

Introducción

En esta práctica veremos cómo diseñar modelos 3D utilizando el programa OpenSCAD. Para ello revisaremos los aspectos principales de la herramienta y del lenguaje de programación que se utiliza en la definición de los modelos.

Una vez introducidos los conceptos básicos se diseñarán los modelos necesarios para construir un sencillo actuador lineal.

En la parte final de la práctica se revisarán conceptos básicos sobre la configuración de impresión de modelos utilizando el software Cura.

Diseño de modelos 3D

OpenSCAD

Es un software gratuito diseñado para crear objetos 3D. A diferencia de otras herramientas similares OpenSCAD no se basa en un modelado activo clásico (con una paleta de herramientas, propiedades, etc.) sino en la compilación de scripts que generan un renderizado 3D.

Existe una gran comunidad de esta herramienta de donde podemos obtener material didáctico, scripts para la creación de objetos y diversas librerías.

<http://www.openscad.org/>

Documentación completa del lenguaje:

https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language

Resumen de comandos principales:

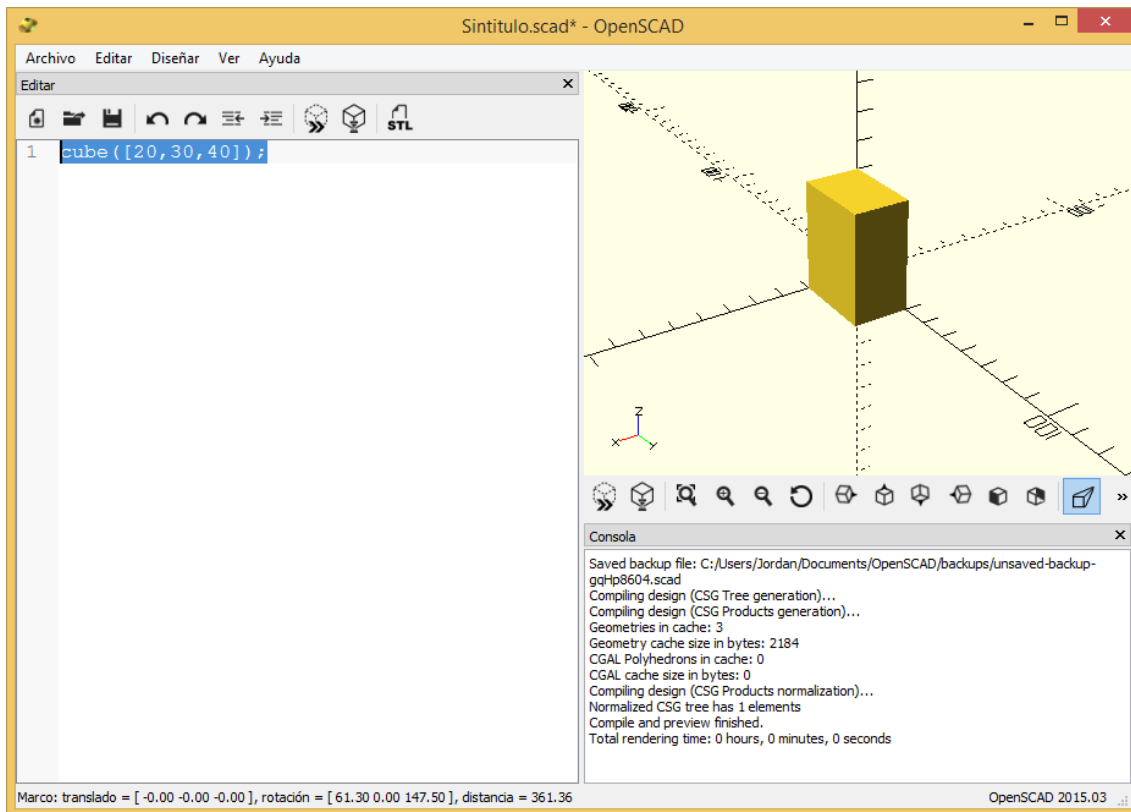
<http://www.openscad.org/cheatsheet/>

Introducción

El ejemplo más básico sería definir una primitiva de tipo **cubo**. Un cubo de 20mm x 30mm x 4mm. OpenScad utiliza como unidad de medida básica el **milímetro**.

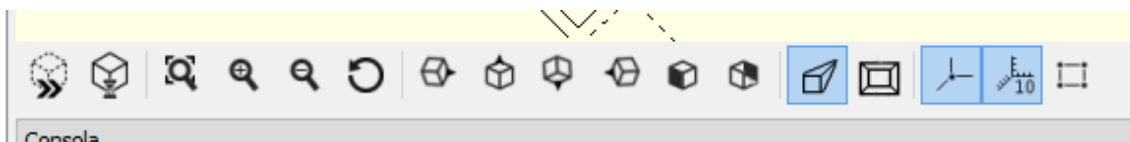
```
cube([20,30,40]);
```

Para compilar el script debemos pulsar el botón **F5**, a continuación se cargará nuestro diseño en la superficie de renderizado.



Podemos modificar la perspectiva de la superficie de renderizado arrastrando el ratón sobre ella (ángulo de visión) y con la rueda del ratón (tamaño).

Tenemos varios botones que nos permiten alinear la pieza respecto a un eje y modificar la información mostrada en los ejes.



Al mover la renderización de la vista es bastante fácil perder la perspectiva de lo que estamos viendo, debemos fijarnos bien en el eje que aparece en la parte inferior izquierda.



Para exportar el diseño en formato STL e imprimirlo posteriormente, debemos compilarlo pulsando F6, a continuación ya podemos obtener el fichero STL. **Archivo -> Export -> Export as STL.**

Más adelante veremos cómo introducir el modelo en el software Cura para estimar el tiempo que tardará en imprimirse la pieza y ver las características de la impresión.

Primitivas

Cubo

Recibe un array de valores numéricos con el tamaño correspondiente en el eje X, Y, Z.

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

Sí queremos construir un cuadrado podemos enviarle un único valor numérico como parámetro.

```
cube (20) ;
```

Esfera

Se pueden crear esferas a partir del radio (r) o el diámetro (d), si no indicamos el tipo de parámetro por defecto será el radio.

```
sphere (d = 20) ;
```

Un aspecto muy importante de la esfera es la **resolución \$fn**, que define el número de

fragmentos que se utilizan para construir la esfera. Observamos que a mayor resolución más lados (polígonos) se utilizan en la definición modelo.

```
sphere(d = 20, $fn = 20);
```

```
sphere(d = 20, $fn = 200);
```

Cilindro

Es uno de los elementos más utilizados gracias a su versatilidad. Los cilindros más básicos se definen en función a un radio (r) o diámetro (d) y una altura (h).

```
cylinder(h = 10, r=20);
```

Existe la posibilidad de que el cilindro no tenga el mismo radio en el lado superior y lado inferior, para ello utilizamos r1 y r2 o d1 y d2.

```
cylinder(h = 10, r1=20, r2=10 );
```

```
cylinder(h = 10, r1=20, r2=0 );
```

Al igual que ocurría con las esferas también podemos especificar el número de polígonos que se utilizan en la construcción de la figura **\$fn**.

```
cylinder(h = 10, r=20, $fn=200);
```

El parámetro **\$fn** nos resultará muy útil para construir **diferentes tipos de polígonos**.

```
cylinder(h = 10, r=20, $fn=5);
```

Poliedro

Nos permite crear una forma a partir de una lista de puntos y de triángulos que se utilizarán para definir las caras del objeto.

Paso 1: se declara primero una lista con todos los puntos que tenga el objeto, por ejemplo 5.

```
polyhedron(  
  points=[ [10,10,0],[10,-10,0],[-10,-10,0],[-10,10,0], [0,0,10] ]  
);
```

Paso 2: Cada punto tiene un índice, 0,1,2,3,4 (es el orden de especificación). Declaramos todas las caras triangulares que queremos que tenga el objeto. La primera une el punto 0, 1, 4

```
polyhedron(  
  points=[ [10,10,0],[10,-10,0],[-10,-10,0],[-10,10,0], [0,0,10] ],  
  faces=[ [0,1,4] ]  
);
```

```
polyhedron(  
  points=[ [10,10,0],[10,-10,0],[-10,-10,0],[-10,10,0], [0,0,10] ],  
  faces=[ [1,2,4],[2,3,4],[3,0,4],[1,0,3],[2,1,3] ]  
);
```

Manipulación de formas

Centrar

Podemos centrar cualquier forma en los ejes X, Y, Z utilizando la propiedad **center = true**.

```
cylinder(h= 20, d = 10, center = true);
```

Trasladar

Traslada la forma en el eje X,Y,Z aplicando el vector especificado.

```
translate([15,0,0])  
cylinder(h= 20, d = 10, center = true);
```

Rotación

Rota un elemento en el eje X,Y,Z aplicando el vector de grados especificado.

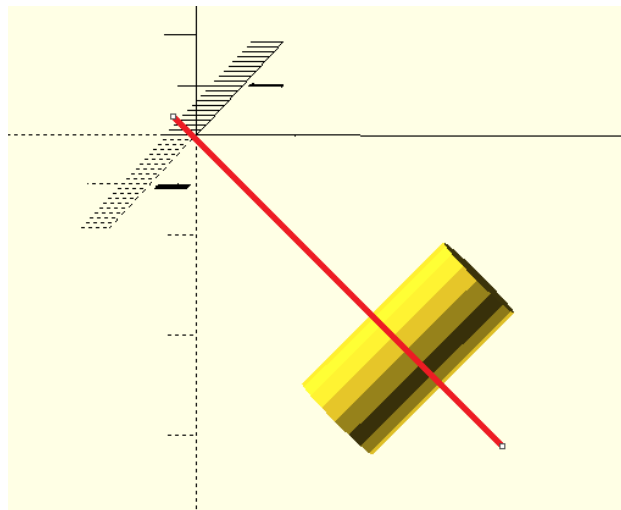
```
rotate([0,45,0])  
cylinder(h= 20, d = 10, center = true);
```

Todas las transformaciones que declaramos se aplican en orden.

