C.M. VAN RIET

OPENSCAD WORKSHOP BASIS

FABLAB AMERSFOORT

C.M. van Riet

UITGEGEVEN DOOR LIGHTBULB MOMENTS IN SAMENWERKING MET FABLAB AMERSFOORT

TUFTE-LATEX.GITHUB.IO/TUFTE-LATEX/

Eerste uitgave, maart 2019

Inhoudsopgave

Introductie

Met deze methode leer je in vier lessen hoe je uiteenlopende modellen kunt maken in OpenSCAD. Wat je daarvoor nodig hebt is dit boekje en OpenSCAD geïnstalleerd op je computer. De modellen worden uitgelegd aan de hand van voorbeelden in code. Je kunt deze code zelf in OpenSCAD zetten om te kijken wat OpenSCAD doet met deze code.

In elk hoofdstuk wordt de theorie van de nieuwe concepten aangevuld met praktische opdrachten. Elk hoofdstuk wordt afgesloten met een aantal 3D-voorbeelden en de bijbehorende code. Als je deze voorbeelden na kunt maken zonder naar de code te kijken, weet je dat je alles uit het hoofdstuk begrepen hebt.

Tweedimensionaal ontwerpen

Onderwerpen

Taal Welke taal spreekt OpenSCAD eigenlijk? Waarom kan ik mijn model niet aanklikken? Moet ik echt alles intypen? Is er een cheat sheet? `http://www.openscad.org/cheatsheet/`

Vormen Hoe maak je 2D-tekeningen in OpenSCAD? Welke vormen zijn er? Hoe pas je die aan? Wat is resolution?

Nieuwe mogelijkheden `translate`, `rotate`, `difference`, `union`, `mirror`, `color`.

Project Ontwerp een glas-in-lood raam met behulp van de figuren die we in dit hoofdstuk behandeld hebben en de `translate`, `rotate`, `difference` en `mirror` functie.

De taal van OpenSCAD

Als je voor het eerst OpenSCAD opstart, valt het op dat je met drie vensters werkt in plaats van één. In het linkervenster kun je typen in de *editor*, rechtsonder zie je een *console* die weergeeft of alles goed gaat (of niet), en rechtsboven zie je een *viewport*. OpenSCAD maakt gebruik van een programmeertaal die wat weg heeft van C. Net zoals in andere programmeertalen kun je functies schrijven, variabelen opgeven en deze weer gebruiken in je functies. Maar je kunt niet op dezelfde manier programmeren in OpenSCAD zoals je dat gewend bent te doen in Python, Javascript of C++. In deze workshop leer je niet hoe de taal van OpenSCAD verschilt van andere programmeertalen. Daarvoor kun je het beste zelf de OpenSCAD-handleiding lezen: `https://en.wikibooks.org/wiki/OpenSCAD_User_Manual`. Wat je wel leert is hoe de taal van OpenSCAD werkt en hoe je zelfstandig aan de slag kunt gaan met OpenSCAD.

Aangezien OpenSCAD werkt met een programmeertaal in het linkervenster, zie je dat je de objecten die je aan het ontwerpen bent niet aan kunt klikken. In de *viewport* kun je in- en uitzoomen en je object roteren, maar je kunt er geen wijzigingen aanbrengen in je object. Alle veranderingen die je wilt aanbrengen, zul je moeten typen. “Wat een gedoe!” kun je nu denken, maar deze aanpak heeft veel voordelen. Je kunt zo erg precies werken en onderdelen exact kopiëren omdat je de tekst die je geschreven hebt ook exact kunt kopiëren.

Om te kunnen werken met OpenSCAD moet je een aantal commando’s kennen. In het begin zul je veel moeten opzoeken of zelf uit moeten vinden wat wel en niet werkt. Gelukkig is er een *cheat sheet*: `http://www.openscad.org/cheatsheet/`. Deze sheet geeft je een mooi overzicht van de verschillende commando’s die je kunt gebruiken in OpenSCAD en welke informatie je mee moet geven aan die commando’s. De *cheat sheet* geeft je een overzicht van de mogelijkheden, maar legt niet uit hoe je de commando’s gebruikt. Om dat te leren, moet je ontdekken, oefenen, en wellicht dit boekje doorlezen.

Vormen

We gaan beginnen met het maken van een paar simpele 2D-vormen zoals een `square` (rechthoek) en een `circle` (cirkel). De maten doen er nu nog even niet toe, maar alle maten die hier genoemd worden zijn in millimeters. Millimeters zijn de standaardmaat van OpenSCAD.

In tegenstelling tot wat de naam doet vermoeden is de `square` hier een rechthoek, en niet per se een vierkant.

Alle vormen die je maakt, worden afgesloten met een puntkomma (`;`). Doe je dit niet, dan krijg je een foutmelding. Om een cirkel te tekenen, is het niet genoeg om alleen maar `circle` te schrijven. OpenSCAD moet meer informatie hebben over de cirkel, zoals de diameter of de positie van het middelpunt. Deze gegevens plaatsen we binnen haakjes `( )`. Hieronder zie je een aantal voorbeelden van vormen die je kunt gebruiken en de informatie die je mee kunt geven aan deze vormen.

We beginnen met de meest gemakkelijke vorm van een cirkel:

```
circle(50);
```

Dit is een cirkel met radius `r` = 50mm. Je ziet aan deze notatie niet of het de diameter of de radius is die de waarde 50mm heeft. Dat maakt op zich niet uit, maar het kan beter.

```
circle(d=50);
```

Dit is dezelfde een cirkel met diameter `d` = 50mm, maar nu is het duidelijk voor de lezer van de code welke afmeting hier bedoeld is. Vaak kunnen we met een letter of woord aangeven welke eigenschap van het object we definiëren. Hier definiëren we alleen de diameter `d`. We kunnen natuurlijk ook de radius definiëren:

```
circle(r=50,center=true);
```

Dit is een cirkel met radius `r` = 50mm waarvan het middelpunt in de oorsprong ligt. Bij cirkels ligt het middelpunt echter altijd in de oorsprong, dus heel veel zin heeft het hier niet om die informatie toe te voegen.

Als we afmetingen niet definiëren met een letter, zoals bij radius, plaatsen we *blokhaken*, `[]`, om de afmetingen. OpenSCAD weet dan bijvoorbeeld dat we voor een rechthoek deze volgorde aanhouden: [breed,hoog] of [x-waarde,y-waarde]. Hieronder zie je een voorbeeld:

```
square([50,30]);
```

Dit is een rechthoek van 50mm bij 30mm. Als je deze code in OpenSCAD schrijft, zul je zien dat de rechthoek linksonder in de oorsprong ligt.

```
square([50,30],center=true);
```

Als je deze code ingeeft, zul je zien dat de rechthoek als middelpunt de oorsprong heeft.

We kunnen vormen combineren tot grotere, complexere vormen. Dit kan gemakkelijk door twee vormen elkaar te laten overlappen. Zo bijvoorbeeld:

```
square([50,30]);
square([50,30],center=true);
```

Je ziet dat deze vormen gecombineerd zijn omdat het rode randje om de groene vormen nu één doorlopende lijn om beide vormen volgt.

OPDRACHT Maak nu zelf een aantal rechthoeken en cirkels. Probeer ook wat complexere vormen te maken door deze rechthoeken en cirkels elkaar te laten overlappen.

Transformaties en operaties

WE HEBBEN nu twee basisvormen in 2D gezien. Door deze elkaar te laten overlappen kun je al een heel aantal vormen maken. Natuurlijk kom je hier niet heel ver mee. Het liefst wil je ook andere bewerkingen uit kunnen voeren op deze vormen. Deze bewerkingen gaan we hieronder behandelen.

Color

De eerste bewerking die we bekijken is een van de meest simpele: kleur. Tijdens *Preview* kun je kleuren geven aan je objecten. Dit kan je helpen om later te herkennen waar bepaalde elementen zich bevinden, of om dingen duidelijker te maken door middel van transparantie. We kijken nu naar een paar mogelijkheden. `Color` kun je gebruiken met accolades om meerdere objecten een kleur te geven, of zonder accolades als je alleen het opvolgende object een kleur wilt geven. Je kunt kleuren oproepen met de Engelse namen (red, blue, etc.) of met RGB-waarden. Deze RGB-waarden gaan niet van 0 tot 255 in OpenSCAD maar van 0 tot 1 (0% tot 100%). De kleur roze is bijvoorbeeld `[0.9,0.6,0.7]`, wat wil zeggen 90% rood, 60% groen en 70% blauw. `Color` kun je toepassen op 2D- en 3D-objecten.

We maken eerst een rood vierkant. Let op: de kleur verdwijnt zodra je op *Render* klikt.

```
color("Red")square(10);
```

Nu maken we een rood vierkant maar ditmaal door de RGB-waarden te gebruiken:

```
color([1,0,0])square(10);
```

Stel je wilt dat je rode vierkant transparant is. Als laatste waarde na de RGB-waarden kunnen we ook een percentage toevoegen voor de transparantie. In dit geval moet het vierkant nog 70% (`0.7`) van zijn oorspronkelijke ondoorzichtigheid behouden:

```
color([1,0,0,0.7])square(10);
```

Overigens kun je transparantie ook meegeven aan `color` als je alleen de naam van de kleur gebruikt:

```
color("Red",0.7)square(10);
```

Let er even op dat je hier geen blokhaken hoeft te gebruiken omdat je geen RGB-waarden gebruikt.

Union

Naast vormen simpelweg te laten overlappen, kun je vormen ook combineren met behulp van `union`. Dit heeft niet direct voordelen voor onze doelen, maar het kan verderop in de cursus handig zijn om `union` te gebruiken als OpenSCAD je gecombineerde vormen niet correct weergeeft. Omdat `union` zelf geen object is, wordt het niet afgesloten met een puntkomma. Daarnaast kan `union` meerdere objecten bevatten, en dus gebruiken we *accolades*, `{}`, zodat OpenSCAD weet welke objecten gecombineerd moeten worden en waar dit stopt.

```
union() {
    square([50,30]);
    square([50,30],center=true);
}
```

Difference

We kijken nu naar `difference`. Ook `difference` wordt niet afgesloten met een puntkomma. Omdat `difference` meerdere objecten kan bevatten, gebruiken we accolades zodat OpenSCAD weet welke objecten van elkaar afgetrokken moeten worden. Het object dat je over wilt houden (en waar de andere objecten dus van afgetrokken moeten worden) staat altijd bovenaan.

```
difference() {
    square([50,30],center=true);    wel de rechthoek overhouden
    circle(d=20);                   niet de cirkel overhouden
}
```

Je kunt deze volgorde ook omdraaien, maar dan houd je niets over omdat de rechthoek groter is dan de cirkel!

```
difference() {
    circle(d=20);                   wel de cirkel overhouden
    square([50,30],center=true);    niet de rechthoek overhouden
}
```

Translate

Bijna alle objecten die we maken raken de oorsprong (met hun hoekpunt of middelpunt). Maar wat als we onze voorwerpen ergens anders willen plaatsen? Dan gebruiken we `translate`. Zoals je nu wel begrijpt, is ook`translate` geen object en dus sluiten we het niet af met een puntkomma. Het kan meerdere objecten bevatten, dus we kunnen het afsluiten met accolades, maar dit is niet nodig als je `translate` maar één object laat transformeren. Wel geven we waarden zonder letteraanduiding, dus gebruiken we blokhaken. Bij `translate` houden we deze volgorde aan: `[breedte,diepte,hoogte]` of `[x-waarde,y-waarde,z-waarde]`. Hieronder zie je hoe dat werkt.

Stel je wilt de rechthoek 10mm in de positieve x-richting verplaatsen, oftewel 10mm naar rechts. Dat doe je als volgt:

```
translate([10,0,0])
square([50,30],center=true);
```

Je kunt 2D-objecten ook in de z-richting verplaatsen. Dit zie je wel terug in *Preview* (F5) maar niet in *Render* (F6) omdat in *Render* de 2D-objecten geprojecteerd worden op het xy-vlak:

```
translate([0,0,10])
square([50,30],center=true);
```

Het is ook mogelijk om twee objecten tegelijk te verplaatsen:

```
translate([0,0,10]) {
    square([50,30],center=true);
    square([30,50],center=true);
}
```

Rotate

Je kunt voorwerpen ook roteren om een as. De notatie is hetzelfde als bij `translate`. De volgorde is als volgt: `[x-as,y-as,z-as]`.

2D-objecten willen we eigenlijk altijd roteren om de z-as. Hierdoor krijgen we dit resultaat:

```
rotate([0,0,45])
square([50,30],center=true);
```

Natuurlijk is het ook mogelijk om 2D-objecten te roteren om de x-as. Je ziet deze handeling terug in *Preview*, maar in *Render* krijg je nu een vorm die lijkt op een minder diepe rechthoek:

```
rotate([45,0,0])
square([50,30],center=true);
```

Op deze manier kun je het 2D-object zelfs laten verdwijnen! Zie hieronder:

```
rotate([90,0,0])
square([50,30],center=true);
```

Hetzelfde geldt voor roteren om de y-as. Nu wordt de rechthoek een stuk smaller in *Render*:

```
rotate([0,45,0])
square([50,30],center=true);
```

Het is dus belangrijk om goed op te letten bij het roteren van 2D-objecten. Als je per ongeluk de verkeerde as neemt, kan je object opeens weg zijn, of een hele andere vorm aannemen aangezien het geprojecteerd wordt op het xy-vlak. Je kunt dit natuurlijk ook opzettelijk doen om interessante resultaten te krijgen.

Wat nu als je `rotate` en `translate` wilt combineren? Welke volgorde moet je aanhouden? Laten we hieronder een paar voorbeelden behandelen.

```
translate([40,0,0])
rotate([0,0,45])
square([50,30],center=true);
```

In het voorbeeld hierboven zie je dat de rechthoek eerst 45° gewenteld wordt om de z-as, en daarna pas 40mm in de x-richting verschoven wordt. OpenSCAD voert dus als eerste de transformatie uit die het dichtste (in regelafstand) bij het object staat, in dit geval `rotate`.

OpenSCAD voert als eerste de transformatie uit die het dichtste bij het object staat. Het leest dus eigenlijk van achter naar voren.

In het voorbeeld hieronder zie je goed wat er gebeurt als je de volgorde omdraait:

```
rotate([0,0,45])
translate([40,0,0])
color("DeepSkyBlue")
square([50,30],center=true);
```

We voegen hier een kleur toe zodat je goed kunt zien wat er gebeurt.

Nu wordt de blauwe rechthoek eerst 40mm verschoven in de x-richting, en daarna wordt de rechthoek om de z-as gedraaid.

Je zou de code ook anders kunnen schrijven, met accolades, om de volgorde van de acties duidelijk te maken:

```
translate([40,0,0]) {
    rotate([0,0,45]) {
        square([50,30],center=true);
    }
}
```

Hier zie je duidelijk wat de structuur is van de commando's die OpenSCAD leest. OpenSCAD ziet dat het een object moet verplaatsen, maar ziet daarmee ook dat het object eerst nog geroteerd moet worden. In dit voorbeeld zal OpenSCAD dus een een geroteerd object verplaatsen, in plaats van een verplaatst object roteren.

Mirror

De laatste transformatie die we hier behandelen is `mirror`. Je kunt objecten spiegelen in de richting van een as. De notatie is als volgt: `[x-as,y-as,z-as]`. Wel spiegelen is `1`, niet spiegelen is `0`. Als je in de x-richting wilt spiegelen, schrijf je `[1,0,0]`.

```
square([50,30]);                                    origineel
mirror([1,0,0])
color("DeepSkyBlue")
square([50,30]);                                    gespiegelde kopie in de x-richting
```

OPDRACHT Gebruik de transformaties die je net hebt geleerd. Voor een overzicht daarvan kun je de *cheat sheet* raadplegen onder "Transformations". Probeer ook vast te werken met `text` en `polygon`. Kijk daarvoor naar de *cheat sheet* onder "2D".
Uitdaging: Experimenteer ook eens met `scale` en `resize`.

Resolutie

SOMS als je cirkels maakt, zie je dat ze niet perfect rond zijn. Ze lijken uit allemaal kleine lijnstukjes of *fragments* te bestaan. Eigenlijk is dat ook zo. Maar in OpenSCAD kun je instellen uit hoeveel fragmentjes je object moet bestaan. Je kunt dit instellen door middel van `$fn` (*fragment number*). Hoe meer fragmentjes je gebruikt, hoe hoger de resolutie van je object wordt maar hoe langer het renderen duurt. Je kunt deze code aan het begin van je document plaatsen om zo de resolutie van alle objecten te definiëren, maar `$fn` kun je ook aan individuele objecten meegeven. Hieronder zie je een paar richtlijnen voor verschillende resoluties en wanneer je welke resolutie het beste kunt gebruiken. Deze resoluties kun je aan het begin van je document plaatsen zodat je de resolutie voor al je objecten kunt instellen. Let er wel even op dat je de resolutie afsluit met een puntkomma.

```
$fn = 50;       werkt goed om OpenSCAD snel te houden
$fn = 100;      is het maximum tijdens prototyping
$fn = 150;      alleen instellen bij de allerlaatste render
```

Je kunt ook met de resolutie spelen om van cirkels andere vormen te maken. Probeer de onderstaande code eens uit om te zien welke geometrische vormen je krijgt in plaats van een cirkel.

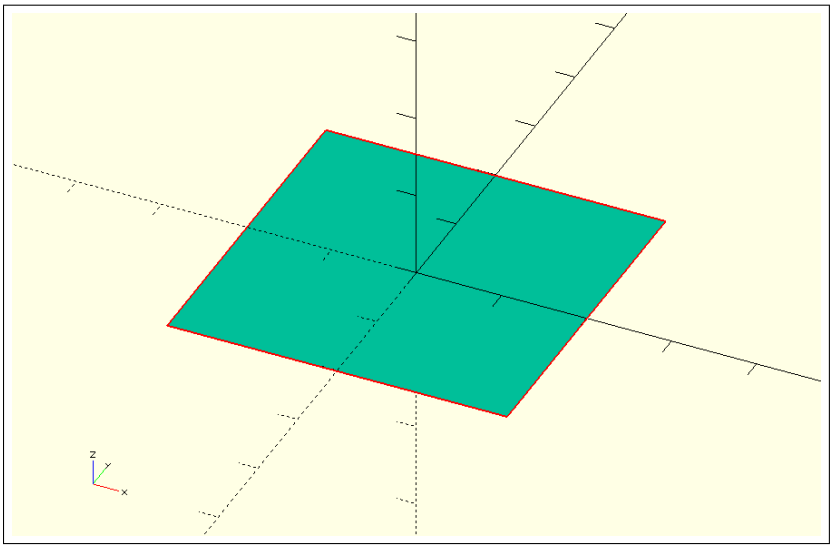
```
circle(d=20,$fn=3);     3 fragmenten om één cirkel te maken
circle(d=20,$fn=4);     4 fragmenten om één cirkel te maken
```

```
circle(d=20,$fn=6);      6 fragmenten om één cirkel te maken
circle(d=20,$fn=8);      8 fragmenten om één cirkel te maken
```

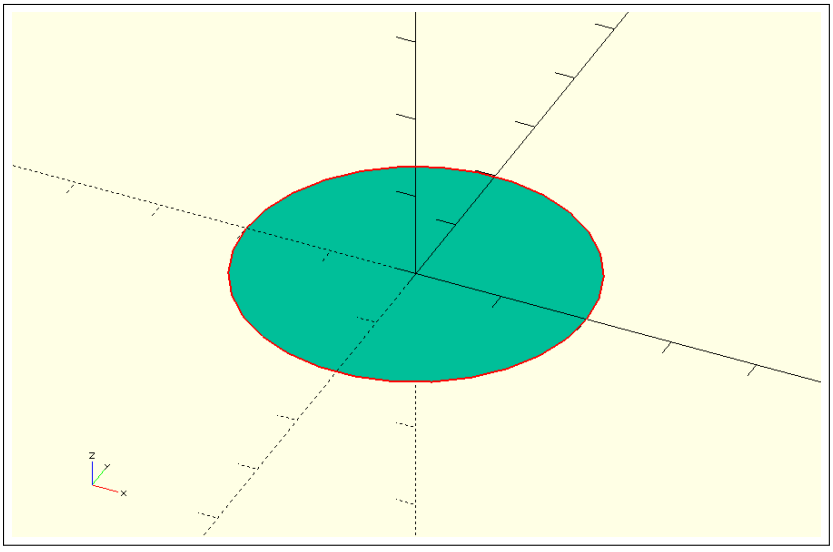
PROJECT Ontwerp een glas-in-lood raam door middel van de vormen die we tot zover behandeld hebben en de `difference`, `rotate` en `translate` functie.

