OpenSCAD

- Allgemein
- 3D-objekte
- 2D-Objekte
- Transformationen
- Boolesche Kombinationen
- Andere Funktionen und Operationen
- Benutzerdefinierte Funktionen und Module
- Debugging Hilfsmittel
- Externe Bibliotheken und Code-Dateien

Allgemein

Einleitung

OpenSCAD ist ein 2D / 3D- und Volumenmodellierungsprogramm, das auf einer funktionalen Programmiersprache basiert, mit der Modelle erstellt werden, die auf dem Bildschirm in der Vorschau angezeigt und in ein 3D-Netz gerendert werden, mit dem das Modell in verschiedenen 2D / 3D-Dateiformaten exportiert werden kann .

Ein Skript in der OpenSCAD-Sprache wird zum Erstellen von 2D- oder 3D-Modellen verwendet. Dieses Skript ist eine frei formatierte Liste von Aktionsanweisungen.

```
Objekt();
Variable = Wert;
operator () action ();
operator () {action (); aktion(); }}
operator () operator () {action (); aktion(); }}
operator () {operator () action ();
operator () {action (); aktion(); }}
```

Objekte

Objekte sind die Bausteine für Modelle, die mit 2D- und 3D-Grundelementen erstellt werden. Objekte enden mit einem Semikolon ';'.

Aktionen

Zu den Aktionsanweisungen gehört das Erstellen von Objekten mithilfe von Grundelementen und das Zuweisen von Werten zu Variablen. Aktionsanweisungen enden auch mit einem Semikolon ';'.

Operatoren

Operatoren oder Transformationen ändern die Position, Farbe und andere Eigenschaften von Objekten. Operatoren verwenden geschweifte Klammern '{}', wenn ihr Bereich mehr als eine Aktion abdeckt. Es kann mehr als ein Operator für dieselbe Aktion oder Gruppe von Aktionen verwendet werden. Mehrere Operatoren werden von rechts nach links verarbeitet, d.h. der Operator, der der Aktion am nächsten liegt, wird zuerst verarbeitet. Operatoren enden nicht mit Semikolons ';', sondern mit den einzelnen darin enthaltenen Aktionen.

Beispiele

```
cube(5);
x = 4+y;
rotate(40) square(5,10);
translate([10,5]) { circle(5); square(4); }
rotate(60) color("red") { circle(5); square(4); }
color("blue")
    {
    translate([5,3,0]) sphere(5); rotate([45,0,45])
    {
    cylinder(10); cube([5,6,7]);
    }
  }
```

2. Kommentare

Kommentare sind eine Möglichkeit, Notizen im Skript oder Code zu hinterlassen (entweder für Sie selbst oder für zukünftige Programmierer), die beschreiben, wie der Code funktioniert oder was er tut. Kommentare werden vom Compiler nicht ausgewertet und sollten nicht zur Beschreibung von selbstverständlichem Code verwendet werden.

OpenSCAD verwendet Kommentare im C ++ - Stil:

```
// Dies ist ein Kommentar
myvar = 10; // Der Rest der Zeile ist ein Kommentar
/ *
```

```
Mehrzeilige Kommentare
kann mehrere Zeilen umfassen.
* /
```

3. Werte und Datentypen

Ein Wert in OpenSCAD ist entweder eine Zahl (wie 42), ein boolescher Wert (wie true), ein String (wie "Hello World"), ein Bereich (wie [0:1:10]), ein Vektor (wie [1,2,3]) oder der undefinierte Wert (undef). Werte können in Variablen gespeichert, als Funktionsargumente übergeben und als Funktionsergebnisse zurückgegeben werden.

OpenSCAD ist eine dynamisch typisierte Sprache mit einem festen Satz von Datentypen. Es gibt keine Typnamen und keine benutzerdefinierten Typen. Funktionen sind keine Werte. Tatsächlich belegen Variablen und Funktionen ***disjunkte Namespaces****.*

3.1. Zahlen

Zahlen sind der wichtigste Werttyp in OpenSCAD und werden in der bekannten Dezimalschreibweise geschrieben, die in anderen Sprachen verwendet wird. ZB -1, 42, 0.5, 2.99792458e+8.
OpenSCAD unterstützt keine oktale oder hexadezimale Notation für Zahlen.

