Variable und deren Berechnungen mit ganzen Zahlen

Variablen werden in Verbindung mit for-Schleifen verwendet, um mit ihnen bestimmte Formen entstehen zu lassen.
Als Variable können generierte Werte direkt verwendet werden oder durch ausgeführte Berechnungen, um anspruchsvollere Schleifen zu erzielen.

Variable benennen

Mit OpenSCAD können die Variablen so lang benannt werden wie man möchte, solange keine Leerzeichen eingefügt oder andere Symbole als Buchstaben, Unterstriche oder Zahlen Verwendung finden.
Es bietet sich an, Namen zu wählen, die einem helfen den Zweck oder Sinn der Variablen zu verstehen um es später bei der Fehlersuche leichter nachvollziehen zu können.

Mathematische Berechnungen mit Variablen

Als Erstes werden die drei Variablen die Werte 12, 7 und 256 über = zugewiesen. Die Variablen tragen die Namen: „Wert_1“ , „Wert_2“ und Wert_3“:

```
Wert_1 = 12;
Wert_2 = 7;
Wert_3 = 256;
```

Um mathematische Berechnungen auszuführen, wie Summe, Subtraktion, Produkt, Quotient oder Rest dieser Werte, werden Standardzeichen verwendet. Die normalen Reihenfolgen der Berechnungen erfolgt wie die, die noch aus dem Matheunterricht bekannt sein dürften (wiePunkt vor Strich). Werden die Ergebnisse von arithmetischen Berechnungen mit einer Variablen versehen, hilft dies die Berechnungen von den Ausgabeanweisungen zu trennen:

```
Summe = Wert_1 + Wert_2;
Differenz = Wert_1 - Wert_2;
Produkt = Wert_1 * Wert_2;
Quotient = Wert_1 / Wert_2;
Rest = Wert_1 % Wert_2;
Hochzahl1 = Wert_1 ^ 2;
Hochzahl2 = Wert_2 ^ 2;
Wurzel3 = sqrt(Wert_3);
```

009-02

Mit der Echo-Funktion wird das Ergebnis jeder mathematischen Berechnung angezeigt. Jede Echo-Funktion verwendet einen markanten Bezeichner, wie („Addition :“ gefolgt von einem Komma , und der entsprechenden Variable Summe) ; für die Ausgabe in der Konsole:

```
echo("Addition: ", Summe);
echo("Subtraktion :", Differenz);
echo("Multiplikation: ", Produkt);
echo("Division: ", Quotient);
echo("Modulo: ", Rest);
echo(Wert_1, " hoch 2 :", Hochzahl1);
echo(Wert_2, " hoch 2 :", Hochzahl2);
echo(" Wurzel aus :", Wert_3, " ist ", Wurzel3);
```

In dieser Zeile `echo(Wert_1, " hoch 2 :", Hochzahl1);` wurde zwischen zwei Werten ein Text eingefügt: echo(ein_wert, „Text“, noch_ein_wert);
Die Werte werden einfach in der runden Klammer von echo eingesetzt und durch Komma getrennt. Wird ein Text dazwischen geschoben, muss für Text am Anfang und am Ende ein doppeltes Apostroph stehen, außerdem auch durch Kommata abgetrennt sein.

+ Summe: addiert beide Zahlen
- Differenz: subtrahiert die Zahlen
* Produkt: multipliziert beide Zahlen miteinander
/ Division: Teilt die eine durch die andere Zahl
% Modulo: dies ist der Rest, der nach dem Teilen übrig bleibt
^ Exponent: das Ergebnis des Potenzierens
sqrt(x): Wurzel aus x

Hier nun der komplette Script:

```
// Variablen-Zuweisungen:
Wert_1 = 12;
Wert_2 = 7;
Wert_3 = 256;

// Alle Berechnungen in
// neue Variablen zuweisen:
Summe = Wert_1 + Wert_2;
Differenz = Wert_1 - Wert_2;
Produkt = Wert_1 * Wert_2;
Quotient = Wert_1 / Wert_2;
Rest = Wert_1 % Wert_2;
Hochzahl1 = Wert_1 ^ 2;
Hochzahl2 = Wert_2 ^ 2;
Wurzel3 = sqrt(Wert_3);

// Ausgabe auf der Konsole:
echo("Addition: ", Summe);
echo("Subtraktion :", Differenz);
echo("Multiplikation: ", Produkt);
echo("Division: ", Quotient);
echo("Modulo: ", Rest);
echo(Wert_1, " hoch 2 :", Hochzahl1);
echo(Wert_2, " hoch 2 :", Hochzahl2);
echo(" Wurzel aus :", Wert_3, " ist ", Wurzel3);
```

Und die Ausgabe der Konsole:

```
ECHO: "Addition: ", 19
ECHO: "Subtraktion :", 5
ECHO: "Multiplikation: ", 84
ECHO: "Division: ", 1.71429
ECHO: "Modulo: ", 5
ECHO: 12, " hoch 2 :", 144
ECHO: 7, " hoch 2 :", 49
ECHO: " Wurzel aus :", 256, " ist ", 16
```

Wahrscheinlich muss etwas gescrollt werden, um alles zu sehen – je nach Einstellungen der Fenstergrößen.