Opdrachten hoofdstuk 1

Probeer de onderstaande voorbeelden na te maken. Op de rechterpagina kun je de code vinden om jouw oplossing mee te vergelijken.



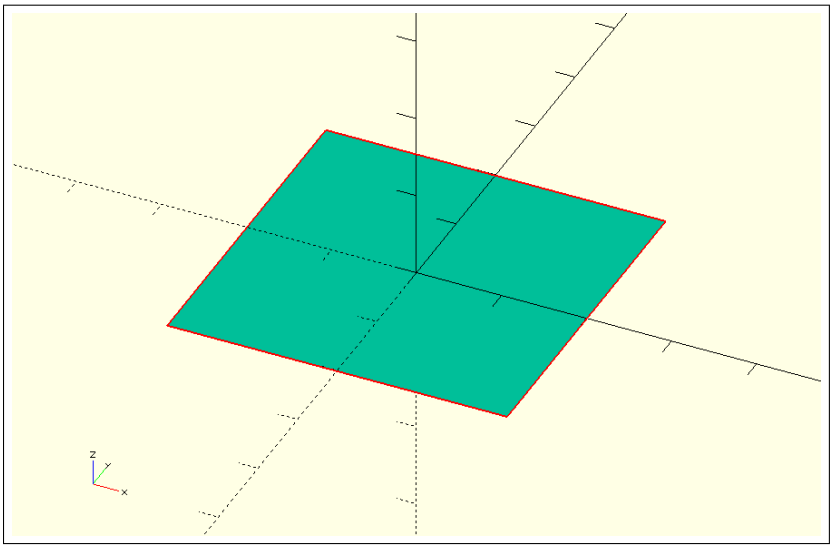
1.1



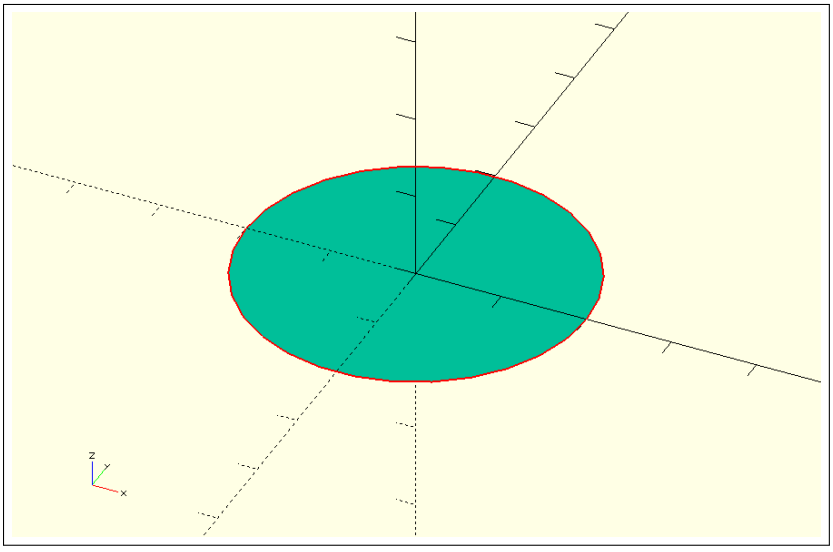
1.2

Antwoorden

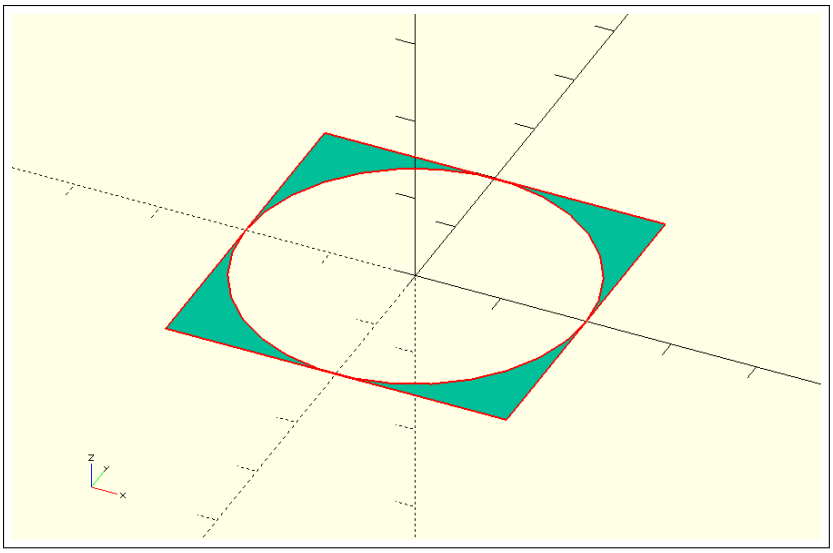
Hieronder zie je wat voorbeeldcode. Deze code kan je helpen om de onderstaande vormen te maken.



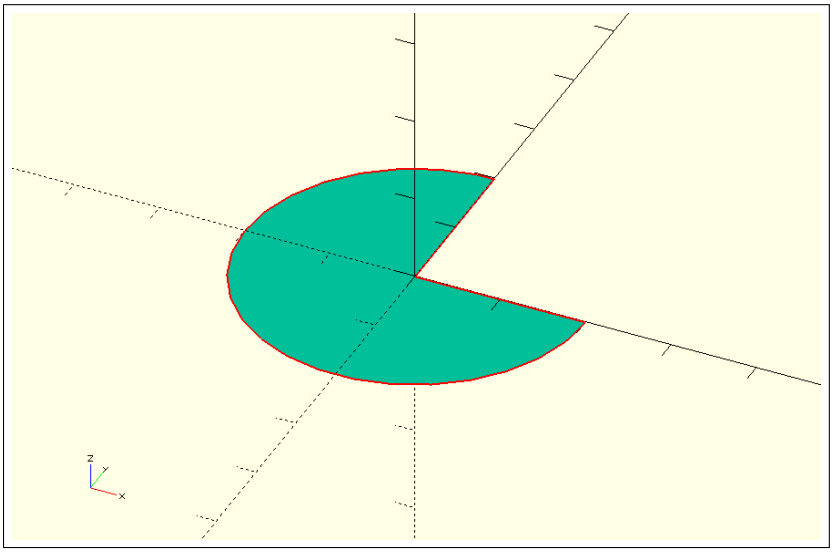
```
square(40,center=true);
```



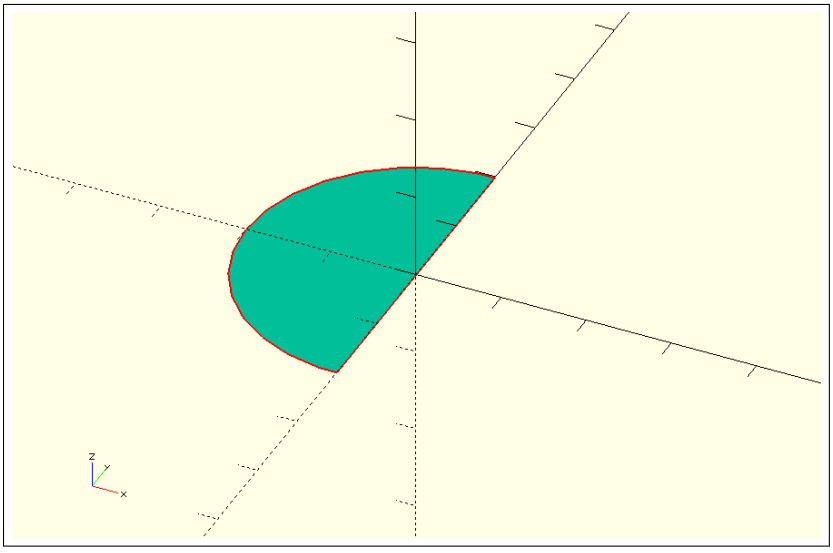
```
circle(20);
```



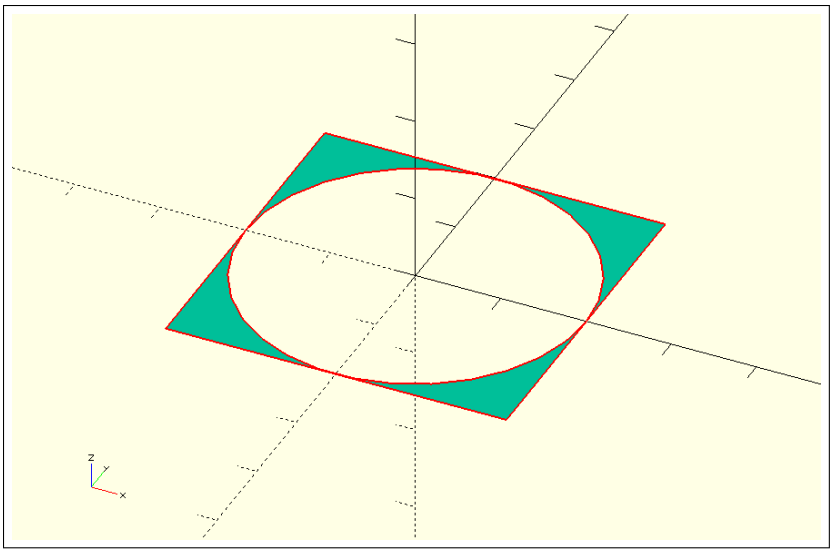
1.3



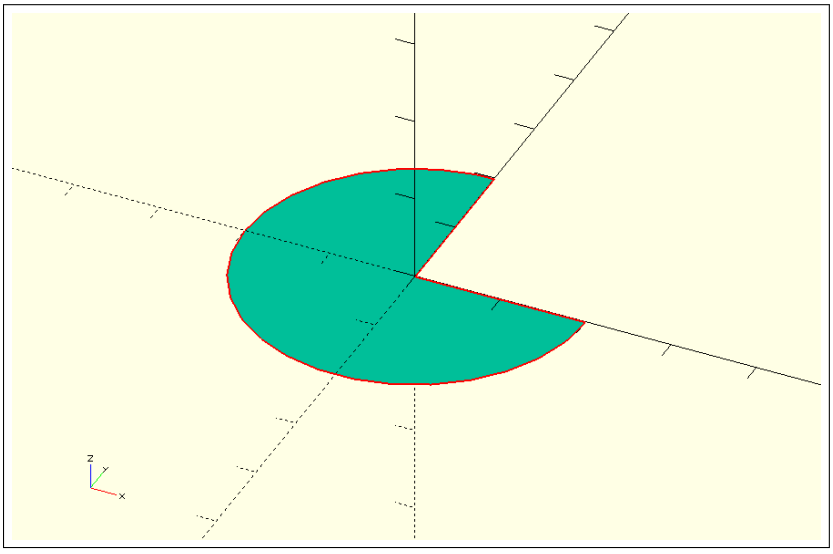
1.4



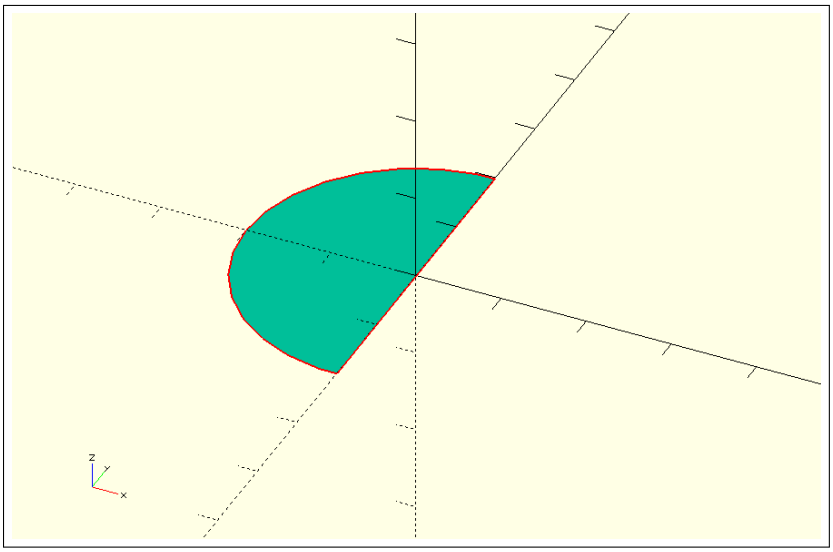
1.5



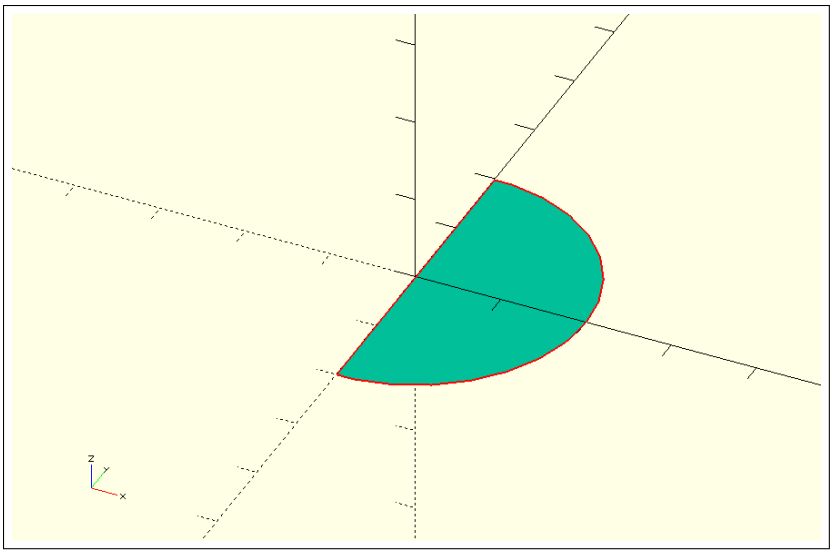
```
difference() {
    square(40,center=true);
    circle(20);
}
```



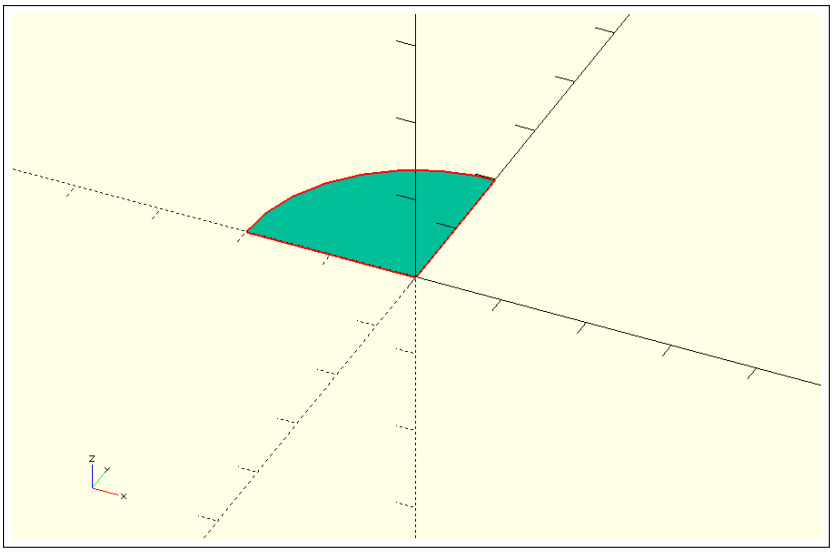
```
difference() {
    circle(20);
    square(20);
}
```



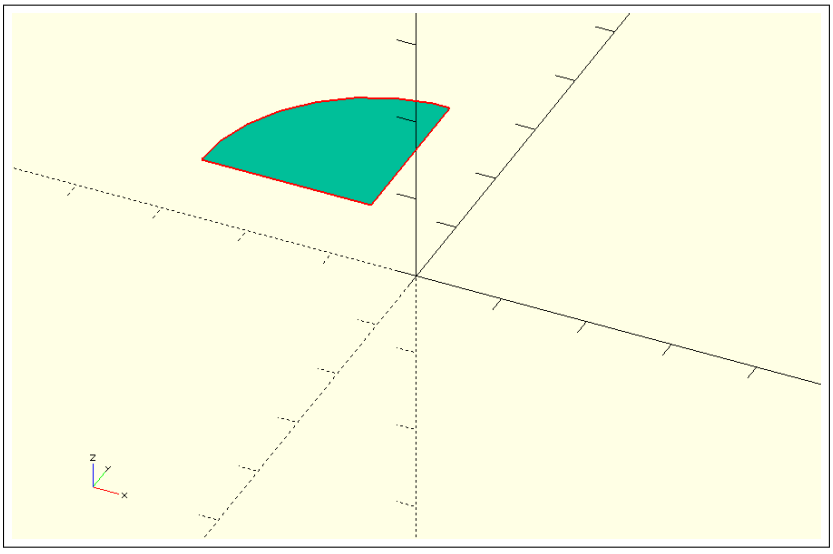
```
difference() {
    circle(20);
    translate([0,-20,0])
    square(40);
}
```



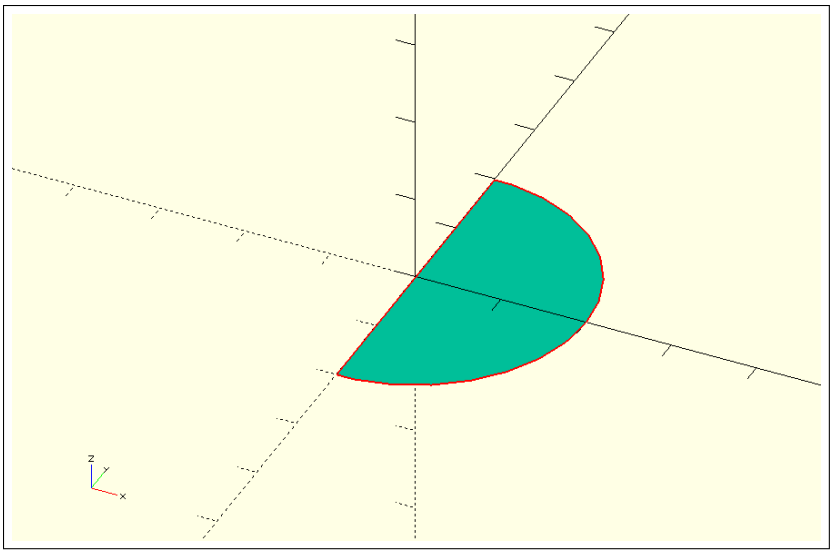
1.6



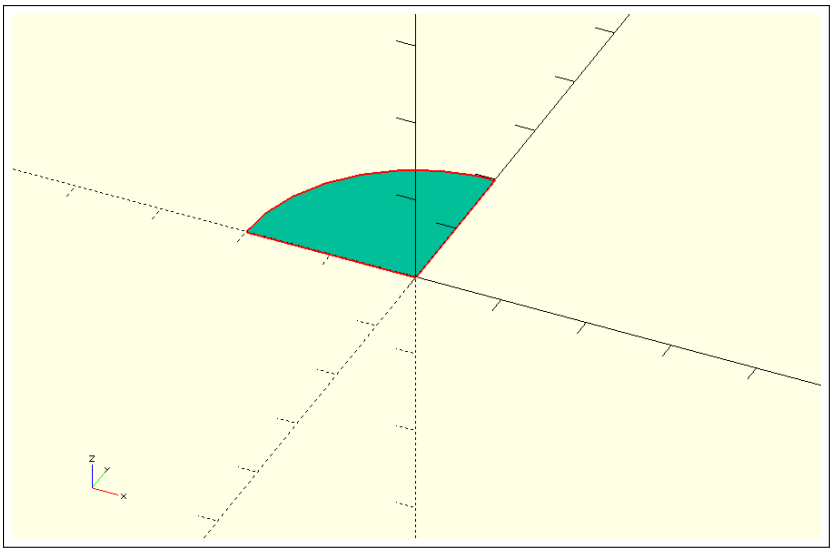
1.7



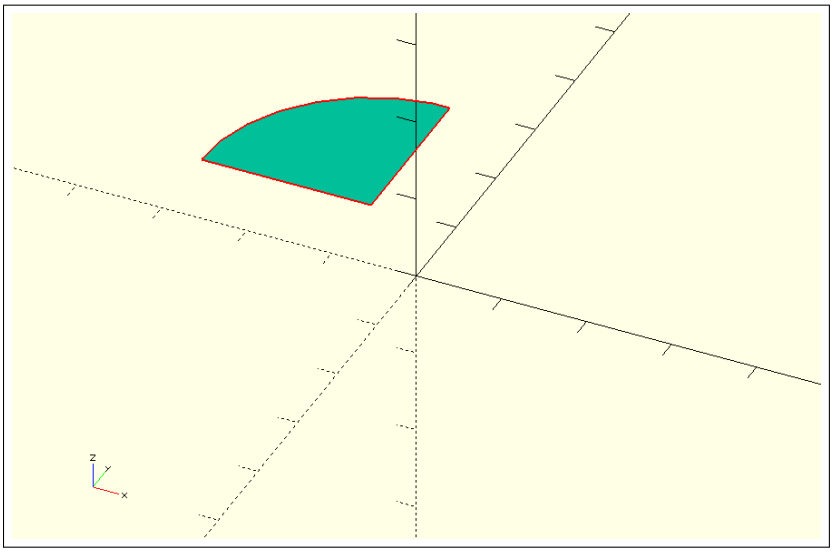
1.8



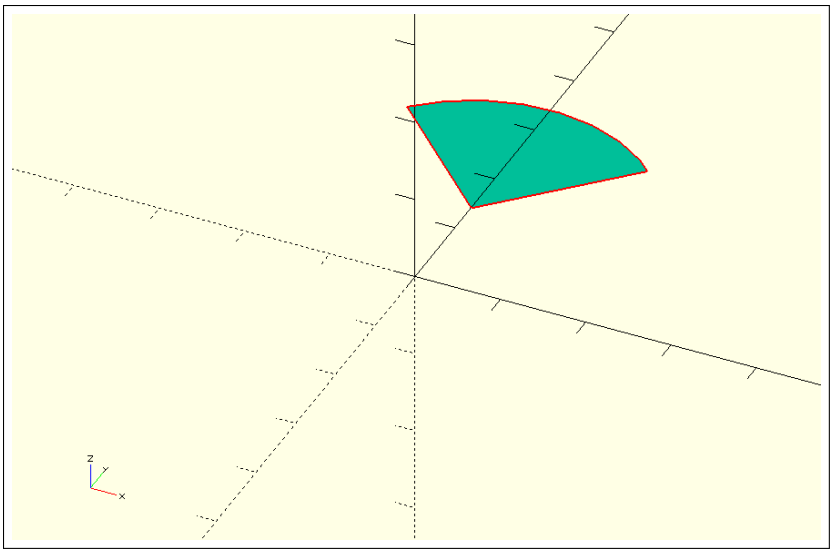
```
mirror([1,0,0])
difference() {
    circle(20);
    translate([0,-20,0])
    square(40);
}
```



