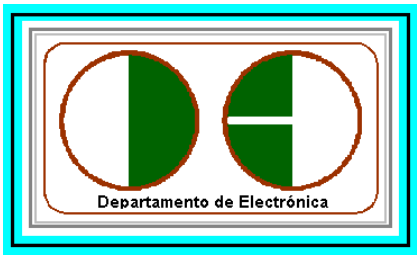
Instituto de Educación Secundaria Virgen de las Nieves



Departamento:

ELECTRICIDAD Y ELECTRÓNICA

Ciclo Formativo de Grado Superior:

DESARROLLO DE PRODUCTOS ELECTRÓNICOS

GranáBoT

Módulo: ***Técnicas de Programación***

Manual breve de OpenScad

Índice

Introducción. Webgrafia

Este manual se basa en la ayuda de OpenScad

http://en.wikibooks.org/wiki/OpenSCAD_User_Manual

y en la wiki: Diseño de piezas con OpenScad

http://www.iearobotics.com

Ficha:

- **Título**: Diseño de piezas con OpenScad
- **Contexto**: Sesión 2 del seminario: Diseño e Impresión de Piezas 3D con herramientas Open Source. UC3M-2011 dentro del Máster de Áutomática y Robótica de la UC3M
- **Ponente**: Juan González
- **Duración**: 2 horas
- **Lugar**: Universidad Carlos III de Madrid
- **Fecha**: 17-Nov-2011

Comienzo con OpenScad

Pasos a realizar:

- Abrir OpenScad.
- Escribir el código del programa.
- Pulsar F5 para renderizar el cubo en pantalla.
- Con la rueda del ratón se hace zoom.
- Pulsando el botón izquierdo y moviendo el ratón se rota la vista del cubo.
- Pulsando el botón central y moviendo el ratón se hace traslación del punto de vista. El mismo efecto se consigue con el botón secundario.
- Se graba el programa openscad creado.
- Se pulsa F6 para hacer un renderizado.
- Se exporta la pieza al formato STL (Design/export as STL)

¡¡Pieza lista para ser impresa en una impresora 3D open-source!!

Operadores matemáticos

&& (AND)	|| (OR)	! (NOT)	
<	<=	==	!=
>=	>	+	-
*	/	%	

Operador condicional: <boolean> ? <valor si True> : <valor si False>

El signo menos "-" también se usa como prefijo de números negativos.

Funciones matemáticas

abs <value>			<value> = cos <degrees>	
<value> = sin <degrees>			<value> = tan <degrees>	
<degrees> = asin <value>			<degrees> = acos <value>	
<degrees> = atan <value>			<degrees> = atan2 <value>	
pow(<base>, <exponent>)				
len	min	max	sqrt	round
ceil	floor			

lookup(<value>, <vector_of_vectors>) (interpola un valor de vectores)

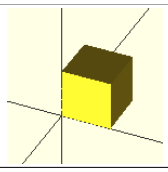
Creación de un cubo

Sintaxis:

cube([x,y,z],center=true|false);

Entre corchetes se dan los valores en mm del cubo correspondientes a los ejes x,y,z.

El parámetro center es para centrar el cubo en los ejes.
Ejemplo:

cube([10,10,10]); //Dibuja un cubo de 1 cm de lado	

Traslación y rotación

La traslación se realiza con el comando translate.
Sintaxis:

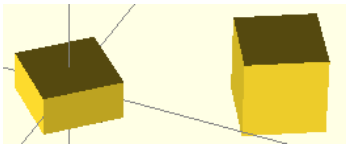
translate([x,y,z])

Damos los valores en mm del valor de traslado en los ejes correspondientes x,y,z. No finaliza con punto y coma porque traslada la figura definida a continuación.
La rotación se realiza con el comando rotate.
Sintaxis:

rotate([x,y,z])

Damos los valores en grados de la rotación en los ejes correspondientes x,y,z. No finaliza con punto y coma porque rota la figura definida a continuación.

Ejemplo:

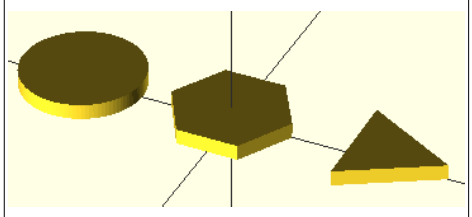
//Ejemplo de traslación y rotación
//Rotación y traslación de un cubo →
rotate([0,0,30])
translate([50,0,0])
cube([20,20,20],center=true); // ←
//Rotación de un cubo →
rotate([0,0,45])
cube([20,20,10],center=true); // ←
	

Cilindros y polígonos

La versatilidad de cylinder permite crear cualquier polígono regular.
Sintaxis:

cylinder(r=m,h=m,$fn=n);

Dibuja un polígono de n lados (parámetro $fn) de un radio y altura dados en mm. El parámetro $fn debe valer como mínimo 3 y, para el propósito de construcción con una impresora 3D, un valor de 100 genera un círculo casi perfecto aunque si queremos podemos dar valores mayores.
Ejemplo:

//Dibujo de una moneda →
translate([-50,0,0])
cylinder(r=40/2, h=5, $fn=100); // ←
//Dibujo de un Hexágono →
cylinder(r=40/2, h=5, $fn=6); // ←
//Dibujo de un Triángulo equilátero →
translate([50,0,0])
cylinder(r=40/2, h=5, $fn=3); // ←
	

Taladros

Con la operación booleana difference() y con cilindros podemos hacer taladros en nuestras piezas.

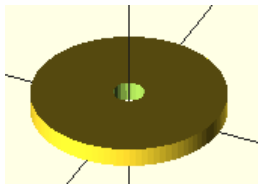
Sintaxis:

difference()

Operador booleano que obtiene la diferencia entre dos polígonos.

Ejemplo:

```
//Rueda simple
difference() {
  cylinder(r=50/2, h=5,$fn=100); //Base de la rueda
  cylinder(r=8/2, h=20,$fn=20,center=true);  //Taladro de 8mm
}
```



Unión de piezas

La operación booleana union() permite pegar objetos.

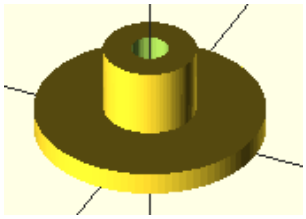
Sintaxis:

union()

Operador booleano que convierte en uno los dos polígonos.

Ejemplo:

```
//Rueda con portaejes y taladro para el eje
difference() {
  //Rueda →
    union() {
      cylinder(r=50/2, h=5, $fn=100); //Base de la rueda
      cylinder(r=20/2, h=20, $fn=80); //Portaejes // ←
    }
   cylinder(r=8/2, h=80, $fn=30,center=true); //Taladro
}
```

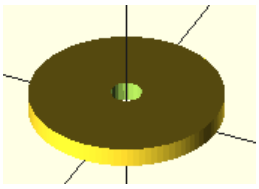


Parámetros

OpenScad es un lenguaje de scripts interpretado que permite parametrizar datos sin necesidad de definir variables. Definimos el objeto usando parámetros y lo construimos a partir de ellos.

Ejemplo: Dibujamos la rueda simple vista en taladros usando parámetros

```
//Parámetros de la rueda →
grosor = 5;
diametro=50;
diam_eje = 8; // ←
  difference() {
    //Base de la rueda
    cylinder(r=diametro/2, h=grosor,$fn=100);
    //Taladro del eje
    cylinder(r=diam_eje/2, h=3*grosor,$fn=20,center=true);
 }
```



Módulos

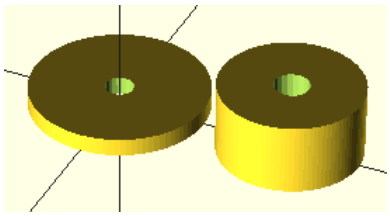
Convertir código en un módulo nos permite reutilizarlo fácilmente.

Sintaxis:

module nombre(<var1>, <var2>, ...) {...}

Ejemplo: Convertimos la rueda simple en un módulo para reutilizarla fácilmente.

```
module rueda_simple(grosor, diametro, diam_eje)
{
//Construcción de la rueda a partir de los parámetros →
difference() {
  //Base de la rueda
  cylinder(r=diametro/2, h=grosor,$fn=100);
  //Taladro del eje
  cylinder(r=diam_eje/2, h=3*grosor,$fn=20,center=true);
  }
}
rueda_simple(diametro=50, grosor=5, diam_eje=8); // ←
translate([50,0,0])
//Uso del módulo definido
rueda_simple(diametro=40, grosor=20, diam_eje=10);
```



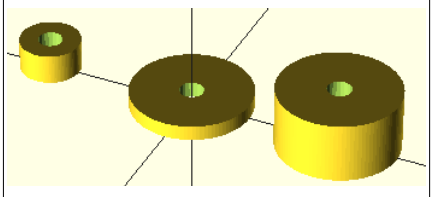
- También podemos crear el módulo dándole parámetros por defecto.

Sintaxis:

module nombre(<var1=value>, <var2=value>, ...) {...}

Ejemplo: Damos parámetros por defecto al módulo de la rueda simple para reutilizarla fácilmente.

```
module rueda_simple(grosor=5, diametro=40, diam_eje=8)
{
//Construcción de la rueda a partir de los parámetros →
  difference() {
     //Base de la rueda
    cylinder(r=diametro/2, h=grosor,$fn=100);
    //Taladro del eje
    cylinder(r=diam_eje/2, h=3*grosor,$fn=20,center=true);
  }
} // ←
//Ejemplos de utilizacion del modulo Rueda simple →
  rueda_simple(); //Rueda por defecto
  translate([50,0,0])
    rueda_simple(grosor=20);
  translate([-50,0,0])
    rueda_simple(diametro=20, grosor=10); // ←
```



Simetrías y comando "use"

El comando mirror nos va a permitir realizar simetrías respecto a los tres ejes.

Sintaxis mirror:

mirror([x,y,z])

Simetría según el eje que pongamos a 1. Los ejes que no intervengan se ponen a 0.

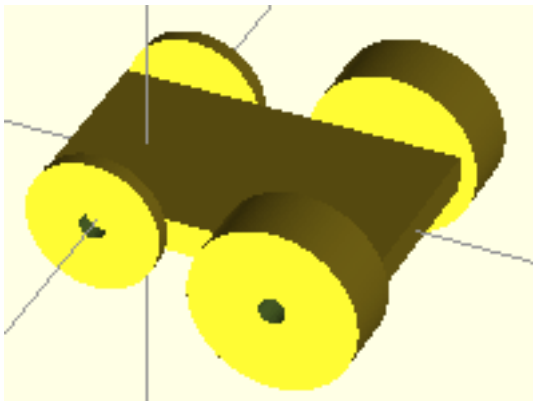
- Una vez definido un módulo y sus parámetros y guardado en formato scad podemos usarlo en nuestros programas mediante el comando “use”.