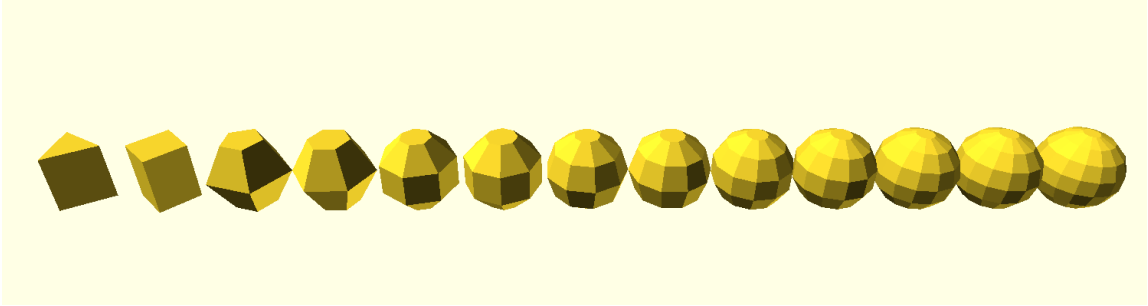
009-04

Berechnungen und Variable in Schleifen

Es können innerhalb einer for-Schleife Berechnungen stattfinden, die zur weiteren Verwendung dienen können.

Der folgende Script erstellt 13 Kugeln, die alle durch die gleiche for-Schleife generiert werden:

```
for (anzahl=[3:1:15]) {
   $fn = anzahl;
   x_position = anzahl*10;
   translate([x_position, 0, 0])sphere(r=5);
   echo("Segmente :", anzahl, " X-Achsenwert :", x_position);
}
```



Die Schleife lässt von 3 bis 15 in Schrittweite von 1 je eine Kugel mit dem Durchmesser von 5 entstehen. Der Wert auf der X-Achse wird mit 10 multipliziert. Dieser Wert für translate sieht dann 30 als Start, 40, 50, 60 … und sofort bis 150 vor. Außerdem wird von 3 bis 15 die Anzahl der Segmente der Darstellung erhöht. So wird von einem zum Nächsten die Gugel immer glatter.

Als Ausgabe für die Konsole erscheint „Segmente:“ mit dessen Wertangabe, ferner noch „X-Achsenwert:“ mit dessen Wert.

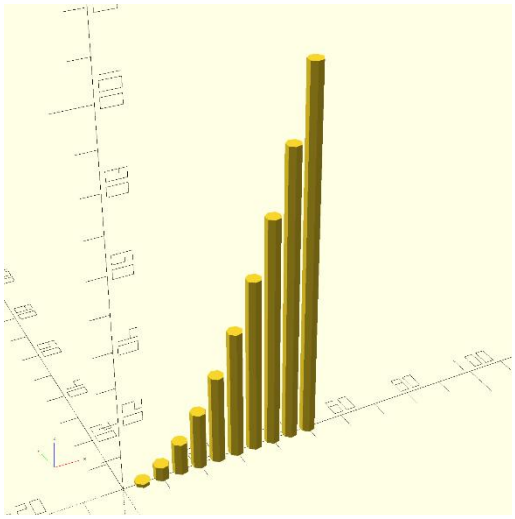
```
ECHO: "Segmente :", 3, " X-Achsenwert :", 30
ECHO: "Segmente :", 4, " X-Achsenwert :", 40
ECHO: "Segmente :", 5, " X-Achsenwert :", 50
ECHO: "Segmente :", 6, " X-Achsenwert :", 60
ECHO: "Segmente :", 7, " X-Achsenwert :", 70
ECHO: "Segmente :", 8, " X-Achsenwert :", 80
ECHO: "Segmente :", 9, " X-Achsenwert :", 90
ECHO: "Segmente :", 10, " X-Achsenwert :", 100
ECHO: "Segmente :", 11, " X-Achsenwert :", 110
ECHO: "Segmente :", 12, " X-Achsenwert :", 120
ECHO: "Segmente :", 13, " X-Achsenwert :", 130
ECHO: "Segmente :", 14, " X-Achsenwert :", 140
ECHO: "Segmente :", 15, " X-Achsenwert :", 150
```

Berechnungen einzigartiger Muster

Es können kombinierte Berechnungen getätigt werden, um interessante Muster zu erstellen.
Der folgende Script generiert eine Reihe von Zylinder mit zunehmender Höhe durch Verwendung eines quadratischen Musters zur Erhöhung:

009-Spezielles Muster.scad

```
for (x=[1:1:10]) {
Hoehe = x*x;                    //  Zuweisung
x_position = 5*x;               //  Berechnung
translate([x_position, 0, 0]) cylinder(h=Hoehe, r=2);   // Ausgabe
}
```



Die for-Schleife enthält die Variable „x“, die von 1 bis 10, in Schrittweite von 1 sich bewegt.
Danach wird die Variable Hoehe als Potenz von x erstellt.
Die Variable „x_position“ wird festgelegt mit der Multiplikation x mal 5.
Jetzt erfolgt die Ausgabe: der Zylinder wird erstellt mit wachsender Höhe und fortschreitendem Wert auf der X-Achse.