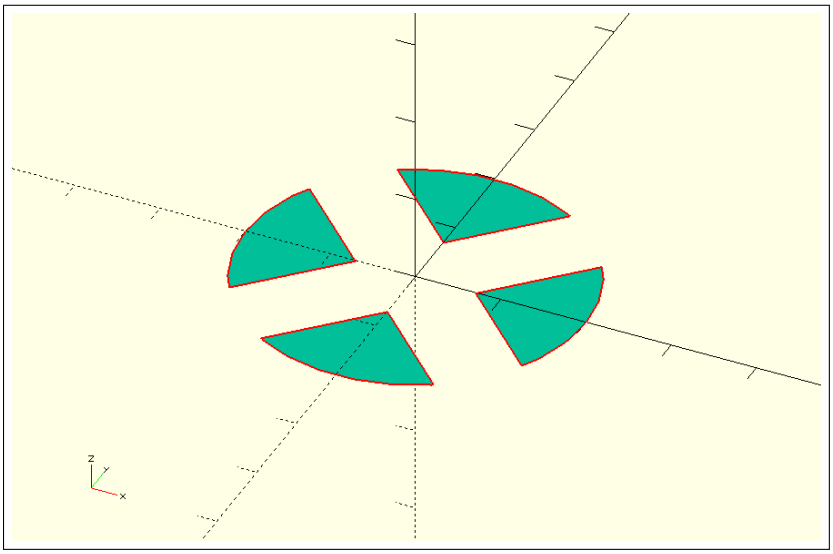
```
difference() {
    circle(20);
    translate([0,-20,0])
    square(40);
    translate([-20,-20,0])
    square(20);
}
```



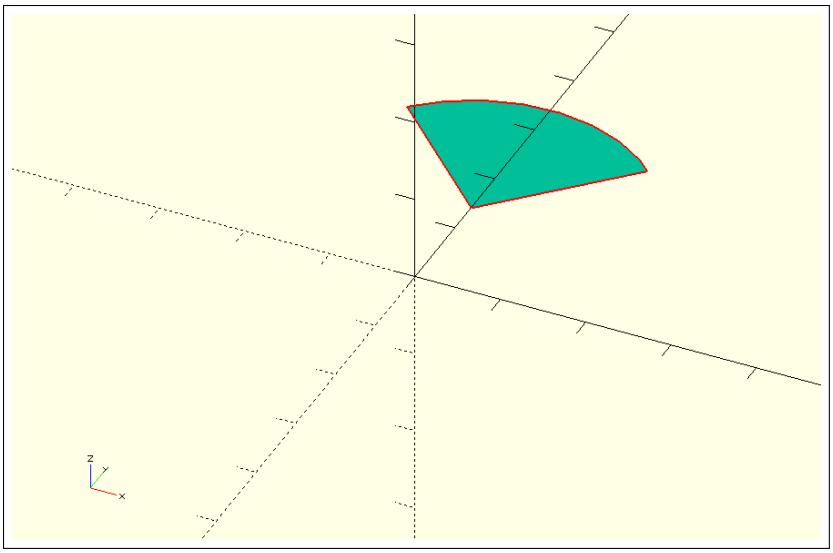
```
translate([-10,10,0])
difference() {
    circle(20);
    translate([0,-20,0])
    square(40);
    translate([-20,-20,0])
    square(20);
}
```



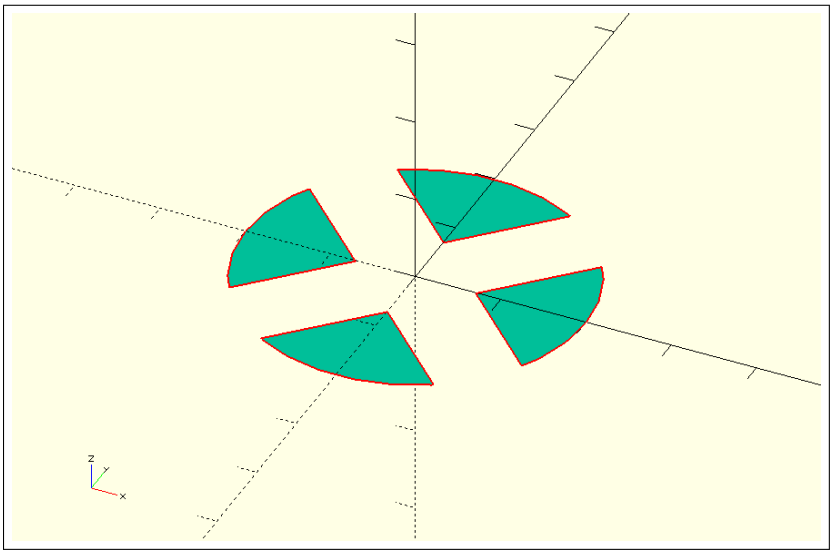
1.9



1.10



```
rotate([0,0,-45])
translate([-10,10,0])
difference() {
    circle(20);
    translate([0,-20,0])
    square(40);
    translate([-20,-20,0])
    square(20);
}
```



```
difference() {
    circle(20);
    rotate([0,0,45])
    square([10,40],center=true);
    rotate([0,0,-45])
    square([10,40],center=true);
}
```

Van tweedimensionaal naar driedimensionaal

Onderwerpen

2D en 3D Kun je 2D en 3D mengen?

Van 2D naar 3D Wat is de *extrude* functie? Waarom zou ik *extrude* gebruiken in plaats van *cylinder* of *cube* om bepaalde vormen te maken?

Nieuwe mogelijkheden `cube`, `sphere`, `cylinder`, `linear_extrude`, `rotate_extrude`, `intersection`.

Modules De kracht achter OpenSCAD. Maak modules en verwijs daar naar terug.

Project Ontwerp een tolletje met behulp van de 3D-objecten die we behandeld hebben en gebruik tenminste `difference` en `module`. Uitdaging: maak twee identieke tolletjes; eentje met `rotate_extrude`, en de andere door vormen te combineren.

Veelgebruikte objecten

In hoofdstuk 1 hebben we met 2D-objecten gewerkt. Hieronder behandelen we kort de drie 3D-objecten die je het meeste zult gebruiken, de balk, bol en cilinder: `cube, sphere, cylinder`. De notatie is bijna hetzelfde als wat je al gewend bent van de 2D-objecten, alleen voegen we nu ook een z-waarde toe: de hoogte.

In tegenstelling tot wat de naam doet vermoeden is de `cube` hier een balk, en niet per se een kubus.

Hieronder zie je een simpele balk van 20mm breed, 15mm diep, en 10mm hoog:

```
cube([20,15,10]);
```

Deze balk kun je ook centreren in de oorsprong:

```
cube([20,15,10],center=true);
```

Als je een blokje wilt maken van 10mm bij 10mm bij 10mm, hoef je niet alle maten in te vullen. Dit werkt ook bij `square`, `circle`, `sphere`.

```
cube(10);
```

Een bol is altijd gecentreerd in de oorsprong. Omdat bij een bol per definitie de breedte, diepte en hoogte hetzelfde zijn, kun je de maat doorgeven zoals bij de balk in het vorige voorbeeld. Let erop dat de 10mm hier standaard de radius betreft, niet de diameter.

```
sphere(10);
```

Om de diameter op te geven moeten we deze expliciet noemen:

```
sphere(d=10);
```

We kunnen een cilinder maken door een hoogte en een radius of diameter op te geven.

```
cylinder(h=30,d=10);
```

Je kunt ook een taps toelopende cilinder maken door de diameter van de basis anders in te stellen dan die van de top:

```
cylinder(h=30,d1=10,d2=5);
```

Andersom zou je ook de basis kleiner kunnen maken en de top groter. Hoe klein kun je deze diameters maken? Als je de top een waarde van `d=0` meegeeft, creëer je een kegel.

```
cylinder(h=30,d1=10,d2=0);
```

In het vorige hoofdstuk heb je al met `rotate` gewerkt in 2D. Belangrijk hier is dat OpenSCAD altijd de volgorde van de assen aanhoudt: eerst roteren om de x-as, dan om de y-as en als laatste om de z-as. We kijken nu naar hoe dit werkt met 3D-objecten. We nemen een balk:

```
cube([10,20,2]);
```

We beginnen met een rotatie om de x-as (ingesprongen om de volgorde duidelijk te maken).

Voorbeeld 1:

```
rotate([90,0,0])
    cube([10,20,2]);
```

En daar voegen we een rotatie om de y-as aan toe:

```
rotate([0,90,0])
    rotate([90,0,0])
        cube([10,20,2]);
```

Deze balk is dus *eerst* om de x-as geroteerd, en daarna om de y-as. We kunnen dit ook omdraaien.

Voorbeeld 2:

```
rotate([90,0,0])
    rotate([0,90,0])
        cube([10,20,2]);
```

Dit produceert een ander resultaat. We zien nu dat de balk eerst om de y-as is geroteerd, en daarna om de x-as. Als laatste bekijken we wat er gebeurt als we deze twee rotaties in een keer uitvoeren:

```
rotate([90,90,0])
    cube([10,20,2]);
```

Welke van de twee resultaten zie je nu? Het resultaat van voorbeeld 1 of 2? Hieruit blijkt dat OpenSCAD altijd de volgorde x, y, z aanhoudt.

OPDRACHT Neem even de tijd om zelf wat te maken met deze basisobjecten. Probeer ook `difference` hierbij te gebruiken.

Tenslotte heb je misschien de vraag: "Kun je 2D en 3D mengen?" Het is beter om dat niet te doen. Hieronder zie je waarom.

```
difference() {
    translate([0,0,-1])
    cube([30,30,1]);
    circle(30);
}
```

Bekijk dit in *Preview* en in *Render*. Valt je wat op? OpenSCAD geeft je een waarschuwing: *WARNING: Mixing* 2D *and* 3D *object is not supported*. We kunnen het gewenste effect wel krijgen door het volgende te doen:

```
difference() {
    translate([0,0,-1])
    cube([30,30,1]);
    cylinder(h=1,r=30,center=true);
}
```

Je ziet nu dat de cilinder echt voor een deel uit de balk gehaald is.

Linear_extrude

SOMS WIL JE beginnen met het maken van een 2D-object en dit daarna een dikte geven zodat het een 3D-object wordt. Dit kan handig zijn bijvoorbeeld wanneer je een tekening importeert. Eigenlijk ben je dan bezig om je vlakke vorm uit te trekken of te 'extruderen'. Deze extrusie kun je bewerkstelligen door middel van `linear_extrude`. Aangezien `Linear_extrude` één of meerdere objecten kan bevatten, heb je geen accolades nodig als je maar één object wilt extruderen. Als je meerdere objecten een dikte mee wilt geven, gebruik je wel accolades. Deze functie kun je erg veel gegevens meegeven, zoals `height, center, convexity, twist, slices, scale`. Allereerst kijken we naar `height`. Hieronder een voorbeeldje:

```
linear_extrude(height=3)
difference() {
    square([50,30],center=true);
    circle(d=20,center=true);
}
```

Je hebt nu een 3D-model gemaakt met hoogte 3mm. Let erop dat je nu geen 2D-object meer ziet als je op *Render* klikt. Dit principe kun je op alle 2D-vormen toepassen. Je kunt ook andere 2D-vormen combineren of van elkaar aftrekken en die daarna een hoogte geven; het principe blijft hetzelfde.

Deze vorm van werken lijkt op het eerste gezicht misschien wat onhandig of veel extra werk. Waarom niet meteen een 3D-object maken? Extrusie gebruiken we graag als we modellen maken voor de lasersnijder. Op deze manier kun je de dikte van het materiaal meegeven als `height` en deze gemakkelijk naderhand aanpassen terwijl je wel een vlakke tekening in je bestand hebt opgenomen die (zonder extrusie) gemakkelijk geëxporteerd kan worden naar een formaat waar een lasersnijder mee kan werken.

Maar dit is maar één voorbeeld van de mogelijkheden. Je kunt veel meer doen met `linear_extrude`. We bekijken nu een aantal van de andere opties.

```
linear_extrude(height=60,twist=90,slices=100)
square([20,20],center=true);
```

Hier is de hoogte op 60mm ingesteld, bestaat het te creëren object uit 100 schijfjes (`slices`), en draait (`twist`) het vierkant 90° ten opzichte van de basis (om de z-as heen). Hoe meer slices, hoe mooier de overgang wordt. We kunnen ook meer dan 90° roteren:

```
linear_extrude(height=60,twist=360,slices=100)
square([20,20]);
```

Let erop dat de rotatie plaatsvindt om de z-as.

Zoals verwacht draaien we nu 360° om de z-as heen.

In het voorbeeld hieronder gaan we experimenteren met `scale`. Wat voegt dit toe aan het voorbeeld?

```
linear_extrude(height=60,twist=360,slices=100,scale=2)
square([20,20]);
```

Wat betekent `scale=2` hier? Je verwacht misschien dat het hele object twee keer zo groot wordt, maar dat is niet het geval. Het lijkt erop dat alleen het eindpunt van de extrusie een factor twee groter is geworden, en dat de extrusie in schijfjes naar die verdubbeling toewerkt. De schijfjes worden dus steeds een stukje groter. Je kunt nu ook zien welk vierkant het origineel is; deze heeft namelijk de originele grootte behouden.

We voegen nog een kenmerk toe, te weten `center=true`:

```
linear_extrude(height=60,center=true,twist=360,slices=100,scale=2)
square([20,20]);
```

Je ziet dat het hele object van positie is veranderd; het middelpunt van het object ligt nu in de oorsprong. Als je `center=true` toevoegt aan `linear_extrude` is het dus niet zo dat de vorm die je extrudeert verplaatst wordt naar de oorsprong en dat daarna de extrusie uitgevoerd wordt.

OPDRACHT Ga nu zelf experimenteren met `linear_extrude`.

Rotate_extrude

Nu we `linear_extrude` onder de knie hebben, gaan we verder met `rotate_extrude`. Let er even op dat deze functie in OpenSCAD 2015.xx minder uitgebreid is dan in latere bètaversies. We gaan hier alleen in op de simpelere bewerkingen die `rotate_extrude` uit kan voeren. Als je wilt weten wat er mogelijk is in de bètaversies kun je het beste de OpenSCAD-handleiding raadplegen.

`rotate_extrude` wentelt 2D- of 3D-objecten om de z-as. 2D-objecten moeten hiervoor in het xy-vlak liggen (of op het xy-vlak geprojecteerd zijn). OpenSCAD roteert het voorwerp eerst 90° om de x-as en wentelt het daarna om de z-as. Ook `rotate_extrude` heeft geen

accolades nodig als je één object wilt wentelen, maar bij meerdere objecten gebruik je wel accolades. Hieronder zie je een voorbeeld:

```
rotate_extrude()
square([10,30]);
```

We hebben nu een cilinder gemaakt! Als we de rechthoek wat opschuiven kunnen we ook een holle cilinder maken:

```
rotate_extrude()
translate([10,0,0])
square([10,30]);
```

Als je niet meteen snapt waarom deze cilinder hol is, laat dan eerst `rotate_extrude()` weg uit je code.

Daarnaast kunnen we met de resolutie spelen. Hierbij kunnen we de resolutie opnemen in onze vorm, bijvoorbeeld `circle`, of in onze bewerking, `rotate_extrude`. We beginnen eerst met het maken van een torus door een cirkel om de z-as heen te wentelen:

```
rotate_extrude()
translate([20,0,0])
circle(5);
```

De resolutie van deze torus ligt nog wat laag. We kunnen goed zien uit welke segmenten het object is opgebouwd. Daar willen we natuurlijk wat aan veranderen door de resolutie te verhogen naar `$fn=100`. Deze resolutie passen we eerst toe op de cirkel:

```
rotate_extrude()
translate([20,0,0])
circle(5,$fn=100);
```

Nu wordt de torus al iets mooier maar nog steeds zijn verschillende segmenten duidelijk zichtbaar. Wat is hier aan de hand? De resolutie van de cirkel is hoog waardoor deze er mooi rond uit blijft zien in de wenteling om de z-as. Maar de stapjes waarmee de cirkel om de z-as gewenteld wordt blijven vrij groot. Om de torus er echt mooi uit te laten zien is het belangrijk om de resolutie niet te geven aan de 2D-vorm die op het platte vlak ligt, maar aan de extrusie zelf:

```
rotate_extrude($fn=100)
translate([20,0,0])
circle(5);
```

Nu is de resolutie van het hele object hoog, zowel van de cirkel als van de stapjes waarmee deze cirkel om de z-as is gewenteld.

We nemen een laatste voorbeeld. We kantelen een rechthoek door deze 45° om de oorsprong te draaien. Daarna wentelen we deze rechthoek om de z-as. We zien dan een soort kommetje ontstaan:

```
rotate_extrude()
rotate([0,0,-45])
square([10,30]);
```

Het valt je misschien op dat de rotatierichting van de rechthoek negatief is. Wat zou er gebeuren als je deze rotatierichting positief maakt? OpenSCAD geeft dan een foutmelding. De rechthoek ligt in dat geval niet meer in zijn geheel rechts van de y-as, maar gaat de y-as over. OpenSCAD kan de wenteling daardoor niet meer goed uitvoeren. Je wil altijd dat de objecten die je gaat wentelen in hun geheel rechts (of eventueel links) van de y-as liggen. Met andere woorden, alle x-waarden van je object moeten positief zijn.

Bij `rotate_extrude` moet je hele object rechts van de y-as liggen.

OPDRACHT Ga zelf experimenteren met `rotate_extrude`. Let erop dat `angle` in OpenSCAD 2015.xx nog niet ondersteund wordt.

