An Introduction to 3D Modeling with openSCAD - openSCAD Basics

Imported and edited from **https://peak5390.wordpress.com/2013/01/08/an-introduction-to-3d-modeling-with-openscad-openscad-basics/** by Eric @ehmatthes.

"I have been introducing some of my math students to 3d modeling using the openSCAD software package. The openSCAD documentation is really good, but can be intimidating to new users. This is a summary of some of the basic shapes and functions that will get you started using openSCAD."

I (ProfHuster) use OpenSCAD to design objects for 3D printing. I started by using the web site **TinkerCAD.com**, which is very capable for designing objects. But I wanted more control over my designs, I wanted to keep my designs on my computer, and I like programming designs.

OpenSCAD is a programming language for 3D objects. You write code, then compile or *render* it into objects. I have found this works better for me than a GUI designing program. This document is an introduction to the basics you need to design objects for 3D printing using OpenSCAD.

Table of Contents

Installing OpenSCAD
Show Your Axes
Cubes and Translation
Parameters
Spheres
Cylinders and Rotate
Difference
Union
Modules
Other openSCAD Features

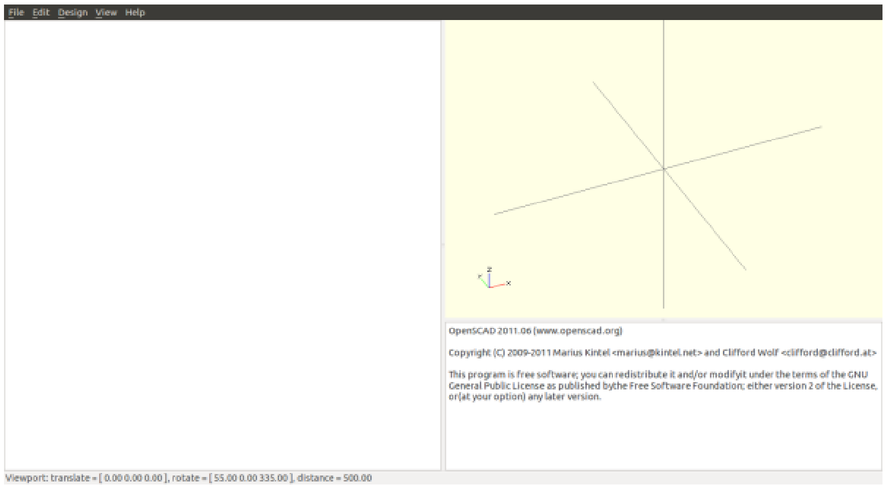
Installing OpenSCAD

Installation of openSCAD is straightforward in most cases. The program is available for Linux, Mac, and Windows from the openSCAD site **http://www.openscad.org/**. Scroll down to the section labeled **Download Releases**, and find the file appropriate for your operating system.

Portable Apps is a program that allows you to carry Windows apps with you on a USB drive. There is an older version you can add to your Portable Apps collection **here**. (ProfHuster recommends Portable Apps, BTW. He often installs portable apps on Windows computers he is forced to use.)

Show Your Axes

Before you begin modeling, go to **View>>Show Axes**. This will make it easier to see where the parts of your model are being placed.



```
OpenSCAD 2011.06 (www.openscad.org)

Copyright (C) 2009-2011 Marius Kintel <marius@kintel.net> and Clifford Wolf <clifford@clifford.at>

This program is free software; you can redistribute it and/or modifyit under the terms of the GNU
General Public License as published bythe Free Software Foundation; either version 2 of the License,
or(at your option) any later version.
```

Viewport: translate = [0.00 0.00 0.00], rotate = [55.00 0.00 335.00], distance = 500.00

