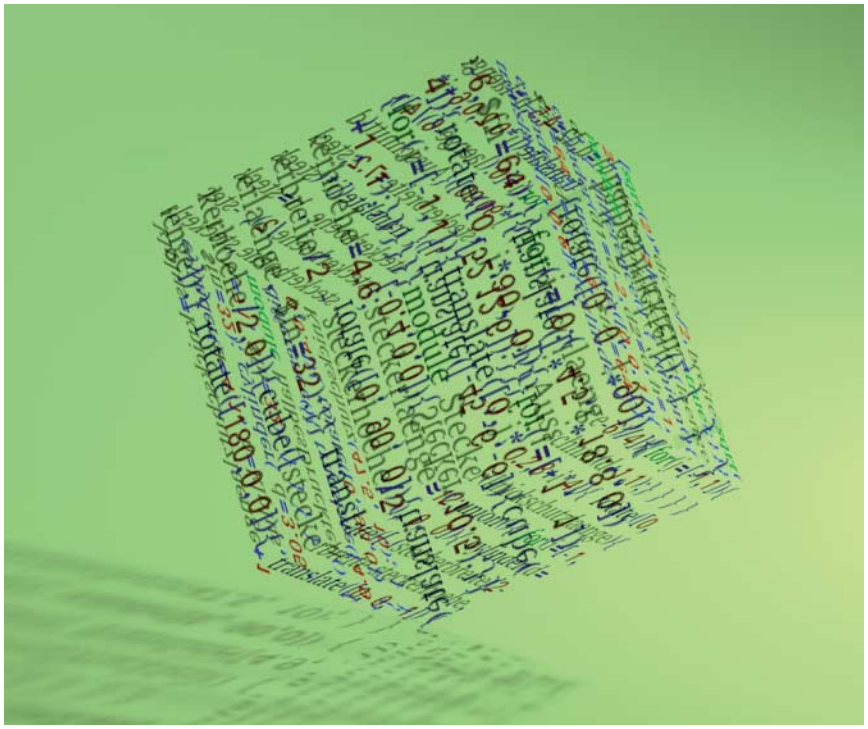


OpenSCAD

Modelle für den 3D-Druck programmieren

Wer programmieren kann – egal in welcher Sprache –, ist es gewohnt, mit Variablen, Schleifen und Funktionen zu arbeiten. Bei OpenSCAD setzt man dieses Vorwissen ein, um ohne viel Einarbeitungszeit eigene 3D-Modelle zu kreieren. Wir zeigen am Beispiel eines 3D-gedruckten Knickschutzes für Apples Lightning-Kabel die ersten Schritte mit der CAD-Programmiersprache.

Von Johannes Merkert

Angeklickt, Maß gezogen, Wert getippt. Der Würfel hängt jetzt an der Kante – nur leider falsch herum. Die Oberflächen von CAD-Programmen sind für Einsteiger oft verwirrend. Weicht man auf ein Modelling-Programm wie Blender aus, wird es nicht besser: Dort geht ohne Tastatur-Shortcuts gar nichts. Wie kommt man also mal schnell zu einem Modell für den heimischen 3D-Drucker?

Einen Königsweg, einen ohne Lernen, gibt es leider nicht. Aber wenn der König eine Programmiersprache beherrscht, kann er Zeit sparen: Bei OpenSCAD programmiert man nämlich CAD in einer C-ähnlichen Sprache und nutzt die eigene Erfahrung im Programmieren. Welche Sprache man bereits beherrscht, ist dabei relativ egal: Die Syntax von OpenSCAD ist einfach und schnell zu erlernen.

Die Bedienoberfläche kommt ohne jeglichen Schnickschnack aus. Links zeigt sie einen Texteditor für den Quellcode, rechts eine mit der Maus drehbare Vorschau des Modells. Rechts unten informiert ein Konsolenfenster über Syntaxfehler – das wars.