Sintaxis use:

use <NombreModulo.scad>

Ejemplo: Utilización del módulo rueda simple para construir un coche sencillo

```
//Ejemplo sencillo de como utilizar los módulos
use <rueda_simple.scad>

//Chasis del coche →
translate([30,0,0])
  cube([100,60,10],center=true); // ←

//Rueda delantera izquierda →
translate([0,-30,0])
  rotate([90,0,0])
   rueda_simple();  // ←
//-- Rueda trasera izquierda →
translate([60,-30,0])
  rotate([90,0,0])
    rueda_simple(grosor=20, diametro=50);  // ←
//Lado derecho del coche.
//Es simétrico con respecto al izquierdo
mirror([0,1,0]) {
//Rueda delantera derecha →
translate([0,-30,0])
  rotate([90,0,0])
    rueda_simple();   // ←
//Rueda trasera derecha →
translate([60,-30,0])
  rotate([90,0,0])
    rueda_simple(grosor=20, diametro=50);  // ←
}
```



Variables especiales

Variables $fa, $fs y $fn

Las variables especiales $fa, $fs y $fn controlan el número de partes o trozos usados para generar un arco:

- $fa es el ángulo mínimo de un fragmento. Por grande que sea el círculo no tiene más de 360 fragmentos dividido por este número. El valor predeterminado es 12 (es decir, 30 fragmentos de un círculo completo). El valor mínimo permitido es de 0,01.
- $fs es el tamaño mínimo de un fragmento. El valor por defecto es 2. El valor mínimo permitido es de 0,01.
- $fn vale normalmente 0. Cuando esta variable tiene un valor mayor que cero, las otras dos variables son ignoradas y el círculo se representa utilizando este número de fragmentos.

Cuando $fa y $fs son usadas para determinar el número de fragmentos de un círculo, el programa OpenSCAD nunca usará menos de 5 fragmentos.

Variable $t

La variable $t se utiliza para animaciones. Si se habilita con view->animate y damos un valor a "FPS" y "Steps", el campo "Time" muestra el valor actual de $t. El diseño se vuelve a compilar cada 1/"FPS" segundos con $t incrementándose en 1/"Steps" para "Steps" veces, finalizando si cualquiera de los dos $t = 1 o $t = 1-1/steps.
Si "Dump Pictures" está seleccionado, las imágenes se crearan en el mismo directorio que el fichero .scad, usando los valores $t y guardando los archivos siguientes:

- $t=0/Steps filename="frame00001.png"
- $t=1/Steps filename="frame00002.png
- $t=2/Steps filename="frame00003.png"
- . . .
- $t=1-3/Steps filename="frame<Steps-2>.png"
- $t=1-2/Steps filename="frame<Steps-1>.png"
- $t=1-1/Steps filename="frame00000.png"

O bien, para otros valores de los pasos, se sigue este patrón:

- $t=0/Steps filename="frame00001.png"
- $t=1/Steps filename="frame00002.png
- $t=2/Steps filename="frame00003.png"
- . . .
- $t=1-3/Steps filename="frame<Steps-2>.png"
- $t=1-2/Steps filename="frame<Steps-1>.png"
- $t=1-1/Steps filename="frame<Steps-0>.png"
- $t=1-0/Steps filename="frame00000.png"

Más sólidos

Esfera

Crea una esfera en el origen de coordenadas. El nombre del argumento es opcional.
Sintaxis:

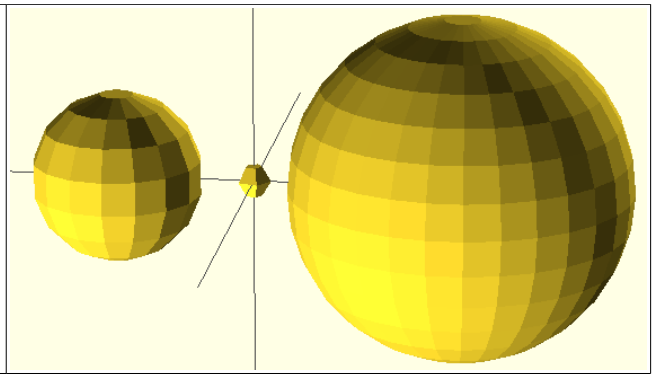
sphere(r = <radius>);

El radio de la esfera 'r' es un número decimal. La resolución de la esfera se basa en el tamaño de la misma y en las variables $fa, $fs y $fn.

$fa: ángulo en grados
$fs: ángulo en mm
$fn: resolución

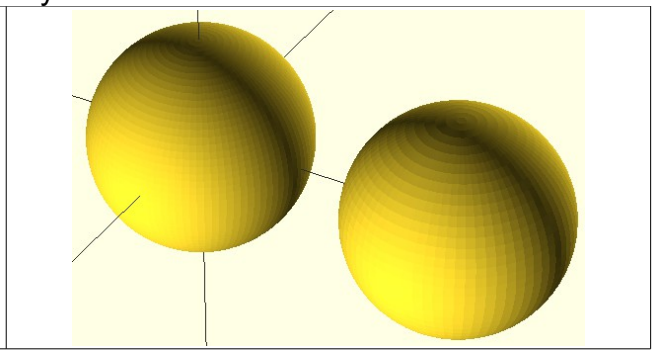
Ejemplo: Dibujamos esferas de radio 1, 5 y 10

```
//Esferas →
sphere(r = 1);
translate([-8,0,0])
  sphere(r = 5);
translate([12,0,0])
  sphere(r = 10); // ←
```



Ejemplo: Dibujamos esferas de alta resolución y radio 2

```
//Esferas de alta resolución
sphere(2, $fn=100);
translate([5,0,0])
//Sin los pequeños triángulos en los polos
sphere(2, $fa=5, $fs=0.1);
```



Cilindros

Crear cilindros simples y piezas cilíndricas.

Sintaxis cilindro:

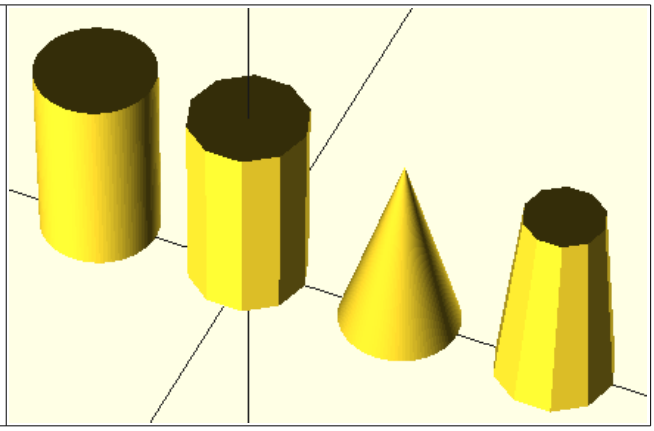
cylinder(h = <height>, r = <radius>);

Sintaxis piezas:

cylinder(h = <height>, r1 = <bottomRadius>, r2 = <topRadius>, center = <boolean>);

Ejemplo: Dibujamos cilindros

```
//Cilindro simple
cylinder(h = 10, r = 3);
translate([-8,0,0])
cylinder(h = 10, r = 3, $fn=100);
//Cono
translate([8,0,0])
cylinder(h = 10, r1 = 3, r2 = 0, $fn=100);
//Tronco cónico
translate([16,0,0])
cylinder(h = 10, r1 = 3, r2 = 2);
```



Transformaciones

Escala

Escala los elementos secundarios utilizando el vector especificado. El nombre del argumento es opcional.

Sintaxis:

scale(v = [x, y, z]) { ... }

Rotación

Girar los grados indicados alrededor del origen de coordenadas o alrededor de un eje arbitrario. Los nombres de los argumentos son opcionales.

Cuando se especifica una rotación en múltiples ejes la rotación se aplica en el orden siguiente: x, y, z.

Sintaxis:

rotate(a = deg, v = [x, y, z]) { ... }

Por ejemplo, para poner un objeto al revés, se puede hacer lo siguiente:

rotate(a=[0,180,0]) { ... }

gira el objeto 180 grados en torno al eje y.

En este otro ejemplo se gira el objeto 45 grados alrededor del eje definido por el vector [1,1,0].

rotate(a=45, v=[1,1,0]) { ... }

Traslación

Mueve los elementos secundarios a lo largo del vector especificado. El nombre del argumento es opcional.

Sintaxis:

translate(v = [x, y, z]) { ... }

Espejo

Refleja el elemento secundario en un plano que pasa por el origen. El argumento para mirror() es el vector normal a dicho plano.

Sintaxis:

mirror([0, 1, 0]) { ... }

Color

Muestra los elementos secundarios utilizando el color RGB especificado mas el valor alfa. Esto sólo se utiliza para la vista previa generada con F5. Para el renderizado CGAL (F6) y STL no se admite color. El valor alfa por defecto es 1.0 (opaco) si no se especifica.

Sintaxis:

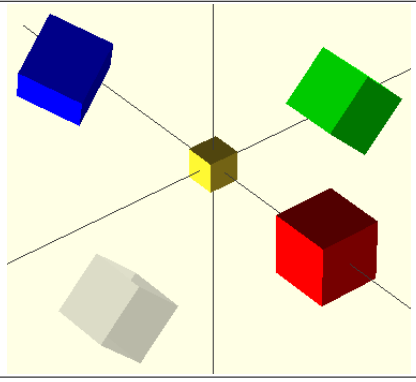
color([r, g, b, a]) { ... }
color([R/255, G/255, B/255, a]) { ... }

Téngase en cuenta que los valores r,g,b están limitados al intervalo {0,0 ... 1,0} en lugar de los tradicionales números enteros {0 ... 255}. Sin embargo, se pueden especificar los valores como fracciones, por ejemplo color ([R/255, G/255, B/255]) {... }
También se pueden definir los colores por su nombre. Por ejemplo, para crear una esfera roja, se puede utilizar este código: color("red",0.5) sphere(5);

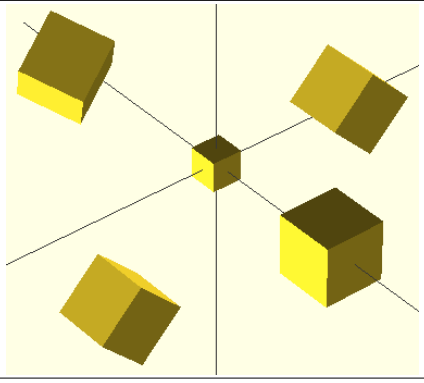
Ejemplo de transformaciones:

```
//Valores por defecto. Color amarillo
cube(center=true);
//Color rojo
color([1,0,0])
translate([0,5,0])
cube(2, center = true);
//Color verde
color([0,1,0])
translate([0,-5,0])
rotate([45,0,0])
cube(2, center = true);
//Color gris y alfa transparente
color([100/256,100/256,100/256, 0.3])
rotate([45,0,0])
translate([0,-5,0])
cube(2, center = true);
//Color azul
color([0,0,1])
translate([-3,-3,3])
rotate(45,v=[1,1,1])
cube(2, center = true);
```

Vista previa con F5



Vista renderizada (F6)



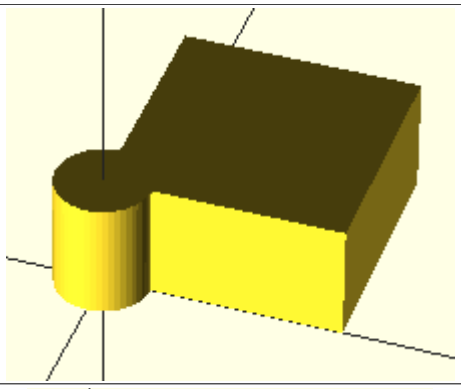
Minkowski

Muestra la suma de Minkowski de nodos secundarios.