Cubes and Translation

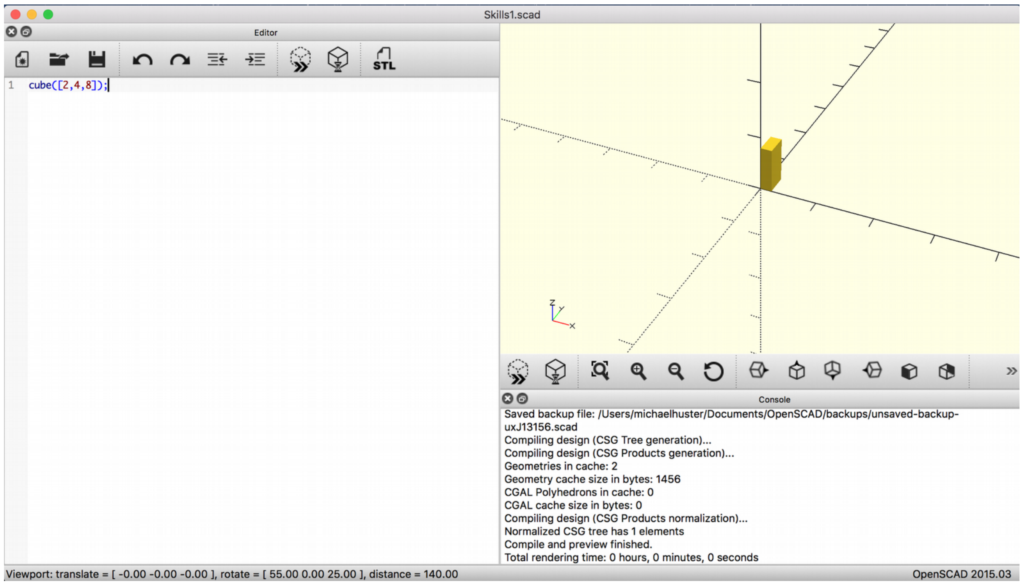
A “cube” in openSCAD simply refers to a rectangular prism. To make a simple cube the code is

```
cube(5);
```

But you rarely want a perfect cube. To make a cube with dimensions (3, 5, 6) the code is

```
cube([2, 4, 8]);
```

Note that the lengths are in square brackets. The coordinates default to millimeters (mm). This code produces this:



```
cube([2,4,8]);
```

```
Saved backup file: /Users/michaelhuster/Documents/OpenSCAD/backups/unsaved-backup-
uxJ13156.scad
Compiling design (CSG Tree generation)...
Compiling design (CSG Products generation)...
Geometries in cache: 2
Geometry cache size in bytes: 1456
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Compiling design (CSG Products normalization)...
Normalized CSG tree has 1 elements
Compile and preview finished.
Total rendering time: 0 hours, 0 minutes, 0 seconds
```

```
Viewport: translate = [ -0.00 -0.00 -0.00 ], rotate = [ 55.00 0.00 25.00 ], distance = 140.00          OpenSCAD 2015.03
```

If this is your first time using openSCAD, there are a few things you should know about the interface:

- To see your shape once you have entered the code, go to press <**F5**>, or click the Preview Icon (>>) or click **View**>>**Compile**.
- To zoom in on the shape, click on the drawing window and go to **View**>>**Zoom In**. The menu will show you the keyboard shortcuts, or zoom with your trackpad or mouse.
- To zoom out, do the opposite of the above.
- To view the shape from different angles, left-click in the drawing window and move your pointer.
- To move the origin of your axes, right-click in the drawing window and move your pointer.

You can center a cube on the xyz axes with the keyword argument **center=true**.

```
cube([2,4,8], center=true);
```