Dies wird als quadratisches Wachstum bezeichnet.

009-06

Verschachtelte Schleifen

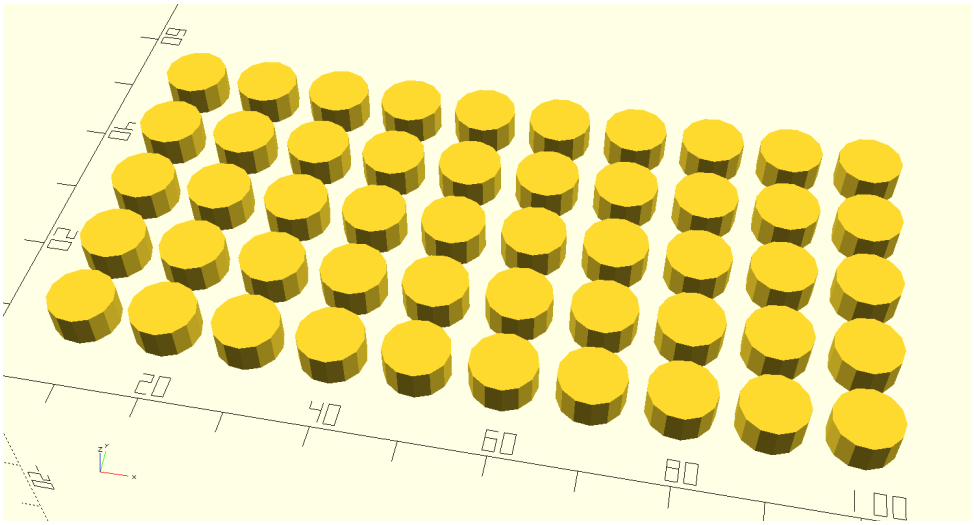
OpenSCAD kann eine Schleife wiederholen, sogar eine for-Schleife die eine weitere for-Schleife darin enthält.

Während eine for-Schleife zur Erstellung einer Zeile verwendet wird, kann eine for-Schleife innerhalb einer anderen for-Schleife abgearbeitet werden.

So können mit wenigen Zeile Formen erzeugt werden, um ein ganzes Gitternetz aus Formen zu erstellen. Das wird als Verschachteln der Schleifen bezeichnet.

Der folgende Script verwendet verschachtelte for-Schleifen:

```
for (y_pos = [10:10:50]) {
    for (x_pos = [10:10:100]) {
        translate([x_pos, y_pos, 0]) cylinder(h=4, r1=4, r2=4);
        echo("x:", x_pos, "y:", y_pos);
    } // x_pos schleife
} // y_pos schleife
```



Der vorhergehende Script verwendet eine Schleife, um in einer Zeile 10 Zylindern zu zeichnen.

Diese for-Schleife wird von der ersten for-Schleife wiederholt, also der Reihe von Zylindern wiederholt.

Zwei Variablen – x_pos und y_pos – arbeiten zusammen um sowohl die X-Position als auch die Y-Position des wiederholten Zylinders zu ändern.

Die innere Schleife wird 10 mal wiederholt, während die äußere Schleife 5 mal wiederholt wird. Dies erzeugt dadurch insgesamt 50 Zylinder.

Mit der Echo-Funktion wird der Überblick der sich ständig ändernden Werte behalten.

Es werden auch Kommentare verwendet werden, damit man weiß, welche Klammer zu welcher Schleife gehört.

Das Kommentieren von Klammern ist nicht erforderlich, kann aber sehr hilfreich sein, wenn viele geschweifte Klammern nebeneinanderstehen.

Mit ein paar Zeilen Code wurden nun eben mal schnell 50 Zylinder generiert.

Das ist sicherlich besser, als eine lange Liste mit 50 Befehlskombinationen zu schreiben, die jeden einzelnen Zylinder generieren.

Dies wäre die perfekte Technik zum Zeichnen von vielen Fenster in einem Wolkenkratzer…..

```
ECHO: "x:", 10, "y:", 10
ECHO: "x:", 20, "y:", 10
ECHO: "x:", 30, "y:", 10
ECHO: "x:", 40, "y:", 10
ECHO: "x:", 50, "y:", 10
ECHO: "x:", 60, "y:", 10
ECHO: "x:", 70, "y:", 10
ECHO: "x:", 80, "y:", 10
ECHO: "x:", 90, "y:", 10
ECHO: "x:", 100, "y:", 10
ECHO: "x:", 10, "y:", 20
ECHO: "x:", 20, "y:", 20
ECHO: "x:", 30, "y:", 20
ECHO: "x:", 40, "y:", 20
ECHO: "x:", 50, "y:", 20
ECHO: "x:", 60, "y:", 20
ECHO: "x:", 70, "y:", 20
ECHO: "x:", 80, "y:", 20
ECHO: "x:", 90, "y:", 20
ECHO: "x:", 100, "y:", 20
ECHO: "x:", 10, "y:", 30
ECHO: "x:", 20, "y:", 30
ECHO: "x:", 30, "y:", 30
ECHO: "x:", 40, "y:", 30
ECHO: "x:", 50, "y:", 30
ECHO: "x:", 60, "y:", 30
ECHO: "x:", 70, "y:", 30
ECHO: "x:", 80, "y:", 30
ECHO: "x:", 90, "y:", 30
ECHO: "x:", 100, "y:", 30
ECHO: "x:", 10, "y:", 40
ECHO: "x:", 20, "y:", 40
ECHO: "x:", 30, "y:", 40
ECHO: "x:", 40, "y:", 40
ECHO: "x:", 50, "y:", 40
ECHO: "x:", 60, "y:", 40
ECHO: "x:", 70, "y:", 40
ECHO: "x:", 80, "y:", 40
ECHO: "x:", 90, "y:", 40
ECHO: "x:", 100, "y:", 40
ECHO: "x:", 10, "y:", 50
ECHO: "x:", 20, "y:", 50
ECHO: "x:", 30, "y:", 50
ECHO: "x:", 40, "y:", 50
ECHO: "x:", 50, "y:", 50
ECHO: "x:", 60, "y:", 50
ECHO: "x:", 70, "y:", 50
ECHO: "x:", 80, "y:", 50
ECHO: "x:", 90, "y:", 50
ECHO: "x:", 100, "y:", 50
```