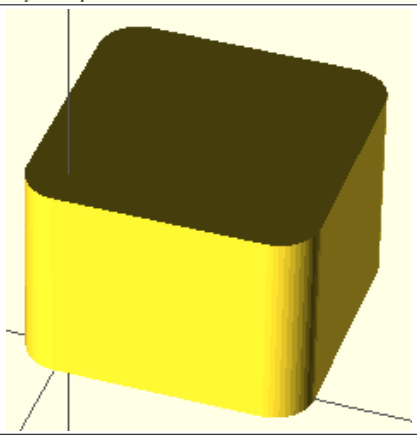
La suma de Minkowski de dos conjuntos de puntos P y Q en $\mathbb{R}$, que se representa por $P \oplus Q$, se define como el conjunto $\{p + q: p \in P, q \in Q\}$.

Ejemplo: Suma Minkowski de un cubo y un cilindro. Es quizá la forma más elegante de redondear las aristas de un cubo.

```
//Representamos un cubo y un cilindro de la misma altura en z
$fn=50;
cube([10,10,5]);
cylinder(r=2,h=5);
```



```
//Le hacemos la suma de Minkowski
$fn=50;
minkowski()
{
cube([10,10,5]);
cylinder(r=2,h=5);
}
```

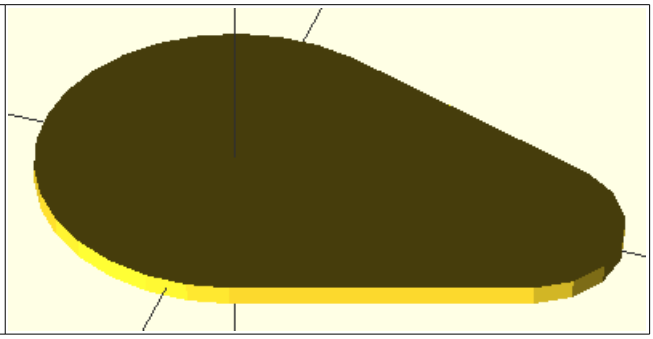


Envolvente (hull)

Muestra la envolvente convexa del nodo secundario.

Ejemplo: Suma Minkowski de un cubo y un cilindro. Es quizá la forma más elegante de redondear las aristas de un cubo.

```
hull() {
translate([15,0,0])
circle(5);
circle(10);
}
```



Operaciones

Unión

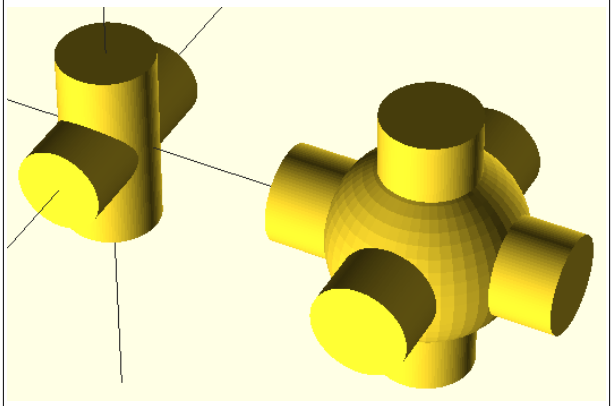
Se unen o suman todos los objetos secundarios entre las llaves.

Sintaxis:

union() {...}

Ejemplo:

```
union() {
  cylinder (h = 4, r=1, center = true, $fn=100);
  rotate ([90,0,0])
  cylinder (h = 4, r=0.9, center = true, $fn=100);
}
translate([7.0,0])
union(){
  sphere(2, $fn=50);
  cylinder(r=1,h=6,$fn=100,center=true);
  rotate([0,90,0])
  cylinder(r=1,h=6,$fn=100,center=true);
  rotate([90,0,0])
  cylinder(r=1,h=6,$fn=100,center=true);
}
```



Diferencia

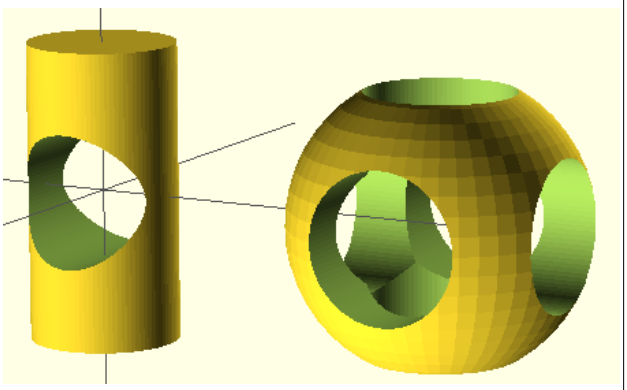
Al primer elemento se le restan los demás

Sintaxis:

difference() {...}

Ejemplo:

```
difference() {
cylinder (h = 4, r=1, center = true, $fn=100);
rotate ([90,0,0])
cylinder (h = 4, r=0.9, center = true, $fn=100);
}
translate([5.0,0])
difference(){
sphere(2, $fn=50);
cylinder(r=1,h=6,$fn=100,center=true);
rotate([0,90,0])
cylinder(r=1,h=6,$fn=100,center=true);
rotate([90,0,0])
cylinder(r=1,h=6,$fn=100,center=true);
}
```



Intersección

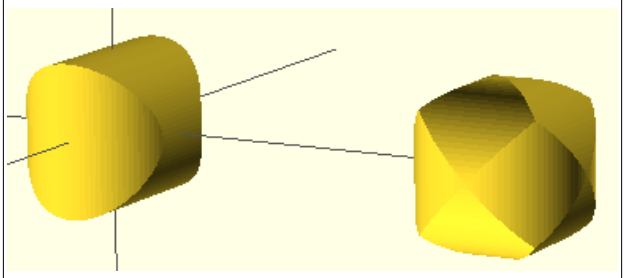
Se obtiene la parte común de dos o más objetos.

Sintaxis:

intersection() {...}

Ejemplo:

```
intersection() {
cylinder (h = 4, r=1, center = true, $fn=100);
rotate ([90,0,0])
cylinder (h = 4, r=0.9, center = true, $fn=100);
}
translate([5.0,0])
intersection(){
sphere(2, $fn=50);
cylinder(r=1,h=6,$fn=100,center=true);
rotate([0,90,0])
cylinder(r=1,h=6,$fn=100,center=true);
rotate([90,0,0])
cylinder(r=1,h=6,$fn=100,center=true);
}
```



Funciones de iteración y condiciones

Bucle for

Cuando vamos a trabajar con piezas que presentan simetría radial (como engranajes, ruedas, etc) este tipo de herramientas resulta bastante útil, pues sólo tenemos que diseñar una parte y ejecutarla 'n' veces.

Esta instrucción actúa de forma similar que en programación, ejecuta la tarea que se encuentre entre las llaves, el número de veces indicado entre paréntesis.

Sintaxis:

Versión rango:

for (variable=<range>) <do_something>

Range: [<start>:<end>]

Range: [<start>:<increment>:<end>]

Recorrer desde start hasta end inclusive. También funciona si <end> si es más pequeño que <start>.

Si en lugar de trabajar con valores continuos se necesita trabajar con puntos concretos podemos especificar los valores a través de un vector con coordenadas discretas.

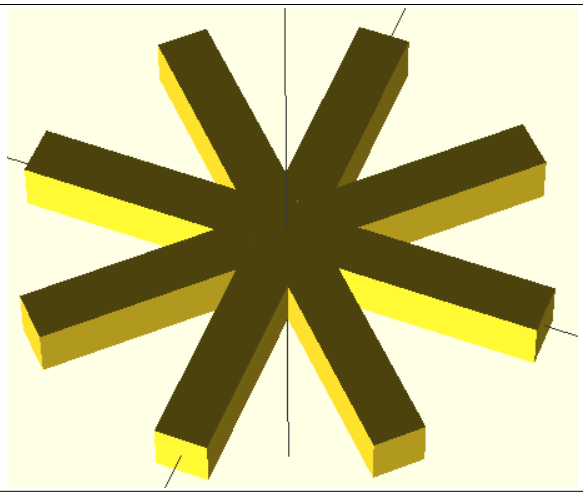
Versión Vector:

for (variable=<vector>) <do_something> - <variable>

Se asigna a cada valor sucesivo en el vector.

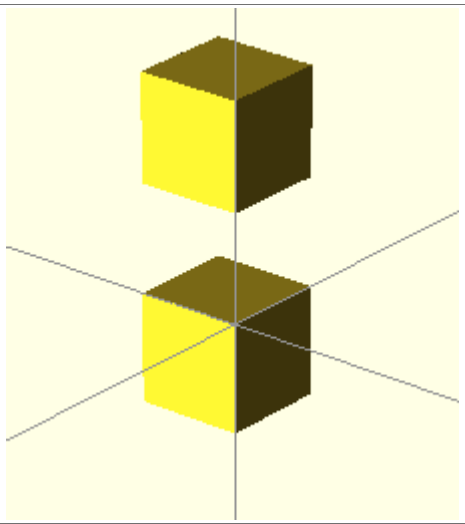
Ejemplo:

```
for (i=[0:8]){
rotate([0,0,i*360/8])
translate([25,0,0])
cube([50,10,10],center=true);
}
```

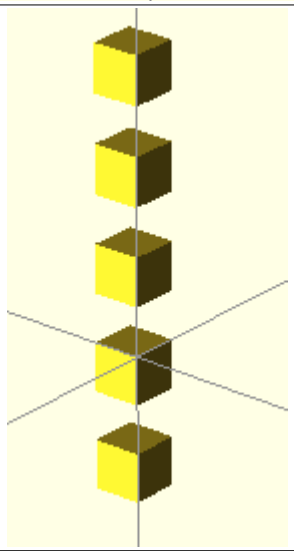


Ejemplo: iteración sobre un vector.

```
// Dos iteraciones, z = -10, z = 10
for (z = [-10, 10])
{
translate([0, 0, z])
cube(size = 10, center = false);
}
```