Translation

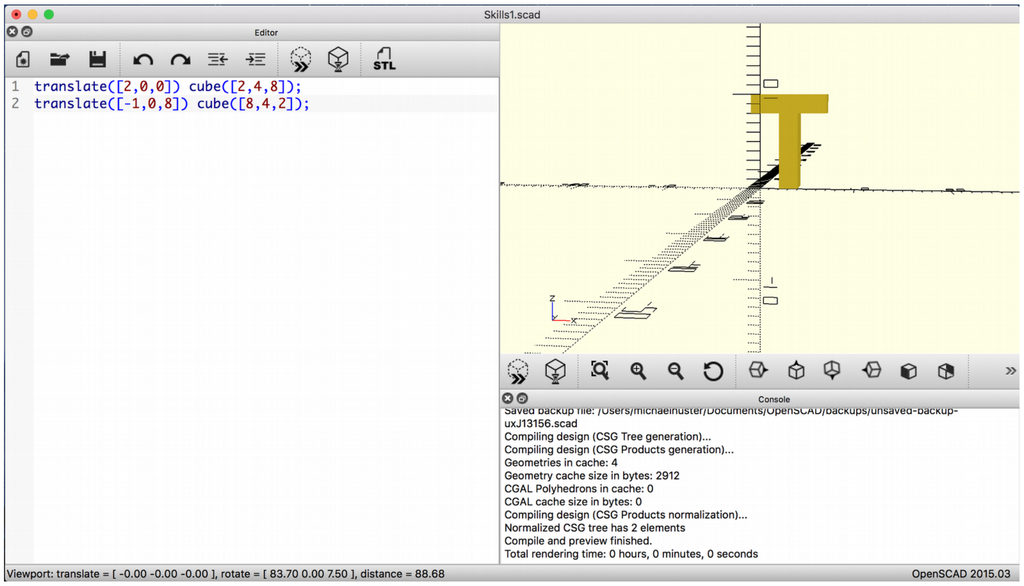
How do you make complex shapes? Well, you have to place objects in different places. You move objects around using the **translate** function. If are only moving one object the syntax is, for example

```
translate([2,0,0]) cube([2,4,8]);
```

You can also group multiple objects together and translate them as a group by using braces, **{** and **}** to group the objects.

The following code makes a 3D letter “T”:

```
translate([2,0,0]) cube([2,4,8]);
translate([-1,0,8]) cube([8,4,2]);
```


Skills1.scad
Editor
STL
Console
Viewport: translate = [-0.00 -0.00 -0.00], rotate = [83.70 0.00 7.50], distance = 88.68
OpenSCAD 2015.03

```
translate([2,0,0]) cube([2,4,8]);
translate([-1,0,8]) cube([8,4,2]);
```

```
Saved backup file: /Users/michaelnuster/Documents/OpenSCAD/backups/unsaved-backup-
uxJ13156.scad
Compiling design (CSG Tree generation)...
Compiling design (CSG Products generation)...
Geometries in cache: 4
Geometry cache size in bytes: 2912
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Compiling design (CSG Products normalization)...
Normalized CSG tree has 2 elements
Compile and preview finished.
Total rendering time: 0 hours, 0 minutes, 0 seconds
```

If you want to make this T, then move it, you could translate to group like this:

```
translate([0,0,10]){
  translate([2,0,0]) cube([2,4,8]);
  translate([-1,0,8]) cube([8,4,2]);
}
```

The indentation is not necessary, but it makes the code more readable.

Intermission: Make something!

At this point, it is probably good to create a model or two using nothing more than translated cubes. Sketch a simple shape using just cubes, and try to create your shape in openSCAD. You might make a tower with a door and two windows, or a smiley face with a cube for a head, two eyes, and a mouth. Create now!

Parameters (or *Variables*)

You can play with openSCAD using just numbers, and things will work. But when you want to make adjustments to your models, you will probably find yourself adjusting a whole bunch of numbers. **openSCAD** addresses this by letting you define parameters (like *variables* in a programming language). Let's make the letter T using parameters:

```
// The letter "T"
THeight = 10;             // along z axis
TWidth = 0.8 * THeight;   // along x
TThick = 0.2 * TWidth;    // how fat lines are
TDepth = 2 * TThick;      // along y

// Top bar
translate([0,0,THeight-TThick]) cube([TWidth,TDepth,TThick]);
// Vertical bar
translate([(TWidth-TThick)/2,0,0]) cube([TThick,TDepth,THeight-TThick]);
```

Working with parameters can require a little more thinking up front, but leads to a model that is much easier to modify later. In line 2, I define our core parameter, the total height of the letter T. In line 3, I make the horizontal width 80% of the height. I define the thickness of the letter as 20% of the height in line 4. Finally, in line 5, I define the 3D depth of the letter as twice the thickness. Now, if I want to change the size of the letter, I only have to change one number, the height.