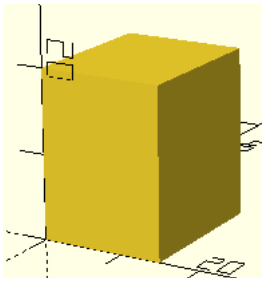
009-08

Fenster in einem Wolkenkratzer mit verschachtelten Schleifen erzeugen

Aus einem Kubus wird ein Büro-Block, aus dem ein weiterer schmaler Kubus als Fenster ausgeschnitten wird:

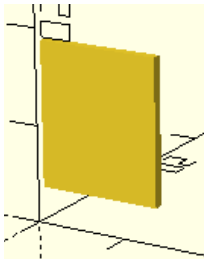
der Büroblock:

```
translate([0,0,0])
cube([15,18,20]); // + Büro
```



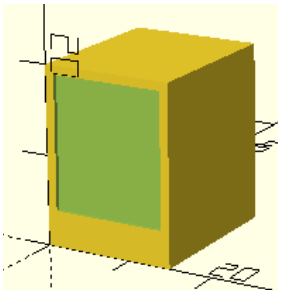
das Fenster:

```
translate([1, -.5, +4])
cube([13,1,15]); //  Fenster
```



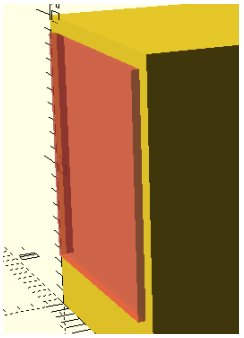
Nun wird der Büroblock erstellt und davon das Fenster abgezogen:

```
difference () {
  translate([0,0,0]) cube([15,18,20]); // + Büro
  translate([1, -.5, 4]) cube([13,1,15]); // - Fenster
}
```



Der Startpunkt des Büroblocks liegt bei ([0, 0, 0]), also im Zentrum.

Das Fenster beginnt weiter innen (x=1) und weiter oben (z=4). Um es sauber zu gestalten wird das „Glas“ mit der Stärke (1) nur um (y= -.5) nach außen gesetzt, um die andere Hälfte auszuschneiden.

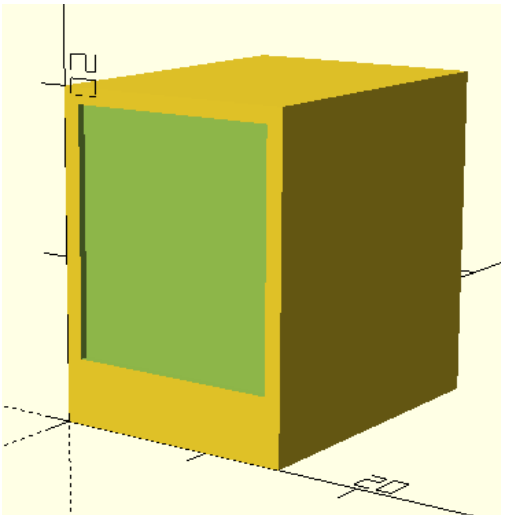


Modul-Aktivierung

Da der Büroblock mehrfach aufgerufen werden muss, ist es ratsam ihn in ein Modul zu setzen:

Über die Parameter-Eingaben von x und z können die Vorgaben überschrieben werden.