Intersection

In het vorige hoofdstuk heb je gezien hoe `difference` en `union` werken. We gaan nu kijken naar een operatie die deze twee op een bepaalde manier combineert. `intersection` onderzoekt twee, of meer, objecten en kijkt waar deze overlappen. Vervolgens haalt het alle delen die niet overlappen weg. Wat uiteindelijk resteert is de doorsnede van de twee objecten. Omdat `intersection` meerdere objecten moet bevatten, gebruik je accolades.

We kijken eerst naar een simpel voorbeeld, een doorsnede van een kubus en een bol.

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

Je ziet hier dat het grootste gedeelte van de kubus intact is gebleven. Alleen de randen van de kubus zijn op een bijzondere manier afgerond omdat het oppervlak van de bol het oppervlak van de kubus daar doorkruist. Haal in deze code ook eens de `intersection` weg om te zien hoe de objecten eruitzien zonder de *intersection*-operatie.

Je kunt allerlei interessante dingen doen door objecten elkaar te laten snijden. Zo kunnen we ook drie cilinders laten overlappen, zoals in een venndiagram, om op deze manier een driehoek te maken die ronde zijden heeft.

```
intersection() {
    cylinder(h=10,d=20);
    translate([4,7,0])
    cylinder(h=10,d=20);
    translate([8,0,0])
    cylinder(h=10,d=20);
}
```

Het zou erg moeilijk zijn om deze vorm op een andere manier te maken.

`intersection` is dus erg geschikt om vormen te creëren die rondingen bevatten. Een ander interessant object kun je maken door twee cilinders nemen, waarvan er eentje loodrecht op de andere staat. Hieronder zie je de code:

```
intersection() {
    cylinder(h=15,d=15,center=true);
    rotate([90,0,0])
    cylinder(h=15,d=15,center=true);
}
```

Ook hier is het erg lastig om dit object op een andere wijze te maken.

Modules

Een van de krachtigste functionaliteiten in OpenSCAD is de 'module'. Hiermee kun je verschillende objecten combineren tot één geheel. Dit geheel kun je dan in een keer aanroepen door middel van `module`. We geven onze module altijd een naam, als volgt: `module naam()`. Zo kun je bijvoorbeeld een paar vormen combineren in een module, en daarna de hele module verplaatsen. Zo hoef je niet alle afzonderlijke objecten één voor één te verplaatsen. Zoals je wel begrijpt kan `module` meerdere elementen bevatten, dus gebruiken we accolades. Let op: een module bevat wel objecten, maar is daarmee nog niet meteen zichtbaar. Eigenlijk creëer je met een module een nieuwe definitie voor een object. De module moet je dus nog aanroepen, anders zie je niets.

NB: roep na het creëeren van een module deze module nog wel aan!

Laten we een L-vorm maken. Hiervoor combineren we een horizontale en verticale balk waardoor een L ontstaat. We kunnen dit doen door twee 2D-objecten te tekenen:

```
square([30,10]);
square([10,50]);
```

Nu willen we onze L naar rechts verplaatsen. Om dit te doen zullen we `translate` moeten gebruiken:

```
translate([10,0,0]) {
    square([30,10]);
    square([10,50]);
}
```

Je begrijpt dat dit al vrij snel lastig wordt, zeker als je meerdere transformaties en operaties hebt gebruikt om je object te creëren. Om ons object in één keer te kunnen verplaatsen kunnen we `module` gebruiken. We maken eerst de module aan:

```
module Lvorm() {
    square([30,10]);
    square([10,50]);
}
```

Als je alleen deze code aan OpenSCAD geeft zul je niets zien. Je zult ook de module aan moeten roepen:

```
module Lvorm() {
    square([30,10]);
    square([10,50]);
}
Lvorm();
```

Als we nu de L naar rechts willen verplaatsen, gebruiken we `translate` voordat we onze module aanroepen:

```
module Lvorm() {
    square([30,10]);
    square([10,50]);
}
translate([10,0,0])
Lvorm();
```

Merk hierbij op dat je ook `union` zou kunnen gebruiken om je L-vorm te creëren en deze in één keer te verplaatsen.

Nu is dit natuurlijk nog niet zo spannend. Het wordt interessanter wanneer we modules gaan gebruiken om objecten van gecombineerde vormen te bewerken. Stel, je wilt de vorm van een maan in een balk stansen. Er is geen manier in OpenSCAD om een 'maan-object' op te roepen. Je zult hiervoor altijd twee cirkels van elkaar af moeten trekken. Dit verschil van twee cirkels kun je vervolgens weer uit een rechthoek halen. Je hebt dan een verschil in een verschil. Als je de maan maar één keer wilt ponsen gaat dit nog goed, maar als je dit 10 keer wilt doen wordt het wel erg onoverzichtelijk zonder modules.

We beginnen met de maan met een hoogte van 2mm:

```
module halve_maan() {
    difference(maan) {
        cylinder(h=2,d=30);
        translate([0,-15,0])
        cylinder(h=2,d=40);
    }
}
```

Let er even op dat je geen minteken (-) gebruikt in de naamgeving van je module. Gebruik in plaats daarvan een liggend streepje (_)

Daarna kunnen we de maan zo vaak uit een rechthoek halen als we maar wilen:

```
difference(rechthoek) {
    cube([50,50,3],center=true);
```

Je kunt je transformaties en operaties namen meegeven tussen de haakjes. Zo blijft je code beter leesbaar.

```
    halve_maan();
    translate([0,-15,0])
    halve_maan();
}
```

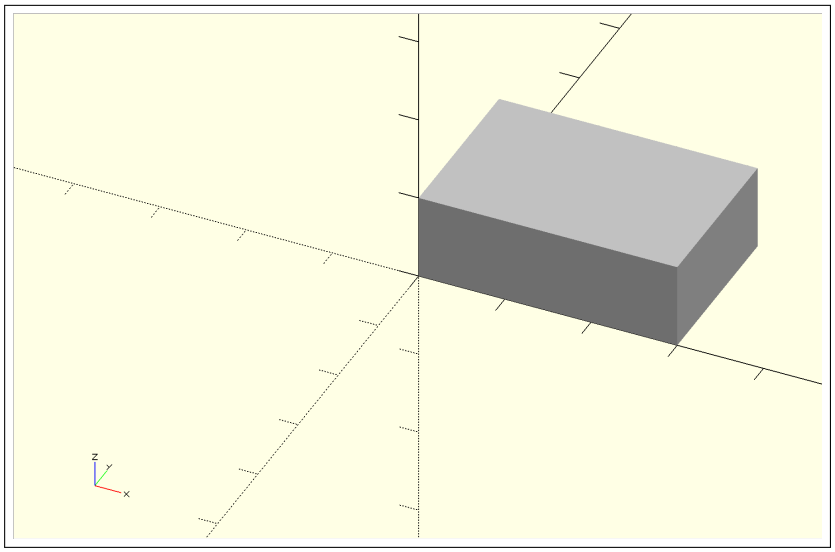
OPDRACHT Ga experimenteren met `module`. Kun je een module in een module schrijven? Wat gebeurt er als je een module oproept voordat je deze geschreven hebt?

PROJECT Ontwerp een tolletje met behulp van de 3D-objecten die we behandeld hebben en gebruik tenminste `difference` en `module`.

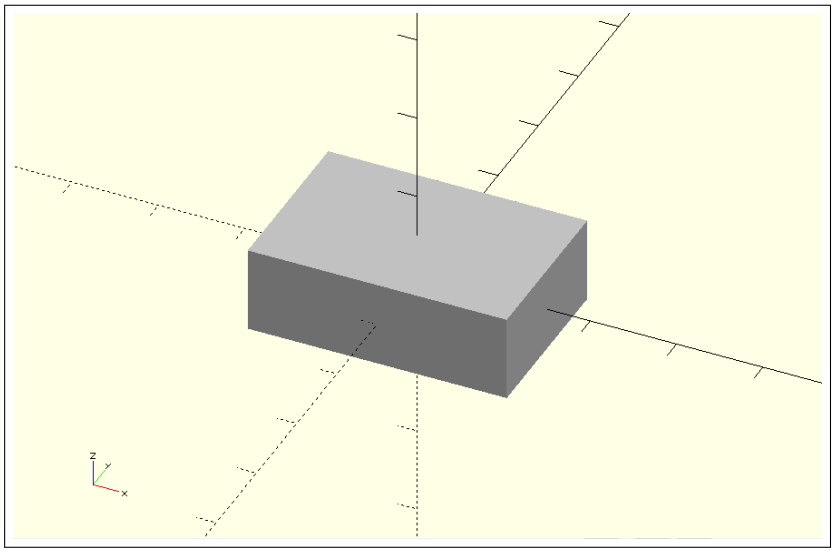
Uitdaging: maak twee identieke tolletjes; eentje met `rotate_extrude`, en de andere door vormen te combineren.