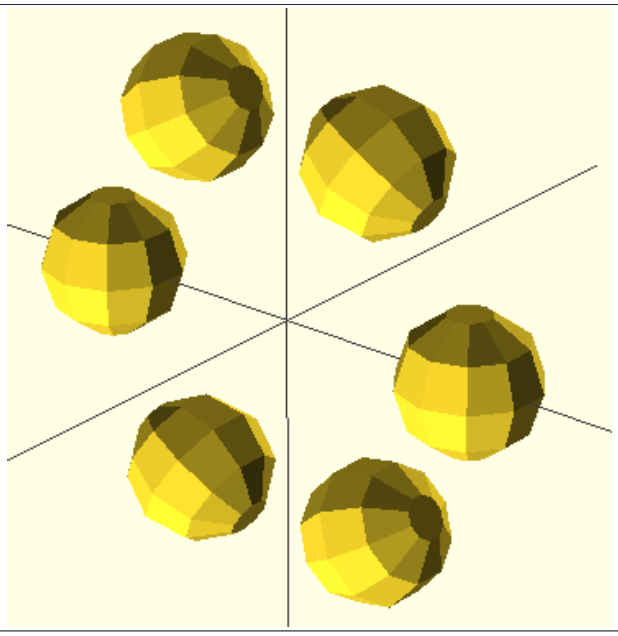


```
// Cinco iteraciones
for (z = [-3,-1, 1,3,5])
{
translate([0, 0, z])
cube(size = 1, center = false);
}
```



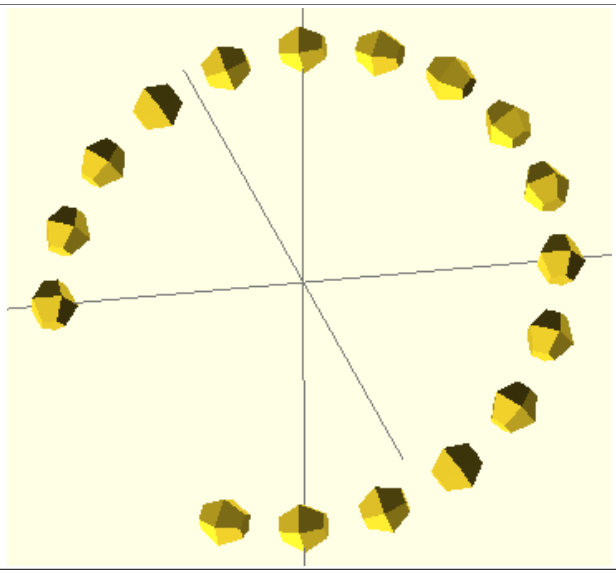
Ejemplo: iteración en un rango.

```
for ( i = [0 : 5] )
{
rotate( i * 360 / 6, [1, 0, 0])
translate([0, 3, 0])
sphere(r = 1,$fn=10);
}
```



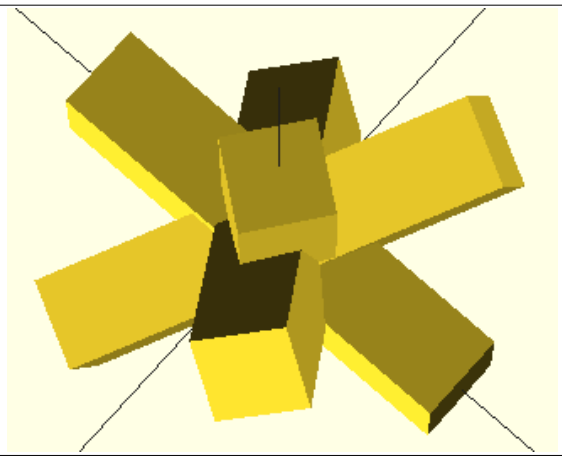
Ejemplo: iteración en un rango con un incremento.

```
//El parámetro central es el incremento.
//Por eso se dibujan 17 esferas.

for ( i = [0 : 0.3 : 5] )
{
rotate( i * 360 / 6, [1, 0, 0])
translate([0, 10, 0])
sphere(r = 1);
}
```



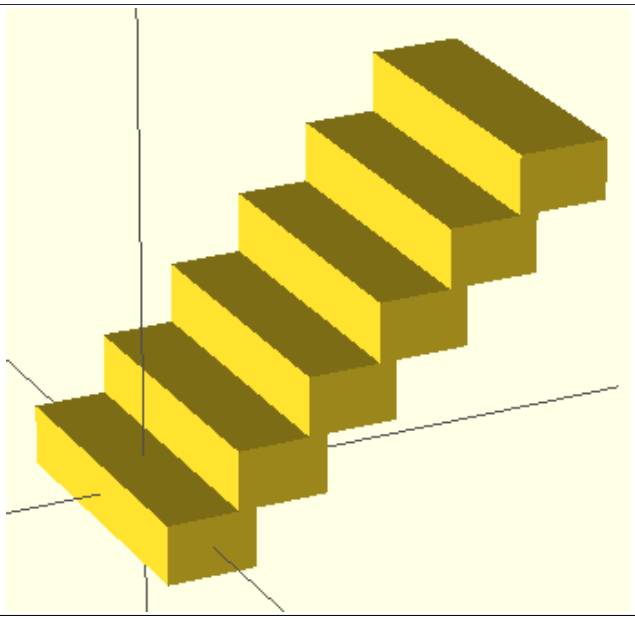
Ejemplo: iteración sobre una matriz con rataciones.

```
for(i = [
[ 0, 0, 0],
[ 10, 20, 300],
[200, 40, 57],
[ 20, 88, 57]
])
{
rotate(i)
cube([100, 20, 20], center = true);
}
```



Ejemplo: iteración sobre una matriz con traslaciones.

```
for(i = [ [ 0, 0, 0],
[ 0, 12, 10],
[0, 24, 20],
[0, 36, 30],
[0, 48, 40],
[0, 60, 50] ])
{
translate(i)
cube([50, 15, 10], center = true);
}
```

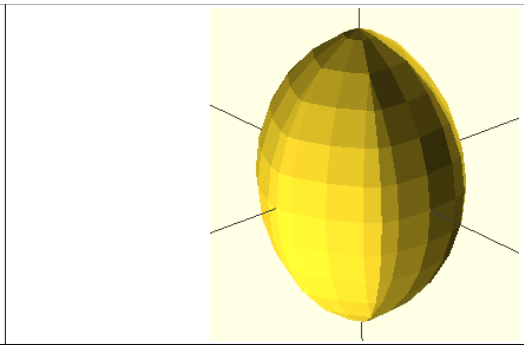


Intersección bucle For

Obtener la intersección sobre los valores de un vector o un rango de valores, obteniendo la parte común de varias piezas.

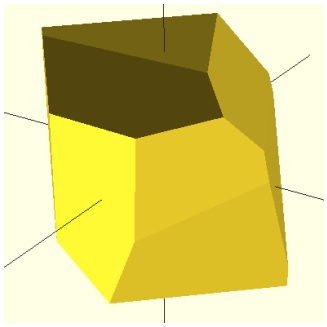
Ejemplo: intersección para un rango

```
intersection_for(n = [1 : 6])
{
rotate([0, 0, n * 60])
{
translate([5,0,0])
sphere(r=12);
}
}
```



Ejemplo: intersección para un vector

```
intersection_for(i = [
[ 0, 0, 0],
[ 10, 20, 300],
[200, 40, 57],
[ 20, 88, 57] ])
{
rotate(i)
cube([5, 2, 2], center = true);
}
```



Sentencia If

Debemos usar una expresión booleana como condición

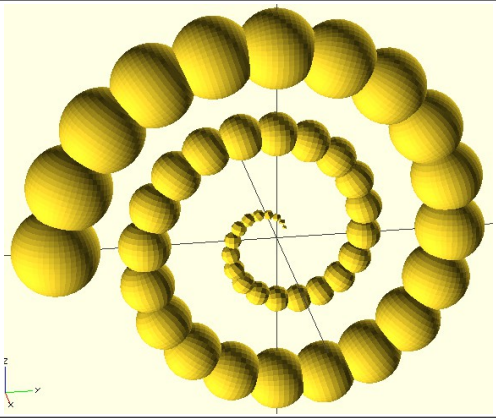
Ejemplo:

```
if (x > y)
  {
  cube(size = 1, center = false);
  }
else
  {
  cube(size = 2, center = true);
  }
```

Sentencia assign

Ejemplo: Convertimos la rueda simple en un módulo para reutilizarla fácilmente.

```
for (i = [1:50])
{
assign (angulo=i*360/20, distancia=i*10, radio=i*2)
{
rotate(angulo, [1, 0, 0])
translate([0, distancia, 0])
sphere(r = radio, $fn=i);
}
}
```



Sentencia Include

Para incluir código desde archivos externos en OpenSCAD, hay dos comandos disponibles:

1. include <filename.scad>

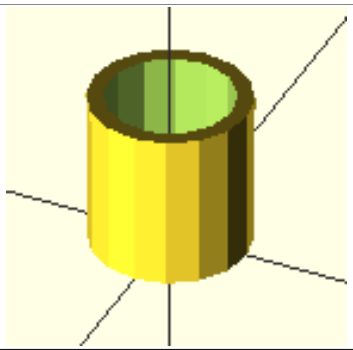
Actúa como si el contenido del archivo incluido se escribiese en el archivo actual.

2. use <filename.scad>

Importa módulos y funciones sin ejecutar ningún comando que no sean esas definiciones.
Los archivos se buscarán en el mismo directorio en el que hemos abierto el diseño o bien en el directorio de instalación ..\OpenSCAD\libraries. Si queremos trabajar con otro directorio debemos indicar la ruta completa.

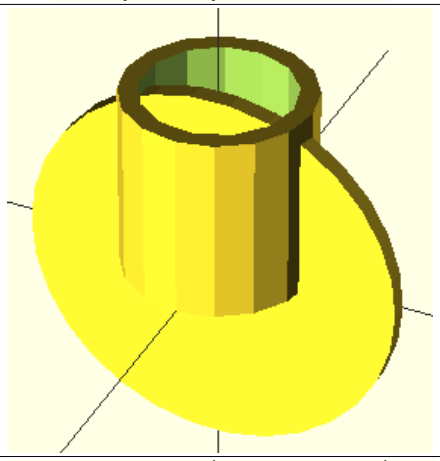
Ejemplo: Creamos un módulo para generar anillos

```
module anillo(r1, r2, h) {
difference() {
cylinder(r = r1, h = h);
translate([ 0, 0, -1 ]) cylinder(r = r2, h = h+2);
}
}
anillo(5, 4, 10);
```



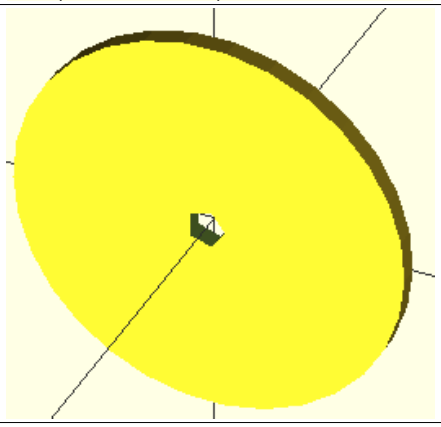
```
include <anillo.scad>;
rotate([90, 0, 0])
anillo(10, 1, 1);
```

Genera como salida el anillo definido en el archivo incluido y el generado en este archivo.



```
use <anillo.scad>;
rotate([90, 0, 0])
anillo(10, 1, 1);
```

Genera como salida sólo el anillo definido en este archivo.