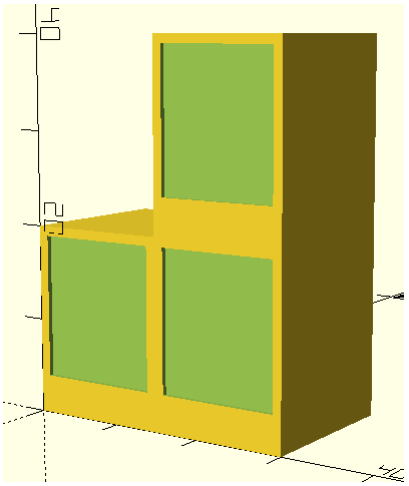
Erstellung eines Büros:



```
//Modulbereich
module buero (x=0, z=0) {
  difference(){
translate([x,0,z])cube([15,18,20]); // + Büro
translate([x+1,-.5,z+4])cube([13,1,15]); // -
Fenster
   } // difference
} // module

//Programmbereich
buero(0,0);
```

Wichtig sin hier die Vorgaben von x und z in den Berechnungen mit einzubauen. So kann nun mittels der Vorgaben x und z jederzeit und überall ein Büroblock entstehen. Es sollte jedoch die Breite von 15 und die Höhe von 20 berücksichtigt werden, damit ein in sich geschlossenes Gebäude entstehen kann.



```
//Modulbereich
module buero (x=0, z=0) {
  difference(){
translate([x,0,z])cube([15,18,20]); // + Büro
translate([x+1,-.5,z+4])cube([13,1,15]); // - Fenster
   } // difference
} // module

//Programmbereich
buero(0,0);
buero(15,0);
buero(15,20);
```

Jetzt lassen sich nun die Büroblöcke einfach zu einem größeren Gebäudekomplex zusammensetzen.

009-10

Die Etagen-Konstruktion:

Mittels der Variablen „etage“ kann jederzeit eine Auswahl getroffen werden.

Nach Einsatz einer Schleife mit „for (a=[0:20:(etage*20)-1])“ werden automatisch die Büroblöcke aufeinander gesetzt. Hier bedeutet jedes „a“ ein Block. Dieser entsteht durch „[0:20:(etage*20)-1]“

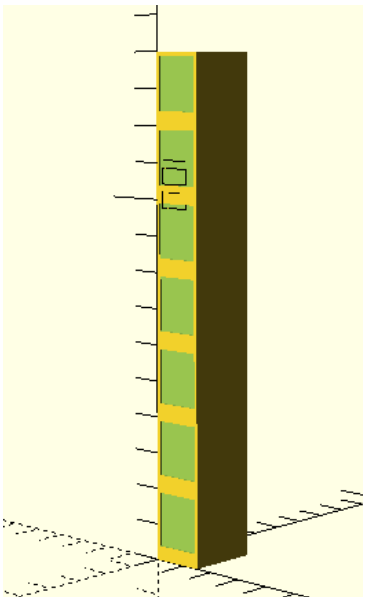
Die Schleife beginnt bei „0“ (Null) Blöcken – nicht bei 1!
Die Schrittweite von „20“ wird durch die Höhe des Blocks bestimmt.
Der Endwert der Schleife setzt sich zusammen aus der Multiplikation der Anzahl der „etagen“ und dem Höhenmaß – abzüglich 1.
Da das Prog bei 0 und nicht bei 1 anfängt zu zählen, muss daher eine Etage abgezogen werden: -1.

Und schon steht ein schmales Hochhaus da:

```
etage=7;

//Modulbereich
module buero (x=0, z=0) {
  difference(){
translate([x,0,z])cube([15,18,20]); // + Büro
translate([x+1,-.5,z+4])cube([13,1,15]); // - Fenster
   } // difference
} // module

for (a=[0:20:(etage*20)-1])   {
//   ([ 0 : Höhe Büro : Etagen - 1])
buero(0,a);
}
```



Das Ganze wird nun mit der Anzahl der Reihen ergänzt.

```
reihe = 15;
…..
for (b=[0:15:(reihe*15)-1])   {

   for (a=[0:20:(etage*20)-1])   {
   //   ([ 0 : Breite Büro : Etagen - 1])
   buero(b,a);
   } // for a

} // for b
```

Die Variable „reihe“ gib nun die Fensteranzahl in der Breite an.
Bei der Breite wird ebenso die Anzahl der „reihen“ mit 15 multipliziert, um das Maß der Blöcke zu erhalten.

009-12

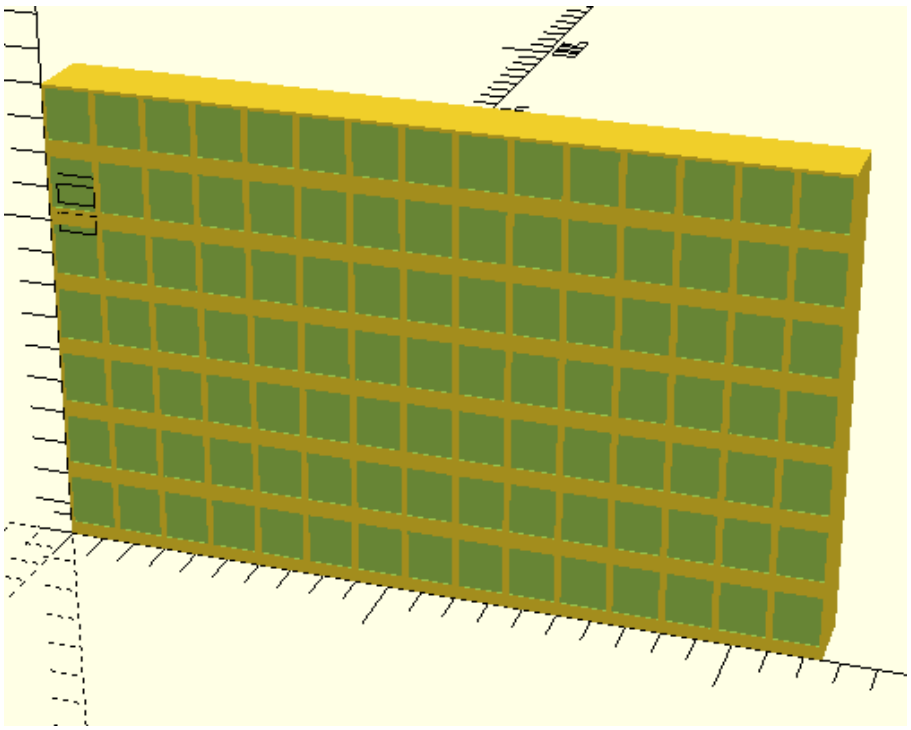
// erstes hochhaus.scad

```
// Erstellung eines Hochhauses

// Variablenbereich
reihe = 15;
etage=7;

//Modulbereich
module buero (x=0, z=0) {
  difference(){
translate([x,0,z])cube([15,18,20]); // + Büro
translate([x+1,-.5,z+4])cube([13,1,15]); // - Fenster
   } // difference
} // module

//Programmbereich
for (b=[0:15:(reihe*15)-1])   {
   for (a=[0:20:(etage*20)-1])   {
   //   ([ 0 : Breite Büro : Etagen – 1])
   buero(b,a);
   } // for a
} // for b
```