Ejemplo 1: queremos **rotar 45º grados (eje Y), trasladar 30 mm eje X.**

```
rotate([0,45,0])  
translate([30,0,0])  
cylinder(h= 20, d = 10, center = true);
```

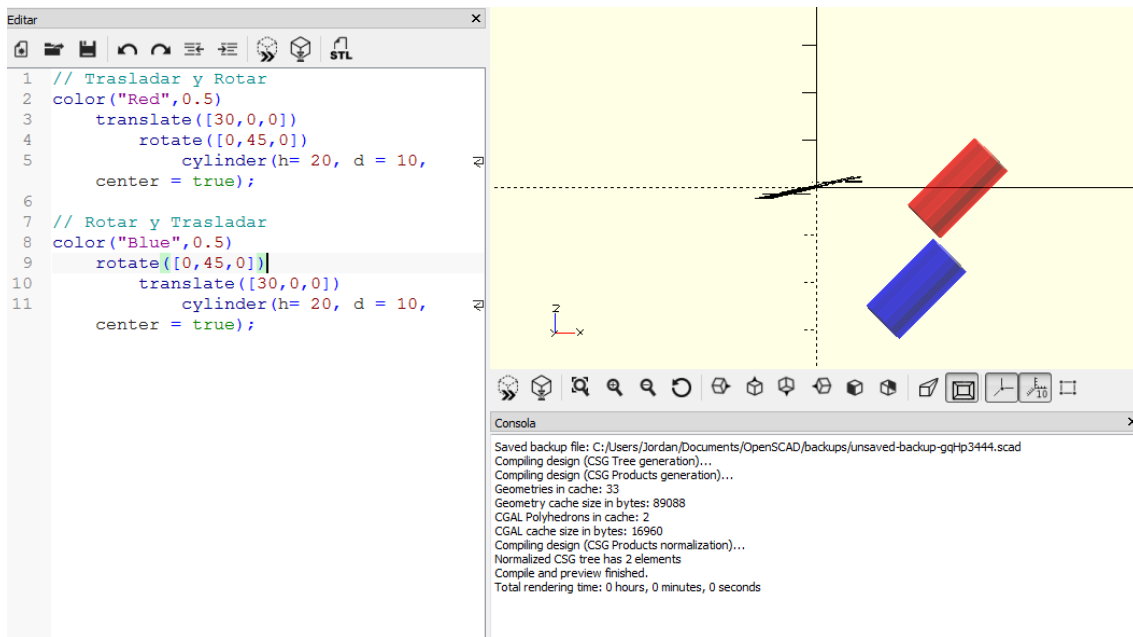
En este ejemplo se puede apreciar con claridad que al rotar 45º el eje del modelo cambia.



Ejemplo 2: queremos **trasladar 30 mm eje X, rotar 45º grados (eje Y)**

En este caso se realizó una rotación en el eje original y luego se trasladó.

```
translate([30,0,0])  
rotate([0,45,0])  
cylinder(h= 20, d = 10, center = true);
```



Agrupaciones

Podemos hacer que las transformaciones afecten a grupos de elementos en lugar de a un único hijo, en lugar de especificar una transformación para cada elemento.

```

translate([15,0,0])
  cylinder(h= 20, d = 10, center = true);

translate([15,0,0])
  cube([20,20,2], center = true);

translate([15,0,0]){
  cylinder(h= 20, d = 10, center = true);
  cube([20,20,2], center = true);
}

```

Escalar

Escala un elemento en el eje X,Y,Z aplicando el vector de escalado especificado.

Es muy útil sobre todo para escalar piezas complejas, formadas a partir de varios elementos.

Podemos definir un factor de escalado que se aplica en todas las direcciones.

```

scale(1)
  cylinder(h= 20, d = 10);

```

O un factor diferente para cada eje, al escalar algo en el eje X,Y,Z podemos modificar su relación de aspecto.


```
scale([0.5,1,2])
cylinder(h= 20, d = 10);
```

Unión de formas

La función unión une todos los nodos especificados.

Ejemplo 1: unir un cubo y un cilindro.

```
union() {
  cube([20,20,1], center = true);

  translate([0,0,10])
    cylinder(h= 20, d = 10, center = true);
}
```

Ejemplo 2: unir dos cilindros.

```
union() {
  cylinder (h = 20, d=10, center = true, $fn=100);

  rotate ([90,0,0])
    cylinder (h = 20, d=10, center = true, $fn=100);
}
```

A partir de la unión podemos manejar los elementos unidos como un solo elemento.

```
translate([0,0,30])
union() {
  cylinder (h = 20, d=10, center = true, $fn=100);

  rotate ([90,0,0])
    cylinder (h = 20, d=10, center = true, $fn=100);
}
```

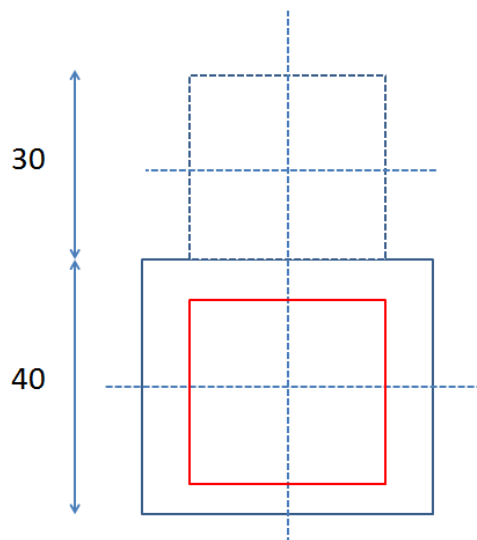
Calculo de posiciones para uniones y diferencias

La posición de las piezas hace referencia a su eje central.

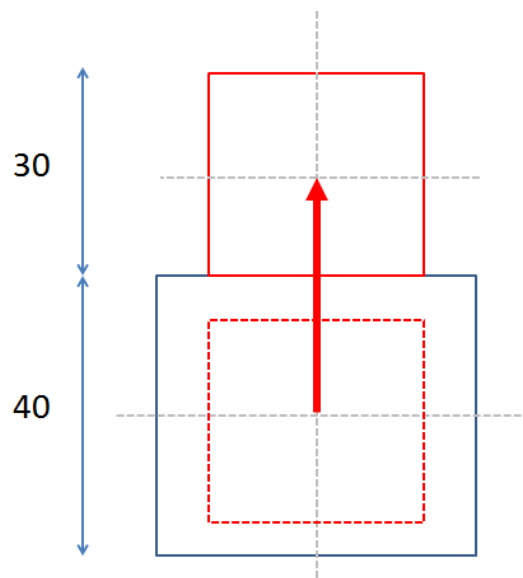
Supongamos que queremos hacer una pirámide con dos cubos de 40x40x40 y 30x30x30.

```
union() {
  cube([40,40,40], center = true);
  cube([30,30,30], center = true);
}
```

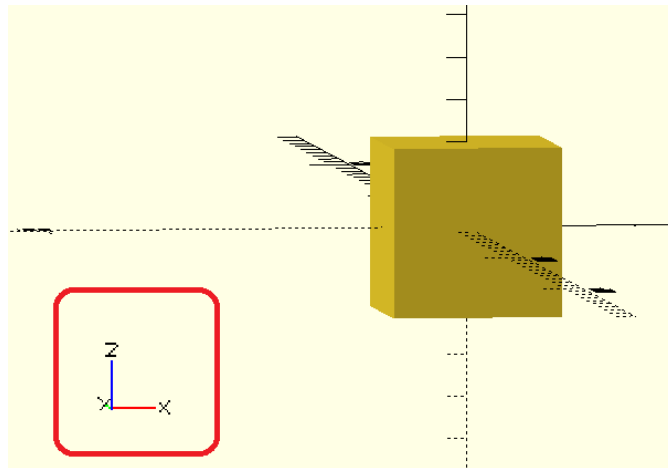
¿Cuánto hay que trasladar el cubo pequeño (rojo en el dibujo) para situarlo exactamente encima del grande?



Como la **referencia de movimiento es el punto medio** serian $40/2 + 30/2$.



Observamos en el programa que la altura se corresponde con el eje Z.



```
union() {
  cube([40,40,40], center = true);
  translate([0,0,40/2 + 30/2])
    cube([30,30,30], center = true);
}
```

La posición de los ejes X, Y, Z cambia dependiendo del punto de vista que utilizemos para diseñar la pieza. Realmente es muy relevante porque obtendremos el mismo modelo tomemos una referencia u otra.

En estos ejemplos siempre se está usando

- X = ancho
- Y = profundidad
- Z = alto

Unión con hull (tangentes)

Permite unir varios modelos mediante tangentes creando una única pieza.

En el siguiente ejemplo vamos a unir estos dos modelos utilizando sus tangentes.

```
cylinder (h = 10, d = 10, center = true);
translate([30,0,0])
  cylinder (h = 30, d = 30, center = true);
```

Introduciendo los modelos dentro del grupo **hull () { ... }** los uniré .

```
hull(){
  cylinder (h = 10, d = 10, center = true);

  translate([30,0,0])
    cylinder (h = 20, d = 30, center = true);
}
```

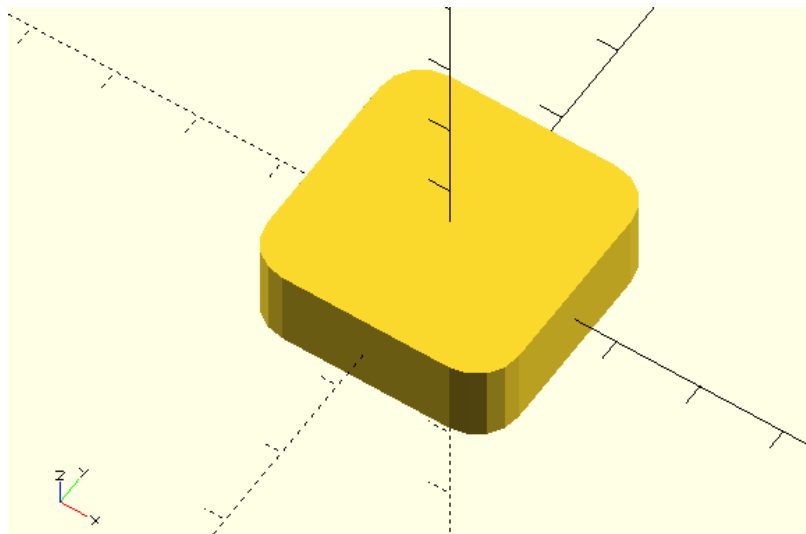
Esta función se puede utilizar para crear formas redondeadas fácilmente.

```
hull(){
  translate([-10,10,0])
    cylinder (h = 10, d = 10, center = true);

  translate([-10,-10,0])
    cylinder (h = 10, d = 10, center = true);

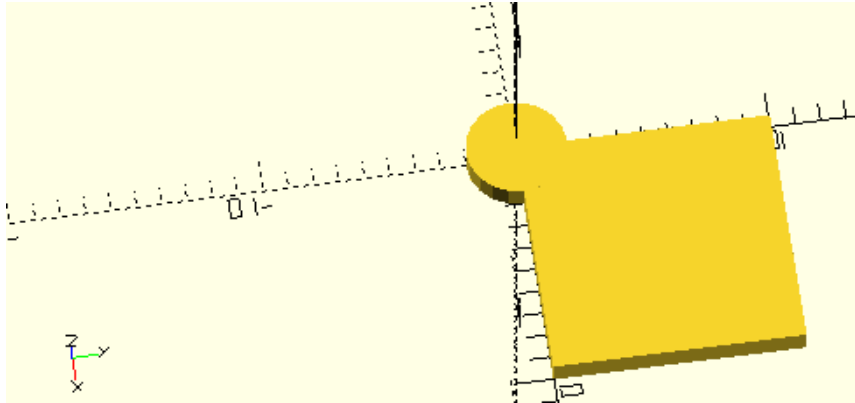
  translate([10,10,0])
    cylinder (h = 10, d = 10, center = true);

  translate([10,-10,0])
    cylinder (h = 10, d = 10, center = true);
}
```