Opdrachten hoofdstuk 2

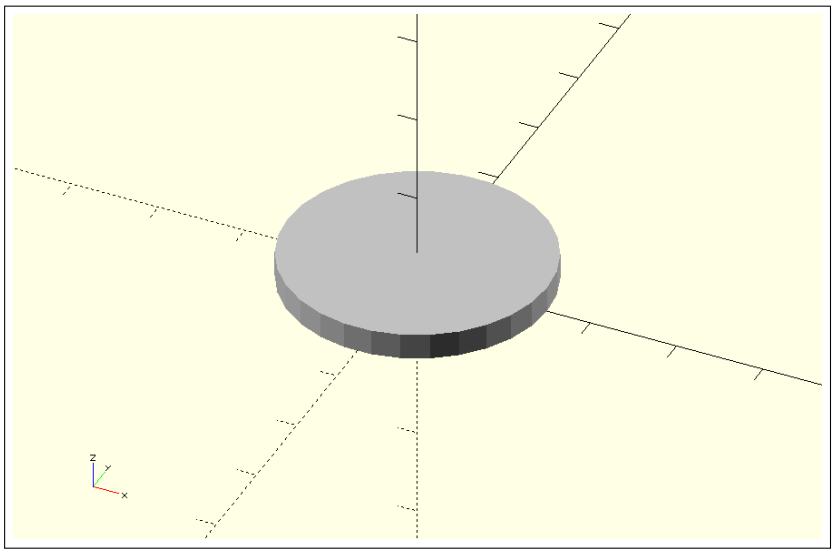
2.1

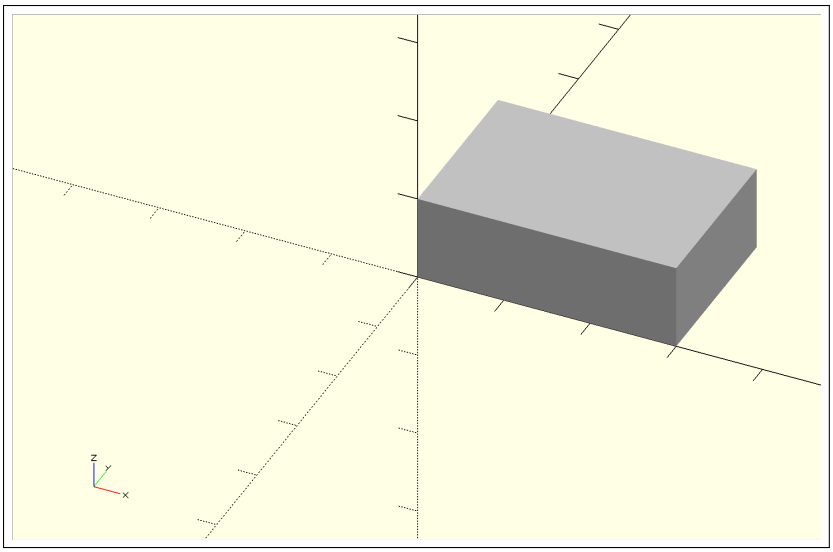


2.2

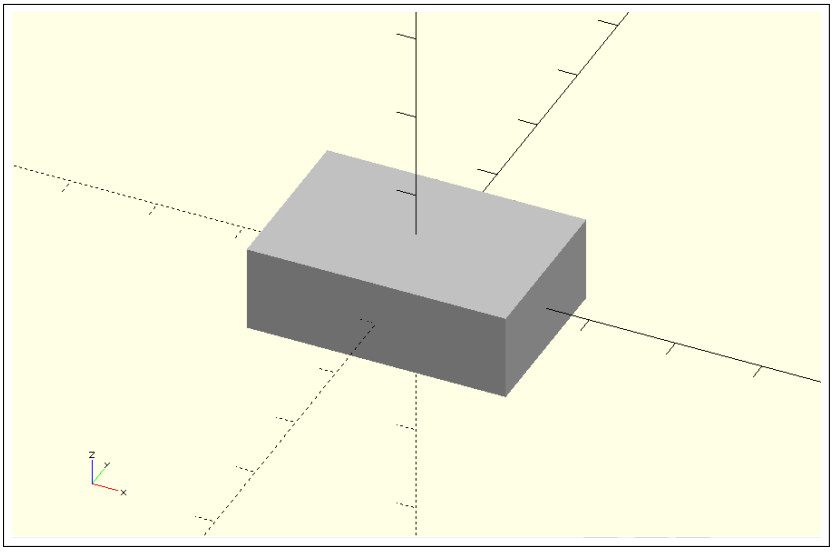


2.3

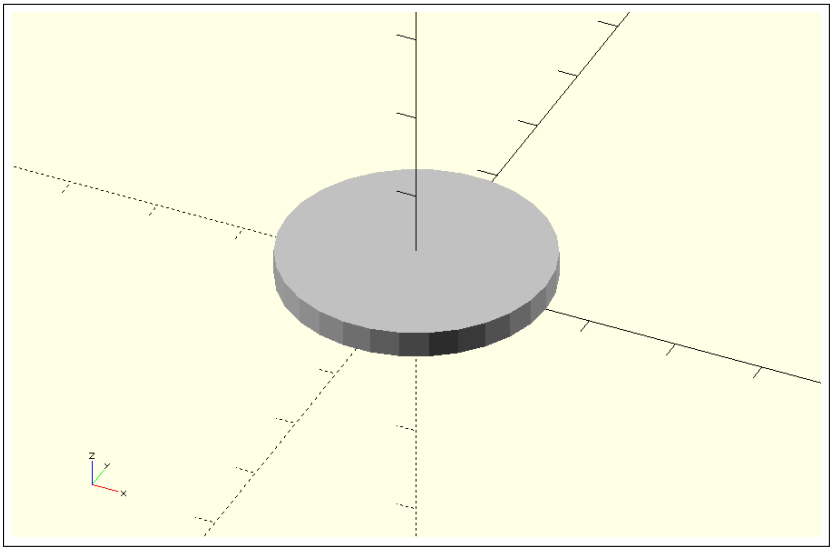




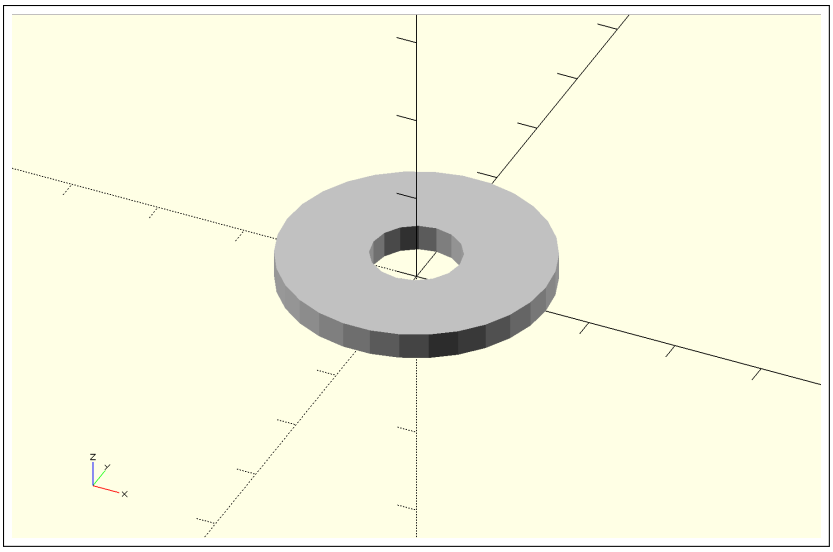
```
cube([30,20,10]);
```



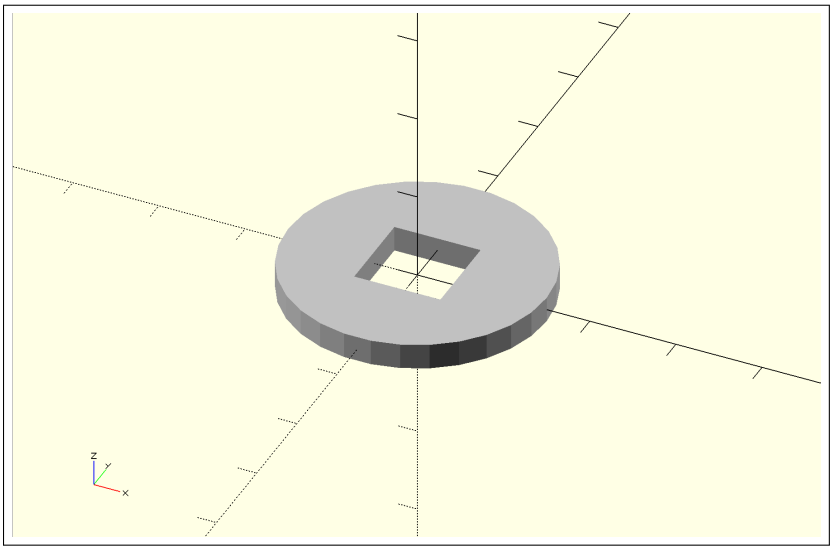
```
cube([30,20,10],center=
true);
```



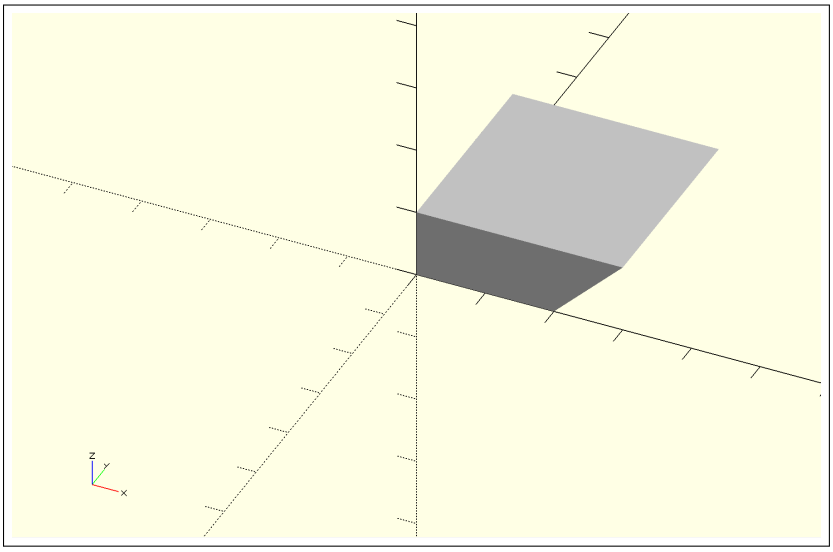
```
cylinder(d=30,h=3);
```



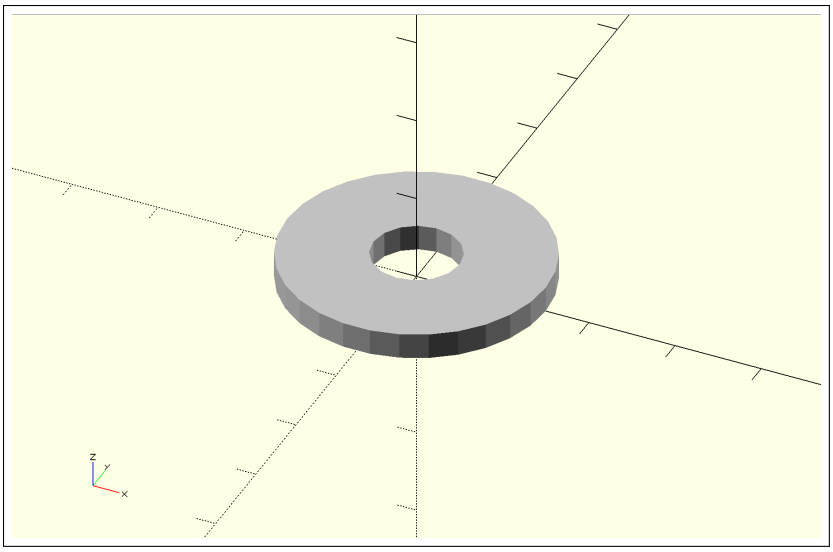
2.4



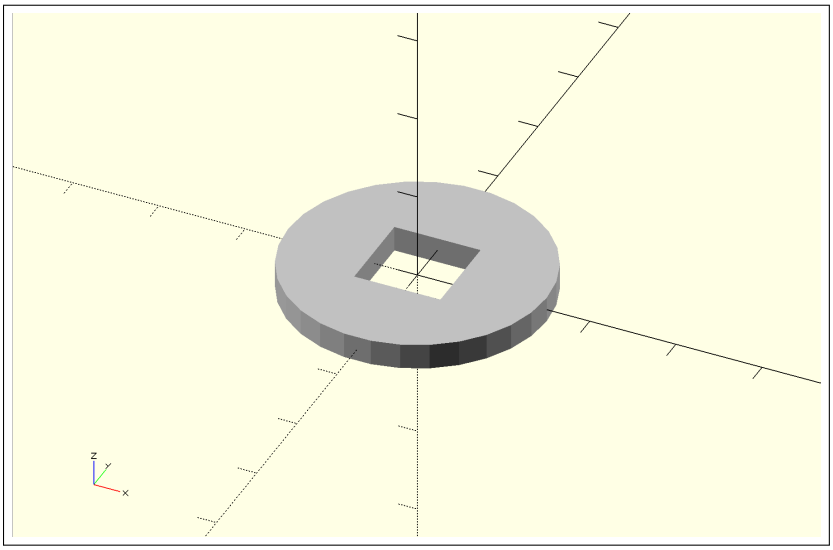
2.5



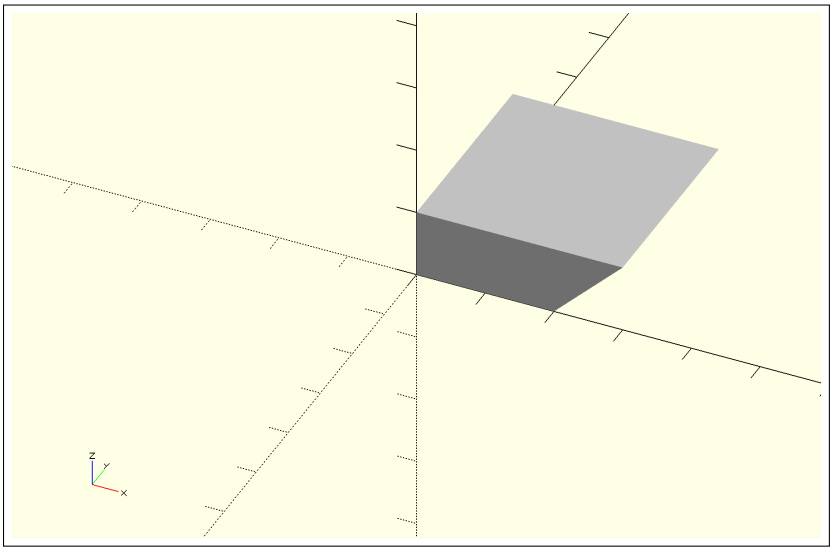
2.6



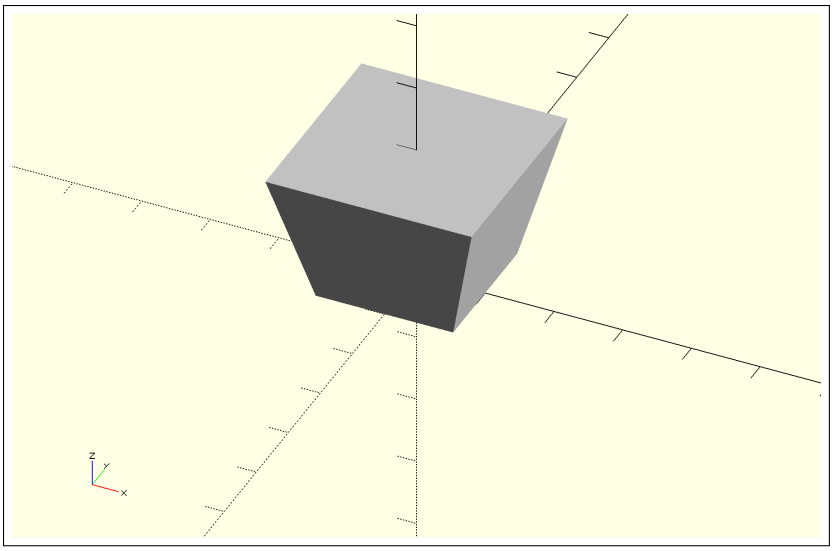
```
difference() {
    cylinder(d=30,h=3);
    cylinder(d=10,h=3);
}
```



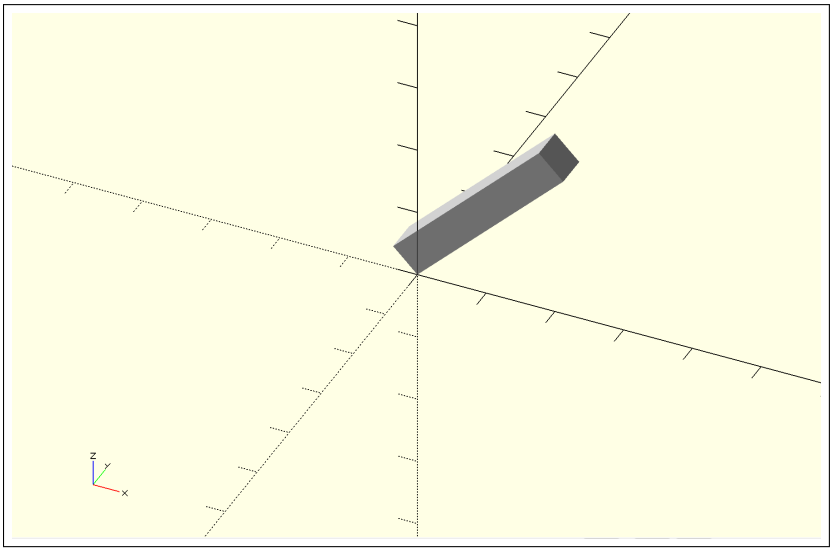
```
difference() {
    cylinder(d=30,h=3);
    cube([10,10,3],center=true);
}
```



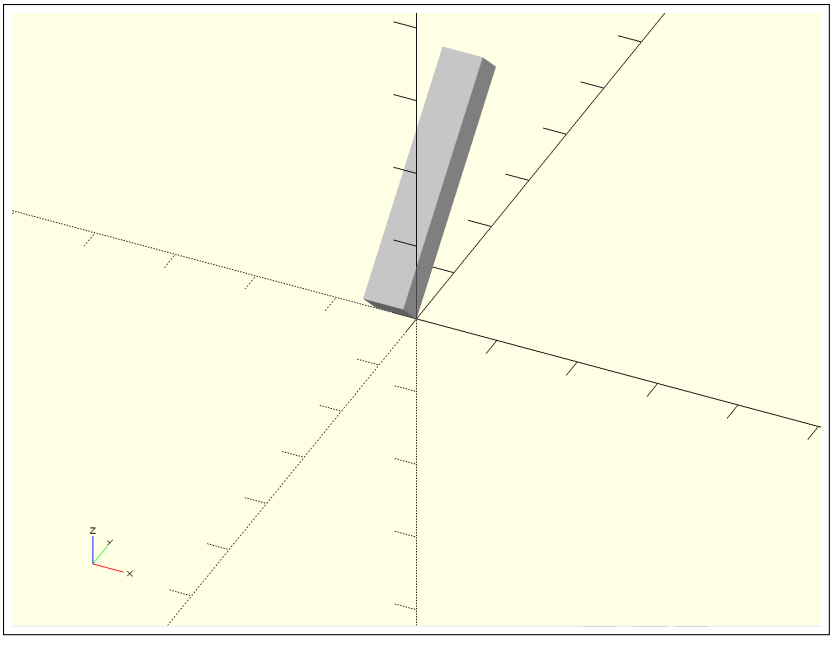
```
linear_extrude(height=10,
scale=1.5)
square(20);
```



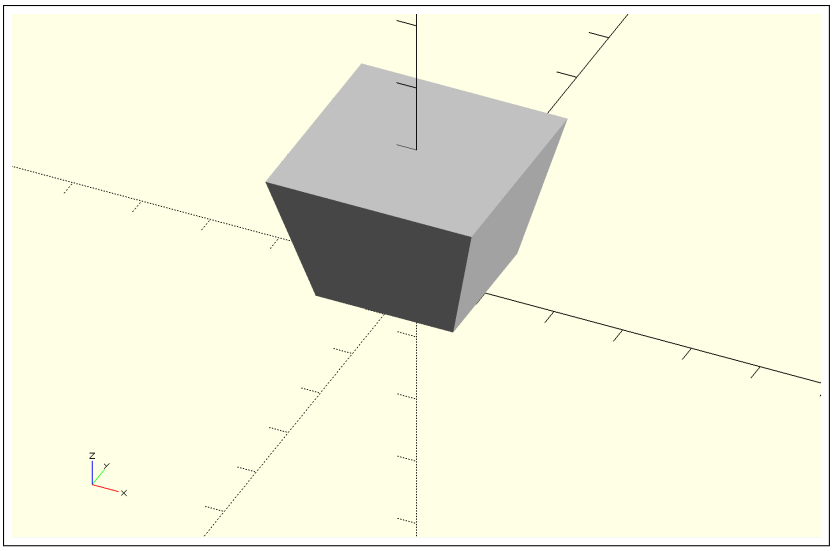
2.7



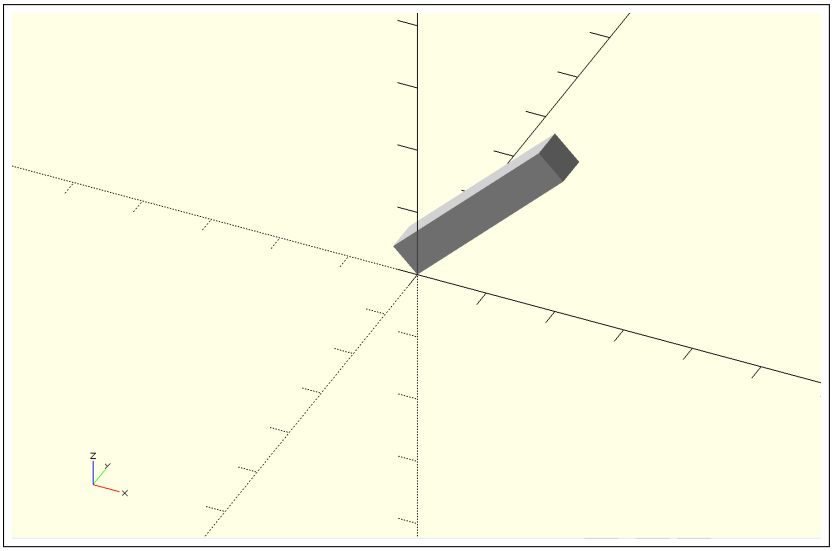
2.8



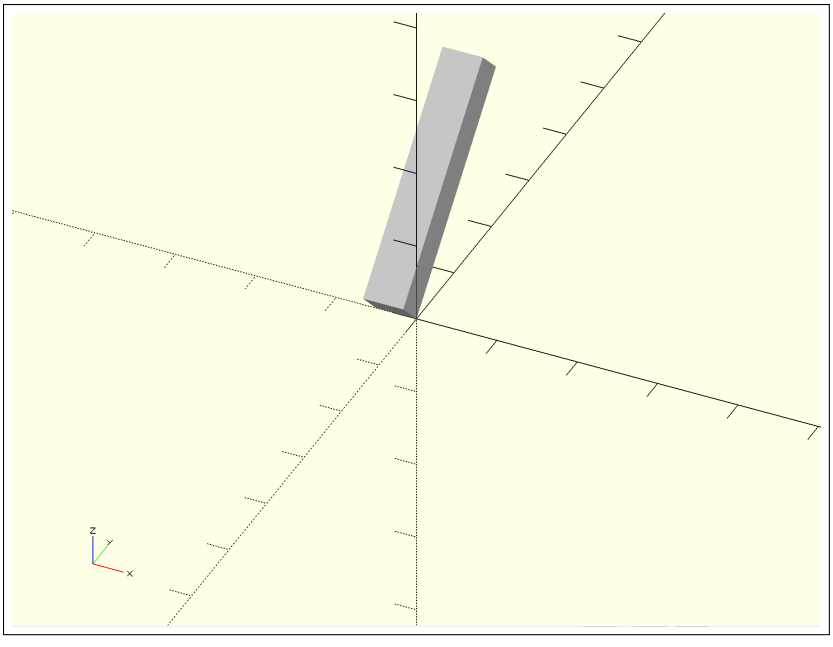
2.9



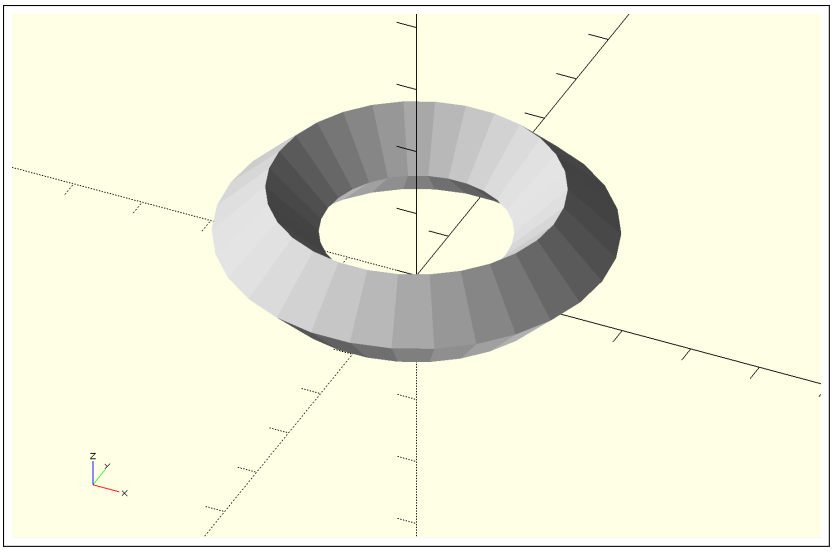
```
linear_extrude(height=20,
scale=1.5)
square(20,center=true);
```



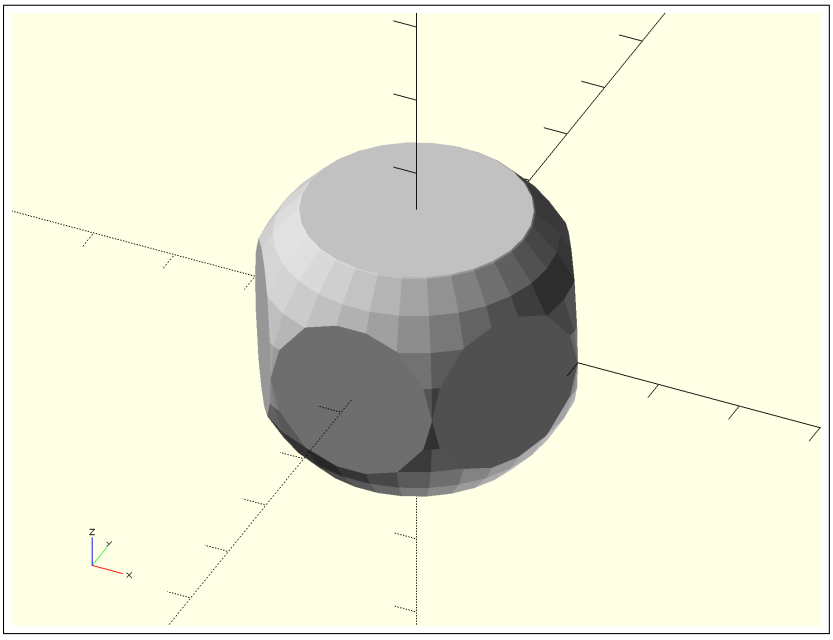
```
rotate([0,-45,0])
cube([30,5,5]);
```



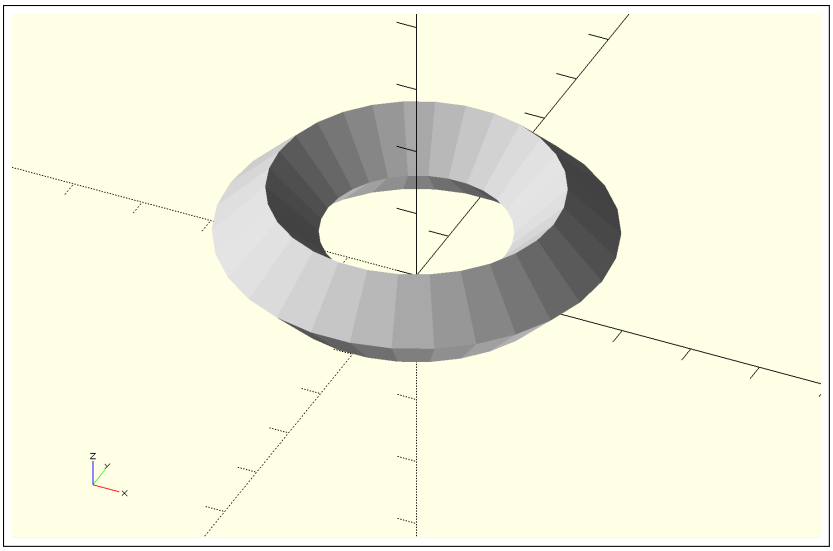
```
rotate([0,-45,90])
cube([30,5,5]);
```



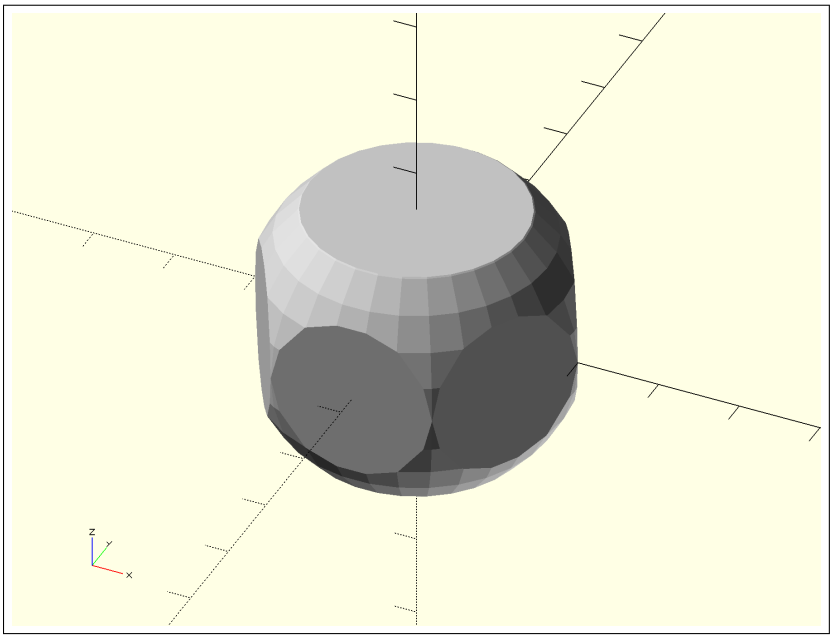
2.10



2.11



```
rotate_extrude()
translate([20,0,0])
rotate([0,0,45])
square(10);
```



```
intersection() {
    cylinder(h=30,d=40,$fn=6,
    center=true);
    sphere(20);
}
```

Parametrisch ontwerpen

Onderwerpen

Parametrisch ontwerpen We gaan aan de slag met variabelen. Maak je model parametrisch.

Nieuwe mogelijkheden Laat OpenSCAD het zware werk doen: `for` en `iteration`, `minkowski`, `hull`.

Comments Zorg dat je code leesbaar is.

Project Ga terug naar je ontwerp van de tol. Maak deze parametrisch.
Uitdaging: ontwerp een tandwiel met behulp van de iteration functie en maak deze parametrisch.