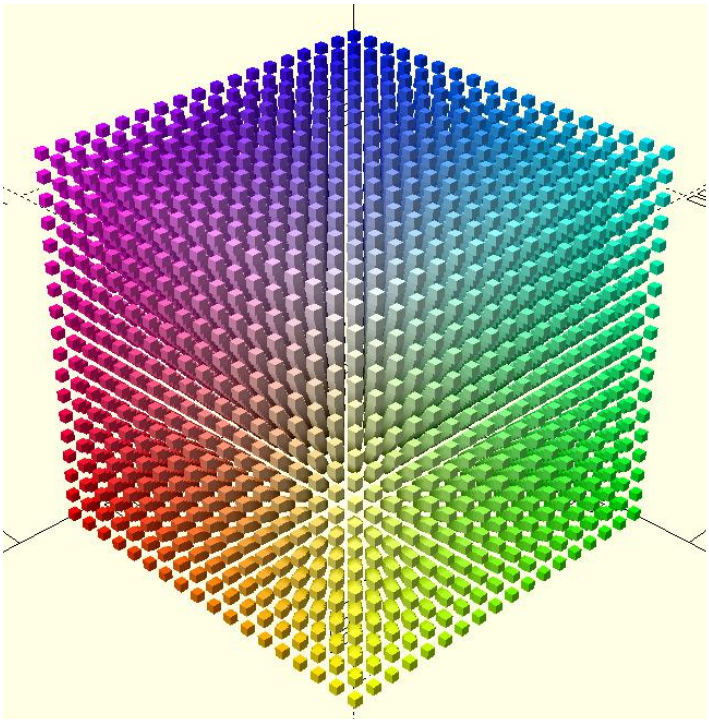
Dreifache Verschachtelung zum Erstellen eines 3D-Gitters

Es kann ein 3D-Gitter aus Formen gezeichnet werden, indem eine weitere Verschachtelungsebene hinzugefügt wird. Dies bedeutet, eine Schleife in einer Schleife, in einer Schleife …. Das Rendern dauert eine Weile, da eine große Anzahl von Formen generiert werden muss:

```
// Großer Farbwürfel aus 18x18x18 kleine Würfel

for (rot = [0:15:255]) {
  for (gruen = [0:15:255]) {
    for (blau = [0:15:255]) {
      translate([rot, gruen, blau])
      color([rot/255, gruen/255, blau/255]) cube(6);
    } // blaue Schleife
  } // gruene Schleife
} // rote Schleife
```

Ein dreifach verschachtelter Würfel aus vielen kleinen Würfeln um das RGB-Farbspektrum darzustellen.



Diese dreifache Verschachtelung verwendet im Wesentlichen die dritte Schleife, um ein Gitter mit Formen zu wiederholen.
Das Script verwendet drei verschachtelte Schleifen, um einen farbigen Würfel aus dem RGB-Farbraum darzustellen.

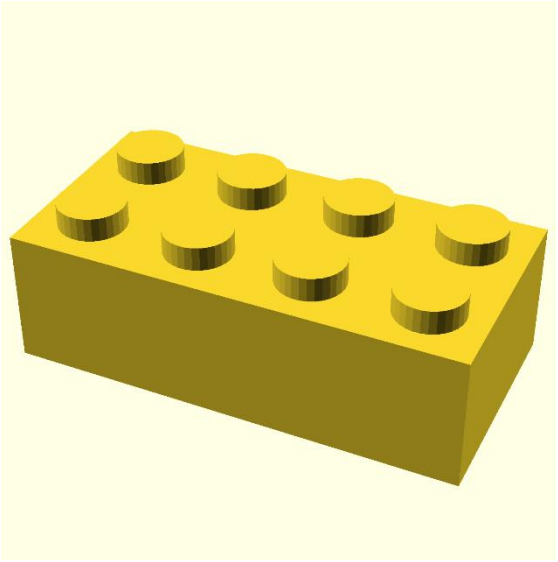
Der 3D-Vektor „color([rot/255, gruen/255, blau/255])“ übernimmt die Farbtransformation, der den Prozentsatz der Farben Rot, Grün und Blau angibt. Da RGB 255 als Maximalwert verwendet, ergibt die Division durch 255 eine Dezimalstelle zwischen 0 und 1.
Die Farbtransformation ist nur im Vorschaumodus wirksam und wird im Rendermodus nicht angezeigt. Geben Sie der Vorschau viel Zeit …….