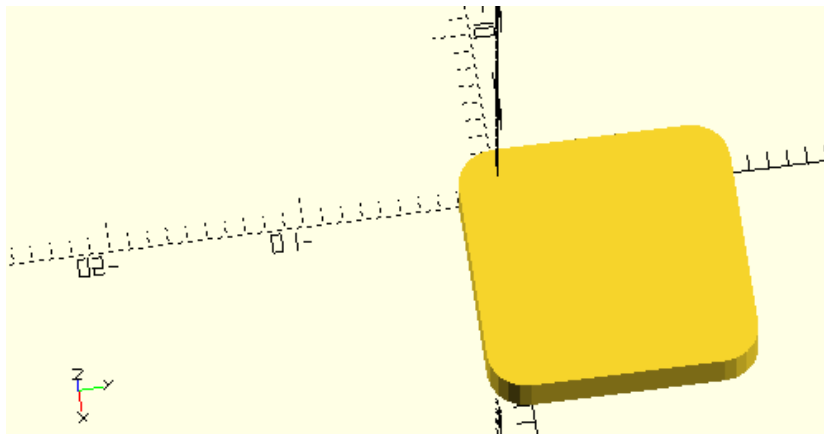
Otra forma de crear piezas con diferentes tipos de bordes es utilizando la operación **minkowski**. Esta operación crea una nueva pieza a partir del primer modelo, agregando otro (u otros) modelo en los puntos de su borde y realizando uniones similares a las del comando hull.

```
cube([10,10,1]);  
cylinder(d=4,h=1, $fn = 20);
```



Si aplicamos la operación **minkowski** al cubo con el cilindro obtendremos lo siguiente.

```
minkowski() {  
  cube([10,10,1]);  
  cylinder(d=4,h=1, $fn = 20);  
}
```



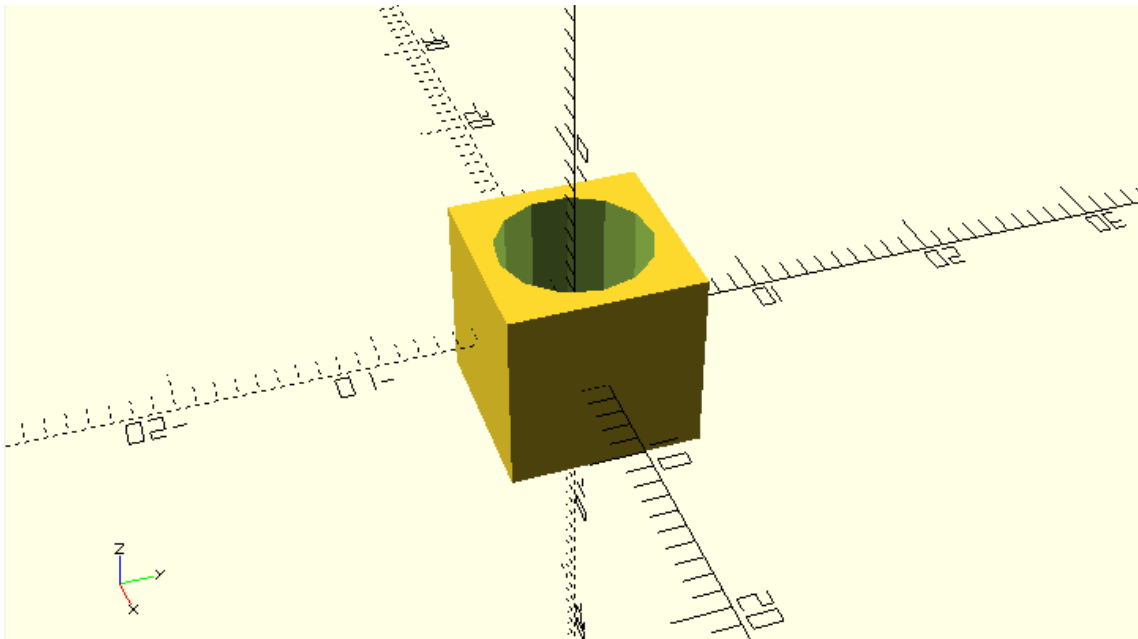
Se pueden combinar varios elementos, por ejemplo, primero aplicar un cilindro y luego una esfera.

```
minkowski() {
  cube([10,10,1]);
  cylinder(d=4,h=1, $fn = 20);
  sphere(d = 4, h = 4, center = true);
}
```

Diferencias

Sustrae al primer elemento todos los elementos que se declaran a continuación, sería algo así como **hacer un agujeros** al primer elemento con las formas que se declaran a continuación.

```
difference() {
  cube (10, center = true);
  cylinder (h = 11, r= 4, center = true);
}
```

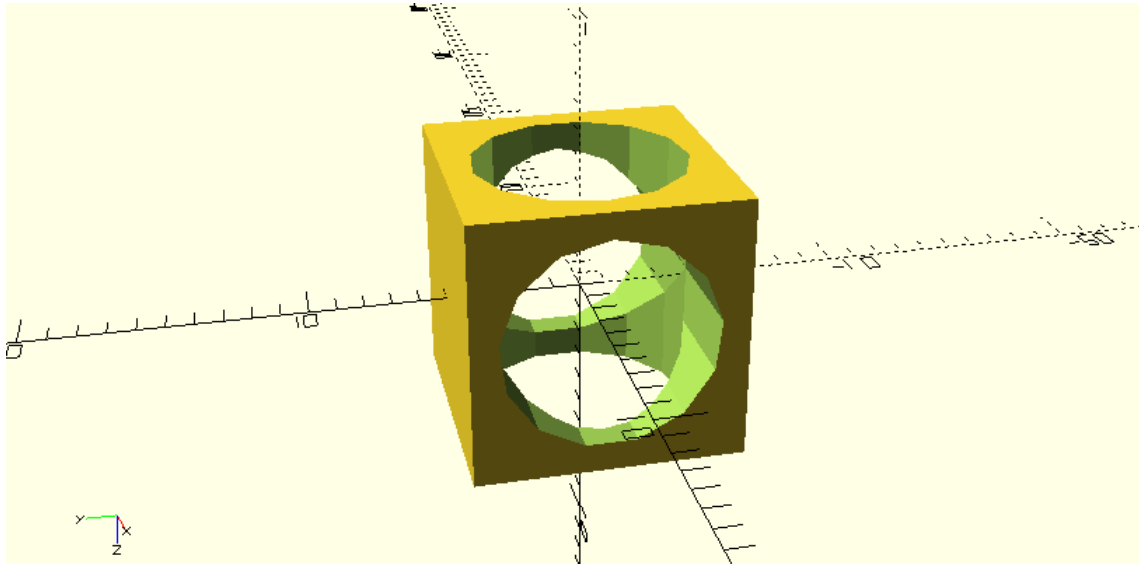


Es recomendable que el elemento que va a hacer el agujero sea más alto que el original.

Es decir para hacer un agujero en un cuadrado de 10mm utilizar un cilindro de 12mm o incluso mucho más.

Todos los elementos que declaramos se aplican como diferencia sobre el primero

```
difference() {  
    cube (10, center = true);  
    cylinder (h = 11, r = 4, center = true);  
    rotate([0,90,0])  
        cylinder (h = 11, r = 4, center = true);  
}
```

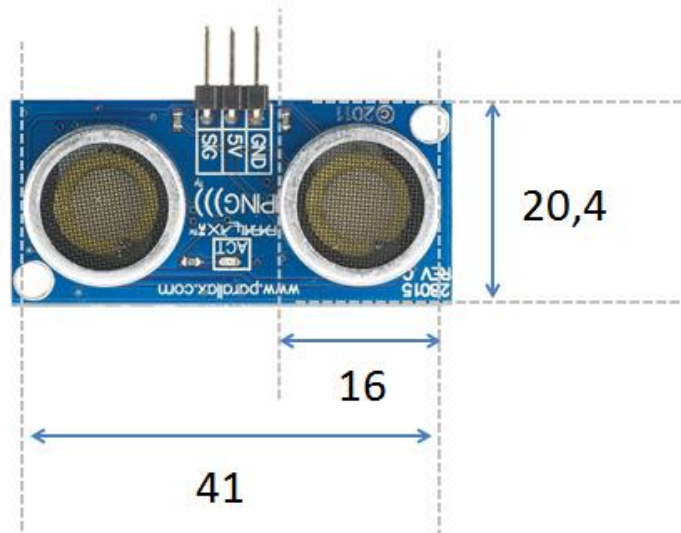


Gracias a la **diferencia** y la **unión** podemos crear piezas complejas con primitivas simples.

Ejemplo: soporte para sensor

Vamos a diseñar un soporte para un sensor de ultrasonidos, el primera paso consiste en obtener las dimensiones del sensor.

Podemos obtener las dimensiones de la web del fabricante o midiendo la pieza con un calibre.



Medidas necesarias tomadas directamente del sensor

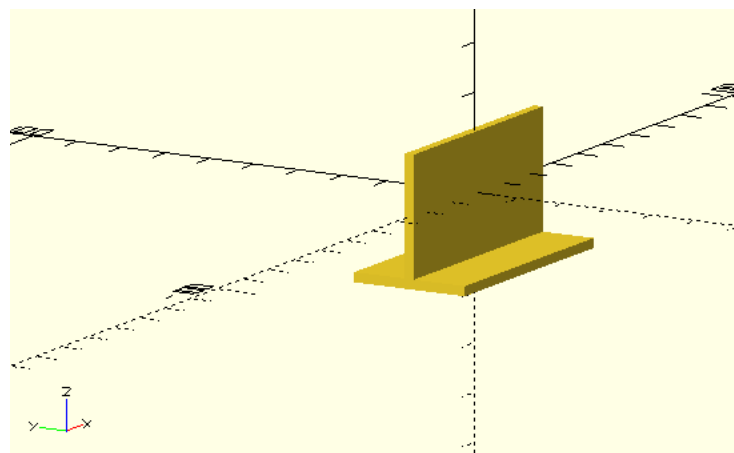
Colocamos una superficie plana de unos 50mm de ancho, 2mm profundidad y 25mm de alto (para que sobre espacio).

```
cube ([50,2,25], center = true);
```

Le unimos una base plana en la parte inferior, (igual que la anterior pero rotada 90° en el eje x)

```
union() {
  cube ([50,2,25], center = true);

  translate([0,0,-25/2 - 2/2])
  rotate([90,0,0])
  cube ([50,2,25], center = true);
}
```



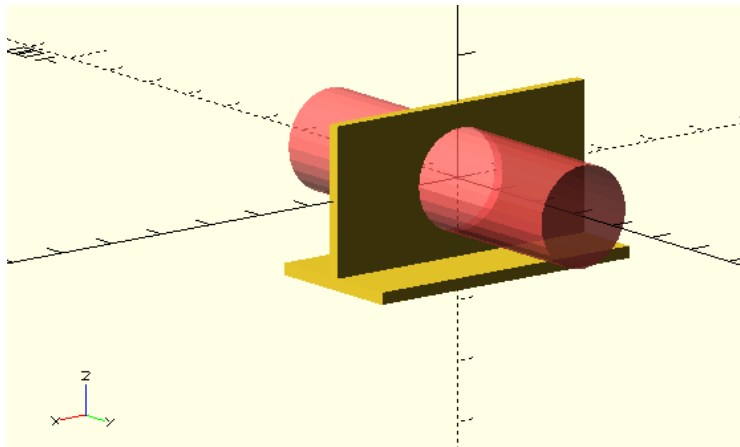
Utilizando la diferencia incluiremos los dos agujeros que se necesitan para encajar la pieza. Según las medidas son de 16mm pero **debemos hacerlos al menos de 17mm** (la impresora suele hacer los cilindros unos mm más pequeños de lo indicado).