Variabelen

Stel je wilt een dekseltje maken voor een rond doosje. Je meet het doosje op, maar je weet niet of de deksel die je gaat printen met die maat precies op het doosje zal passen. Misschien zet het filament wel wat uit, of krimpt het wat. Eigenlijk wil je dus terug kunnen gaan naar je ontwerp en het gemakkelijk een millimeter groter of kleiner maken zonder dat je in al je functies en operaties je waarden aan hoeft te passen. Variabelen kunnen je daarbij helpen.

Het is good practice om je variabelen bovenaan je document te zetten. Zo heb je alles handig in één oogopslag op dezelfde plek. Het is ook belangrijk dat je de variabele een waarde geeft voordat je deze gaat gebruiken. Nog een reden om alle variabelen bovenaan je document te zetten.

We gaan aan het doosje werken. We beginnen met de variabelen die we nodig gaan hebben, allereerst de diameter van ons doosje en de waarde die daarbij hoort, in ons geval is dat 134mm. Let erop dat je de waarde afsluit met een puntkomma zodat OpenSCAD kan zien dat de waarde daar eindigt. We geven ook de dikte van de rand op als variabele. Hier willen we graag dat onze rand 3mm dik wordt. Nadat we de variabelen hebben opgegeven, schrijven we de code voor de deksel. De deksel bestaat uit een grote, dikke cilinder waar een kleinere, dunnere cirkel vanaf is gehaald.

```
diameter_deksel = 134;                    dit is de diameter in millimeters
dikte_rand = 3;                           dit is de dikte in millimeters van de
                                          rand

difference() {
    cylinder(h=10,d=diameter_deksel+dikte_rand*2);
    translate([0,0,3])
    cylinder(h=7,d=diameter_deksel);
}
```

Nu hebben we een deksel met diameter $134 + 3 \times 2 = 140$mm. Als we willen dat de rand niet 3mm maar 5mm dik is, kunnen we de waarde van de dikte van de rand gemakkelijk bovenaan in ons document aanpassen. Hetzelfde geldt voor het veranderen van de diameter van het doosje. We hoeven dan niet door onze code heen te struinen om alles aan te passen. Natuurlijk zou dat hier niet zo'n probleem zijn, maar je kunt je voorstellen dat het wel een probleem wordt op het moment dat je code in de honderden regels loopt.

Variabelen zijn dus enorm handig als je wilt dat een object zich aanpast aan verschillende situaties. Hieronder zie je een voorbeeld waarin de cirkel altijd het midden van de rechthoek uitsnijdt zonder `center=true` te gebruiken voor zowel rechthoek als cirkel.

```
breedte = 20;
hoogte = 10;
diameter_cirkel = 5;

difference() {
    square([breedte,hoogte]);
    translate([breedte/2,hoogte/2,0])
    circle(diameter_cirkel);
}
```

Nu kun je alle variabelen aanpassen en zal de cirkel altijd in het midden van de rechthoek blijven. Dit komt doordat de verplaatsing van de cirkel gedefinieerd is als de helft van de hoogte en breedte van de rechthoek.

OPDRACHT Ga eens terug naar eerdere lessen en maak een aantal van je ontwerpen parametrisch.

For en Iteration

IN OPENSCAD kun je ook een beetje programmeren. Je kunt OpenSCAD keuzes laten maken op basis van bepaalde input, of je laat het bepaalde handelingen herhalen. Zo doet het programma het zware werk voor je.

In dit boekje behandelen we niet alle programmeermogelijkheden van OpenSCAD maar beperken we ons tot een paar basisopties die je gemakkelijk in je eigen werk kunt gebruiken. We kijken hier eerst naar for-statements die gebruik maken van `i`. De `i` staat voor *index*. Hiermee gaat OpenSCAD tellen en voor (`for`) elke stap een bepaalde handeling uitvoeren.

We kunnen een lijstje maken met `i`. In de cheatsheet zie je dat `i` de volgende notatie kan gebruiken: `[start:end]`. We krijgen dus de volgende notatie: `for (i = [start:end])`. Omdat `for` meerdere elementen kan bevatten, gebruiken we accolades. Dit is overigens niet strikt noodzakelijk maar maakt je code wel een stuk overzichtelijker.

OpenSCAD heeft een iets andere notatie voor `for` dan de for-loop in bijvoorbeeld Java of C++, maar functioneel zijn ze hetzelfde.

We gaan een paar toepassingen bekijken. Stel, je wilt 5 blokjes elke keer 20mm opschuiven in de x-richting door middel van `translate([20,0,0])`. Je zou dit allemaal handmatig kunnen doen, als volgt:

```
cube(10);                                   blokje 1
translate ([20,0,0])
cube(10);                                   blokje 2
translate ([40,0,0])
cube(10);                                   blokje 3
translate ([60,0,0])
```

```
cube(10);                                   blokje 4
translate ([80,0,0])
cube(10);                                   blokje 5
```

Voor vijf blokjes is dit nog wel te doen, maar wat als je er opeens toch 10 nodig blijkt te hebben, of 20? Dan wil je OpenSCAD dat werk laten doen. We laten OpenSCAD eerst een lijst maken. We hebben gezien dat we ons begin en ons eind in moeten vullen, dus proberen we hier 1 en 5 in te vullen omdat we 5 blokjes willen hebben. Daarna laten we OpenSCAD voor elke iteratie (1, 2, 3, 4, 5) een bepaalde handeling uitvoeren. In dit geval willen we dat elk blokje 20mm opschuift in de x-richting, vandaar dat we `translate([20,0,0])` gebruiken. Om te zorgen dat OpenSCAD dit uitvoert per blokje, voegen we wat toe aan onze `translate`-functie. Blokje 2 moet bijvoorbeeld 20mm opschuiven, blokje 3 moet 2x20=40mm opschuiven, enzovoorts. We willen het nummer van ons blokje dus vermenigvuldigen met de verplaatsing in de x-richting. Daarom passen we de functie als volgt aan: `translate([i*20,0,0])`. Nu hebben we alles om onze code te schrijven:

```
for (i = [1:5]) {                           i geeft aan hoe vaak OpenSCAD de
    translate([i*20,0,0])                   handeling uit moet voeren.
    cube(10);
}
```

We zien nu dat we vijf blokjes hebben maar dat het eerste blokje al 20mm is opgeschoven ten opzichte van de oorsprong! Dit komt omdat we zijn begonnen te tellen bij 1. Bij `i=1` wordt het blokje namelijk al 20mm verplaatst. Als we exact ons eerdere voorbeeld na willen maken, zullen we onze lijst anders moeten definiëren en bij 0 moeten beginnen. Op deze manier maakt OpenSCAD wel het eerste blokje, maar zal dit niet verplaatst worden: $0 \times 20 = 0$, wat betekent dat OpenSCAD `translate([0*20,0,0])` uitvoert voor het eerste blokje. Als je dit aanpast in je code zul je merken dat je opeens zes blokjes krijgt; je moet dus ook de bovengrens meeveranderen. Dit komt doordat de lijst `[0:5]` zes iteraties bevat (0, 1, 2, ,3, 4, 5). Dit betekent dat we een lijst moeten maken van `[0:(5-1)]` of `[0:4]`. Nu we dit weten, zien we meteen dat we deze lijst parametrisch kunnen maken.

```
aantal_blokjes = 40;

for (i = [0:aantal_blokjes-1]) {            Hier staat onze lijst. We gebruiken
    translate([i*20,0,0])                   aantal_blokjes-1 omdat we hebben
    cube(10);                               gezien dat de for-statement bij 0 begint
}                                           te tellen.
```

Deze functie werkt perfect en nu hoeven we niet handmatig 40 blokjes te maken en te verplaatsen.

Je kunt iteration ook gebruiken voor andere transformaties, zoals `rotate`, `mirror`, `color`, wat je maar wilt! Hier volgt een voorbeeld van `rotate`. We roteren hier elke balk 20° verder om de y-as.

```
for (i = [0:8]) {
    rotate([0,i*20,0])
    cube([2,2,40],center=true);
}
```

Nu hebben we een ster gemaakt. Pas het aantal iteraties maar eens aan van 9 naar 4 en kijk wat er gebeurt.

We kunnen een object ook steeds groter maken door `i` in de maten van het object te plaatsen.

```
for (i = [0:5]) {
    translate([i*20,0,0])
    cube(10+i);
}
```

Nu wordt elk opvolgend blokje 1mm groter.

Nu worden de blokjes steeds groter!

OPDRACHT Ga zelf aan de slag met iteration. Probeer ook eens `translate` en `rotate` daarin te combineren.

Minkowski

Een elegante manier om voorwerpen rondingen mee te geven, is `minkowski`. Je kunt `minkowski` gebruiken bij 2D- en 3D-objecten. Deze functie laat een bepaald object de omtrek van een ander object volgen. De combinatie van die twee wordt het resultaat. Net als `difference` kan `minkowski` meerdere objecten bevatten en heeft het dus accolades nodig.

We beginnen met een aantal voorbeelden die gebruik maken van cirkels en bollen om zo de scherpe randjes van onze objecten af te halen. In andere CAD-programma's wordt deze functionaliteit ook wel *fillet* genoemd. Een voorbeeld in 2D:

```
square([60,20]);
circle(5);
```

Dit zijn gewoon een rechthoek en een cirkel. Maar met `minkowski` kun je de cirkel de omtrek van de rechthoek laten volgen, als volgt:

```
minkowski() {
    square([60,20]);
    circle(5);
}
```

Je ziet dat de rechthoek nu afgerond is bij de hoeken. Maar omdat we nu een combinatie gebruiken van de rechthoek en cirkel zul je ook merken dat je object iets groter is geworden. In dit geval 5mm. We kunnen hetzelfde doen met 3D-objecten:

```
minkowski() {
    cube(30);
    sphere(5);
}
```

Ook in dit voorbeeld is ons uiteindelijke object iets groter geworden dan de oorspronkelijke kubus waar we mee begonnen zijn. De uiteindelijke kubus heeft een breedte, diepte en hoogte van 35mm in plaats van de originele 30mm. Om te zorgen dat je een kubus krijgt met een breedte, diepte en hoogte van 30mm kun je `cube(25);` gebruiken.

`Minkowski` is niet alleen te gebruiken voor simpele objecten maar ook voor gecombineerde objecten of modules. We gaan terug naar onze L-vorm maar deze keer willen we dat een cirkel alle randen volgt. We maken eerst onze module:

```
module Lvorm() {
    square([30,10]);
    square([10,50]);
}
```

Daarna kunnen we er simpel onze `minkowski`-transformatie op loslaten:

```
module Lvorm() {
    square([30,10]);
    square([10,50]);
}
minkowski() {
    Lvorm();
    circle(5);
}
```

Let er wel op dat `minkowski` een zware transformatie is en dat OpenSCAD dus lang zal moeten rekenen als het op ingewikkeldere vormen toegepast wordt. Wachttijden van 10 minuten zijn niet ongewoon. Je kunt dit toch werkbaar houden door het aantal fragments in de codeerfase laag te houden (`$fn=25 of $fn=50`). Als je tevreden bent kun je voor het uiteindelijke renderen de resolutie omhoog gooien.

Natuurlijk is `minkowski` ook te gebruiken in combinatie met andere vormen en objecten. Zo kun je bijvoorbeeld twee vierkanten gebruiken:

```
minkowski() {
    square(10);
```

```
    square(5,center=true);
}
```

Je ziet nu dat het resulterende vierkant 15mm bij 15mm groot is. Dit heeft natuurlijk niet zoveel zin, maar je krijg interessante resultaten als je het kleine vierkant 45° draait ten opzichte van de z-as:

```
minkowski() {
    square(10);
    rotate([0,0,45])
    square(5,center=true);
}
```

Je ziet nu een groter vierkant waarbij de hoeken zijn afgevlakt. Interessant is hier om op te merken dat de afmetingen van het vierkant nu $10 + \sqrt{50}$ is omdat de diagonaal van het kleine vierkantje bij de breedte en hoogte van het vierkant wordt opgeteld. Dit is een andere manier om je scherpe hoekjes wat af te vlakken, ook wel *chamfer* genoemd.

OPDRACHT Ga terug naar de objecten die je hebt gemaakt in hoofdstuk 2. Kun je daar `minkowski` op toepassen?

Hull

Waar `minkowski` je wat mogelijkheden geeft om meer organische vormen te ontwerpen, gaat `hull` hierin nog een stapje verder. `hull` is de perfecte manier om een soepele overgang tussen twee vormen te verkrijgen. In principe pakt `hull` twee (of meer) vormen en verbindt deze door de kortste weg tussen die vormen op te vullen. Eigenlijk gaat de ene vorm dus langzaam over in de andere. `hull` kan meerdere objecten bevatten, dus is het netjes om met accolades te werken.

Net als minkowski, is `hull` een vrij zware bewerking, dus enig geduld zul je af en toe wel moeten hebben. Ook hier is het weer een goed idee om bij het ontwerpen gebruik te maken van een lage resolutie (onder de $fn = 50;) en pas voor het renderen de resolutie te nemen die je daadwerkelijk nodig hebt voor het eindproduct.

We beginnen met twee cirkels die enige afstand van elkaar hebben.

```
    circle(5);
    translate([60,0,0])
    circle(5);
```

Deze cirkels verbinden we nu door middel van `hull`.

```
hull() {
    circle(5);
    translate([60,0,0])
    circle(5);
}
```

Je ziet dat de kortste weg tussen de cirkels als het ware is opgevuld. Hier heb je twee vormen met elkaar verbonden, maar je kunt op dezelfde manier ook meerdere vormen met elkaar verbinden. In plaats van twee cirkels gaan we nu vier cirkels met elkaar verbinden.

Zojuist heb je in de sectie over `minkowski` gezien hoe je een afgeronde rechthoek kunt maken:

```
minkowski() {
    square([60,20]);
    circle(5);
}
```

Nu kunnen we deze vorm ook maken door vier cirkels met elkaar te verbinden door middel van `hull`.

```
hull() {
    circle(5);
    translate([60,0,0])
    circle(5);
    translate([60,20,0])
    circle(5);
    translate([0,20,0])
    circle(5);
}
```

Dit is natuurlijk een vrij omslachtige manier van werken. Het is dus goed om na te denken over wat je wilt bereiken en welke methode de beste is om je doel te bereiken. In dit voorbeeld is `minkowski` duidelijk de beste manier om een afgeronde rechthoek te maken.

Nu we hebben gezien hoe `hull` werkt in 2D, gaan we kijken naar de manier waarop je `hull` in kunt zetten om een 3D-object te creëren.

We beginnen eerst met een cilinder en daarboven een in de oorsprong gecentreerde balk:

```
cylinder(r=10,h=1);
translate([0,0,20])
cube([10,10,1],center=true);
```

We laten deze twee vormen nu in elkaar overlopen door `hull`:

```
hull() {
    cylinder(r=10,h=1);
    translate([0,0,20])
    cube([10,10,1],center=true);
}
```

Zoals je kunt zien is dit een hele elegante manier om twee objecten met elkaar te verbinden, in dit geval een cilinder en een balk. Let erop dat `hull` de cilinder en balk aan elkaar maakt door niet de onderzijde maar de bovenzijde van de balk te verbinden met de cilinder.

Je kunt beter zien hoe dit werkt door de balk vlak boven de cylinder te plaatsen, als volgt:

```
hull() {
    cylinder(r=10,h=1);
    translate([0,0,1.5])
    cube([10,10,1],center=true);
}
```

De balk verdwijnt als het ware in het nieuwe object.

Je kunt op deze manier talloze objecten aan elkaar bevestigen, maar `hull` zal altijd alle ruimte tussen de objecten opvullen. Het is dus niet geschikt om een 'deuk' of inkeping in een oppervlak te creëren. Stel, we nemen vier bollen en die verbinden we door middel van `hull`.

```
hull() {
    sphere(10);
    translate([30,15,40])
    sphere(10);
    translate([60,0,0])
    sphere(10);
    translate([30,40,0])
    sphere(10);
}
```

Je zou deze piramide ook kunnen creëren door een normale piramide te maken en deze door middel van `minkowski` afgeronde punten te geven.

Je ziet nu een soort van piramide. Als je nu een vijfde bol in de code van de piramide plaatst, zal dat niets uitmaken voor de uiteindelijke vorm. We zien dit hieronder.

```
hull() {
    sphere(10);
    translate([30,15,40])
    sphere(10);
    translate([60,0,0])
    sphere(10);
    translate([30,40,0])
    sphere(10);
    translate([30,15,15])
    sphere(10);
}
```

bol nummer vijf

Zoals je ziet is er niets aan de vorm van de piramide veranderd. Elke vorm die je met behulp van `hull` maakt zal dus altijd de omvang van de buitenste objecten aannemen. Elk object dat zich binnen de grens van die buitenste objecten bevindt, zal simpelweg 'opgeslokt' worden.

Comments

Als je je ontwerpen na een tijdje ook nog goed wilt kunnen begrijpen, is het wijs om comments in je code te zetten die per onderdeel

beschrijven wat je code doet. Ook als je je ontwerpen wilt delen met anderen, kunnen comments hen helpen. En natuurlijk kun je door middel van de comments-functionaliteit ook delen van de functionele code 'wegcommenten' zodat deze, al dan niet tijdelijk, niet gelezen wordt door OpenSCAD. Zo kun je bekijken wat de rest van je code doet, wat handig is om te troubleshooten.

We kijken hieronder kort naar twee soorten comments: *line comments* en *multi-line comments*. Multi-line comments stellen je in staat om hele blokken tekst als commentaar in je code plaatsen. Je maakt deze comments door een forward slash (/) en een asterisk (*) te combineren. Om een comment te openen gebruik je `/*` en om deze te sluiten gebruik je de tegenovergestelde combinatie `*/`. Je kunt meerdere *opening comments* sluiten met maar één *closing comment*. Line comments werken maar voor één regel en maak je door middel van twee forward slashes: `//`. Je hoeft deze vorm niet af te sluiten omdat OpenSCAD weet dat de comment nooit langer is dan één regel.

Als je een stukje code geschreven hebt waar je niet helemaal tevreden mee bent, wil je het soms weghalen. Het kan dan slimmer zijn om het even tussen multi-line comments te zetten. Zo kun je kijken wat de rest van de code doet, maar hoef je niet meteen je nieuwe code te verwijderen en daarna weer opnieuw te typen. Line comments zijn erg handig als je modules snel aan en uit wilt kunnen zetten. Hieronder een kort voorbeeld hiervan:

```
module prisma() {
    cylinder(h=5,d=10,$fn=3);
}
//prisma();
```

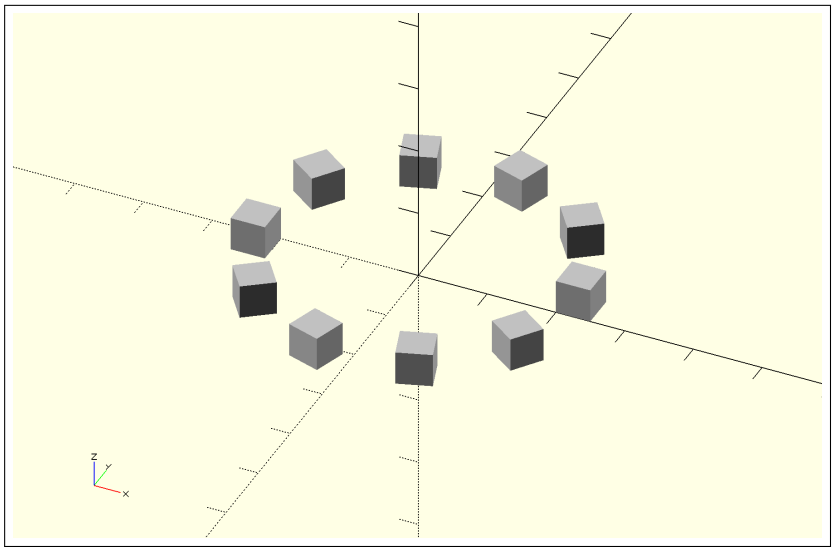
Je ziet nog niets aangezien het aanroepen van de module nu onzichtbaar is. Als je de `//` weghaalt zal OpenSCAD de prisma wel laten zien.

OPDRACHT Ga eens terug naar een eerder project en schrijf in comments bij de verschillende onderdelen wat ze doen. Kon je alles nog gemakkelijk terugvinden zonder comments?