Primitivas 2D

Cuadrado (square)

Crea un cuadrado en el origen del sistema de coordenadas. Si center es true se centrará en el origen, de lo contrario, se crea en el primer cuadrante.

Sintaxis:

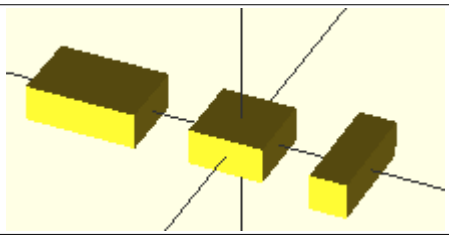
square(size = <val>, center=<boolean>);
square(size = [x,y], center=<boolean>);

El parámetro size puede ser un decimal o una matriz de dos valores. Si sólo damos un número el resultado será un cuadrado con sus lados de esa longitud. Si damos dos valores en forma de matriz, entonces los valores se corresponden a las longitudes de los lados en los ejes X y Y lados. El valor por defecto es 1.

El parámetro center determina la posición del objeto. Si es true el objeto estará centrado respecto a (0,0). De lo contrario, el objeto se coloca en el primer cuadrante con una esquina en (0,0). El valor predeterminado es false.

Ejemplo:

```
square (2,center =true);
translate([3,0,0])
square ([1,3],center =true);
translate([-4,0,0])
square ([3,2],center =true);
```



Círculo (circle)

Crea un círculo en el origen de coordenadas. El nombre del argumento es opcional.

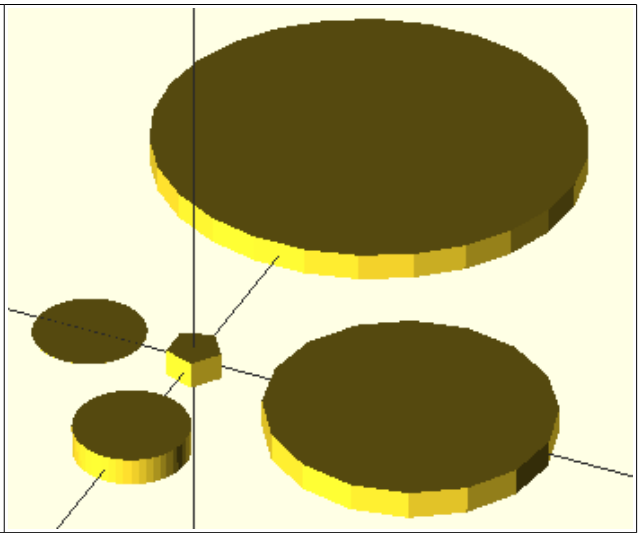
Sintaxis:

circle(r = <val>);

El parámetro r es un número decimal que expresa el radio del círculo. La resolución del círculo se basará en el tamaño del círculo.

Ejemplo:

```
//Círculo radio=1 y baja resolución →
circle(r = 1); // ←
translate([8,0,0])
//Círculo radio=5 y baja resolución →
circle(r = 5); // ←
translate([0,15,0])
//Círculo radio=8 y baja resolución →
circle(r = 8); // ←
translate([-4,0,0])
//Círculo radio=2 mm y alta resolución →
scale([1/100, 1/100, 1/100]) circle(200); // ←
translate([0,-5,0])
//Círculo radio=2 mm y alta resolución →
circle(2, $fn=50); // ←
```



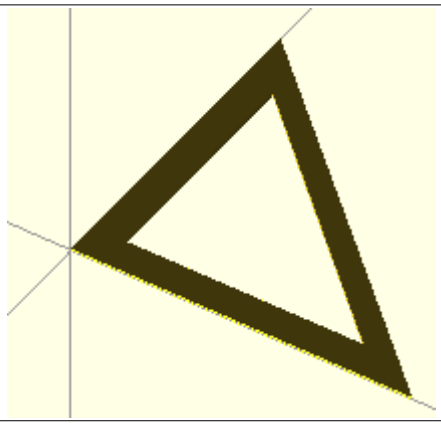
Polígono (polygon)

Crea un polígono con los puntos y caminos especificados.
Sintaxis:

polygon(points = [[x, y], ...], paths = [[p1, p2, p3..], ...], convexity = N);

Ejemplo: Hacer una escuadra con 6 puntos (tres para el triángulo "exterior" y tres para el "interior"). Conectamos los puntos de dos en dos mediante el path.

```
polygon(
points=[
                [0,0],[100,0],[0,100],
                        [10,10],[80,10],[10,80]],
paths=[
                [0,1,2],
                        [3,4,5]
]);
```



Cada elemento de un camino o path se debe corresponder con la posición de un punto definido en el vector de puntos, por ejemplo, "4" se refiere a [80,10].

Importar dxf

Importa un archivo dxf como un objeto 2D.

Ejemplo:

import_dxf(file="design.dxf", layer="layername", origin = [100,100], scale = 0.5);

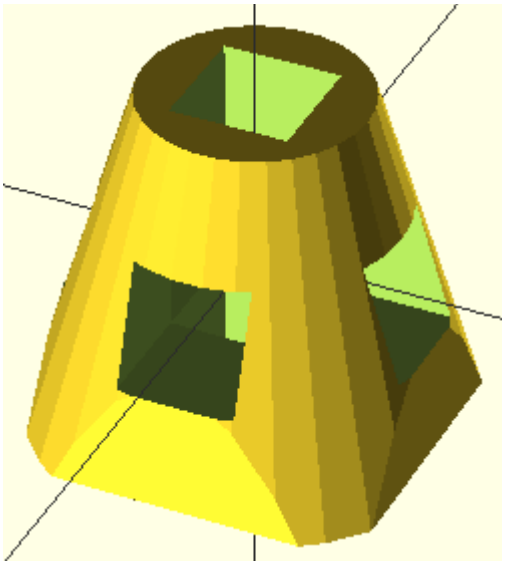
Proyección 3D a 2D

La función projection() permite crear dibujos 2D a partir de modelos 3D y exportarlos a formato dxf. Se realiza una proyección del modelo 3D al plano (x,y) con z = 0. Si cut=true, sólo son considerados los puntos con z=0 (corte efectivo del objeto), con cut=false se consideran los puntos por encima y por debajo del plano (creación de una proyección propia).
Sintaxis:

projection(cut = <boolean>)

Ejemplo: Consideremos el archivo example002.scad que está incluido en OpenSCAD.

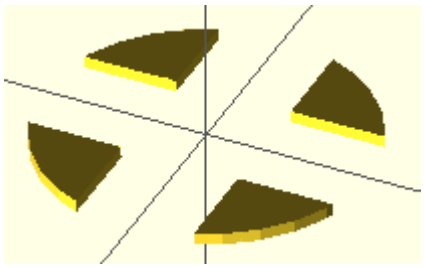
```
module example002()
{
intersection() {
difference() {
union() {
cube([30, 30, 30], center = true);
translate([0, 0, -25])
cube([15, 15, 50], center = true);
}
union() {
cube([50, 10, 10], center = true);
cube([10, 50, 10], center = true);
cube([10, 10, 50], center = true);
}
}
translate([0, 0, 5])
cylinder(h = 50, r1 = 20, r2 = 5, center = true);
}
}
example002();
```



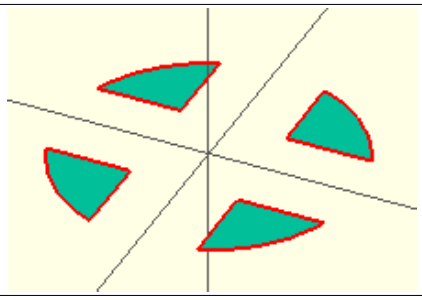
Abrimos (o creamos el archivo) y generamos el código example002.stl tras renderizar con F6. Seguidamente ya podemos ejecutar el siguiente código.
Ejemplo: Hacemos un corte paralelo al plano (x,y) con z=0.

```
projection(cut = true)
import_stl ("example002.stl");
```

F5

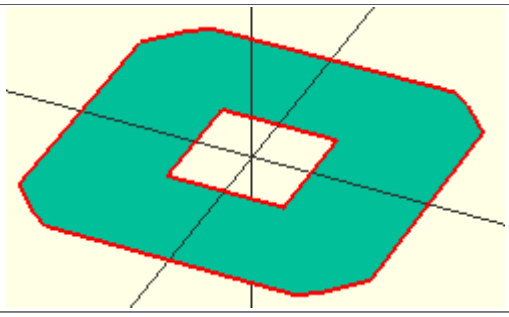


F6



Ejemplo: Hacemos la proyección ordinaria sobre el plano (x,y).

```
projection(cut = false)
import_stl ("example002.stl");
```



Extrusión 2D a 3D

Es posible utilizar los comandos de extrusión para convertir objetos 2D en objetos 3D. Esto se puede hacer con las primitivas 2D vistas, como cuadrados y círculos, pero también con polígonos arbitrarios.

Sintaxis:

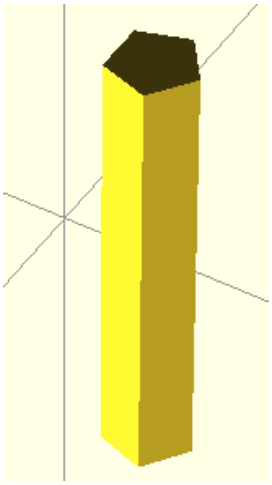
linear_extrude(height=<val>,center=<boolean>,convexity=<val>,twist=<degrees>[,slices=<val>, $fn=...,$fs=...,$fa=...]){...}
rotate_extrude(convexity = <val>[, $fn = ...]){...}

La Extrusión lineal es una operación de modelado que toma un polígono 2D como entrada y lo extiende en la tercera dimensión. De esta manera se crea una forma 3D.