Zusätzlich zu den Dezimalzahlen werden die folgenden Namen für spezielle Zahlen definiert:

- PI

OpenSCAD hat nur eine einzige Art von Zahl, nämlich eine 64-Bit-IEEE-Gleitkommazahl.
OpenSCAD unterscheidet weder Ganzzahlen und Gleitkommazahlen als zwei verschiedene Typen, noch unterstützt es komplexe Zahlen.
Da OpenSCAD den IEEE-Gleitkommastandard verwendet, gibt es einige Abweichungen vom Verhalten von Zahlen in der Mathematik:

- Wir verwenden binäre Gleitkommazahlen. Eine Bruchzahl wird nicht genau dargestellt, es sei denn, der Nenner ist eine Potenz von 2. Beispielsweise hat 0,2 (2/10) keine exakte

interne Darstellung, aber 0,25 (1/4) und 0,125 (1/8) werden genau dargestellt .

- Die größte darstellbare Zahl ist ungefähr 1e308. Wenn ein numerisches Ergebnis zu groß ist, kann das Ergebnis unendlich sein (durch Echo als inf gedruckt).
- Die kleinste darstellbare Zahl ist ungefähr -1e308. Wenn ein numerisches Ergebnis zu klein ist, kann das Ergebnis -infinity sein (durch Echo als -inf gedruckt).
- Wenn ein numerisches Ergebnis ungültig ist, kann das Ergebnis Not A Number sein (durch Echo als nan gedruckt).
- Wenn ein negatives numerisches Ergebnis zu nahe an Null liegt, um darstellbar zu sein, ist das Ergebnis -0, wenn das Ergebnis negativ ist, andernfalls ist es 0. Null (0) und negative Null (-0) werden als zwei unterschiedliche Zahlen behandelt durch einige der mathematischen Operationen und werden durch "Echo" unterschiedlich gedruckt, obwohl sie als gleich verglichen werden.

Die Konstanten 'inf' und 'nan' werden von OpenSCAD nicht als numerische Konstanten unterstützt, obwohl Sie Zahlen berechnen können, die auf diese Weise mit 'echo' gedruckt werden. Sie können Variablen mit diesen Werten definieren, indem Sie Folgendes verwenden:

```
inf = 1e200 * 1e200;
nan = 0/0;
Echo (inf, nan);
```

Der Wert 'nan' ist der einzige OpenSCAD-Wert, der keinem anderen Wert, einschließlich sich selbst, entspricht. Obwohl Sie mit 'x == undef' testen können, ob eine Variable 'x' den undefinierten Wert hat, können Sie mit 'x == 0/0' nicht testen, ob x keine Zahl ist. Stattdessen müssen Sie 'x! = X' verwenden, um zu testen, ob x nan ist.

3.2. Boolesche Werte

Boolesche Werte sind Wahrheitswerte. Es gibt zwei Boolesche Werte, nämlich `true` und `false`. Ein Boolescher Wert wird als Argument an die bedingte Anweisung `if ()` übergeben, bedingter Operator `?` und logische Operatoren `!` (nicht), `&&` (und) und `||` (oder). In all diesen Kontexten können Sie tatsächlich eine beliebige Menge übergeben. Die meisten Werte werden in einem booleschen Kontext in `true` konvertiert. Die Werte, die als `false` gelten, sind:

- `false`
- `0` und `-0`
- `""`
- `[]`
- `undef`

Beachten Sie, dass `"false"` (die Zeichenfolge), `[0]` (ein numerischer Vektor), `[[]]` (ein Vektor, der einen leeren Vektor enthält), `[false]` (ein Vektor, der den Booleschen Wert false enthält) und `0/0` (Not A Number) alle als wahr gelten.