La separación entre los cilindros según las medidas tomadas es de $41 - 16 * 2 = 9\text{mm}$.

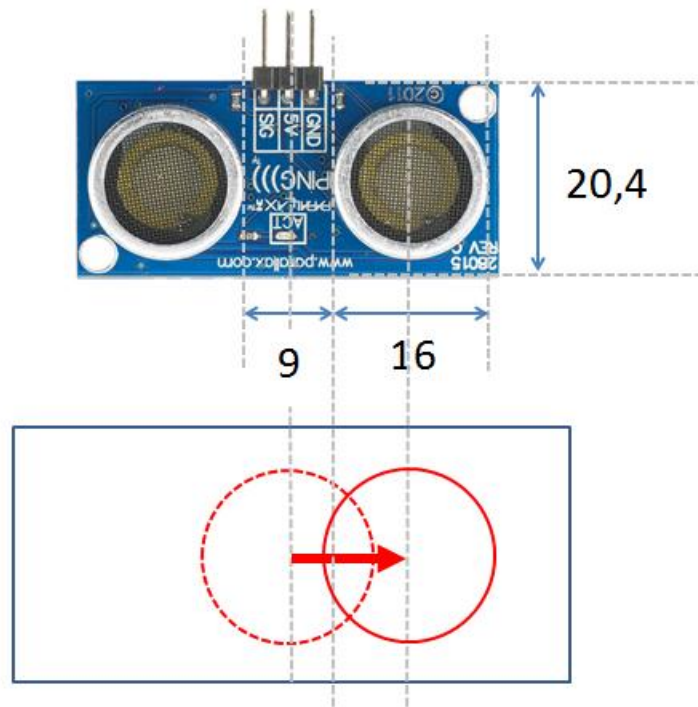
Colocamos primeramente las dos diferencias de los cilindros en el centro, hacemos que los cilindros de las diferencias tenga una altura mayor al elemento que queremos perforar.

```
difference() {  
  union() {  
    cube ([50,2,25], center = true);  
  
    translate([0,0,-25/2 - 2/2])  
      rotate([90,0,0])  
      cube ([50,2,25], center = true);  
  }  
  
  #rotate([90,0,0])  
  cylinder (h = 65, d = 17, center = true);  
  
  #rotate([90,0,0])  
  cylinder (h = 65, d = 17, center = true);  
}
```

El operador # sirve para resaltar elementos, pero no tiene ningún efecto en el modelo final.



¿Cuanto debemos trasladar el punto medio de los cilindros para que coincidan con los puntos del sensor?



```

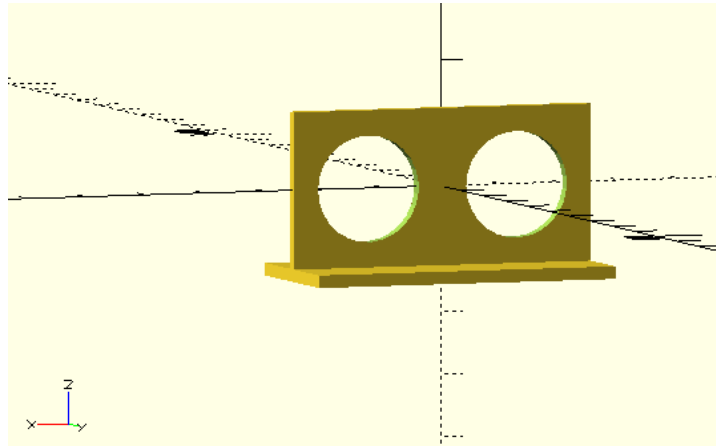
difference() {
  union() {
    cube ([50,2,25], center = true);

    translate([0,0,-25/2 - 2/2])
      rotate([90,0,0])
      cube ([50,2,25], center = true);
  }

  translate([9/2 + 16/2,0,0])
    #rotate([90,0,0])
    cylinder (h = 65, d = 17, center = true);

  translate([-9/2 - 16/2,0,0])
    #rotate([90,0,0])
    cylinder (h = 65, d = 17, center = true);
}

```



Para finalizar el modelo solo falta un detalle, siempre que podamos debemos evitar plataformas rectas (se quedan pegadas al cristal de la impresora y pueden romperse durante la extracción).

Incluimos una diferencia con un **plano inclinado que corte un fragmento de la base** (este plano se puede incluir de manera aproximada, no hace falta ser exacto).

```
difference() {
  union() {
    cube ([50,2,25], center = true);

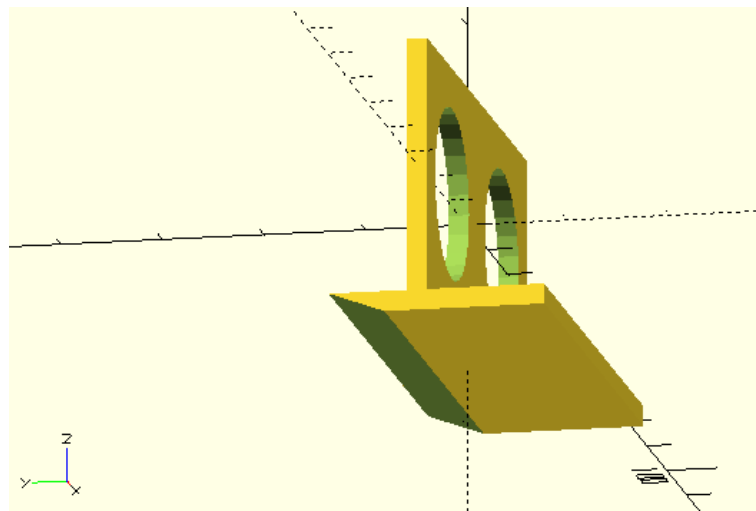
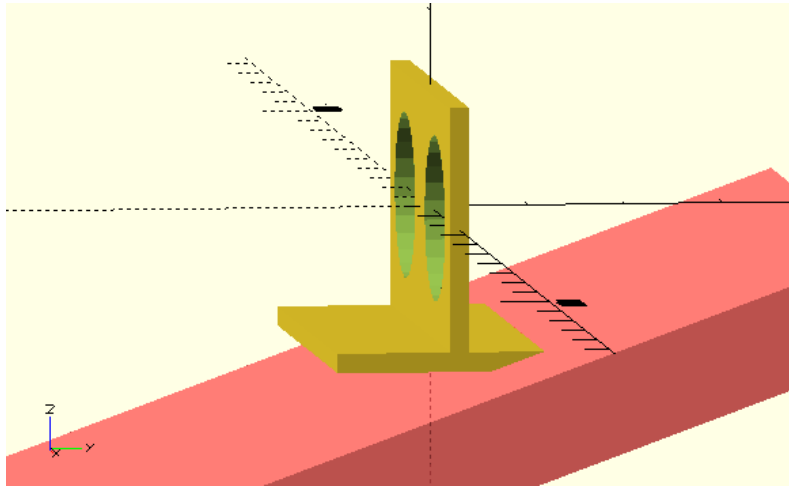
    translate([0,0,-25/2 - 2/2])
    rotate([90,0,0])
    cube ([50,2,25], center = true);
  }

  translate([9/2 + 16/2,0,0])
  rotate([90,0,0])
  cylinder (h = 65, d = 17, center = true);

  translate([- 9/2 - 16/2,0,0])
  rotate([90,0,0])
  cylinder (h = 65, d = 17, center = true);

  #translate([0,0,-21])
  rotate([20,0,0])
  cube ([90,90,10], center = true);
}
```

Ahora será mucho mas sencillo extraer la pieza con la espátula.



Intersecciones

Crea una intersección entre varios nodos.

Ejemplo 1 : intersección entre un cilindro y un cubo.

```
intersection() {
  cylinder (h = 11, r= 4, center = true);
  translate([5,0,0])
    cube (10, center = true);
}
```

Ejemplo 2 : intersección entre tres cilindros.

```
intersection() {
```

```

cylinder (h = 11, r= 4, center = true);
translate([2,0,0])
  cylinder (h = 11, r= 4, center = true);
translate([0,-2,0])
  cylinder (h = 11, r= 4, center = true);
}

```

En ocasiones resulta un poco difícil ver el resultado de las intersecciones y diferencias, puede resultar de gran ayuda aplicar **modificadores de fondo y depuración** (siguiente sección).

Modificadores de fondo y depuración

El operador % permite resaltar piezas haciéndolas transparentes, se suele utilizar para resaltar las piezas en una diferencia.

```

difference() {
  cube (10, center = true);
  %cylinder (h = 11, r= 4, center = true);
}

```

El operador # tiene una visualización similar al %, pero las piezas se resaltan utilizando un color rojo.

El operador * sirve para deshabilitar elementos, es muy útil si tenemos un elemento superpuesto o contenido dentro de otro y queremos ver los elementos ocultos.

El operador ! sirve para mostrar únicamente un elemento (sub-árbol), todo lo demás desaparece de la renderización.

La utilización de diferentes colores también nos puede servir para ver cómo está compuesta una pieza.

```

union() {
  cylinder (h = 20, d=10, center = true, $fn=100);
  color("red")
    rotate ([90,0,0])
      cylinder (h = 20, d=10, center = true, $fn=100);
}

```

Valores y tipos de datos

OpenScad nos ofrece la posibilidad de utilizar variables, resultan muy útiles para paramétrica nuestro código. Entre otros tipos permite manejar números, booleanos, cadenas de texto y vectores.

En el siguiente fragmento de código tenemos una rueda creada a partir de dos cilindros.

```

difference() {
  // rueda
  cylinder(h= 10, r = 10, center = true);
  // eje
  #cylinder(h= 15, r = 5, center = true);
}

```

Este script se podría parametrizar fácilmente mediante el uso de **variables**, en este caso podrían ser: radio del eje, radio de la rueda, y el ancho.

```

radioRueda = 10;
ancho = 10;
radioEje = 5;

```

En lugar de utilizar los valores utilizamos las variables en el resto del script.

```

radioRueda = 10;
ancho = 10;
radioEje = 5;

difference() {
  // rueda
  cylinder(h= ancho, r = radioRueda, center = true);
  // eje
  #cylinder(h= ancho*2, r = radioEje, center = true);
}

```

A partir de ahora vamos a procurar utilizar variables en todos los casos, y hacer los modelos lo más paramétricos posibles.

Ejemplo, caja paramétrica:

```

an_caja = 10;
pr_caja = 20;
al_caja = 10;

borde = 2;

difference() {
  cube ([an_caja,pr_caja,al_caja], center = true);

  translate([0,0,borde])
  cube ([an_caja-borde*2,pr_caja-borde*2,al_caja], center = true);
}

```

Podemos aplicar las operaciones matemáticas comunes sobre los valores, * + -...