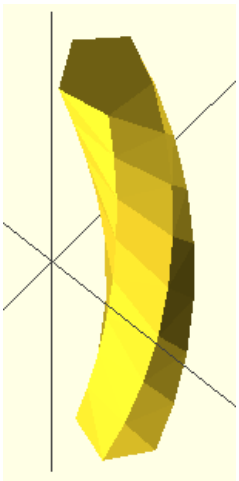
Twist (Giro) es el número de grados a través de los cuales se extruye la forma. Ajustándolo a 360 se extruye a través de una revolución. El sentido de giro sigue la regla de la mano izquierda.

Ejemplo: Distintos grados de giro

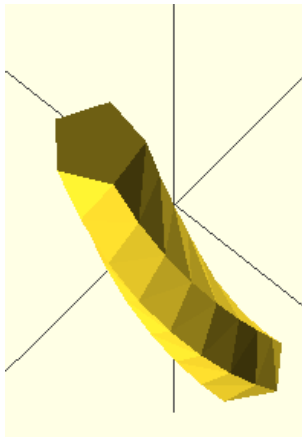
```
linear_extrude(height = 10, center = true, convexity = 10, twist = Giro)
translate([2, 0, 0])
circle(r = 1);
```



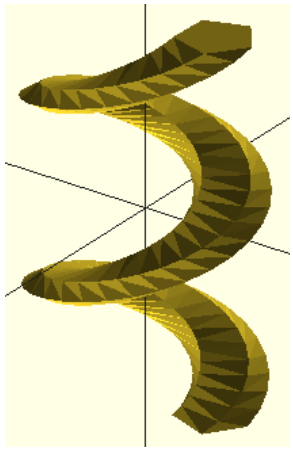
Giro = 0



Giro = -100

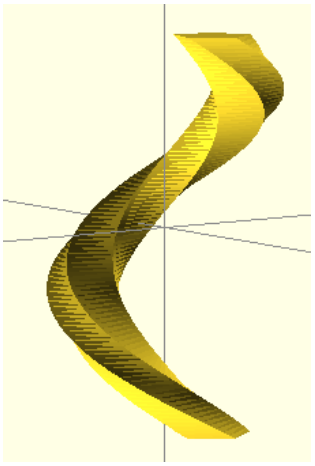


Giro = 100

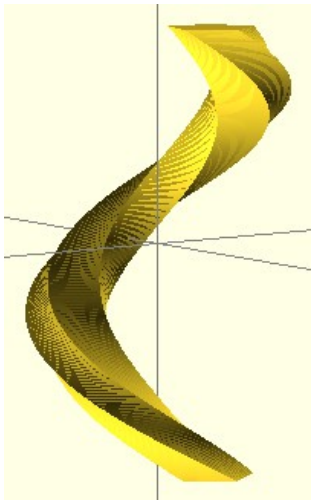


Giro = -720

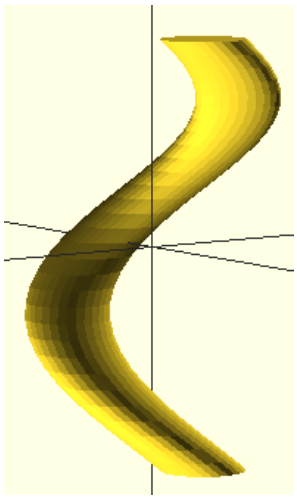
El parámetro slices (rodajas) se puede usar para mejorar la salida.
Las variables especiales $fn, $fs y $fa también se pueden usar con este fin.



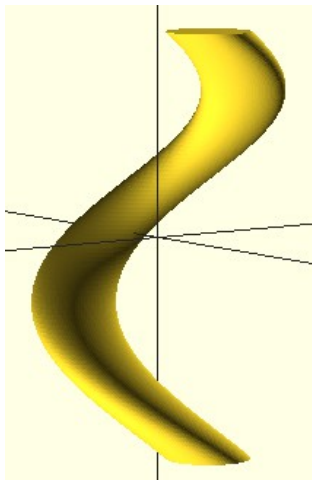
slices = 100



slices = 500



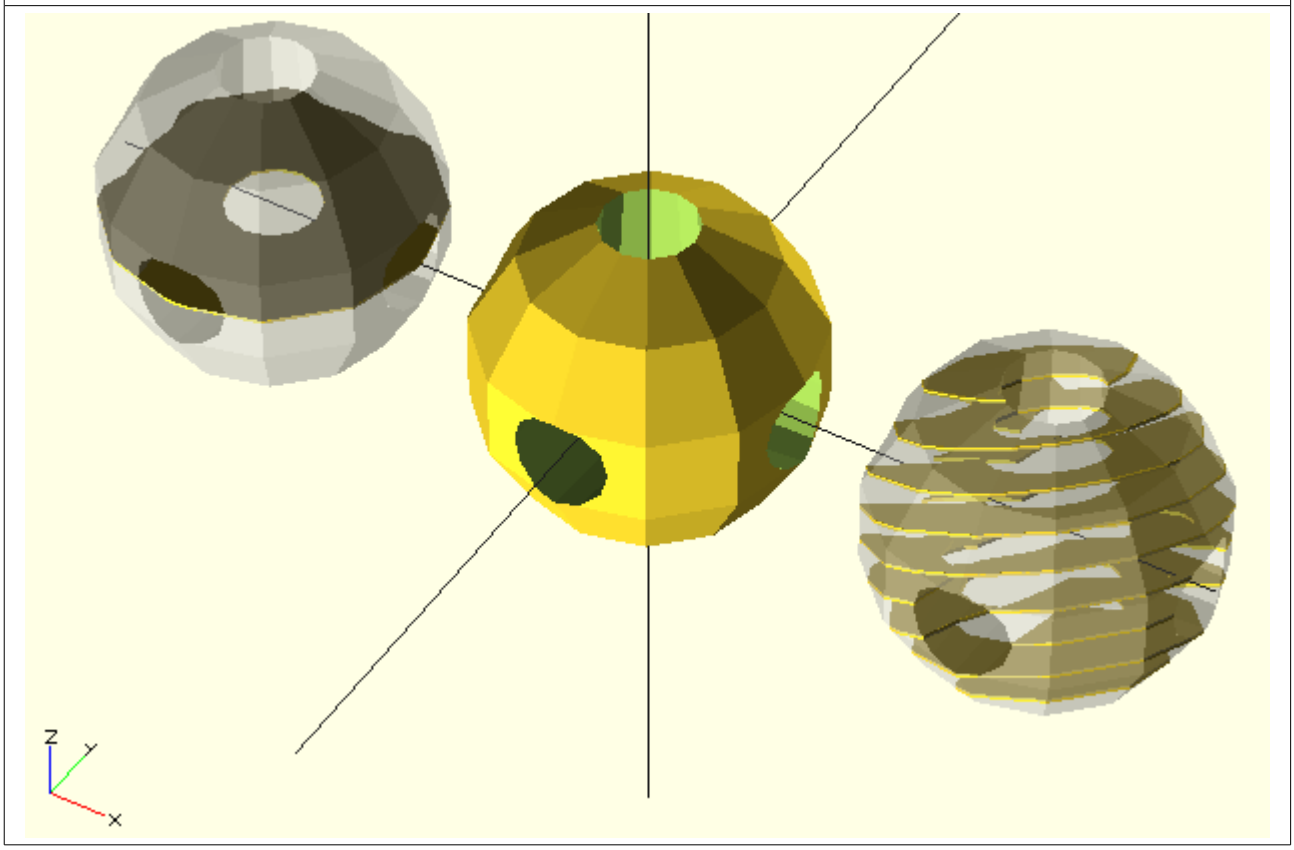
$fn = 30



$fn = 100

Ejemplo: A continuación vemos un ejemplo más complicado tomado del blog de Giles Bathgate.

```
module thing() {
$fa = 30;
  difference() {
    sphere(r = 25);
    cylinder(h = 62.5, r1 = 12.5, r2 = 6.25, center = true);
    rotate(90, [ 1, 0, 0 ]) cylinder(h = 62.5,
    r1 = 12.5, r2 = 6.25, center = true);
    rotate(90, [ 0, 1, 0 ]) cylinder(h = 62.5,
    r1 = 12.5, r2 = 6.25, center = true); }
}
module demo_proj() {
linear_extrude(center = true, height = 0.5) projection(cut = false) thing();
% thing(); }
module demo_cut() {
   for (i=[-20:5:+20]) {
   rotate(-30, [ 1, 1, 0 ]) translate([ 0, 0, -i ])
   linear_extrude(center = true, height = 0.5) projection(cut = true)
   translate([ 0, 0, i ]) rotate(+30, [ 1, 1, 0 ]) thing(); }
% thing(); }
thing();
translate([ -60, 0, 0 ]) demo_proj();
translate([ +60, 0, 0 ]) demo_cut();
```

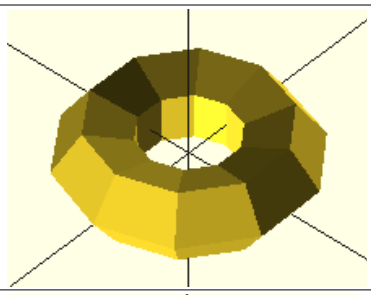


Rotar extrusión

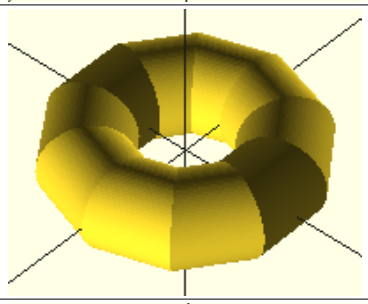
Rotar una extrusión es simplemente una extrusión lineal con giro.

Ejemplo:

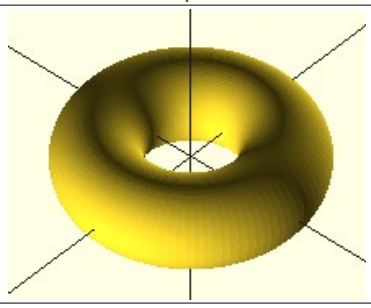
```
//Un toro sencillo
rotate_extrude(convexity = 10)
translate([2, 0, 0])
circle(r = 1);
```



```
//Definimos los círculos
rotate_extrude(convexity = 10)
translate([2, 0, 0])
circle(r = 1, $fn = 100);
```



```
//Definimos los círculos y la extrusión
rotate_extrude(convexity = 10, $fn = 100)
translate([2, 0, 0])
circle(r = 1, $fn = 100);
```

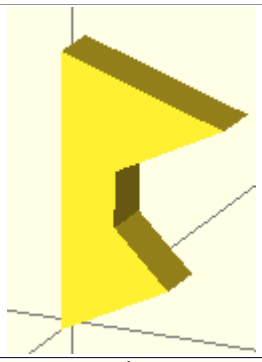


Extrusión de un polígono

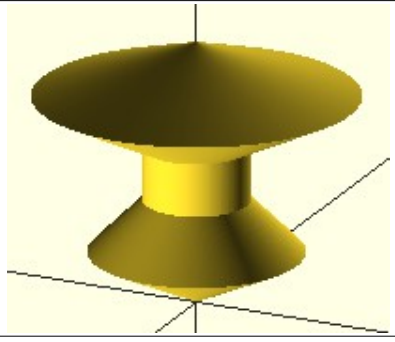
La extrusión también se puede realizar en polígonos con puntos elegidos por el usuario.