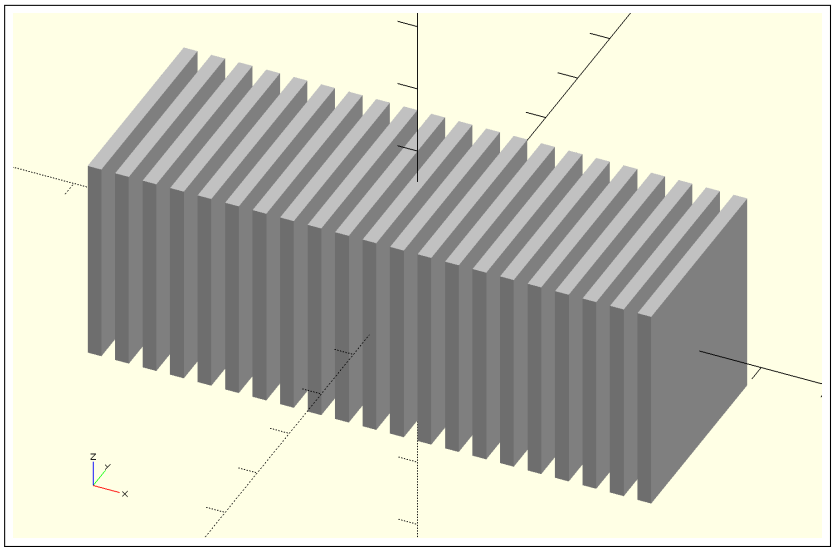
PROJECT Ga terug naar je ontwerp van de tol. Maak deze parametrisch.
Uitdaging: ontwerp een tandwiel met behulp van de iteration-functie en maak dit tandwiel parametrisch. Hint: denk terug aan het ontwerp van de ster in deze les.

Opdrachten hoofdstuk 3

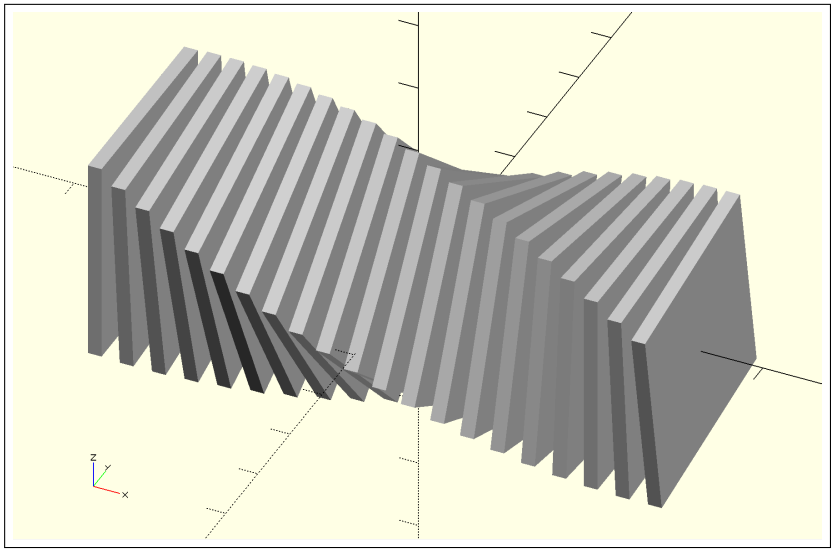
3.1

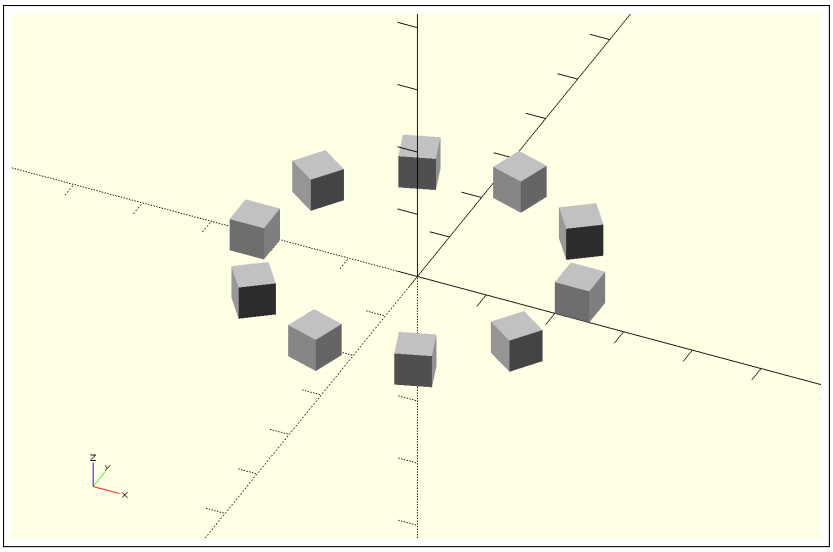


3.2

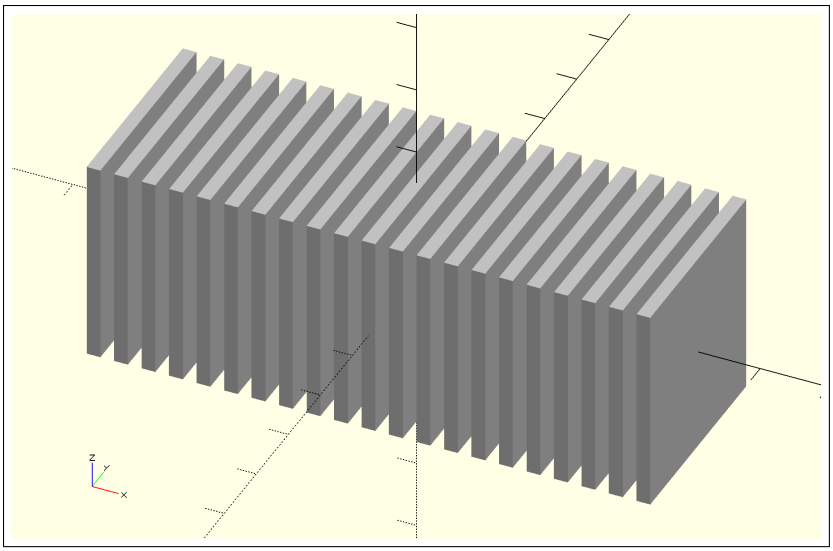


3.3

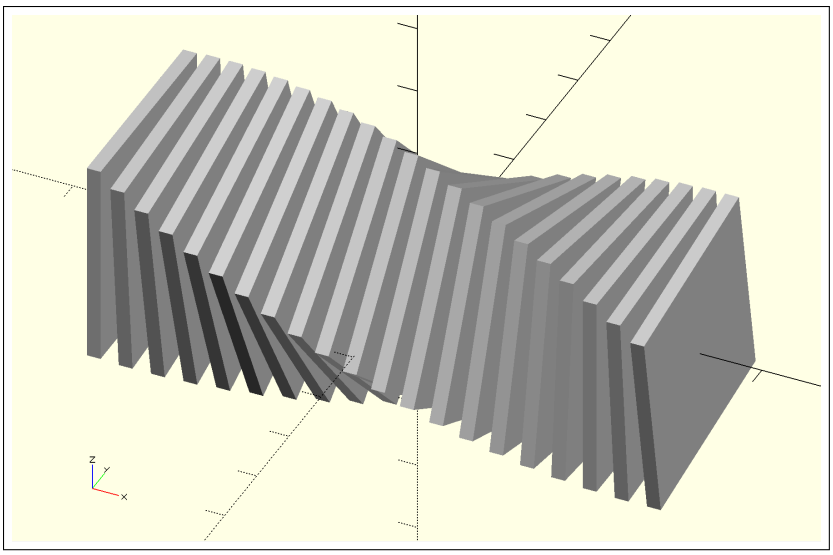




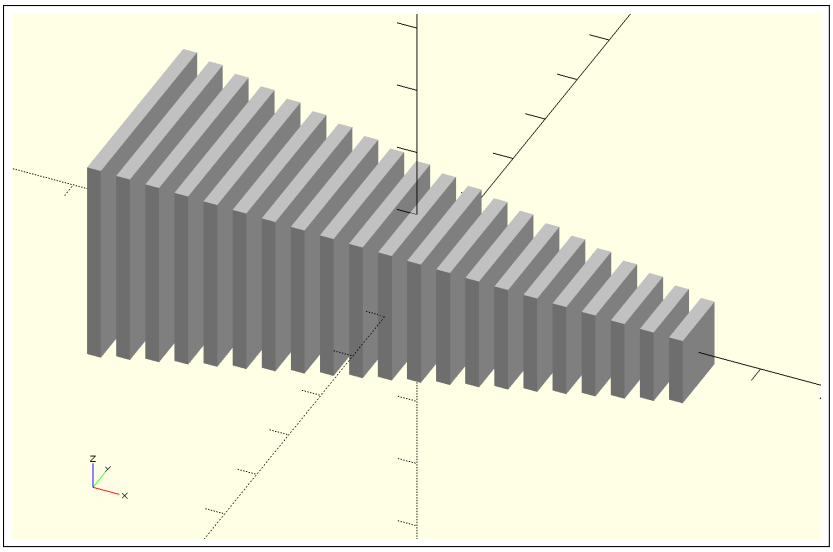
```
for (i = [0:10])
    {
    rotate([0,0,i*36])
    translate([20,0,0])
    cube(5);
    }
```



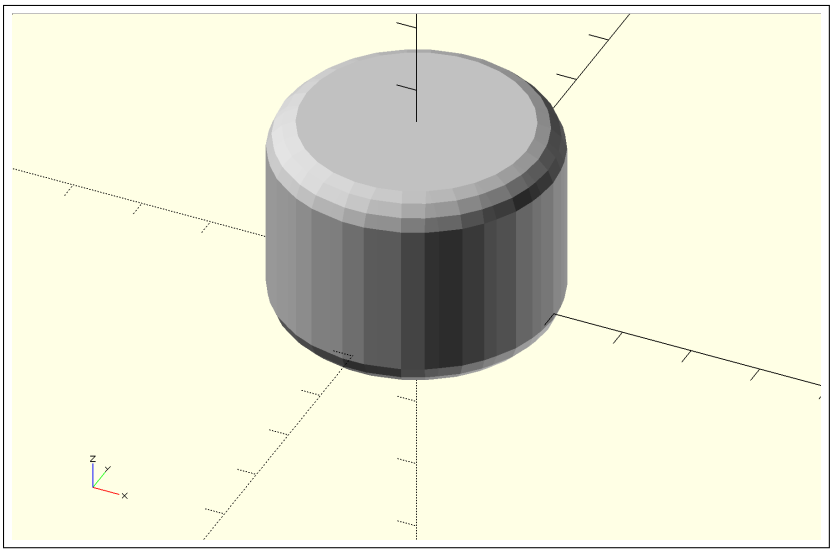
```
for (i = [0:20])
    {
    translate([i*4,0,0])
    translate([-40,0,0])
    cube([2,30,30],center=true);
    }
```



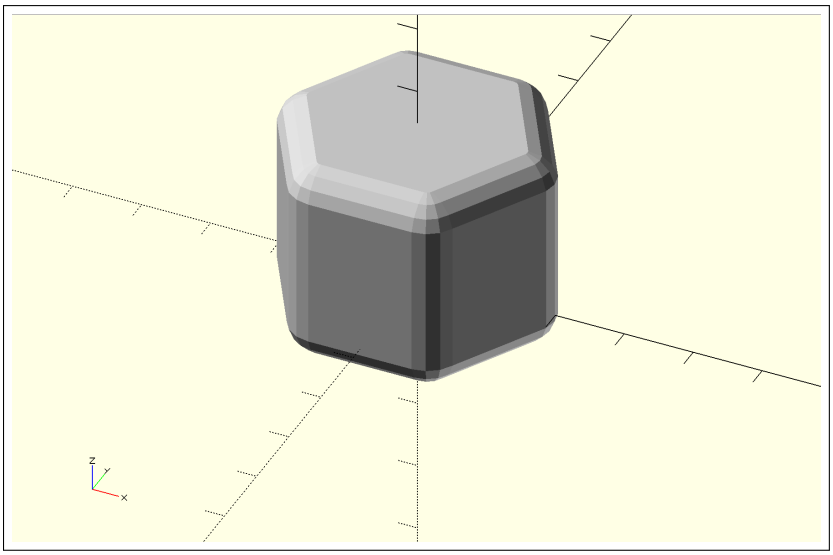
```
for (i = [0:20])
    {
    translate([i*4,0,0])
    rotate([i*5,0,0])
    translate([-40,0,0])
    cube([2,30,30],center=true);
    }
```



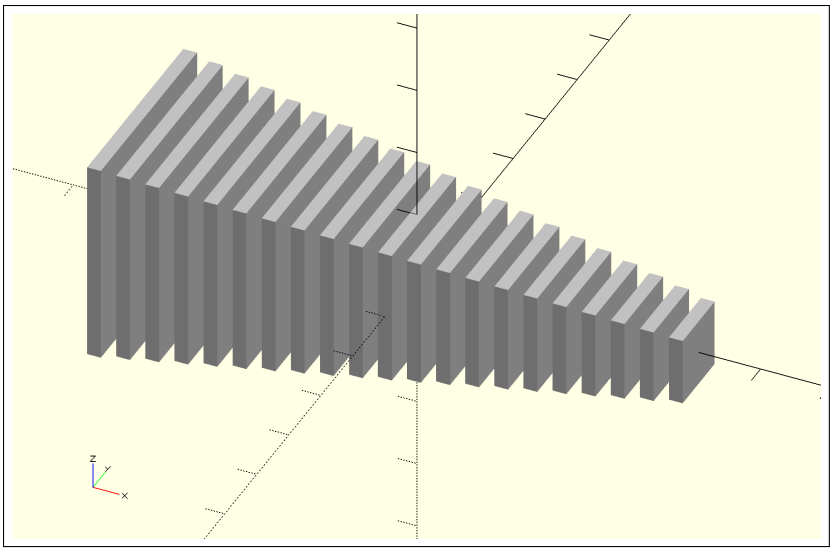
3.4



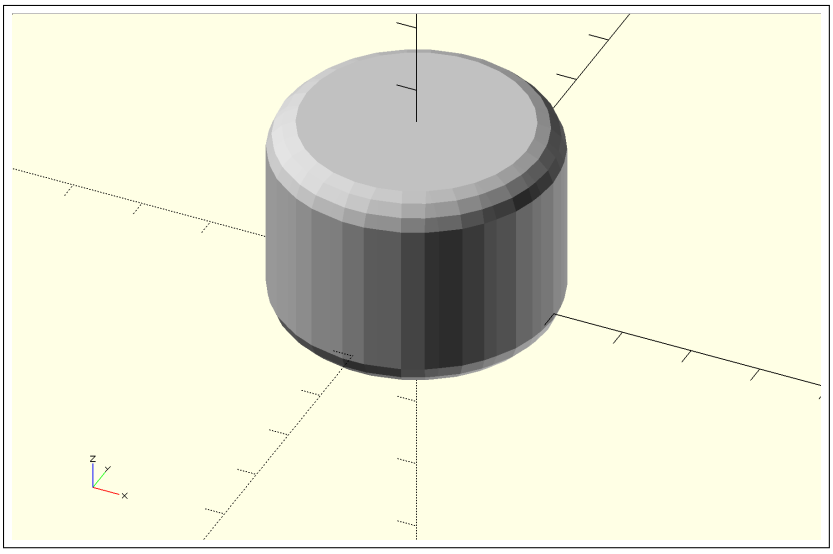
3.5



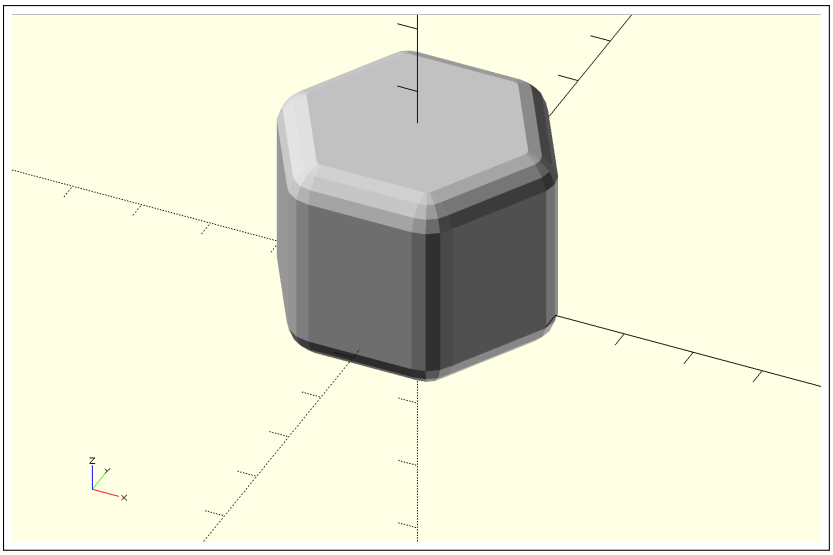
3.6



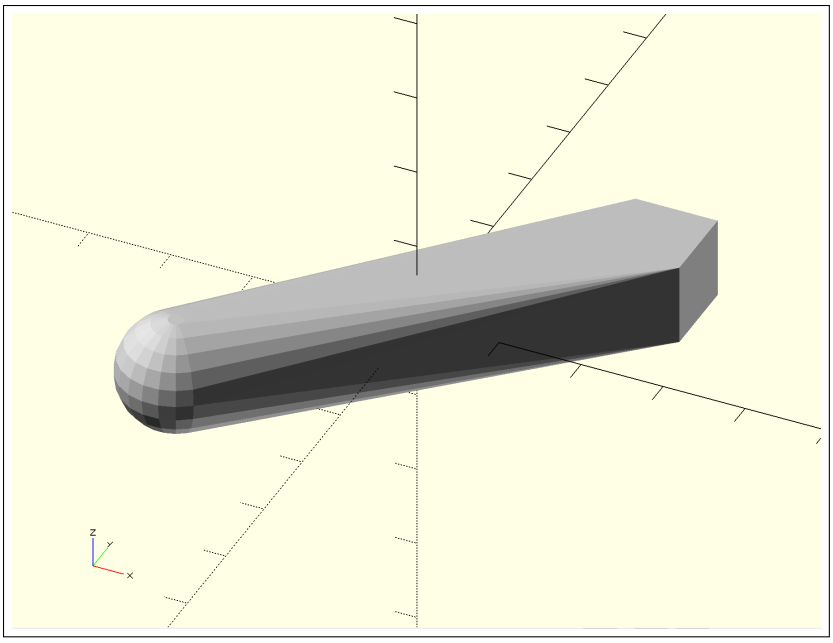
```
for (i = [0:20])
    {
    translate([i*4,0,0])
    translate([-40,0,0])
    cube([2,30-i,30-i],center=true);
    }
```



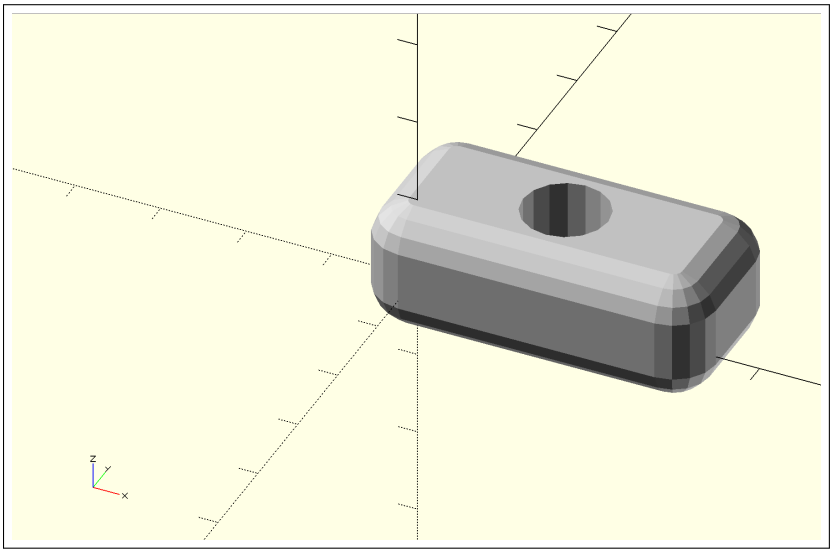
```
minkowski() {
    cylinder(h=20,d=30);
    sphere(5);
}
```