Además de muchas otras funciones matemáticas abs, exp, log , cos, sin, max, min, etc.

https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/The_OpenSCAD_Language#Other_Mathematical_Functions

Resulta sencillo crear nuestras propias funciones para calcular valores.

https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/User-Defined_Functions

Vectores

Otro tipo de variable muy utilizado es el vector, podemos declararlos y utilizarlos al igual que en la mayor parte de lenguajes de programación.

Resulta especialmente útil porque los vectores se utilizan constantemente.

```
dim_caja = [10,20,10]; // an, pr, al

borde = 2;

difference(){
    cube (dim_caja, center = true);

    translate([0,0,borde])
    cube ([dim_caja[0]-borde*2,dim_caja[1]-borde*2,dim_caja[2]],
center = true);
}
```

Bucles for

Los bucles **for** nos permiten definir modelos aplicando patrones.

Sí por ejemplo queremos crear un soporte con varios agujeros cada 8mm podríamos hacerlo con un bucle.

```
an_sop = 6;
pr_sop = 70;
al_sop = 3;

d_agujero = 4;
distancia = 8;

difference(){
    cube([an_sop,pr_sop,al_sop], center = true);

    // 5 repeticiones de 1 a 5
```

```

for (i = [1 : 5])
{
    #translate([0,- pr_sop/2 + d_agujero/2 + distancia*i,0])
    cylinder(h= an_sop*2, d = d_agujero, center = true );
}

```

Por defecto funciona con un incremento de 1, pero podemos especificarle otro.

```

for (i = [1 : <incremento> : 5])

for (i = [1 : 0.5 : 5])

```

El bucle for también se puede utilizar para recorrer **los elementos de un vector**.

```

for (i = [1, 3, 5])

for(i = [ [ 0, 0, 0],
          [10, 12, 10],
          [20, 24, 20],
          [30, 36, 30],
          [20, 48, 40],
          [10, 60, 50] ])

```

El bucle **for** permite anidaciones.

```

for (i =[0: 10 :50 ], j = [0: 10 :50 ]){
    translate([i, j, 0])
    cube([6, 6, 6]);
}

```

En los casos anteriores especificamos el número de agujeros, pero también podrían **calcularse dinámicamente** en función del tamaño de la pieza.

```

an_sop = 6;
pr_sop = 70;
al_sop = 3;

d_agujero = 4;
distancia = 8;

n = pr_sop / (d_agujero/2 + distancia );
echo("Hay que hacer ", n);

difference() {
    cube([an_sop,pr_sop,al_sop], center = true);
}

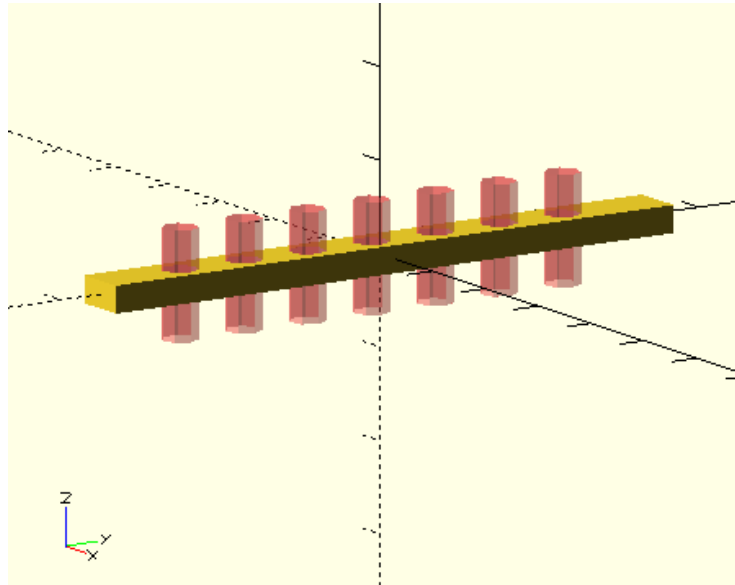
```



```

for (i = [1 : n])
{
    #translate([0,- pr_sop/2 + d_agujero/2 + distancia*i,0])
    cylinder(h= an_sop*2, d = d_agujero, center = true );
}

```



Ejemplo escalera, hay un patrón claro que se sigue en cada peldaño que se coloca (cada peldaño es más alto y está en otra posición):

```

an_escalon = 15;
pr_escalon = 10;
al_escalon = 5;

union() {

    cube ([an_escalon, pr_escalon, al_escalon] , center = true);

    translate([0,pr_escalon, al_escalon/2])
        cube ([an_escalon, pr_escalon, al_escalon*2] , center = true);

    translate([0,pr_escalon*2, al_escalon/2 * 2])
        cube ([an_escalon, pr_escalon, al_escalon*3] , center = true);

}

```

Se podría definir fácilmente el tamaño y la posición de los escalones con un bucle.

```

an_escalon = 15;
pr_escalon = 10;
al_escalon = 5;

union() {
    for (i = [1:10]) {

```

```

        translate([0,pr_escalon * i, al_escalon/2 * i])
        cube ([an_escalon, pr_escalon, al_escalon + al_escalon*i] ,
            center = true);
    }
}

```

Condiciones if

Permiten evaluar una condición y ejecutar instrucciones si se cumple o no (else).

Ejemplo, condición con Boolean.

```

an_cubo = 10;
pr_cubo = 10;
al_cubo = 10;

grande = true;

if (grande) {
    scale(2)
    cube([an_cubo,pr_cubo,al_cubo], center = true);
} else {
    cube([an_cubo,pr_cubo,al_cubo], center = true);
}

```

Ejemplo, condición con expresión matemática.

```

if (an_cubo >= 10 && pr_cubo > 10) {

```

Resultan bastante útiles para la depuración y creación de modelos con patrones iterativos y el uso de recursividad.

Módulos

Los módulos son algo similar a las funciones, podemos encapsular un script en un módulo y reutilizarlo mediante llamadas.

Al igual que las funciones pueden recibir parámetros.

En el siguiente ejemplo convertimos en **un módulo** el script que realizamos anteriormente para crear la rueda. Recibe tres parámetros que le indican cómo construir la rueda.

```

module rueda(radiusRueda, ancho, radiusEje) {
    difference() {
        // rueda
        cylinder(h= ancho, r = radiusRueda, center = true);
        // eje
        #cylinder(h= ancho*2, r = radiusEje, center = true);
    }
}

```

```
}
```

Para invocar al módulo y que se generen el modelo debemos escribir el nombre del módulo y los parámetros requeridos.

```
rueda(10,10,3);  
  
translate([0,20,0])  
    rueda(10,10,3);
```

Los módulos permiten establecer **valores por defecto**.

```
module rueda(radius=10, ancho=10, radioEje=5) {  
    difference() {  
        // rueda  
        cylinder(h= ancho, r = radius, center = true);  
        // eje  
        #cylinder(h= ancho*2, r = radioEje, center = true);  
    }  
}
```

Para llamar a un módulo con valores por defecto podemos llamarlo sin parámetros, o declarar en la llamada **solo aquellos parámetros que nos interese cambiar**.

```
rueda();  
  
translate([0,20,0])  
    rueda(radiusEje= 9);
```

Los módulos se pueden guardar en ficheros independientes (suele ser lo más recomendable para mantener una buena organización) y usarse en otros scripts. Para guardar un módulo -> **Archivo -> Salvar Como -> rueda.scad**

Para agregar el módulo debemos utilizar la directiva use (Debemos especificar el path del fichero .scad , nosotros lo colocaremos en el mismo directorio).

```
use <rueda.scad>  
  
rueda();
```

Existe gran cantidad de módulos que podemos descargar y utilizar en nuestros proyectos para crear piezas fácilmente, en posteriores secciones veremos algunas.

<http://www.thingiverse.com/search?q=openscad+module&sa=>

Recursividad

Los módulos pueden ser llamados de forma recursiva.

Las funciones recursivas son muy útiles para crear piezas siguiendo un patrón recursivo, esta característica ha sido incluida en la última versión, todavía no está totalmente madura.

Ejemplo de modelo recursivo.

```
an_pr_piso= 40;  
al_piso= 5;  
  
crearPiso(t = 11);  
  
module crearPiso(n = 0, t){  
    if ( n >= t){  
        // Se acabo  
    } else {  
        translate([0,0, al_piso*n ])  
        cube([an_pr_piso/t * n,  
            an_pr_piso/t * n,  
            al_piso],  
            center = true);  
  
        crearPiso(n + 1, t);  
    }  
}
```

En ocasiones la compilación falla si le ponemos nombres de más de un carácter a los parámetros del módulo.

Extrusión

La extrusión es un proceso industrial que se utiliza para crear objetos con una sección transversal definida y fija. El material se extrae mediante un troquel, se suele utilizar para crear tubos, barras y otro tipo de elementos similares.

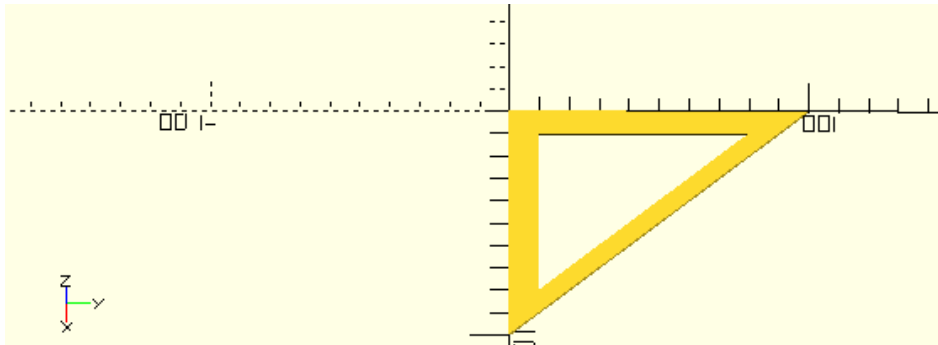


En este ejemplo lo primero que vamos a hacer es crear la "forma" mediante un polígono.

Para crear una forma 2D, existen diferentes formas en 2D como: circle, square, polygon.

El polygon nos permite definir piezas libremente, especificando sus puntos **points**, y luego el orden en el que se unen esos puntos **paths** (un único orden, si todos los puntos están unidos, o varios ordenes si el polígono tiene agujeros).