Ejemplo:

```
//Polígono simple definido por puntos
rotate([90,0,0])
polygon(points=[[0,0],[2,1],[1,2],[1,3],[3,4],[0,5]]);
```



```
//Extrusión del polígono
rotate_extrude($fn=200)
polygon(points=[[0,0],[2,1],[1,2],[1,3],[3,4],[0,5]]);
```



Extrusión DXF

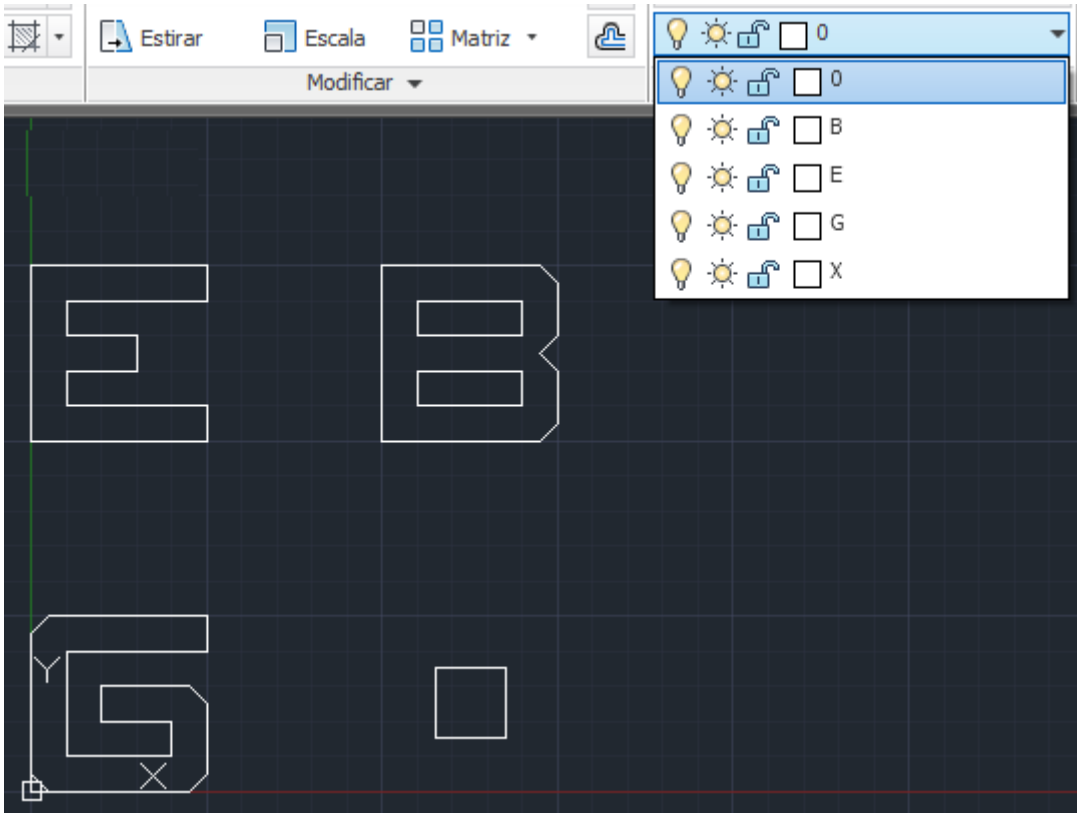
Con las declaraciones de extrusión DXF podemos convertir archivos DXF en 2D directamente en objetos 3D.

Extrusión dxf lineal

Sintaxis:

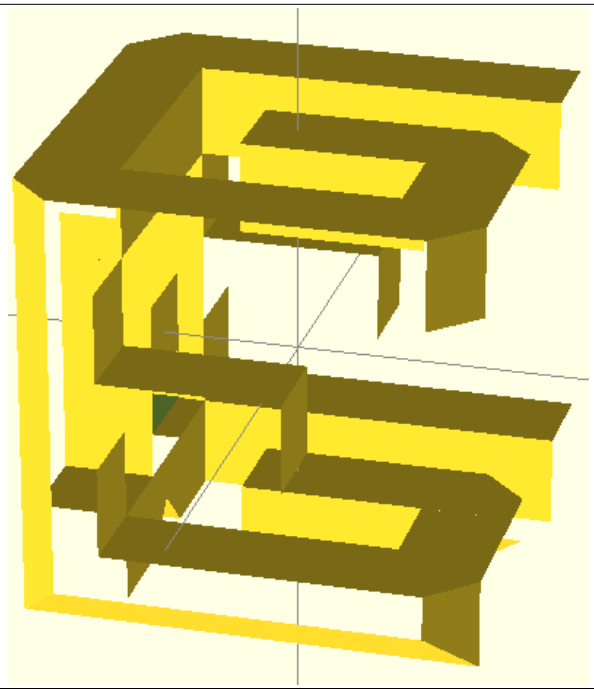
```
linear_extrude(file = "example008.dxf", layer = "G");
```

En la carpeta de instalación del programa está el example008.scad y en esta misma carpeta el archivo example008.dxf que es un dibujo en 2D de las letras E, B, G y un punto con las capas que vemos en la imagen dxf.



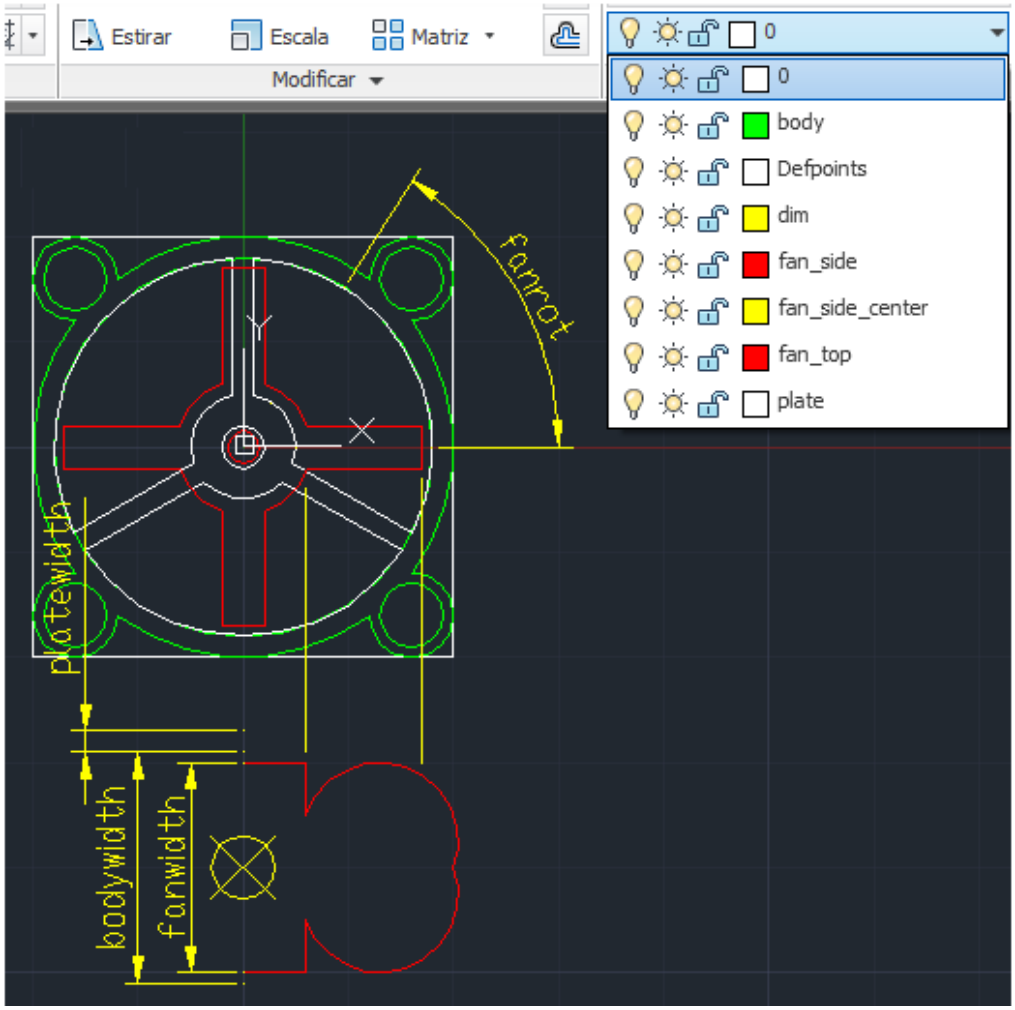
Ejemplo:

```
difference()
{
intersection()
{
translate([ -25, -25, -25])
linear_extrude(height = 50, convexity = 3)
import(file = "example008.dxf", layer = "G");
rotate(90, [1, 0, 0])
translate([ -25, -125, -25])
linear_extrude(height = 50, convexity = 3)
import(file = "example008.dxf", layer = "E");
rotate(90, [0, 1, 0])
translate([ -125, -125, -25])
linear_extrude(height = 50, convexity = 3)
import(file = "example008.dxf", layer = "B");
}

intersection()
{
translate([ -125, -25, -26])
linear_extrude(height = 52, convexity = 1)
import(file = "example008.dxf", layer = "X");

rotate(90, [0, 1, 0])
translate([ -125, -25, -26])
linear_extrude(height = 52, convexity = 1)
import(file = "example008.dxf", layer = "X");
}
}
```



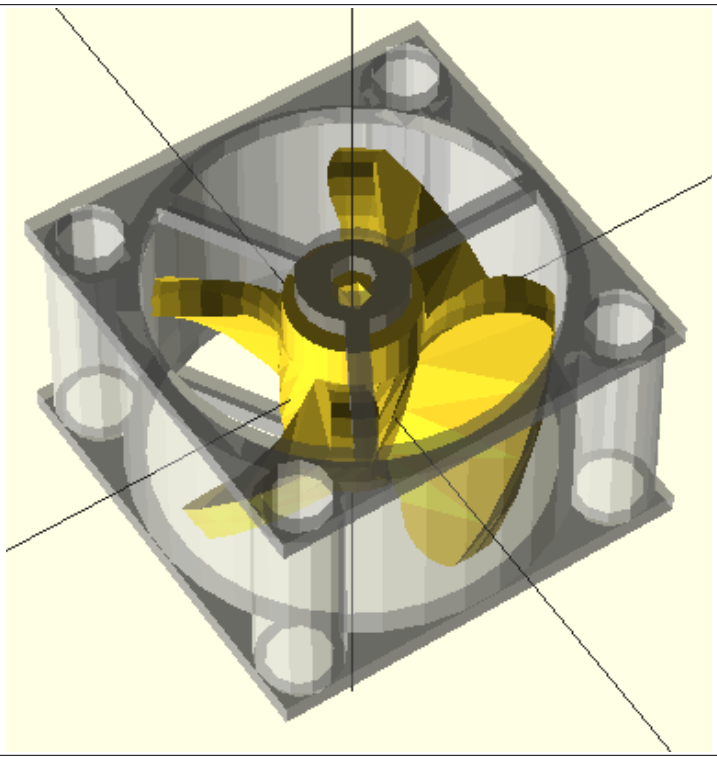
Rotar extrusión dxf

Rotar una extrusión es simplemente una extrusión lineal con giro.
En la carpeta de instalación del programa está el example009.scad y en esta misma carpeta el archivo example009.dxf que es un dibujo en 2D de un ventilador de ordenador con las capas que vemos en la imagen dxf.