In line 8, I move the top leg up to the height of the letter minus the thickness of the horizontal bar. Finally, in the last line I create the vertical bar, and then translate it to the right half the letter width minus half the letter thickness. If these numbers are confusing in the model you are creating, you can still use guess-and-check to work things out.

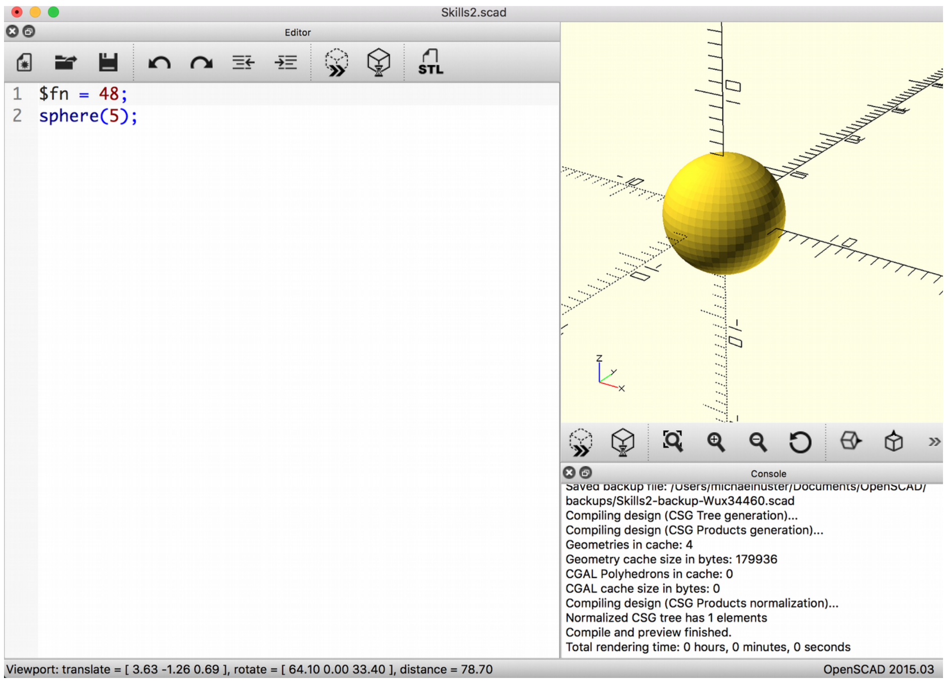
Now let's look at the rest of the shapes you can use in openSCAD.

Spheres

A simple sphere can be defined in one line:

```
sphere(5);
```

I can control the resolution of the sphere with the special parameter "**$fn**". This setting controls the number of faces that are used to create a curve in openSCAD. If you set **$fn** very high, you will get a smooth sphere but openSCAD will take a long time to render your models. It is good practice to use a moderate number, such as **48**, while working, and then use a high number such as **96** for rendering your final versions. Try making a sphere with different values of **$fn**.



```
$fn = 48;
sphere(5);
```

```
Saved backup file: /Users/michaelnuster/Documents/OpenSCAD/
backups/Skills2-backup-Wux34460.scad
Compiling design (CSG Tree generation)...
Compiling design (CSG Products generation)...
Geometries in cache: 4
Geometry cache size in bytes: 179936
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Compiling design (CSG Products normalization)...
Normalized CSG tree has 1 elements
Compile and preview finished.
Total rendering time: 0 hours, 0 minutes, 0 seconds
```