3.3. Strings

Eine Zeichenfolge ist eine Folge von null oder mehr Unicode-Zeichen. Zeichenfolgenwerte werden verwendet, um beim Importieren einer Datei Dateinamen anzugeben und bei Verwendung von `echo ()` Text für Debugging-Zwecke anzuzeigen. Strings können auch mit dem neuen Grundelement `text ()` verwendet werden, das in 2015.03 hinzugefügt wurde.

Ein Zeichenfolgenliteral wird als Folge von Zeichen geschrieben, die in Anführungszeichen `"`, wie folgt: `" "` (eine leere Zeichenfolge) oder `"Dies ist eine Zeichenfolge"` eingeschlossen sind.

Verwenden Sie `\"` , um ein `"`-Zeichen in ein Zeichenfolgenliteral aufzunehmen. Verwenden Sie `\\` , um ein `\`-Zeichen in ein Zeichenfolgenliteral aufzunehmen. Die folgenden Escape-Sequenzen, die mit \ beginnen, können in Zeichenfolgenliteralen verwendet werden:

- `\"` → `"`
- `\\` → `\`
- `\t` → tab
- `\n` → newline
- `\r` → carriage return
- `\u03a9` → `Ω` - Weitere Informationen zu Unicode-Zeichen finden Sie in **text ()**

Hinweis: Dieses Verhalten ist seit OpenSCAD-2011.04 neu. Sie können alte Dateien mit dem folgenden Befehl sed aktualisieren: `sed 's/\\/\\\\/g' non-escaped.scad > escaped.scad`

Beispiel:
`echo("The quick brown fox \tjumps \"over\" the lazy dog.\rThe quick brown fox.\nThe \\lazy\\ dog.");`

Ergebnis

`ECHO: "The quick brown fox jumps "over" the lazy dog. The quick brown fox.`
`The \lazy\ dog."`

altes Ergebnis
`ECHO: "The quick brown fox \tjumps \"over\" the lazy dog. The quick brown fox.\nThe \\lazy\\ dog."`

3.4. Bereiche

Bereiche werden von `for()`-Schleifen und `children()` verwendet. Sie haben 2 Sorten:

`[<start>: <ende>]`
`[<start>: <Inkrement>: <ende>]`

Obwohl in eckigen Klammern `[]` eingeschlossen, sind sie keine Vektoren. Sie verwenden Doppelpunkte `:` für Trennzeichen anstelle von Kommas.

```
r1 = [0:10];
r2 = [0,5:2,5:20];
Echo (r1); // ECHO: [0: 1: 10]
Echo (r2); // ECHO: [0,5: 2,5: 20]
```

Sie sollten Schrittwerte vermeiden, die nicht genau als binäre Gleitkommazahlen dargestellt werden können. Ganzzahlen sind in Ordnung, ebenso wie Bruchwerte, deren Nenner eine Zweierpotenz ist. Zum Beispiel sind 0,25 (1/4) und 0,125 (1/8) sicher, aber 0,2 (2/10) sollten vermieden werden. Das Problem bei diesen Schrittwerten besteht darin, dass Ihr Bereich aufgrund ungenauer Arithmetik möglicherweise zu viele oder zu wenige Elemente enthält.

Ein fehlendes `<Inkrement>` ist standardmäßig 1. Ein Bereich in der Form `[<start>: <ende>]` mit `<start>` größer als `<ende>` generiert eine Warnung und entspricht `[<ende>: 1: <start>]`. Ein Bereich in der Form `[<start>: 1: <ende>]` mit `<start>` größer als `<ende>` generiert keine Warnung und entspricht `[]`. Das `<Inkrement>` in einem Bereich kann negativ sein (für Versionen nach 2014).

3.5. Der undefinierte Wert

Der undefinierte Wert ist ein spezieller Wert, der als `undef` geschrieben wird. Dies ist der Anfangswert einer Variablen, der kein Wert zugewiesen wurde, und er wird häufig als Ergebnis von Funktionen oder Operationen zurückgegeben, denen unzulässige Argumente übergeben werden. Schließlich kann `undef` als Nullwert verwendet werden, der in anderen Programmiersprachen `null` oder `NULL` entspricht.