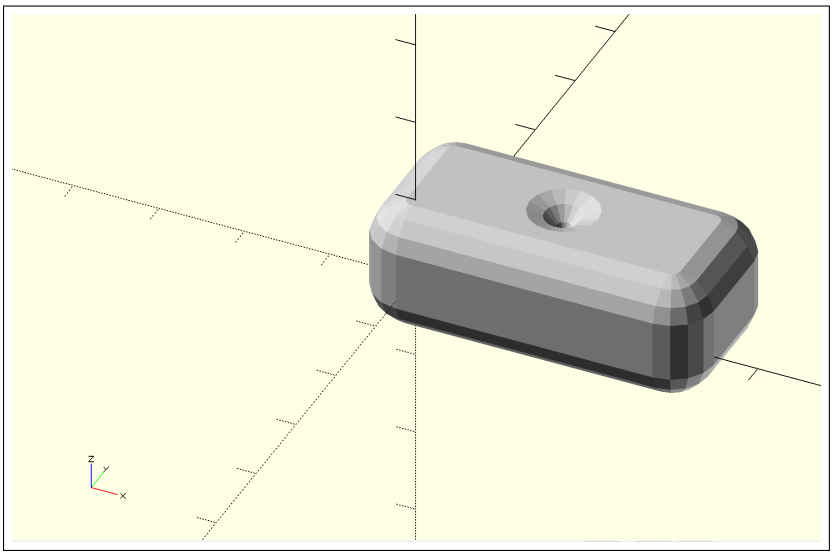
```
minkowski() {
    cylinder(h=20,d=30,$fn=6);
    sphere(5);
}
```



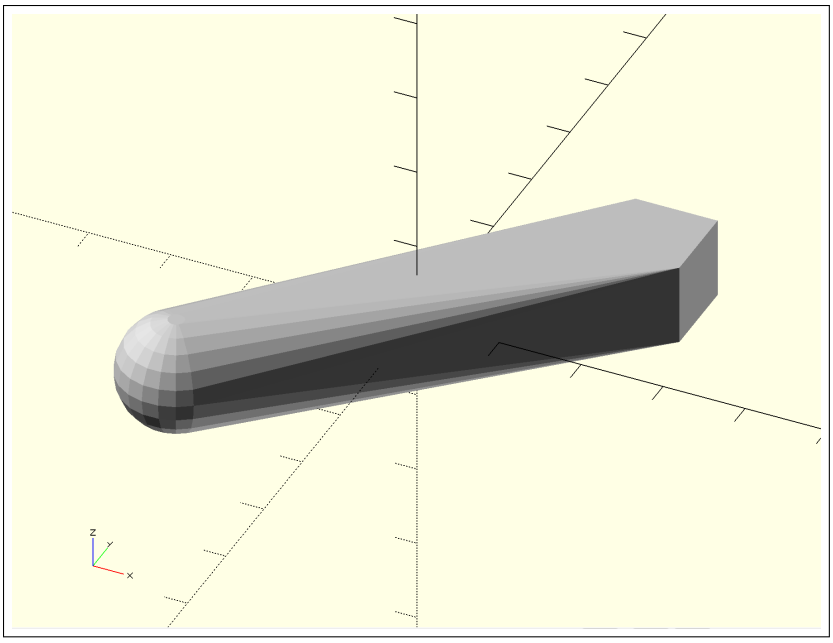
3.7



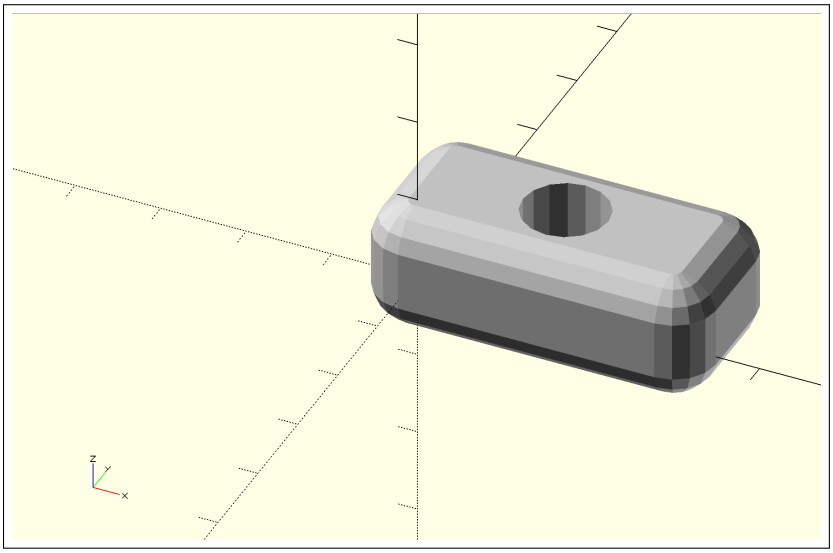
3.8



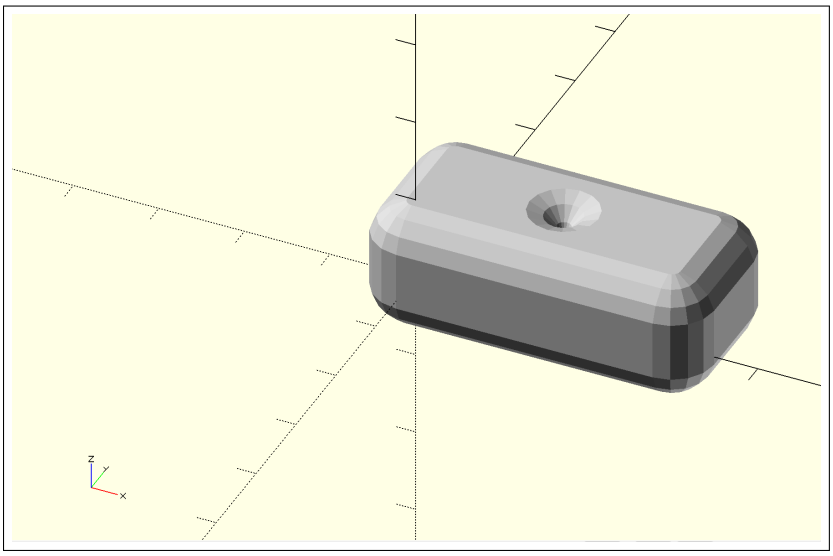
3.9



```
hull (){
    translate([-20,-20,0])
    sphere(7);
    translate([20,20,0])
    cube(10,center=true);
}
```



```
difference() {
    minkowski() {
        cube([30,10,5]);
        sphere(5);
    }
translate([15,5,0])
cylinder(h=10,d=10);
}
```



```
minkowski() {
    difference() {
        cube([30,10,5]);
        translate([15,5,0])
        cylinder(h=10,d=10);
    }
sphere(5);
}
```

In de praktijk

Onderwerpen

Nieuwe mogelijkheden Maak je ontwerp veelzijdiger: `project` en `if`-statements.

Wegen naar Rome Hacks die je leven gemakkelijker maken. Allerlei manieren om een kegel te maken, een cilinder, een kubus en een hexagoon.

Analyse Hoe kunnen we alledaagse vormen namaken?

Project Kijk rond in je omgeving. Is er iets stuk? Wil je iets verbeteren? Weet je een hack? Ontwerp dit in OpenSCAD, maar dan wel parametrisch!

Nieuwe mogelijkheden

Projection

Soms wil je een projectie op het xy-vlak maken van een 3D-object. Dit kun je doen omdat je een balletje door een gat in een plaat triplex wilt laten vallen, bijvoorbeeld. Dit gat wil je uitsnijden in de lasersnijder. Daarvoor zul je dus een 2D-tekening moeten maken. Je hebt al veel gesleuteld aan dat balletje, en nu wil je wel weten hoe groot dat gat dan precies moet zijn. We hebben al eerder gezien dat je niet een 3D-object van een 2D-object af kan trekken. Om dit probleem te omzeilen kan `projection` je helpen. `projection` werpt eigenlijk een schaduw op het xy-vlak. Je kunt het met en zonder accolades gebruiken. Hieronder zie je hoe `projection` werkt. We maken eerst een projectie van een bol.

```
projection()
translate([0,0,20])
sphere(10);
```

Je maakt met projection eigenlijk een nieuw 2D-object, in dit geval een cirkel. Daarom zien *Preview* en *Render* er nu anders uit. Probeer de hoogte eens te veranderen van de bol. Verandert de vorm van je `projection` nu ook?

Nu gaan we de cirkel als projectie uit onze plaat snijden.

```
difference() {
    square(50, center=true);
    projection()
    translate([0,0,20])
    sphere(10);
}
```

Zoals gezegd kun je 2D en 3D niet mengen. We kunnen onze projectie dus uit een `square` snijden, maar niet uit een `cube`.

We kunnen ook een dwarsdoorsnede maken van ons object door `projection(cut=true)` te gebruiken. We snijden het object dan doormidden op het xy-vlak. We nemen het voorbeeld van de spiraal uit hoofdstuk 2 maar nu gecentreerd in de oorsprong.

```
module twist() {
    linear_extrude(height=60,twist=90,slices=100,center=true)
    square([20,20],center=true);
}
projection(cut=true)
translate([0,0,0])
twist();
```

Je ziet dat hier een `translate`-functie is opgenomen die (nog) niets doet. Als je je object op een andere hoogte doormidden wilt snijden,

kun je dit niet bewerkstelligen door in `projection` iets aan te passen. Je zult het hele object moeten verplaatsen in de richting van de z-as. Verander de z-waarde in de `translate`-functie en kijk wat er gebeurt met je *Render*!

OPDRACHT Ga zelf experimenteren met `projection`. Ga ook eens terug naar eerdere ontwerpen en probeer verschillende projecties te krijgen met `projection(cut=true)` en `projection(cut=false)`

If-statements

In de vorige les hebben we gezien dat je for-statements kunt gebruiken om iteraties uit te voeren. In deze les gaan we kijken naar if-statements. Deze statements evalueren of een bepaalde variabele waar (true) is of onwaar (false) en voeren aan de hand van die informatie een bepaalde handeling wel of niet uit.

We gaan een vorm uit een plaat snijden, te weten een vierkant of een cirkel, afhankelijk van de waarde die een bepaalde variabele heeft. Hiervoor maken we eerst de variabele 'vorm' aan met een naam tussen dubbele aanhalingstekens. We kunnen hier kiezen tussen twee namen: "vierkant" en "rond". Deze naam is een *string*.

Je gebruikt `if` op de volgende wijze: `if (voorwaarde) {voer dit uit}`. Dit ziet er in OpenSCAD zo uit: `if (variable == waarde) {voer dit uit}`. Let erop dat je een variabele een waarde geeft door middel van het het is-gelijk-teken (=). Als we de if-statement willen laten controleren of een variabele een bepaalde waarde heeft, gebruiken we dubbele is-gelijk-tekens (==).

```
vorm = "vierkant";

if (vorm == "vierkant") {
    difference() {
        cube([50,50,3],center=true);
        cube([30,30,4],center=true);
    }
}
if (vorm == "cirkel") {
    difference() {
        cube([50,50,3],center=true);
        cylinder(h=4,d=30,center=true);
    }
}
```

Kies hier "vierkant" of "rond"

Als de variabele "vierkant" heeft als waarde
Voer dan dit uit.

Let op de dubbele is-gelijk-tekens!

Probeer nu eens de variabele te veranderen naar "rond". Niet de dubbele aanhalingstekens vergeten!

Nu hebben we eigenlijk twee if-statements achter elkaar gezet. Dit kan ook anders. Je kunt aan de if-statement ook een `else if`

toevoegen. OpenSCAD gaat dan eerst kijken of je if-statement waar is, en zo niet, of dan je else if-statement waar is. Dat ziet er zo uit:

```
vorm = "vierkant";                                  Kies hier “vierkant” of “cirkel”

if (vorm == "vierkant") {                           Als de variabele “cirkel” heeft als
    difference() {                                  waarde
        cube([50,50,3],center=true);                Voer dan dit uit.
        cube([30,30,4],center=true);
    }
}
else if (vorm == "cirkel") {                        Let op de dubbele is-gelijk-tekens!
    difference() {
        cube([50,50,3],center=true);
        cylinder(h=4,d=30,center=true);
    }
}
```

Wat is hier het voordeel van? Welnu, als je vorm inderdaad een “vierkant” is, dan hoeft OpenSCAD alleen het eerste deel van je code te doorlopen. Het programma zal dan niet controleren of je else if-statement ook waar is. Als je twee if-statements achter elkaar schrijft, zoals we in ons vorige voorbeeld hebben gedaan, dan zal OpenSCAD altijd allebei de statements op hun waarheid controleren. Je kunt dus optimaliseren door `else if` te gebruiken. Er is ook een derde optie. Naast `if` en `else if` bestaat er ook `else`. `Else` wordt altijd uitgevoerd als je if-statement niet waar blijkt te zijn. We zouden ons voorbeeld daarop aan kunnen passen:

```
vorm = "vierkant";                                  Kies hier “vierkant” of “cirkel”

if (vorm == "vierkant") {                           Als de variabele “cirkel” heeft als
    difference() {                                  waarde
        cube([50,50,3],center=true);                Voer dan dit uit.
        cube([30,30,4],center=true);
    }
}
else {                                              Let op de dubbele is-gelijk-tekens!
    difference() {
        cube([50,50,3],center=true);
        cylinder(h=4,d=30,center=true);
    }
}
```

Nu zou je bij je variabele ‘vorm’ vanalles in kunnen vullen, bijvoorbeeld “stoel”. OpenSCAD gaat dan de statements doorlopen. De if-conditie blijkt niet waar te zijn: de variabele ‘vorm’ heeft immers niet de waarde “vierkant”. Dus zal OpenSCAD altijd de else-statement uitvoeren en daarmee een rond gat in de plaat snijden. Als de if-statement wel waar is, dan zal OpenSCAD wederom niet de moeite nemen om de else-statement ook nog te controleren.

Je kunt ontelbaar veel else if-statements aan je if-statement toevoegen, maar het if-gedeelte en het else-gedeelte zijn uniek. Je code kan er dus ook zo uit komen te zien:

```
if (conditie1) {
}
else if (conditie2) {
}
else if (conditie3) {
}
else if (conditie4) {
}
else if (conditie5) {
}
else {
}
```

Belangrijk is hier om op te merken dat OpenSCAD zal stoppen met controleren wanneer het programma een conditie heeft gevonden die waar blijkt te zijn. Het zal dus nooit al deze statements uit proberen te voeren.

OPDRACHT Schijf je eigen if-statement.

Wegen naar Rome

Het is je misschien al opgevallen, in OpenSCAD bestaan er meerdere manieren om hetzelfde object te maken. Hieronder zie je elke keer een drietal verschillende manieren om een kubus, cilinder, hexagoon en kegel te maken. Voordat je naar deze voorbeelden kijkt is het goed om zelf alvast na te denken over de genoemde objecten en manieren te verzinnen om deze objecten te maken.

kubus

```
cube(10);
```

kubus 1

```
translate([20,0,0])
linear_extrude(height=10)
square(10);
```

kubus 2

```
translate([45,5,0])
rotate([0,0,45])
cylinder(h=10,r=sqrt(50),$fn=4);
```

kubus 3

cilinder

```
cylinder(h=10,d=10);
```

cilinder 1

```
translate([20,0,0])
linear_extrude(height=10)
circle(d=10);
```

cilinder 2

```
translate([40,0,0])
rotate_extrude()
square([5,10]);
```

cilinder 3

hexagoon

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

hexagoon 1

```
translate([20,0,0])
linear_extrude(height=10)
circle(d=10,$fn=6);
```

hexagoon 2

```
translate([40,0,0])
for (i = [0:5]) {
    rotate([0,0,i*60])
    translate([-2.5,0,0])
    cylinder(h=10,d=5,$fn=3);
}
```

hexagoon 3

kegel

```
cylinder(h=10,d1=10,d2=0);
```

kegel 1

```
translate([20,0,0])
linear_extrude(height=10,scale=0)
circle(d=10);
```

kegel 2

```
translate([40,0,0])
rotate_extrude()
difference() {
    translate([0,2.5,0])
    rotate([0,0,90])
    circle(d=10,$fn=3);
    translate([-10,0,0])
    square(10);
}
```

kegel 3

Waarom is dit nuttig? Het dwingt je om na te denken over de verschillende manieren waarop je bepaalde doelen kunt bereiken. Het is altijd slim om voordat je iets gaat modelleren in OpenSCAD na te denken over waar OpenSCAD goed in is, en welke aanpak

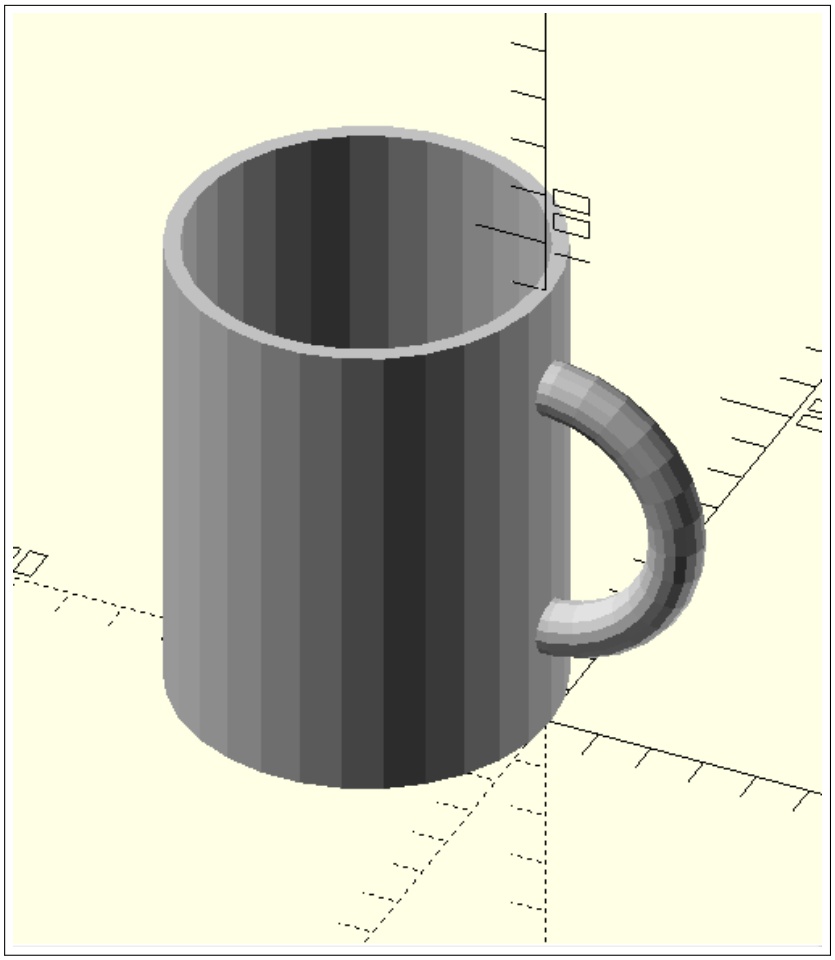
daarbij het snelste of gemakkelijkste is. Als je bijvoorbeeld het trucje van de fragments niet kent, zoals te zien in de voorbeelden van de hexagonen, zul je veel meer moeite moeten doen om een hexagoon te maken.

Analyse

Hoe maken we een voorwerp na uit het echt leven? Bijvoorbeeld een koffiekopje? Is dat lastig?

OPDRACHT Geef jezelf 20 minuten om een koffiekopje te modelleren. Denk eerst na over de gehele aanpak en hoe je dit snel kunt doen. Je zult vast meteen ideeën hebben om het ronde lichaam van het kopje te maken, maar hoe maak je het handvat? Op de volgende pagina vind je een voorbeeld. Het is maar 26 regels lang!

Koffiekopje



```
module lichaam() {
    difference() {
        cylinder(h=90,d=70);
        translate([0,0,3])
        cylinder(h=90,d=70-6);
    }
}
translate([-34,0,0])
lichaam();

module handvat() {
    rotate_extrude()
    translate([25,0,0])
    circle(5);
}

module half_handvat() {
    difference() {
        handvat();
        translate([-100,-50,-50])
        cube(100);
    }
}
translate([0,0,45])
rotate([90,0,0])
half_handvat();
```