Die spartanische Oberfläche hat einen entscheidenden Vorteil: Selbst wer keinerlei Erfahrung mit CAD-Programmen hat, findet sich hier extrem schnell zurecht. Alle komplizierten Funktionen stecken in der Programmiersprache, in der man die Modelle definiert – und für die gibt es ein umfangreiches WikiBook (siehe ct.de/y6kw). Wer also nur ein paar Quader mit Bohrungen versehen will, lernt einfach diesen Teil der Sprache und ignoriert, was OpenSCAD sonst noch alles kann. Für Programmierer gelingt der Einstieg daher sehr schnell.

Um den Einstieg noch mehr zu beschleunigen, gibt es diesen Artikel. Wir haben uns ein Beispiel gesucht, das ein reales Problem löst, und erklären damit die wichtigsten Elemente der Sprache.

Knickstelle und Kabelbruch

Apples originale Lightning-Kabel, um iPhones an USB-Ladegeräte anzuschließen, erleiden oft schon nach kurzer Zeit einen Kabelbruch kurz hinter dem kleinen Stecker. Das liegt daran, dass sich das Kabel im Alltag oft an dieser Stelle mit kleinem Radius biegt und Apple keine ausreichenden Maßnahmen gegen diese mechanische Belastung unternommen hat. Abhilfe schafft ein Knickschutz mit Rippen aus Gummi, wie man ihn von den Steckerenden vieler Stecker anderer Hersteller kennt. Die Rippen schieben sich beim Biegen des Kabels so zusammen, dass nur ein Abknicken mit erheblich größerem Radius möglich ist. Das schont die Litzen im Inneren und das Kabel bleibt heil.

Einen solchen Knickschutz druckt auch der heimische 3D-Drucker, sofern er flexibles Filament verarbeiten kann. Für flexible Filamente eignen sich besonders 3D-Drucker mit sogenannten Direct-Drive-Extrudern, bei denen der Extrudermotor direkt über der Druckdüse sitzt. Außerdem hilft es, wenn der Drucker Filament mit 3 mm Durchmesser verarbeitet statt 1,75 mm, da sich das dickere Filament weniger verbiegt und die Reibung im Extruder so etwas kleiner bleibt. Wir

Das Problem: Ohne Knickschutz erleiden Apples Lightning-Kabel schon nach kurzer Zeit einen Kabelbruch.

haben rotes 3 mm InnoFlex 45 auf einem Ultimaker 2 Plus gedruckt – für das Teil eignen sich aber auch viele andere flexible Filamente und Drucker.

Addieren und Abziehen

OpenSCAD bringt eine Reihe von Grundkörpern wie Quader, Zylinder und Kugeln mit, die sich über Parameter wie Höhe, Breite, Durchmesser et cetera anpassen lassen. Designs basieren darauf, dass man Formen zusammenfügt (Addition) oder eine aus der anderen herausschneidet (Differenz). Eine Platte mit Bohrungen entsteht also als flacher Quader, von dem die Bohrungen als Zylinder herausgezogen wurden.

Für unseren Knickschutz programmierten wir zwei Objekte: einen grob zylinderförmigen Knickschutz ohne Aussparung in der Mitte und einen Teil des Lightning-Steckers samt Kabel, um diese Form vom Knickschutz abzuziehen. So entsteht ein Teil mit einer Aussparung, die exakt der Form des Steckers folgt. Damit sich das entstehende Objekt drucken lässt, muss man es in zwei Teile zerschneiden und auf dem Drucktisch positionieren. Dafür klonten wir das Teil und entfernten jeweils die untere Hälfte, indem wir einen größeren Quader davon abzogen.

Variablen und Vektoren

Da OpenSCAD eine Programmiersprache ist, erlaubt sie, Variablen zu definieren. Das kann sehr sinnvoll sein, da gerade beim 3D-Druck nicht immer klar ist, welche Toleranzen man einbauen muss, damit Teile exakt passen. Ein Testdruck zeigt zwar, wo es hakt, es nervt aber, dann den Code nach allen Stellen zu durchsuchen, an denen das falsche Maß steht. CAD-Programmierer definieren sich daher sinnvollerweise Variablen für die wichtigsten Werte:

```
steckerbreite = 7.2;
steckerhoehe = 4.6;
steckerlaenge = 12;
```

Mit den Variablen lässt sich natürlich auch rechnen. Beispielsweise berechnet `sqrt(2)*dicke/2` die Hälfte der Dicke multipliziert mit der Wurzel von zwei. Über solche Formeln sorgt man dafür, dass das gesamte Objekt nur von wenigen Grundwerten abhängt. Außerdem spart man sich den Taschenrechner. Um Platz zu sparen, haben wir in unserem Quellcode jedoch dort, wo sich Maße nicht ändern, direkt Zahlen angegeben.