Alle arithmetischen Ausdrücke, die `undef`-Werte enthalten, werden als `undef` ausgewertet. In logischen Ausdrücken entspricht `undef` `false`. Vergleichsoperatorausdrücke mit `undef` werden als `false` ausgewertet, mit Ausnahme von `undef == undef`, was wahr ist.

Beachten Sie, dass numerische Operationen auch `'nan'` (keine Zahl) zurückgeben können, um ein unzulässiges Argument anzuzeigen. Zum Beispiel: `0/false` ist `undef`, aber `0/0` ist `'nan'`. Vergleichsoperatoren wie `<` und `>` geben `false` zurück, wenn unzulässige Argumente übergeben werden. Obwohl `undef` ein Sprachwert ist, ist `'nan'` dies nicht.

4. Variablen

OpenSCAD-Variablen werden durch eine Anweisung mit einem Namen oder einer Kennung, eine Zuweisung über einen Ausdruck und ein Semikolon erstellt. Die Rolle von Arrays, die in vielen imperativen Sprachen zu finden sind, wird in OpenSCAD über Vektoren behandelt.

```
var = 25;
xx = 1,25 * cos (50);
y = 2 * xx + var;
Logik = wahr;
MyString = "Dies ist eine Zeichenfolge";
a_vector = [1,2,3];
rr = a_vector [2]; // Mitglied des Vektors
Bereich 1 = [-1,5: 0,5: 3]; // for () Schleifenbereich
xx = [0: 5]; // Alternative für () Schleifenbereich
```

OpenSCAD ist eine funktionale Programmiersprache, da solche Variablen aufgrund der Anforderungen an die referenzielle Transparenz an Ausdrücke gebunden sind und während ihrer gesamten Lebensdauer einen einzigen Wert behalten. In imperativen Sprachen wie C wird dasselbe

Verhalten als Konstanten angesehen, die typischerweise normalen Variablen gegenübergestellt werden.

Mit anderen Worten, OpenSCAD-Variablen ähneln eher Konstanten, weisen jedoch einen wichtigen Unterschied auf. Wenn Variablen mehrmals ein Wert zugewiesen wird, wird an allen Stellen im Code nur der zuletzt zugewiesene Wert verwendet. Weitere Informationen finden Sie unter Variablen werden zur Kompilierungszeit und nicht zur Laufzeit festgelegt. Dieses Verhalten ist auf die Notwendigkeit zurückzuführen, über die Verwendung der Option -D variable = value eine Variableneingabe in der Befehlszeile bereitzustellen. OpenSCAD platziert diese Zuweisung derzeit am Ende des Quellcodes und muss daher zulassen, dass der Wert einer Variablen zu diesem Zweck geändert wird.

Werte können zur Laufzeit nicht geändert werden. Alle Variablen sind effektiv Konstanten, die sich nicht ändern. Jede Variable behält ihren zuletzt zugewiesenen Wert zur Kompilierungszeit gemäß den funktionalen Programmiersprachen. Im Gegensatz zu imperativen Sprachen wie C ist OpenSCAD keine interative Sprache, und als solche ist das Konzept von `x = x + 1` nicht gültig. Das Verständnis dieses Konzepts führt zum Verständnis der Schönheit von OpenSCAD.

Vor der Version 2015.03
Es war an keiner Stelle möglich, Aufgaben zu erledigen, außer auf der obersten Ebene der Datei und der obersten Ebene des Moduls. Innerhalb einer if / else- oder for-Schleife wurde assign () benötigt.

Seit Version 2015.03
Variablen können jetzt in jedem Bereich zugewiesen werden. Beachten Sie, dass Zuweisungen nur innerhalb des Bereichs gültig sind, in dem sie definiert sind. Sie dürfen weiterhin keine Werte an einen äußeren Bereich weitergeben. Weitere Informationen finden Sie unter Umfang der Variablen.

```
a = 0;
if (a == 0)
  {
 a = 1; // vor 2015.03 würde diese Zeile einen Kompilierungsfehler erzeugen
      // seit 2015.03 kein Fehler mehr, aber der Wert a = 1 ist auf die geschweiften Klammern
{} beschränkt
  }}
```

4.1. Undefinierte Variable

Eine nicht zugewiesene Variable hat den Sonderwert undef. Es könnte im bedingten Ausdruck getestet und von einer Funktion zurückgegeben werden.