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



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

Se puede agregar torsión a la extrusión lineal con la variable **twist = n** de grados de torsión. Como en casos anteriores podemos definir la resolución con **\$fn**.

```
linear_extrude(height = 50, twist = 90, $fn=100, center = true)
square([20,20], center = true);
```

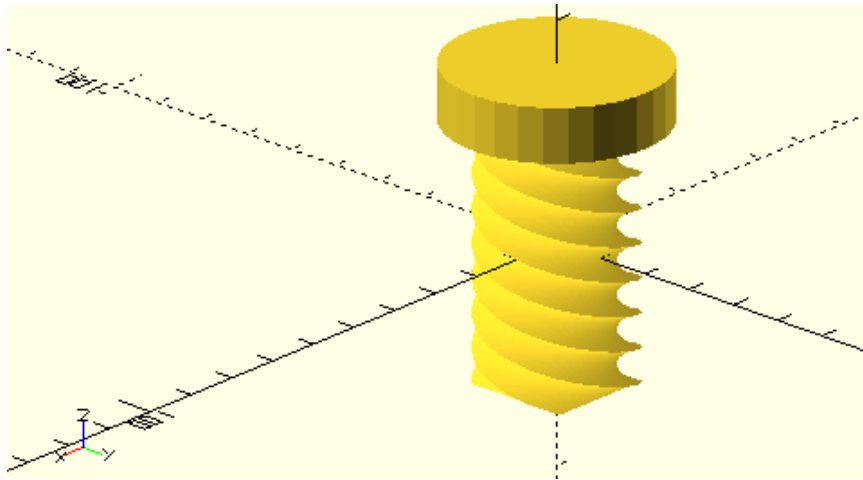
Lo transformamos en el cuerpo de un tornillo que da dos vueltas (360×2).

```
linear_extrude(height = 50, twist = 360*2, $fn=100, center = true)
square([20,20], center = true);
```

Completamos el modelo añadiéndole una cabeza.

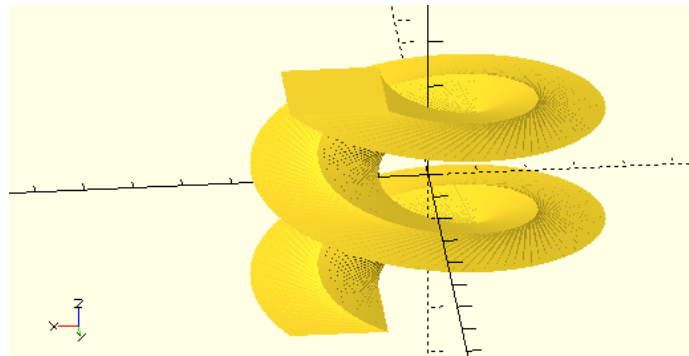
```
union() {
  linear_extrude(height = 50, twist = 360*2, $fn=100, center = true)
  square([20,20], center = true);

  translate([0,0,20/2 + 35/2])
  cylinder(d= 40, h = 10, center= true);
}
```



Volviendo al cuerpo del tornillo, podemos aplicar una torsión a la extrusión incluyendo una **translación** . El modelo se va a extruir usando como base el eje central, si no está justo en el eje central (no lo está lo hemos trasladado) parecerá que gira alrededor del eje.

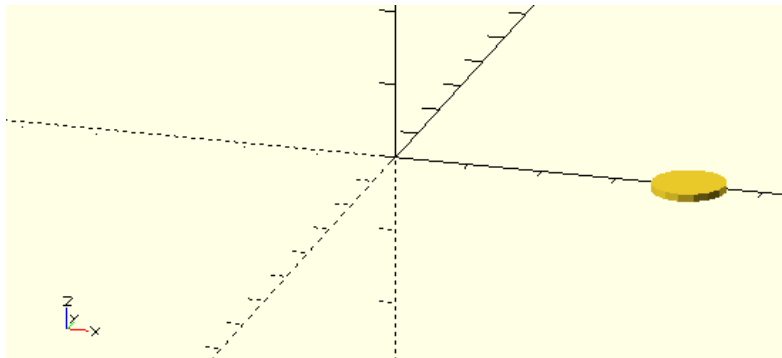
```
linear_extrude(height = 50, twist = 360*2, $fn=100, center = true)
  translate([20,0,0])
    square([20,20], center = true);
```



OpenScad también permite aplicar rotación de extrusión

Creamos un círculo (lo trasladamos no dejarlo en el centro del eje).

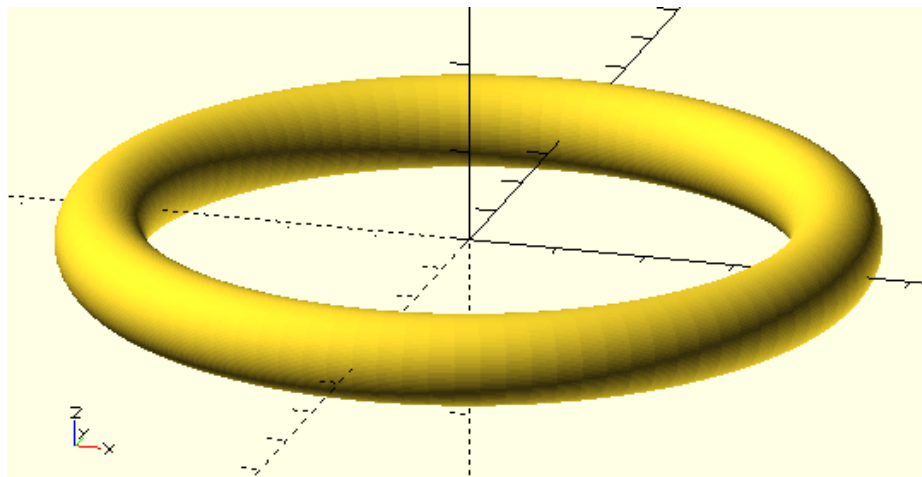
```
translate([40,0,0])
  circle(d = 10);
```



Incluimos la rotación de **extrusión de rotación** con resolución 100 (no confundir el \$fn de la extrusión con el \$fn del círculo).

Lo que hará esta extrusión es girar el modelo y completar la extrusión rotando alrededor del eje.

```
rotate_extrude($fn= 100)
  translate([40,0,0])
    circle(d = 10);
```



Probamos una acción similar con otro polígono más complejo.

```
rotate_extrude($fn= 100)
  translate([30,0,0])
    polygon(points=[[0,0],[0,10],[10,10],[5,5],[10,0]],
      paths=[ [0,1,2,3,4] ]);
```

Importar modelos

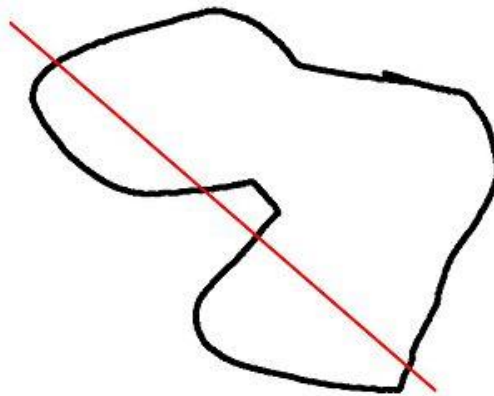
En ocasiones resulta muy útil importar modelos stl para realizar modificaciones sobre ellos.

```
import("Base.stl", convexity=10);
```

El objeto importado se comporta como un bloque sobre el que podemos aplicar uniones, diferencias, etc.

La propiedad convexity indica el número máximo de lados frontales que se permitirán en una intersección con el objeto.

Ejemplo: al cortar la siguiente imagen con un plano bastaría con una convexity de 10, no obstante, el manual de OpenScad nos recomiendan utilizar una convexity de 10 (suele ser suficiente incluso para piezas relativamente complejas)



Librerías

Existe un gran número de librerías y módulos de código libre que podemos utilizar para diseñar modelos en nuestros proyectos.

En estas librerías se pueden encontrar gran parte de las piezas más comúnmente utilizadas para construir diferentes sistemas robóticos, tornillos, engranajes, soportes, etc.

Gran parte de estas librerías se basan en la parametrización, por lo que resulta muy sencillo crear modelos que se adapten a nuestras necesidades. No obstante, los modelos creados por las librerías pueden volver a ser modificados (uniones, diferencias, etc.)

Las librerías están desarrolladas utilizando módulos y lenguaje OpenScad por lo que podemos explorarlas para ver su funcionamiento.

Al haber tantas librerías normalmente hay mucho donde elegir, lo más recomendable es usar una librería que permita hacer **justo** lo que necesitamos. Las librerías potentes a menudo son bastante complejas de utilizar.

MCAD

Es una de las librerías más populares y completa <https://github.com/elmom/MCAD> (aunque algunas funciones están deprecadas) contiene módulos para la creación de formas, 2d, curvas, cajas, engranajes, formas complejas, y muchas más utilidades.

Descomprimos la librería en una carpeta y abrimos el fichero **involute_gears.scad**. En este fichero encontramos un ejemplo de definición de engranajes.

Descomentamos el ejemplo *SimpleTest* para probarlo.

```
gear (circular_pitch=700,  
      gear_thickness = 12,  
      rim_thickness = 15,  
      hub_thickness = 17,  
      circles=8) ;
```

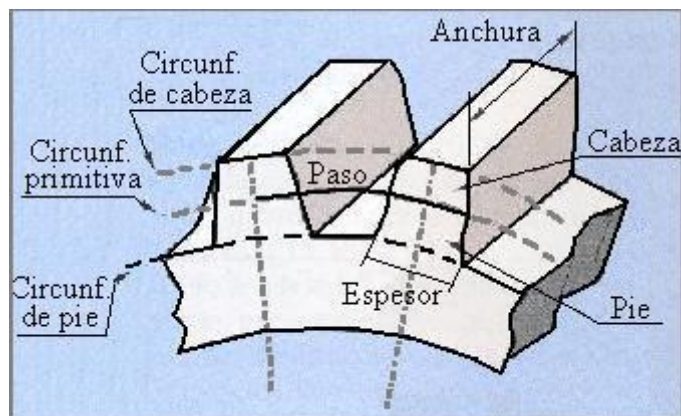
- **circular_pitch**: tamaño de los dientes (el tamaño será proporcional).
- **gear_thickness** : grosor del engranaje.
- **rim_thickness**: grosor de la llanta exterior (si la queremos incluir).
- **hub_thickness**: grosor del soporte para el agujero del eje (**no es el diámetro, es el alto**).
- **circles**: número de agujeros a repartir por la superficie del engranaje.