009-14

Einen LEGO Stein erstellen

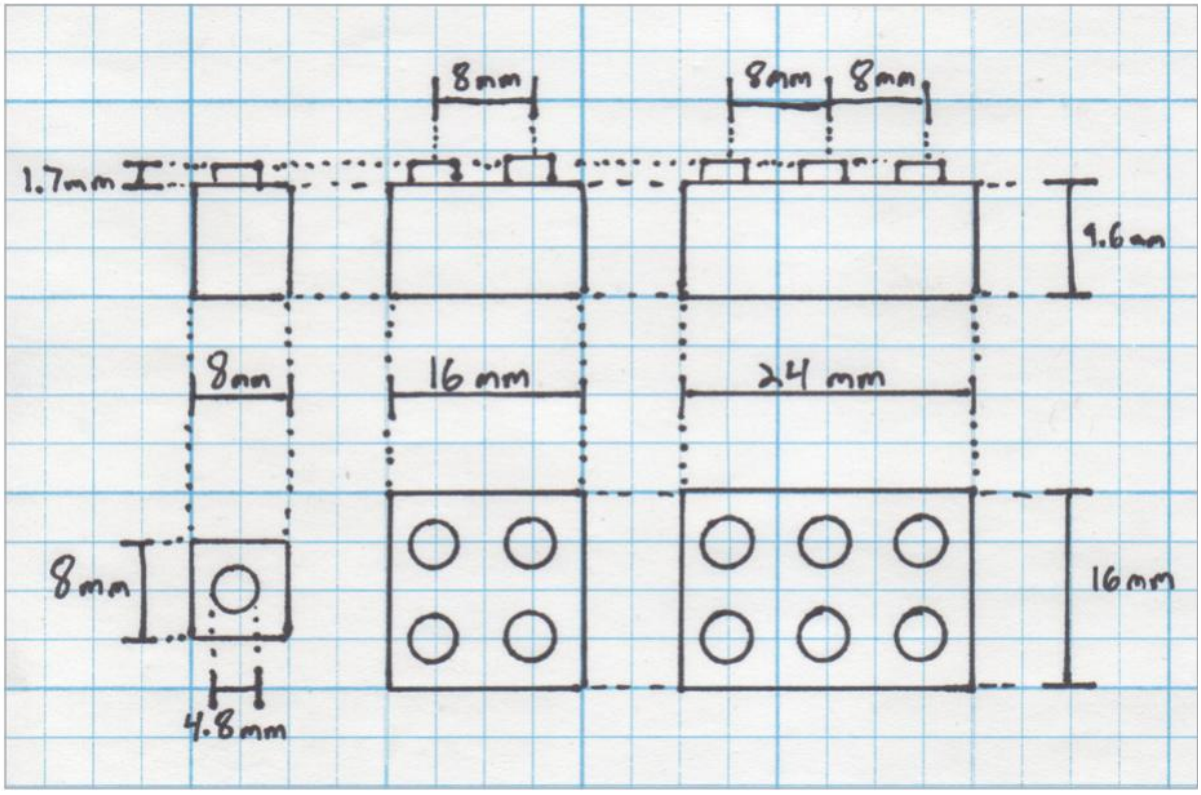
Es wird nun ein komplexes Modellierungsprojekt erstellt, das Parameter, Module und Schleifen in einer einzigen Form verwendet.

Die Steinform besteht aus zwei Noppen in einer Richtung und beliebiger Anzahl von Noppen in die andere Richtung. Die Noppen sind die kleinen Zylinder auf der Oberseite vom Stein, die in die anderen Steine passen um diese zusammenzuhalten.



Das Bild zeigt einen LEGO-Stein mit zwei Zeilen und vier Noppen pro Reihe:
ein Stein mit 2x4 Noppen

Für eine bessere Vorstellung des Projektes ist es immer ratsam, erst einmal eine kleine Handskizze anzufertigen. In diese können dann die Maßangaben eingetragen werden.



Die Abmessungen der BAUsteine sind leicht online verfügbar.
Die verwendeten Abmessungen sind von Wikipedia entnommen.

Jeder Stein ist 8 breit und hat eine Höhe von 9,6, zusätzlich den Noppen von 1,7 Höhe.
Der Noppen ist mittig angebracht und hat einen Durchmesser von 4,8.
Jede weitere Noppe lässt den Stein um 8 wachsen.

Die Länge eines BAUsteins hängt auch von der Anzahl der Noppen ab. Es werden nur Steine mit zwei Noppenreihen erstellt, was demnach nur eine feste Steinbreite von 16 mm vorweist.