Beispiel

```
Echo ("Variable a ist", a); // Variable a ist undef
if (a == undef) {
  echo ("Variable a wird undefiniert getestet"); // Variable a wird undefiniert getestet
}}
```

4.2. Umfang der Variablen

Wenn Operatoren wie `translate()` und `color()` mehr als eine Aktion umfassen müssen (Aktionen enden mit;), werden geschweifte Klammern `{}` benötigt, um die Aktionen zu gruppieren und einen neuen inneren Bereich zu erstellen. Wenn nur ein Semikolon vorhanden ist, sind geschweifte Klammern normalerweise optional.

Jedes Klammerpaar erstellt einen neuen Bereich innerhalb des Bereichs, in dem sie verwendet wurden. Seit 2015.03 können in diesem neuen Bereich neue Variablen erstellt werden. Variablen, die in einem äußeren Bereich erstellt wurden, können neue Werte zugewiesen werden. Diese Variablen und ihre Werte stehen auch weiteren inneren Bereichen zur Verfügung, die in diesem Bereich erstellt wurden, stehen jedoch keinem Objekt außerhalb dieses Bereichs zur Verfügung. Variablen haben immer noch nur den letzten Wert, der innerhalb eines Bereichs zugewiesen wurde.

```
Bereich 1
a = 6;                  // erstelle a
Echo (a, b);            // 6, undef
translate ([5,0,0]) {   // Bereich 1.1
  a = 10;
  b = 16;               // erstelle b
  Echo (a, b);          // 100, 16 a = 10; wurde später von a = 100 überschrieben;
  Farbe ("blau") {      // Bereich 1.1.1
    Echo (a, b);        // 100, 20
```

```
    Würfel();
    b = 20;
  }                     // zurück zu 1.1
  Echo (a, b);          // 100, 16
  a = 100;              // überschreibe a in 1.1
}                       // zurück zu 1
Echo (a, b);            // 6, undef
Farbe ("rot") {         // Bereich 1.2
  Würfel();
  Echo (a, b);          // 6, undef
}                       // zurück zu 1
Echo (a, b);            // 6, undef

// In diesem Beispiel sind die Bereiche 1 und 1.1 äußere Bereiche von 1.1.1, 1.2 jedoch nicht.
```

Anonyme Bereiche gelten nicht als Bereiche:

```
{
  Winkel = 45;
}}
Quadrat (10) drehen (Winkel);
```

`For()`-Schleifen sind keine Ausnahme von der Regel für Variablen mit nur einem Wert innerhalb eines Bereichs. Für jeden Durchgang wird eine Kopie des Schleifeninhalts erstellt. Jeder Durchlauf erhält einen eigenen Bereich, sodass alle Variablen eindeutige Werte für diesen Durchgang haben können. Nein, du kannst immer noch nicht `a = a + 1` machen;

4.3. Variablen werden zur Kompilierungszeit und nicht zur Laufzeit festgelegt

Da OpenSCAD seine Variablenwerte zur Kompilierungszeit und nicht zur Laufzeit berechnet, gilt die letzte Variablenzuweisung innerhalb eines Bereichs überall in diesem Bereich oder in inneren Bereichen davon. Es kann hilfreich sein, sie als überschreibbare Konstanten und nicht als Variablen zu betrachten.

```
// Der Wert von 'a' gibt nur den zuletzt eingestellten Wert wieder
   a = 0;
   Echo (a); // 5
   a = 3;
   Echo (a); // 5
   a = 5;
```

Dies scheint zwar nicht intuitiv zu sein, ermöglicht Ihnen jedoch einige interessante Dinge: Wenn Sie beispielsweise Ihre gemeinsam genutzten Bibliotheksdateien so einrichten, dass Standardwerte als Variablen auf ihrer Stammebene definiert sind, wenn Sie diese Datei in Ihren eigenen Code aufnehmen können Sie diese Konstanten neu definieren oder überschreiben, indem Sie ihnen einfach einen neuen Wert zuweisen.