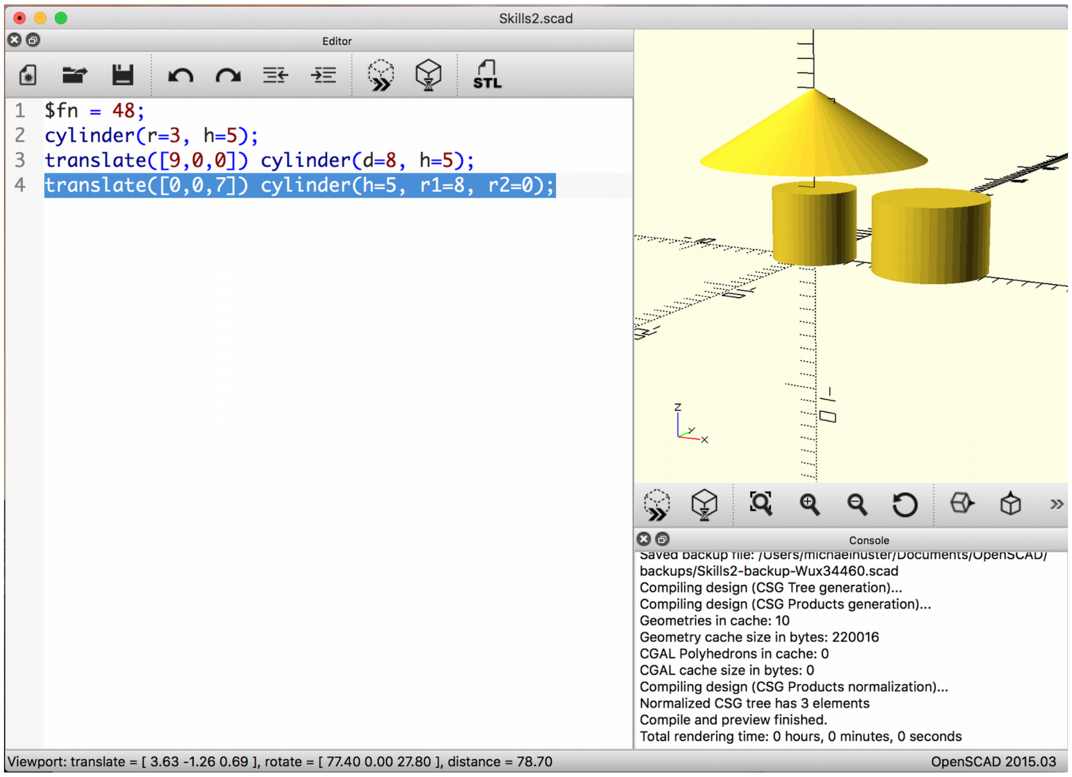
Viewport: translate = [3.63 -1.26 0.69], rotate = [64.10 0.00 33.40], distance = 78.70

OpenSCAD 2015.03

Cylinders and Rotate

A cylinder takes two variables to define and responds to the **$fn** variable just as a sphere does. A cylinder needs either a radius and a height, or a diameter and a height. The example below draws two cylinders. Note that the keyword parameters are **r** for radius, **d** for diameter, and **h** for height. You can also specify a top and bottom radius and make a cone or truncated cone.

```
$fn = 48;
cylinder(r=3, h=5);
translate([9,0,0]) cylinder(d=8, h=5);
translate([0,0,7]) cylinder(h=5, r1=8, r2=0);
```



```
$fn = 48;
cylinder(r=3, h=5);
translate([9,0,0]) cylinder(d=8, h=5);
translate([0,0,7]) cylinder(h=5, r1=8, r2=0);
```

```
Saved backup file: /Users/michaelnuster/Documents/OpenSCAD/
backups/Skills2-backup-Wux34460.scad
Compiling design (CSG Tree generation)...
Compiling design (CSG Products generation)...
Geometries in cache: 10
Geometry cache size in bytes: 220016
CGAL Polyhedrons in cache: 0
CGAL cache size in bytes: 0
Compiling design (CSG Products normalization)...
Normalized CSG tree has 3 elements
Compile and preview finished.
Total rendering time: 0 hours, 0 minutes, 0 seconds
```

```
Viewport: translate = [ 3.63 -1.26 0.69 ], rotate = [ 77.40 0.00 27.80 ], distance = 78.70
OpenSCAD 2015.03
```

Cylinders are oriented along the z-axis by default. If I want a cylinder to lie horizontally, I can use the **rotate** function. The rotate function, like translate, takes 3 arguments. The arguments are components of the vector you are rotating around. The units are degrees. So if you want to rotate an object around the x-axis by 90 degrees, the command is **`rotate([90,0,0])`**. The rotation point is the origin **`(0,0,0)`**.