Drei Werte in eckigen Klammern interpretiert OpenSCAD als Elemente eines 3D-Vektors. Mit Vektoren definiert man Quader (`cube([breite, tiefe, hoehe])`), Verschiebungen (`translate([x, y, z])`) und Drehungen (`rotate([drehung_um_x, drehung_um_y, drehung_um_z])`).

Eckige Klammern definieren aber auch beliebig lange Listen:

```
winkel_liste = [-5, 45, 90, 180, 360];
```

Alle Werte sind Längen in Millimetern oder Winkel in Grad.

Vorschau: F5

Ein Druck auf die Taste F5 erzeugt eine Vorschau des gerade programmierten Objekts. Die erscheint im rechten Fenster, wo man sie mit der Maus drehen kann. Bewegungen mit gedrückter linker Maustaste drehen den Sichtwinkel um das Objekt. Bei gedrückter rechter Maustaste verschiebt sich die Ansicht und das Mausrad sorgt für einen Zoom.

Schieben und Schleifen

Der Kunststoffteil des Lightning-Steckers ist 7 mm breit, aber nur 4,5 mm dick und hat abgerundete Kanten. Die Rundungen entsprechen jeweils einem halben Zylinder mit 4,5 mm Durchmesser. Einen Zylinder mit gleich bleibendem Durchmesser definiert:

```
cylinder(d=steckerhoehe,
         h=steckerlaenge, $fn=32);
```

OpenSCAD kann keine echten Rundungen darstellen. Deshalb legt der Parameter `$fn=32` fest, dass die kreisförmigen Kanten des Zylinders durch ein regelmäßiges Vieleck mit 32 Ecken angenähert werden. Braucht man beispielsweise eine Vertiefung im Druckteil, um eine Mutter einzulassen, definiert man einfach einen Zylinder mit `$fn=6`.

Für die Rundungen des Steckers könnte man zwei dieser Zylinder definieren und per `translate()` gleichmäßig nach links und rechts verschieben. Das wäre aber Handarbeit, die man sich mit einer Schleife sparen kann:

```
for(x=[-1, 1]) {
  translate([x * (steckerbreite / 2 -
             steckerhoehe / 2), 0, 0]) {
    cylinder(d=steckerhoehe,
             h=steckerlaenge, $fn=32);
}}
```

Die Schleife iteriert über die kleine Liste. Das heißt, `translate()` und `cylinder()` werden zweimal ausgeführt, wobei die Variable `x` zuerst den Wert -1 und danach den Wert 1 hat. Die Verschiebungsweite `(steckerbreite / 2 – steckerhoehe / 2)` sorgt dafür, dass die Außenkanten der beiden Zylinder an der weitesten Stelle genau die `steckerbreite` ergeben.

Damit die Form dem Lightning-Stecker entspricht, füllt ein Quader den Zwischenraum:

```
translate([-(steckerbreite / 2 -
            steckerhoehe / 2),
           -steckerhoehe / 2, 0])
  cube([steckerbreite - steckerhoehe,
        steckerhoehe, steckerlaenge]);
```

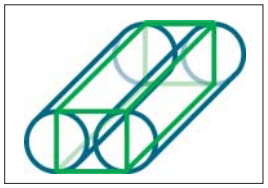
Das `translate()` ist hier nötig, weil ein Quader von einer Ecke aus definiert wird, während Zylinder aus dem Mittelpunkt des Kreises um ihre Basis emporwachsen.