Ejemplo:

```
bodywidth = dxf_dim(file = "example009.dxf", name = "bodywidth");
fanwidth = dxf_dim(file = "example009.dxf", name = "fanwidth");
platewidth = dxf_dim(file = "example009.dxf", name = "platewidth");
fan_side_center = dxf_cross(file = "example009.dxf", layer = "fan_side_center");
fanrot = dxf_dim(file = "example009.dxf", name = "fanrot");

% linear_extrude(height = bodywidth, center = true, convexity = 10)
import(file = "example009.dxf", layer = "body");

% for (z = [+(bodywidth/2 + platewidth/2),
-(bodywidth/2 + platewidth/2)])
{
translate([0, 0, z])
linear_extrude(height = platewidth, center = true, convexity = 10)
import(file = "example009.dxf", layer = "plate");
}

intersection()
{
linear_extrude(height = fanwidth, center = true, convexity = 10, twist = -fanrot)
import(file = "example009.dxf", layer = "fan_top");
// NB! We have to use the deprecated module here since the "fan_side"
// layer contains an open polyline, which is not yet supported
// by the import() module.
rotate_extrude(file = "example009.dxf", layer = "fan_side",
origin = fan_side_center, convexity = 10);
}
```



Caracteres modificadores

Los cambios de color provocados por los modificadores de caracteres sólo se muestra en modo "Compile" no en modo "Compile and Render (CGAL)".

Modificador de fondo

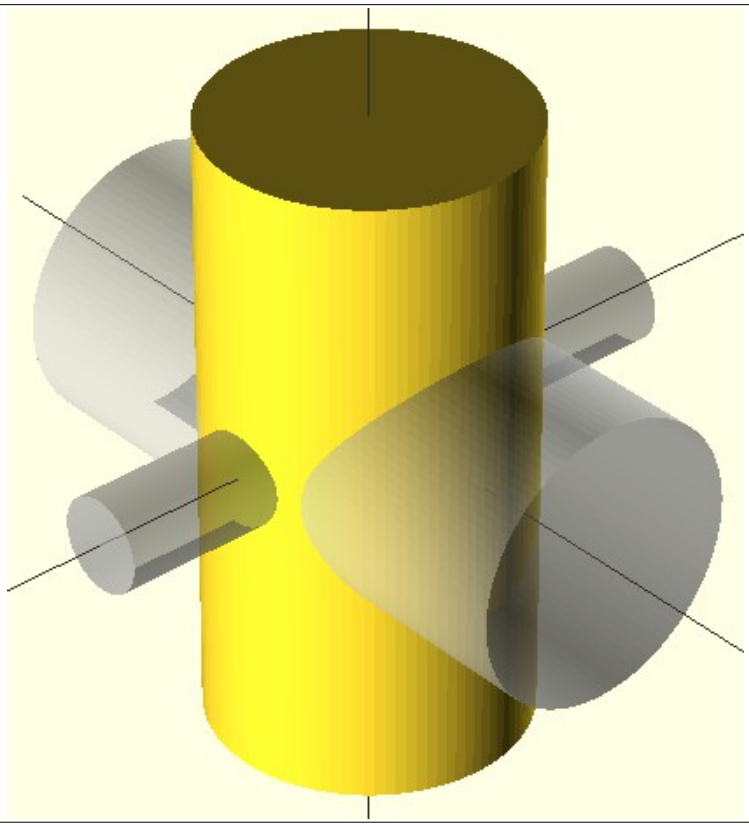
Ignora el CSG (Constructive solid geometry) del subcódigo para dibujarlo en gris transparente.

Sintaxis:

{% ... }

Ejemplo:

```
difference() {
// Objeto inicial
cylinder (h = 4, r=1, center = true, $fn=100);
// Primer objeto que se resta
% rotate ([90,0,0]) cylinder (h = 4, r=0.3, center = true, $fn=100);
// Segundo objeto que se resta
% rotate ([0,90,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```



Modificador de depuración

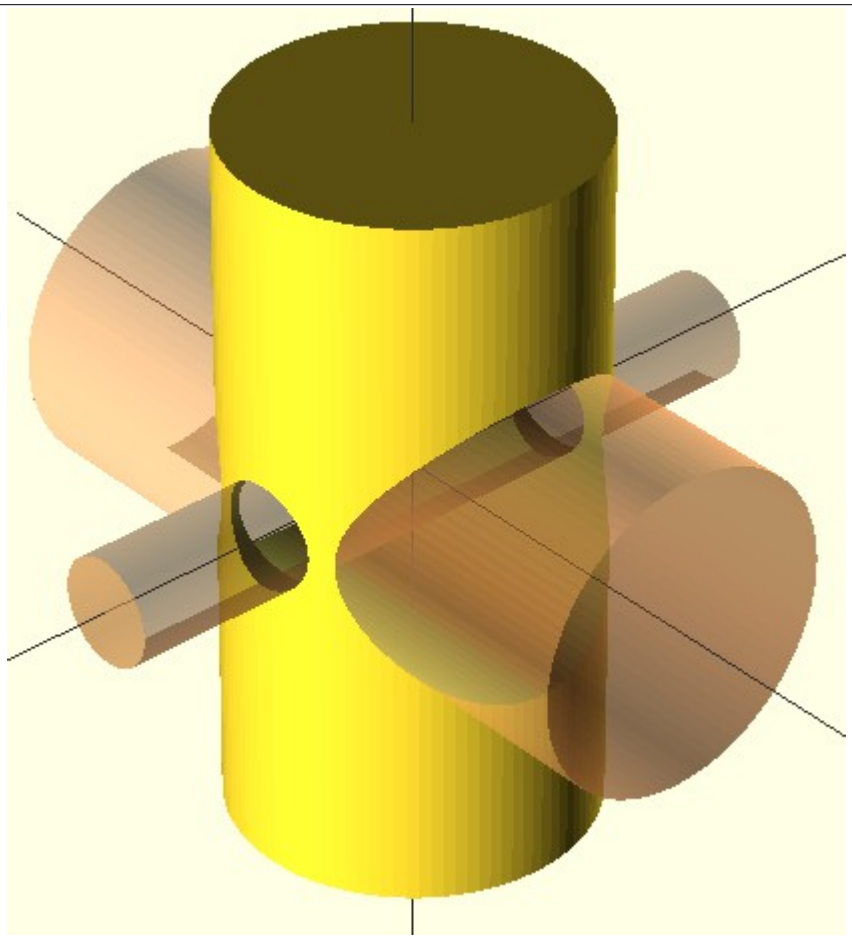
Es habitual su uso en el proceso de renderizado, pero también para dibujar en rosa transparente.

Sintaxis:

{... }

Ejemplo:

```
difference() {
// Objeto inicial
cylinder (h = 4, r=1, center = true, $fn=100);
// Primer objeto que se resta
# rotate ([90,0,0]) cylinder (h = 4, r=0.3, center = true, $fn=100);
// Segundo objeto que se resta
# rotate ([0,90,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```



Modificador Root

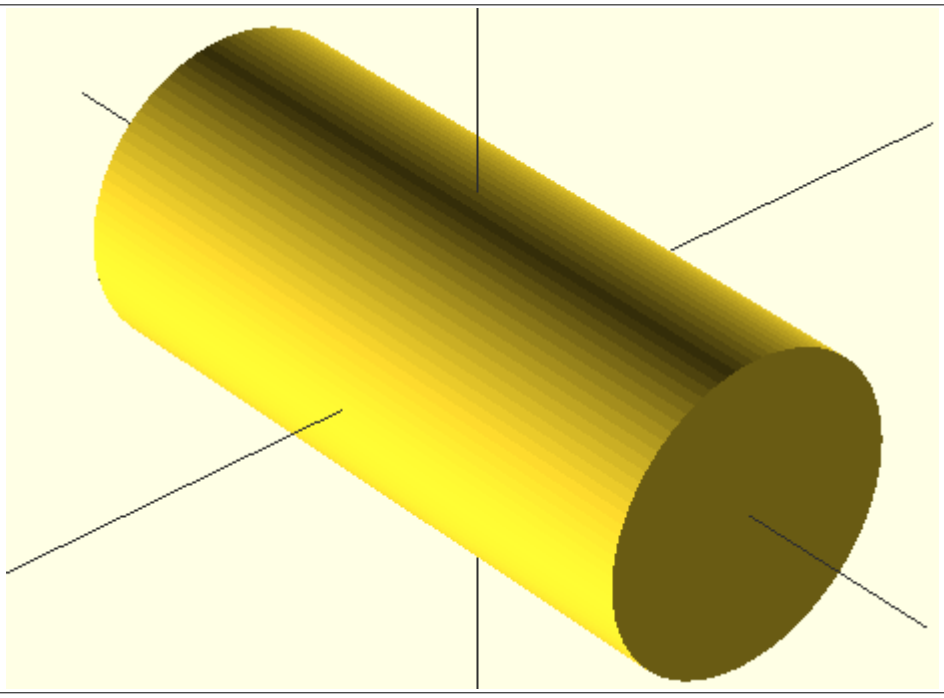
Ignora el resto del diseño y usa este subcódigo como raíz del diseño.

Sintaxis:

! {... }

Ejemplo:

```
difference() {
// Objeto inicial
cylinder (h = 4, r=1, center = true, $fn=100);
// Primer objeto que se resta
% rotate ([90,0,0]) cylinder (h = 4, r=0.3, center = true, $fn=100);
// Segundo objeto que se resta
! rotate ([0,90,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```



Desactivar Modificador

Ignorar este código

Sintaxis:

* {... }

Ejemplo:

```
difference() {
// Objeto inicial
cylinder (h = 4, r=1, center = true, $fn=100);
// Primer objeto que se resta
% rotate ([90,0,0]) cylinder (h = 4, r=0.3, center = true, $fn=100);
// Segundo objeto que se resta
* rotate ([0,90,0]) cylinder (h = 4, r=0.9, center = true, $fn=100);
}
```