The commands like translate and rotate are applied in order from right to left, so these commands give different results:

```
rotate([90,0,0])translate([0,0,6]) cylinder(r=3, h=5);
translate([0,0,6]) rotate([90,0,0])cylinder(r=3, h=5);
```

So, you have to think carefully how to want to orient and move an object. If you want to move a horizontal cylinder, make sure to put the **`translate`** function before the **`rotate`** function.

Like the **`translate`** function, you can rotate multiple objects by grouping them with braces, **`{`** and **`}`**, after the **`rotate`** call.

Difference

Everything I have looked at so far can be used to build complex shapes by ***adding*** a bunch of smaller parts together. But sometimes I need to take pieces away. A good example of this is a ring; a ring is a cylinder, with a smaller cylinder removed from the middle. The **`difference`** function builds the first thing in the list, and removes everything else:

```
// A simple ring with a hole
$fn = 50;
difference() {
    // This piece will be created:
    cylinder(r=10, h=10, center=true);

    // Everything else listed will be taken away:
```

```
    cylinder(r=8, h=10.1, center=true);
    rotate([0,90,0]) cylinder(r=3,h=10);
}
```

Sometimes it is very helpful to see what is being taken away from your model. Putting a pound sign in front of the pieces that are taken away makes them visible.

The piece that is being taken away must be slightly larger in one dimension than the piece it is being subtracted from in order to "poke through". For example, if the cylinder removed from the middle has the same height as the outer cylinder a thin skin shows up. Note that if you *Render* (the F6 key) this skin will disappear. It only shows up when you do a *Preview* (F5). Render is much slower, so I usually do a fix by defining a small parameter **`eps = 0.1`** and adding this to dimensions to make the skin disappear.

```
// A simple ring.
$fn = 50;
difference() {
    // This piece will be created:
    cylinder(r=10, h=5, center=true);
    // Everything else listed will be taken away:
    cylinder(r=8, h=5, center=true);
}
```

Here is my fix. Also note I define parameters at the top of the program to make it easy to make changes to the sizes. This is ***always*** a good programming practice.

```
// A simple ring.
$fn = 50;
// Parameters
eps = 0.1;
inner_radius = 8;
thickness = 2;
outer_radius = inner_radius + thickness;
height = 5;

difference() {
    // This piece will be created:
    cylinder(r=outer_radius, h=height, center=true);

    // Everything else listed will be taken away:
    cylinder(r=inner_radius, h=height+eps, center=true);
}
```

UNION

If you want to build up a piece out of other shapes before removing some parts, you can use the **`union`** function. The following code makes a ring with a lip or flange on it. (Remember, you can delete the # to get rid of the pink "missing" parts.)

```
// A ring with a lip at the top
$fn = 50;
// Parameters
eps = 0.1;
inner_radius = 8;
thickness = 2;
outer_radius = inner_radius + thickness;
lip_radius = outer_radius + thickness;
lip_height = thickness/2;
height = 5;

difference() {
    // These pieces will be created. Union makes them one piece
    union() {
        // main body of ring
        cylinder(r=outer_radius, h=height, center=true);
        // lip at top of ring
        translate([0,0,height/2+lip_height/2]) {
            cylinder(r=lip_radius, h=lip_height, center=true);
        }
    }
    // Everything else listed will be taken away:
    #cylinder(r=inner_radius, h=height+2*lip_height+eps, center=true);
}
```

This may seem to get complicated quickly, if you are used to a mouse-driven modeling program. But it is quite powerful to be able to write down an exact recipe for your models. I can quickly get into an efficient process of creating a model, and then making very precise refinements to our model. This approach certainly works better for some models than others.