Im kompletten Quellcode auf GitHub verschmilzt die `union()`-Funktion die drei Objekte noch zu einem einzigen.

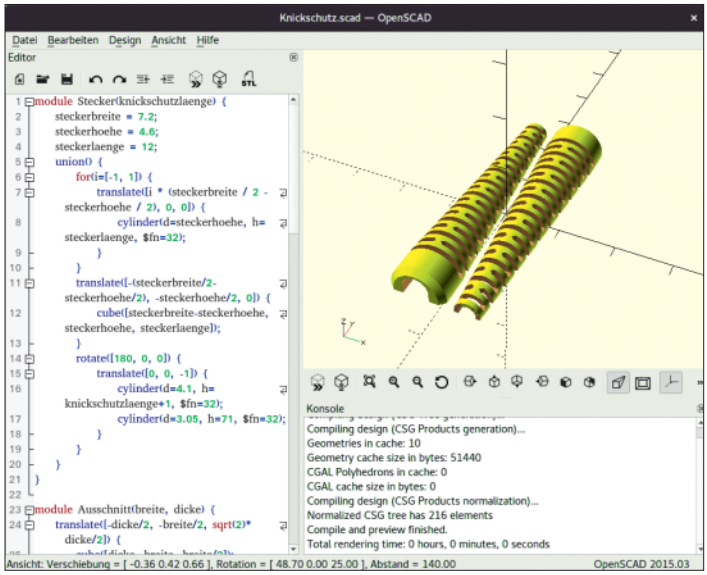
Konvexer Konus

Die Grundform des Knickschutzes ist rund, basiert also auf einem Zylinder. Damit am Ende des Knickschutzes keine Kante entsteht, an der das Kabel wieder abknicken könnte, soll sich der Zylinder zum Kabel hin verjüngen. Im Code definiert man das mit den Parametern `d1` und `d2`, die man statt `d` an die `cylinder()`-Funktion übergibt. `d1` legt den Durchmesser am Fuß und `d2` den am oberen Ende des Zylinders fest, der dadurch zum Kegelstumpf wird. Aus ästhetischen Gründen wollten wir den Teil am Stecker mit konstantem Durchmesser modellieren und das Steckerende mit einer Fase versehen. Die Grundform entsteht damit aus drei `cylinder()`-Objekten:

```
union() {
  cylinder(d1=durchmesser - 2,
           d2=durchmesser,
           h=1, $fn=64);
  translate([0, 0, 1])
    cylinder(d=durchmesser,h=5,$fn=64);
  translate([0, 0, 6])
    cylinder(d1=durchmesser, d2=4.5,
             h=laenge-6, $fn=64); }
```



Stecken mehrere Formen ineinander, verschmelzen sie.



Links Quelltext, rechts Vorschau: Die Oberfläche von OpenSCAD durchschaut jeder in ein paar Sekunden.

Modularisierte Messer

Ein solider Kegelstumpf wäre viel zu steif als Knickschutz. Erst durch Aussparungen aus vier Richtungen erhält er die nötige Flexibilität. Wir wollten aber nicht einfach nur Quader aussparen, sondern sie auch hübsch im 45-Grad-Winkel anfasen. Da dieses „Messer“ eine große Menge an Aussparungen schneiden soll, lohnt es sich, dafür ein Modul zu definieren:

```
module Ausschnitt(breite, dicke) {
  translate([-dicke/2, -breite/2,
             sqrt(2)*dicke/2])
    cube([dicke, breite, breite/2]);
  translate([0, 0, sqrt(2)*dicke/2])
    rotate([0, 45, 0])
      cube([sqrt(2)*dicke/2,
            breite,
            sqrt(2)*dicke/2],
           center=true); }
```

Module definieren parametrisierbare Objekte. Über den vergebenen Namen lassen sie sich wiederverwenden.

```
difference() {
  // hier wird die Grundform definiert
  for(i=[0:(laenge-6)/4]) {
    for(j=[-1, 1]) {
      translate([j*0.2, 0, 6+i*4])
        rotate([0, j*90, 0])
          Ausschnitt(durchmesser+1,1);
      translate([0, j*0.2, 8+i*4])
        rotate([0, j*90, 90])
          Ausschnitt(durchmesser+1,1);
}}}
```

Der `Ausschnitt()` schlitzt nun die Grundform. Damit liegen sich immer zwei Ausschnitte um 180 Grad gedreht gegenüber (zweite `for`-Schleife) und ab 6 mm Abstand hinterm Stecker kommt alle 4 mm ein `Ausschnitt()`-Paar (erste `for`-Schleife).