Das Erkunden einer Vielzahl von handgezeichneten BAUsteinFormen macht es einfacher Identifizieren Sie die OpenSCAD-Anweisungen, die zum Definieren eines BAUsteins erforderlich sind

ein BAUstein Modul:

```
// Aufruf mit Zahlenübergabe:
BAUstein(6);

//Modulbereich
module BAUstein(noppen_pro_reihe=4) {
$fn=30;
laenge = noppen_pro_reihe * 8;
cube([laenge, 16, 9.6]);
    for (x_position=[4 : 8 : laenge-4]) {
    translate([x_position, 4, 1.7]) cylinder(h=9.6, d=4.8);
    translate([x_position, 12, 1.7]) cylinder(h=9.6, d=4.8);
    }
}  // module
```

Ein BAUstein wird nun mit einem Modul erstellt.

Zuerst wird ein Modul namens BAUstein mit einem Parameter „noppen_pro_reihe“ erstellt. Dieser Parameter repräsentiert die Anzahl der Noppen entlang der Oberseite des BAUsteins, der die Gesamtlänge des Steines bestimmt.

Dieser Parameter ist sehr nützlich, da dadurch das Modul mehrfach wieder verwenden werden kann. Somit kann eine hohe Anzahl von Steinen mit unterschiedlichen Größen erzielt werden.

Der Ausgangs-Wert ist ein Stein mit 2x2, also mit 4 Noppen.

009-16

Eine Variable namens „laenge“ wird erstellt, um die Gesamtlänge des Steines zu bestimmen der auf „noppen_pro_reihe“ basiert. Jeder weitere Bolzen erhöht die Breite des Steins um 8:

laenge = noppen_pro_reihe * 8;

Andere Abmessungen des BAUsteins bleiben fest, ohne jeglichen Bezug zur Anzahl der Noppen pro Reihe:

cube([width, 16, 9.6]);

Eine for-Schleife wird verwendet, um jeden wiederholten Bolzen an seiner richtigen Position zu zeichnen:

```
for (x_position=[4 : 8 : laenge-4]) {
translate([x_position, 4, 1.7]) cylinder(h=9.6, d=4.8);
translate([x_position, 12, 1.7]) cylinder(h=9.6, d=4.8);
}
```

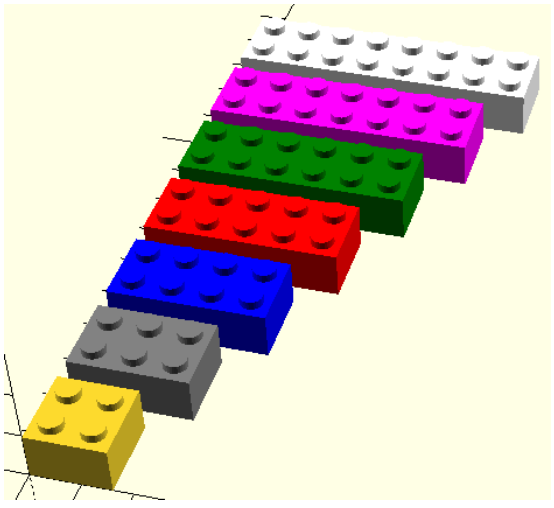
Innerhalb der for-Schleife verfolgt die Variable „x_position“ die X-Position jedes einzelnen Noppens. Der erste Noppen ist bei x = 4 zentriert, und jeder weitere Noppen ist 8 vom vorherigen Noppen entfernt positioniert.

Zwei Reihen von Noppen werden mit den exakt gleichen Werten auf der X-Achse gezeichnet.

Da der Stein auf nur zwei Noppen auf der Y-Achse beschränket wird, ist es einfacher, die Zeilen explizit bei y = 4 mm und y = 12 mm zu positionieren.

Das BAUstein-Modul ist nun vollständig, was bedeutet, dass Sie es verwenden können.

BAUsteine in verschiedenen Größen und Farben erstellt



Hier den Skript für eine Handvoll Steine:

```
// Aufruf mit Zahlenübergabe:
BAUstein(2);
color("grey",1)translate([0,20,0])BAUstein(3);
color("blue",1)translate([0,40,0])BAUstein(4);
color("red",1)translate([0,60,0])BAUstein(5);
color("green",1)translate([0,80,0])BAUstein(6);
color("magenta",1)translate([0,100,0])BAUstein(7);
color("white",1)translate([0,120,0])BAUstein(8);

//Modulbereich

module BAUstein(noppen_pro_reihe=4) {
$fn=30;
laenge = noppen_pro_reihe * 8;
cube([laenge, 16, 9.6]);
    for (x_position=[4 : 8 : laenge-4]) {
    translate([x_position, 4, 1.7]) cylinder(h=9.6, d=4.8);
    translate([x_position, 12, 1.7]) cylinder(h=9.6, d=4.8);
    }
}  // module
```

Dieses Modul ist leider nur ein vereinfachtes Design eines LEGO-Steines.
Es kann nicht wie ein Echter funktionieren, da das Innenleben zum Einrasten der Steine untereinander noch fehlt.

Das überlasse ich gern den Anderen als Herausforderung……