MODULES

Modules are a powerful feature of openSCAD. Once I have created a shape I like, I can give it a name. From that point on, making that shape is as simple as using the name I have given it. A module is like a function for creating shapes. You can also give the module parameters. Then the module becomes a general purpose ring creation function. Let’s wrap our ring in a module with parameters for it inner radius, thickness, and height:

```
// A simple ring.
$fn = 50;
eps = 0.1;
inner_radius = 8;
thickness = 2;
height = 5;
// This line tells openSCAD to find the instructions for how to make a ring,
//  and then make the ring
ring(inner_radius, thickness, height);

// Instructions (or function) for how to make a ring
module ring(innerR, thick, hite) {
    // Parameters
    outer_radius = innerR + thick;
    height = 5;
    difference() {
        // This piece will be created:
        cylinder(r=outer_radius, h=hite, center=true);

        // Everything else listed will be taken away:
        #cylinder(r=innerR, h=hite+eps, center=true);
    }
}
```

Now I can make a ring any size I want, I can easily make any number of rings I want, and I can use any of the functions I have learned earlier on our rings. I will just demonstrate this by rotating and translating a ring and making a second ring of half the size:

```
// A simple ring.
$fn = 50;
eps = 0.1;
inner_radius = 8;
thickness = 2;
```

```
height = 5;
// This line tells openSCAD to find the instructions for how to make a ring,
//  and then make the ring
translate([0,0,15]) {
    rotate([0,90,0]) {
        ring(inner_radius,thickness,height);
    }
}
ring(inner_radius/2,thickness/2,height/2);

// Instructions (or function) for how to make a ring
module ring(innerR, thick, hite) {
    // Parameters
    outer_radius = innerR + thick;
    height = 5;
    difference() {
        // This piece will be created:
        cylinder(r=outer_radius, h=hite, center=true);

        // Everything else listed will be taken away:
        #cylinder(r=innerR, h=hite+eps, center=true);
    }
}
```

Modules are indispensable when you are working with a feature that appears multiple times in your overall model.

Preparing for a 3D Printer

Once you have the object designed, you have to *render* your model. You can use the **`F6`** key (or **`<fn>`** + **`<F6>`** on Mac portables.) or click **`Design → Render`**. Once you have it rendered, you have to expert it in the STL format. There may be an **`STL`** icon to click or click **`File → Export → STL.`**

Other openSCAD Features

If you have understood most of what has been described here, then you can probably make sense of the rest openSCAD's features using the project's documentation. Here are a few things to look at, which build on what was described here:

Comments

Comments are useful for writing notes in English about the parts of your model. They can also be used to hide parts of your model, so you can focus on certain details. To hide a piece, highlight the code for that piece and go to **`Edit>>Comment (Ctrl-D)`**. When you compile your model, that part will disappear. To show it again, highlight the code and choose **`Edit>>Uncomment (Ctrl-Shift-D)`**.

Import

You can define a module in another file, and then import that file into your main project file. This is useful for isolating parts of your model, and keeping yourself from dealing with very large files. It also means you can reuse parts in different projects, share parts with other designers, and use parts that other designers have created.

Extrude

You can define two dimensional shapes with the commands **`circle`**, **`square`** or **`polygon`**, then use **`extrude`** to make a three dimensional object.

Scale

You can wrap any block of code in the **`scale`** function, and openSCAD will scale that part of your model. You can make things larger or smaller. This is no substitute for parameters, however, because it scales everything in your model by the same factor.

Color

You can wrap any part of your model in the color function, and bring color into your models. OpenSCAD uses a decimal color model, where "**`color([0,0,0]){}`**" is black, and "**`color([1,1,1]){}`**" is white. You can use any value from 0 to 1. The values represent red, green, and blue respectively. Keep in mind that many 3d printers do not use this color information.

For Loop

The **`for`** loop can be used to automate the creation of similar parts. For example, if you wanted a series of spheres on the outside of your ring, spaced 45 degrees apart, you could use a **`for`** loop to create these spheres as a group.

Intersection

The union function combines two objects into one; the intersection finds only the areas that overlap between two objects.

Animation

There is a special variable called **`$t`**. The value of **`$t`** increases with time. So if you use **`$t`** to define the size of a cube, that cube will grow over time. You can use simple trigonometric functions to create cyclic behavior in your models.