Die erste `for`-Schleife soll über die gesamte `laenge` Ausschnitte wegstanzen. Die Anzahl der Werte, die die Laufvariable `i` dabei annehmen muss, soll daher mit der `laenge` wachsen. Neben Listen, die alle Werte einzeln angeben, erlaubt OpenSCAD auch, Bereiche zu definieren. Die Syntax dafür verwendet einen Doppelpunkt: `[startwert:zielwert]`. Im Beispiel nimmt `i` also ganzzahlige Werte von `0` bis `(laenge-6)/4` an.

Professionell positioniert

Für den fertigen Knickschutz zieht der Code noch den Lightning-Stecker von der Grundform ab und halbiert das Objekt, damit es flach auf dem Druckbett aufliegt. Da wir Stecker und Grundform aufrecht stehend entworfen haben, müssen wir beide Objekte passend drehen und verschieben. Lässt man die geschweiften Klammern hinter einem Befehl wie `rotate()` weg, bezieht er sich nur auf den nächsten Befehl oder das nächste Objekt:

```
module halbesDruckteil() {
  difference() {
    translate([4.9, 0, 0.4])
      rotate([0, -90, 0])
        Knickschutz(47.5, 10);
    rotate([0, 90, 0])
      translate([-0.4, 0, 0])
        Stecker(10.5);
    translate([-45, -6, -6])
      cube([52, 12, 6]); }}
```

Um beide Hälften in einem Schritt zu drucken, positioniert sie der folgende Code platzsparend nebeneinander:

```
for(i=[-1, 1]) {
  translate([i*4.5, i*18.8, 0])
    rotate([0, 0, i*90])
      halbesDruckteil(); }
```

In der Vorschau sieht das Modell nun fertig aus. Um eine STL-Datei exportieren zu können, muss man OpenSCAD jedoch noch mit F6 anweisen, eine aufwendigere Berechnung der Polygone durchzuführen, was mehrere Sekunden dauert. Danach speichert OpenSCAD die Datei ohne Murren über den STL-Button.

Brandheiße Bastelei

Wir haben die beiden Hälften mit einem heißen Messer verschweißt. Dafür schiebt man die Klinge zwischen die Klebeflächen und zieht sie langsam heraus, während man die beiden Hälften aufeinanderdrückt. Das Messer verschmiert dabei mit flexiblem Plastik – nehmen Sie also kein wertvolles Küchenmesser dafür.

Der fertige Knickschutz schützt nun effektiv das Lightning-Kabel vor Kabelbrüchen. Den kompletten OpenSCAD-Code von überschaubaren 90 Zeilen finden Sie über ct.de/y6kw.

Unser Beispiel zeigt, dass man als Programmierer mit OpenSCAD schnell komplexe Vorlagen für Druckobjekte programmieren kann. Dass man sich dabei in keine unübersichtliche Oberfläche einarbeiten muss, spart viel Zeit. Die CAD-Programmiersprache eignet sich jedoch nicht für jedes Objekt gleich gut. Beispielsweise bringen andere CAD-Programme spezialisierte Funktionen für das Entwerfen von Urmodellen für Gussformen mit. Bei organischen Formen eignen sich Programme wie Blender besser, da sie Subdivision-Surfaces beherrschen. Für 3D-gedruckte Halterungen, Gehäuse oder auch Getriebe eignet sich OpenSCAD jedoch hervorragend. *(jme@ct.de)* c't

3D-Modell auf GitHub, Wiki-Book: ct.de/y6kw

Die Lösung: Der selbst gedruckte Knickschutz verhindert Abknicken und Kabelbruch.