Sí bajamos un poco en el fichero, vemos todo lo que nos permite definir el módulo **gear**, la mayor parte de los parámetros tienen valores por defecto.

```

296 module gear (
297     number_of_teeth=15,
298     circular_pitch=false, diametral_pitch=false,
299     pressure_angle=28,
300     clearance = 0.2,
301     gear_thickness=5,
302     rim_thickness=8,
303     rim_width=5,
304     hub_thickness=10,
305     hub_diameter=15,
306     bore_diameter=5,
307     circles=0,
308     backlash=0,
309     twist=0,
310     involute_facets=0,
311     flat=false)

```



Por ejemplo si queremos crear un engranaje para reducir a la mitad la potencia de un motor. Creamos un engranaje con 15 dientes para el motor y otro con 30 dientes (los dos con el mismo **circular_pitch** para que encajen).

```

gear (
    number_of_teeth=15,
    circular_pitch=400,
    hub_diameter= 10,
    bore_diameter=5);

gear (
    number_of_teeth=30,
    circular_pitch=400,
    hub_diameter= 20,
    bore_diameter=15);

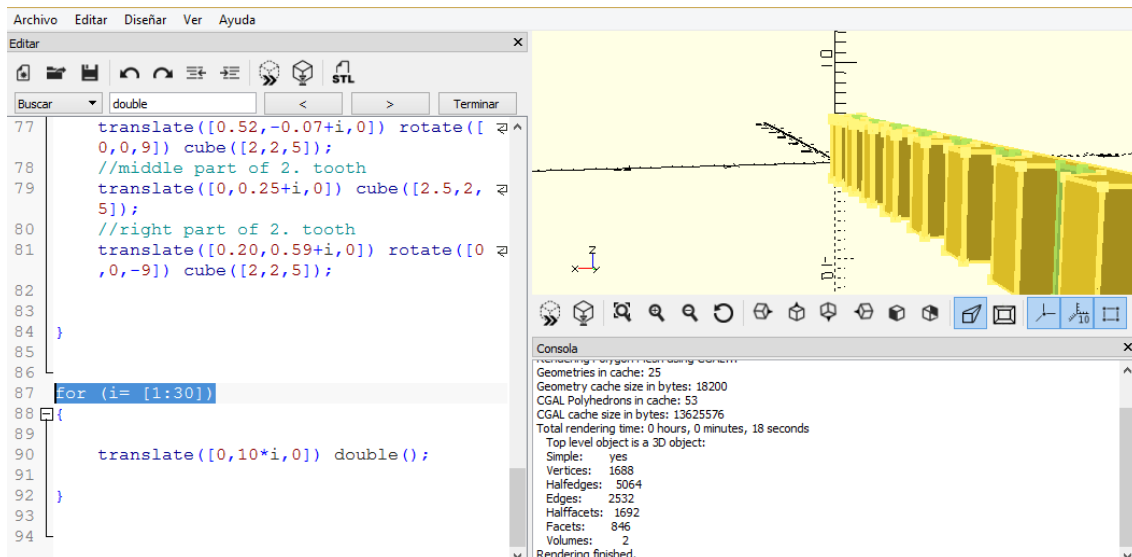
```

Correas dentadas

Módulo para crear correas dentadas, estos modelos requieren elasticidad por lo que deben ser impresos con un tipo de material elástico como **FilaFlex** .

<http://www.thingiverse.com/thing:6800>

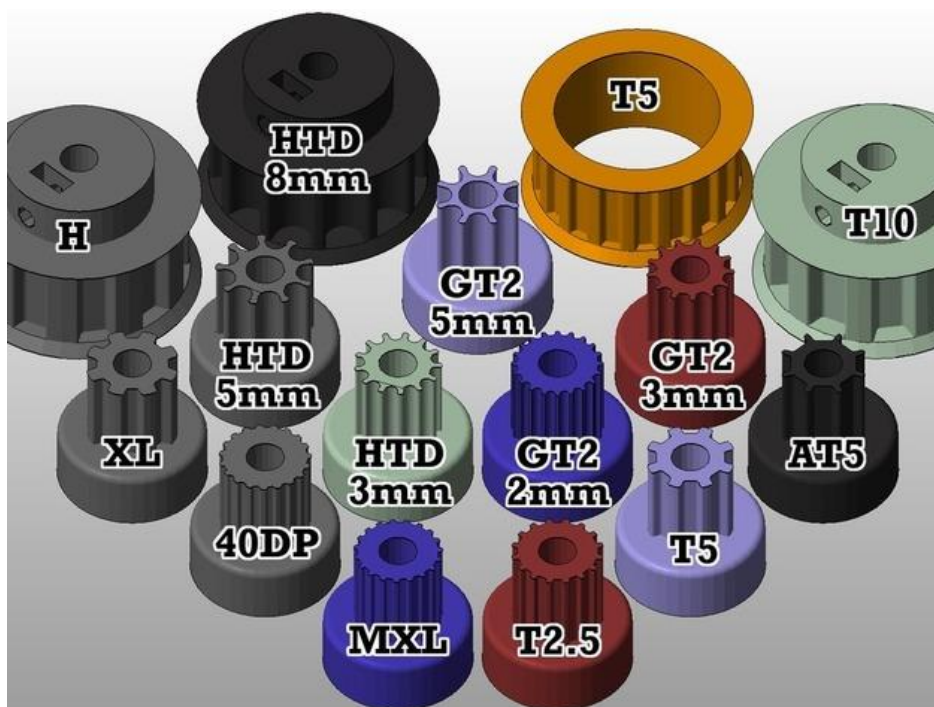
En la parte final del fichero .scad vemos como se define la correa.



Esta librería no está demasiado bien parametrizada. Si queremos modificar el tamaño de los dientes o el ancho de la correa habrá que modificar la implementación de los módulos.

Poleas para correas

Esta otra librería <http://www.thingiverse.com/thing:16627> tiene una gran potencia, permite definir poleas para correas parametrizadas, podemos especificar prácticamente todos los aspectos de la polea mediante variables.

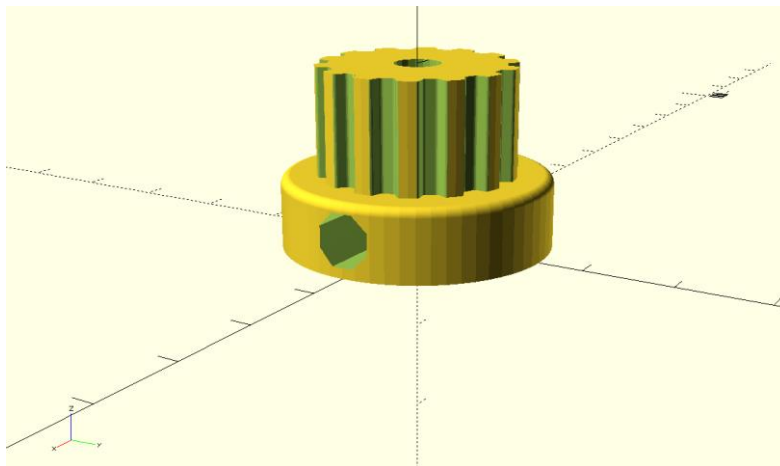


Al descargar el fichero y abrirlo podemos ver que toda la configuración de la polea está parametrizada:

```

23 teeth = 8;           // Number of teeth, standard Mendel T5 belt
   = 8, gives Outside Diameter of 11.88mm
24 profile = 6;         // 1=MXL 2=40DP 3=XL 4=H 5=T2.5 6=T5 7=T10
   8=AT5 9=HTD_3mm 10=HTD_5mm 11=HTD_8mm 12=GT2_2mm 13=GT2_3mm
   14=GT2_5mm
25
26 motor_shaft = 5.2;   // NEMA17 motor shaft exact diameter = 5
27 m3_dia = 3.2;        // 3mm hole diameter
28 m3_nut_hex = 1;      // 1 for hex, 0 for square nut
29 m3_nut_flats = 5.7;   // normal M3 hex nut exact width = 5.5
30 m3_nut_depth = 2.7;  // normal M3 hex nut exact depth = 2.4,
   nyloc = 4
31
32 retainer = 0;        // Belt retainer above teeth, 0 = No, 1 = Yes
33 retainer_ht = 1.5;    // height of retainer flange over pulley,
   standard = 1.5
34 idler = 0;           // Belt retainer below teeth, 0 = No, 1 = Yes
35 idler_ht = 1.5;      // height of idler flange over pulley,
   standard = 1.5
36
37 pulley_t_ht = 12;     // length of toothed part of pulley,
   standard = 12
38 pulley_b_ht = 8;      // pulley base height, standard = 8.
   Set to same as idler_ht if you want an idler but no pulley.
39 pulley_b_dia = 20;    // pulley base diameter, standard = 20
40 no_of_nuts = 1;       // number of captive nuts required,
   standard = 1
41 nut_angle = 90;       // angle between nuts, standard = 90
42 nut_shaft_distance = 1.2; // distance between inner face of
   nut and shaft, can be negative.

```



La librería <http://www.thingiverse.com/thing:60433/#files> es relativamente similar a la anterior, pero con bastante menos poder de configuración, no obstante, contiene todos los parámetros más utilizados.

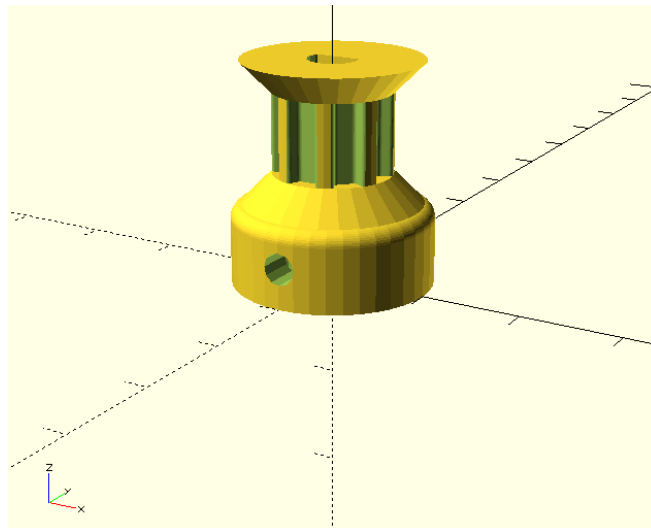
En la parte final del documento tenemos un ejemplo de cómo definir mediante parámetros los aspectos más importantes de la polea.

```

292
293 teeth=8;
294 belt = find("T5",Belts);
295 nut = find("M3 large",Nuts);
296 pulley_OD = pulley_OD(teeth,belt[1]);
297 echo (str("Belt type = ",belt[0]," Number of teeth = ",teeth,
           " Pulley Outside Diameter = ",pulley_OD,"mm "));
298
299 base_diameter=20;
300 base_height=8;
301 retainer_height=3;
302 gear_height=8;
303 nut_offset=5;

```

- **teeth:** nº de dientes
- **belt:** tipo de diente en la correa (T5, paso métrico 5mm una de las más comunes, otras T2,5, T10). las puede haber de diferentes anchos, pero con el mismo tipo de diente.
- **nut:** tipo de tornillo para el encaje inferior (M3 Largo)
- **gear_height :** ancho del engranaje, (solo la parte del engranaje) tiene que ser mayor que la correa.



Por si los parámetros de configuración no fueran suficientes podemos aplicar directamente diferencias sobre la polea generada, por ejemplo, para modificar el tamaño de los ejes o de los huecos para los tornillos.

Tornillos, tuercas y varillas roscadas

La librería <http://www.thingiverse.com/thing:8793/#files> permite crear diferentes varillas roscadas y tuercas.