4.4. Spezielle Variablen

Spezielle Variablen bieten eine alternative Möglichkeit, Argumente an Module und Funktionen zu übergeben. Alle Variablen, die mit einem '$' beginnen, sind spezielle Variablen, ähnlich wie spezielle Variablen in lisp. Als solche sind sie dynamischer als reguläre Variablen. (Weitere Informationen finden Sie unter Andere Sprachfunktionen.)

5. Vektoren

Ein Vektor ist eine Folge von null oder mehr OpenSCAD-Werten. Vektoren sind eine Sammlung (oder Liste oder Tabelle) von numerischen oder booleschen Werten, Variablen, Vektoren, Zeichenfolgen oder einer beliebigen Kombination davon. Sie können auch Ausdrücke sein, die einen dieser Ausdrücke ergeben. Vektoren übernehmen die Rolle von Arrays, die in vielen imperativen Sprachen vorkommen. Die Informationen hier gelten auch für Listen und Tabellen, die Vektoren für ihre Daten verwenden.

Ein Vektor hat eckige Klammern `[]`, die null oder mehr Elemente enthalten, die durch Kommas getrennt sind. Ein Vektor kann Vektoren enthalten, die Vektoren usw. enthalten.

```
// Beispiele
```

```
[1,2,3]
[a, 5, b]
[]
[5.643]
["a", "b", "string"]
[[1, r], [x, y, z, 4,5]]
[3, 5, [6,7], [[8,9], [10, [11,12], 13], c, "string"]
[4/3, 6 * 1,5, cos (60)]
```

Verwendung in OpenSCAD:

```
cube ([Breite, Tiefe, Höhe]); // Zur Verdeutlichung werden optionale Leerzeichen angezeigt
translate ([x, y, z])
polygon ([[x0, y0], [x1, y1], [x2, y2]]);
```

Erstellung

Vektoren werden erstellt, indem die Liste der Elemente geschrieben, durch Kommas getrennt und in eckigen Klammern eingeschlossen wird. Variablen werden durch ihre Werte ersetzt.

```
Würfel ([10,15,20]);
a1 = [1,2,3];
a2 = [4,5];
a3 = [6,7,8,9];
b = [a1, a2, a3]; // [[1,2,3], [4,5], [6,7,8,9]] beachten Sie die erhöhte
Verschachtelungstiefe
```

Elemente innerhalb von Vektoren

Elemente innerhalb von Vektoren sind von 0 bis n-1 nummeriert, wobei n die von `len ()` zurückgegebene Länge ist. Adresselemente innerhalb von Vektoren mit der folgenden Notation:

```
e [5]              // Element Nr. 5 (sechstes)      1. Verschachtelungsebene
e [5] [2]          // Element 2 von Element 5       2. Verschachtelungsebene
e [5] [2] [0]      // Element 0 von 2 von 5         3. Verschachtelungsebene
e [5] [2] [0] [1]  // Element 1 von 0 von 2 von 5   4. Verschachtelungsebene
```

Beispielelemente mit Längen von len ()

```
e = [[1], [], [3,4,5], "string", "x", [[10,11], [12,13,14], [[15,16], [17 ]]]]; // Länge 6



Adresslängenelement
e [0] 1 [1]
e [1] 0 []
e [5] 3 [[10,11], [12,13,14], [[15,16], [17]]]
e [5] [1] 3 [12, 13, 14]
e [5] [2] 2 [[15,16], [17]]
e [5] [2] [0] 2 [15, 16]
e [5] [2] [0] [1] undef 16

e [3] 6 "string"
e [3] [2] 1 r

s = [2,0,5]; a = 2;
s [a] undef 5
e [s [a]] 3 [[10,11], [12,13,14], [[15,16], [17]]]
```

alternative Punktnotation

Auf die ersten drei Elemente eines Vektors kann mit einer alternativen Punktnotation zugegriffen werden:

```
e.x // äquivalent zu e [0]
e.y // äquivalent zu e [1]
e.z // äquivalent zu e [2]
```

5.1. Vektoroperatoren

5.1.1 concat

Hinweis: Benötigt Version 2015.03

`concat ()` kombiniert die Elemente von 2 oder mehr Vektoren zu einem einzigen Vektor. Es wird keine Änderung der Verschachtelungsebene vorgenommen.

```
Vektor1 = [1,2,3]; Vektor2 = [4]; Vektor3 = [5,6];
new_vector = concat (Vektor1, Vektor2, Vektor3); // [1,2,3,4,5,6]

string_vector = concat ("abc", "def"); // ["abc", "def"]
one_string = str (string_vector [0], string_vector [1]); // "abcdef"
```

5.1.2. len

`len ()` ist eine Funktion, die die Länge von Vektoren oder Strings zurückgibt. Die Indizes der Elemente reichen von `[0]` bis `[Länge-1]`.

> **Vektor**
>
> Gibt die Anzahl der Elemente auf dieser Ebene zurück.
>
> Einzelwerte, die keine Vektoren sind, geben `undef` zurück.
>
> **Zeichenfolge**
>
> Gibt die Anzahl der Zeichen in der Zeichenfolge zurück.

```
a = [1,2,3]; Echo (len (a)); //  3
```

Sieh dir Beispielelemente mit Längen an

6. Matrix

Eine Matrix ist ein Vektor von Vektoren.

```
Beispiel, das eine 2D-Rotationsmatrix definiert
mr = [
     [cos (Winkel), -sin (Winkel)],
     [sin (Winkel), cos (Winkel)]
    ];
```

Input bekommen

Jetzt haben wir Variablen, es wäre schön, Eingaben in diese zu bekommen, anstatt die Werte aus dem Code festzulegen. Es gibt einige Funktionen zum Lesen von Daten aus DXF-Dateien, oder Sie können eine Variable mit dem Schalter -D in der Befehlszeile festlegen.

Einen Punkt aus einer Zeichnung ziehen

Das Abrufen eines Punkts ist nützlich, um einen Ursprungspunkt in einer 2D-Ansicht in einer technischen Zeichnung zu lesen. Die Funktion `dxf_cross` liest den Schnittpunkt zweier Linien auf einer von Ihnen angegebenen Ebene und gibt den Schnittpunkt zurück. Dies bedeutet, dass der Punkt mit zwei Zeilen in der DXF-Datei angegeben werden muss und nicht mit dem Objekt.

```
OriginPoint = dxf_cross(file="drawing.dxf", layer="SCAD.Origin", origin=[0, 0], scale=1);
```

Abrufen eines Dimensionswerts

Sie können Maße aus einer technischen Zeichnung lesen. Dies kann nützlich sein, um einen Drehwinkel, eine Extrusionshöhe oder einen Abstand zwischen Teilen abzulesen. Erstellen Sie in der Zeichnung eine Bemaßung, die nicht den Bemaßungswert, sondern einen Bezeichner anzeigt. Um den Wert zu lesen, geben Sie diesen Bezeichner aus Ihrem Programm an:

```
TotalWidth = dxf_dim(file="drawing.dxf", name="TotalWidth", layer="SCAD.Origin", origin=[0, 0], scale=1);
```

Ein schönes Beispiel für beide Funktionen finden Sie in Beispiel 009 und im Bild auf der Homepage von OpenSCAD.

original in english

3D-objekte

bisher nur in english

2D-Objekte

bisher nur in english

Transformationen

bisher nur in english

Boolesche Kombinationen

bisher nur in english

Andere Funktionen und Operationen

bisher nur in english

Benutzerdefinierte Funktionen und Module

bisher nur in english

Debugging Hilfsmittel

bisher nur in english

Externe Bibliotheken und Code-Dateien

bisher nur in english