En la propia librería se incluye un ejemplo similar a este:

```

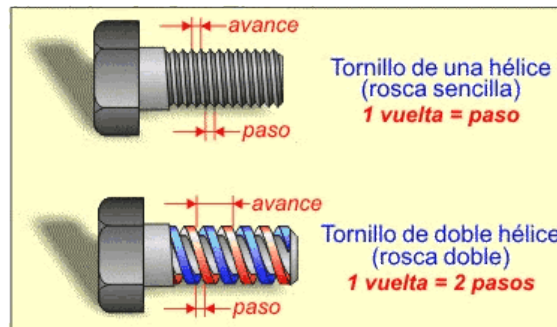
trapezoidThread(
  length=65,           // alto
  pitch=10,            // distancia entre
  pitchRadius=10,      // radial distance from center to mid-profile

```

```

    threadHeightToPitch=0.5,    // ratio between the height of the
    profile and the pitch
                                // std value for Acme or metric lead screw is
0.5
    profileRatio=0.5,           // ratio between the lengths of the
    raised part of the profile and the pitch
                                // std value for Acme or metric lead screw is
0.5
    threadAngle=30,             // angle between the two faces of the
    thread
                                // std value for Acme is 29 or for metric lead
screw is 30
    RH=true,                    // true/false the thread winds clockwise
    looking along shaft, i.e.follows the Right Hand Rule
    clearance=0.1,              // radial clearance, normalized to thread
    height
    backlash=0.1,               // axial clearance, normalized to pitch
    stepsPerTurn=24             // number of slices to create per turn
);

```



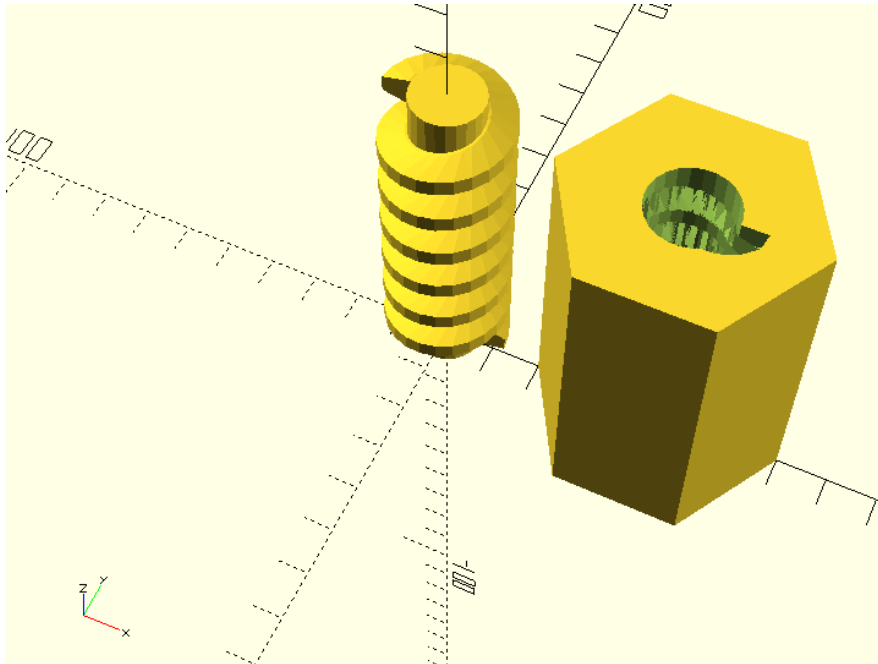
Sí en lugar del módulo llamamos a **trapezoidNut** obtenemos la tuerca para el modelo anterior.

```

translate([45,0,0])
trapezoidNut (
    length=65,                 // axial length of the threaded rod
    pitch=10,                   // axial distance from crest to crest
    pitchRadius=10,            // radial distance from center to mid-profile
    threadHeightToPitch=0.5,    // ratio between the height of the
    profile and the pitch
                                // std value for Acme or metric lead screw is
0.5
    profileRatio=0.5,           // ratio between the lengths of the
    raised part of the profile and the pitch
                                // std value for Acme or metric lead screw is
0.5
    threadAngle=30,             // angle between the two faces of the
    thread
                                // std value for Acme is 29 or for metric lead
screw is 30
    RH=true,                    // true/false the thread winds clockwise
    looking along shaft, i.e.follows the Right Hand Rule
    clearance=0.1,              // radial clearance, normalized to thread
    height
    backlash=0.1,               // axial clearance, normalized to pitch
);

```

```
    stepsPerTurn=24                // number of slices to create per turn
);
```



La librería <http://www.thingiverse.com/thing:311031/#files> también permite definir tornillos y tuercas, tiene una capacidad de configuración bastante más limitada que la librería anterior.

Abrimos el fichero y descomentamos el ejemplo, también podemos repetir algunos de los ejemplos que aparecen en la parte inicial.

Tornillo y tuerca:

```
hex_bolt(10,36); // Tornillo M10 de 36mm de largo, cabeza hex
```

```
hex_nut(10); // Tuerca estándar para tornillo M10
```

Varillas roscadas

```
thread_out(8,16); // Espiral M8 16mm de largo
thread_out_centre(8,16); // Relleno para la espiral anterior
```

Tubo roscado (interior).

```
thread_in(8,10); // Espiral M8, 16mm de largo, para interior de un tubo
thread_in_ring(8,10,2); // Tubo exterior para la espiral de 2mm de ancho.
```

Debido a los pequeños errores de precisión en la impresión 3D la creación de tornillos y tuercas no resulta lo más recomendado, se obtiene una mayor precisión y resistencia con elementos de metal.

Engranajes

Sí realizamos una búsqueda veremos que hay decenas de librerías para crear engranajes (gran parte de ellas basadas en MCAD).

Dependiendo de la complejidad del sistema se pueden querer personalizar muchos aspectos de los engranajes, en nuestro caso posiblemente nos interese utilizar engranajes muy simples por lo que es preferible una librería con funcionalidad reducida pero fácil de utilizar <http://www.thingiverse.com/thing:268787/#files> es una de las más simples (sí la funcionalidad se nos queda corta siempre podemos buscar otra librería más completa).

Sí abrimos el fichero **.scad** vemos que la definición de los engranajes es bastante simple.

- **height** : alto del engranaje
- **type**: tipo de engranaje (Spur - normal, Cog - 2D, Double/Single Helix , con hélice)
- **teeth**: número de dientes, calculará el diámetro del engranaje en función del nº de dientes, todos los engranajes creados con la librería son interoperables
- **hole**: incluir eje
- **holeSize** = radio del eje (no confundir con el diámetro).



Helix Gear

Ejemplo, para aumentar la velocidad de giro que un servomotor de 360° puede darle a un eje podemos crear un sistema de engranajes.

Si unimos un engranaje grande al servomotor y hacemos que este mueva uno de la mitad del tamaño conseguiremos duplicar la velocidad de giro.

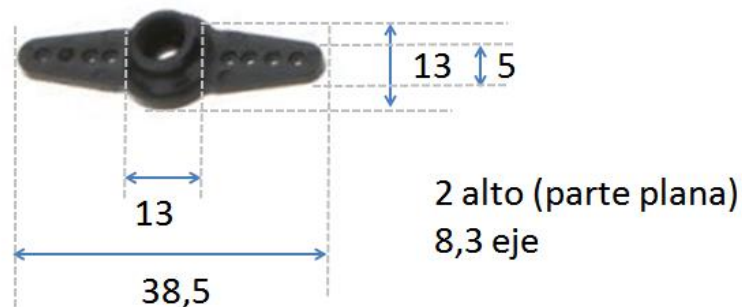
Creemos un engranaje de 50 dientes para el servomotor

```
height = 5;
type = "Spur"; // [Spur,Cog,Double Helix, Single Helix]
teeth = 50; // [18:50]

hole = "Yes"; //[Yes,No]
rotate_Offset_double_Helix_only = 5; //[0:20]

holeSize = 4.5; //don't make too big or your gear will disappear
scale = 100; //this is here temporarily I would not recommend changing!
```

Hacemos una diferencia para encajar una de las cabezas del servomotor. Hacemos una figura similar a la cabeza del servomotor con 1 o 2 mm de holgura.



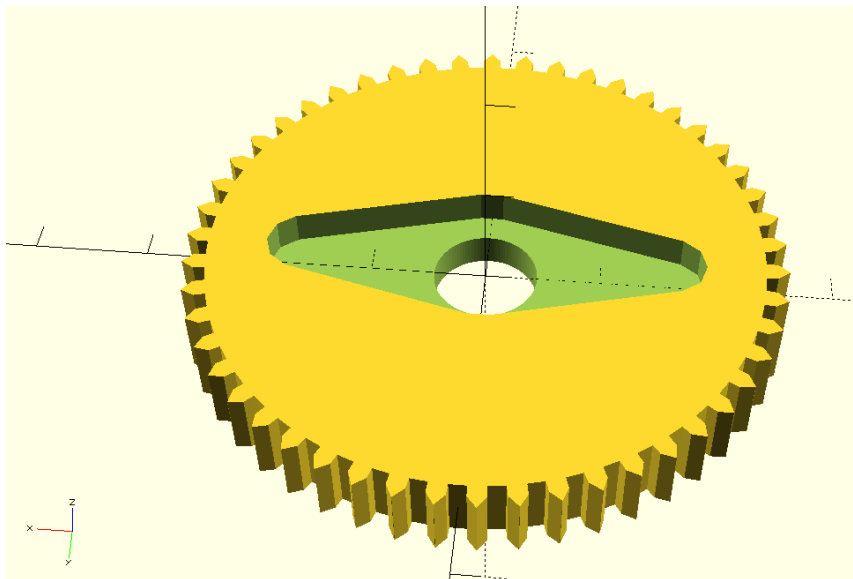
```

if (type == "Spur")
{
if(hole == "Yes")
{
    difference() {
        hole();
        union() {
            hull() {
                // (38,5/2 - 5/2) + 2
                translate([-38.5/2 + 5/2 ,0, 5/2 - 1])
                cylinder (h = 2 + 1, d = 5 + 1, center = true);

                translate([0,0,3/2])
                cylinder (h = 2 + 1, d = 13 + 1, center = true);

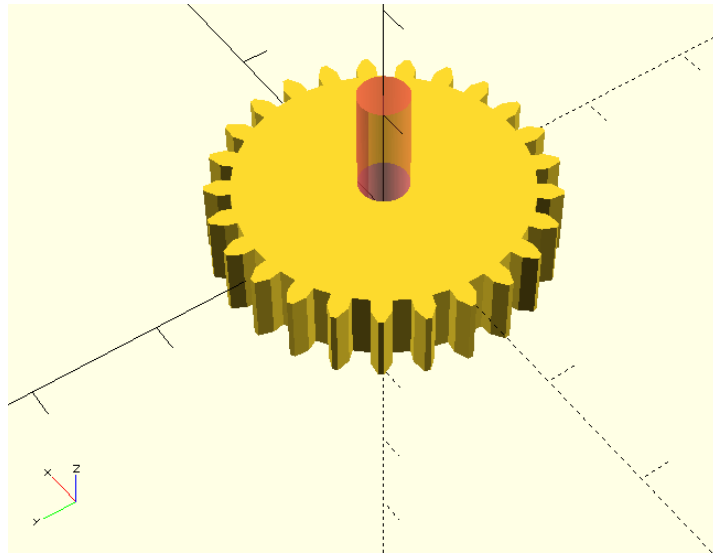
                #translate([38.5/2 - 5/2, 0, 5/2 - 1])
                cylinder (h = 2 + 1, d = 5 + 1, center = true);
            }
        }
    }
}
}

```



Creamos un engranaje de 25 dientes al que el engranaje anterior transmitirá el movimiento.

```
height = 5;  
type = "Spur"; // [Spur,Cog,Double Helix, Single Helix]  
teeth = 50; // [18:50]  
  
hole = "Yes"; //[Yes,No]  
rotate_Offset_double_Helix_only = 5; //[0:20]  
  
holeSize = 2.5; //don't make too big or your gear will disappear  
scale = 100; //this is here temporarily I would not recommend changing!
```



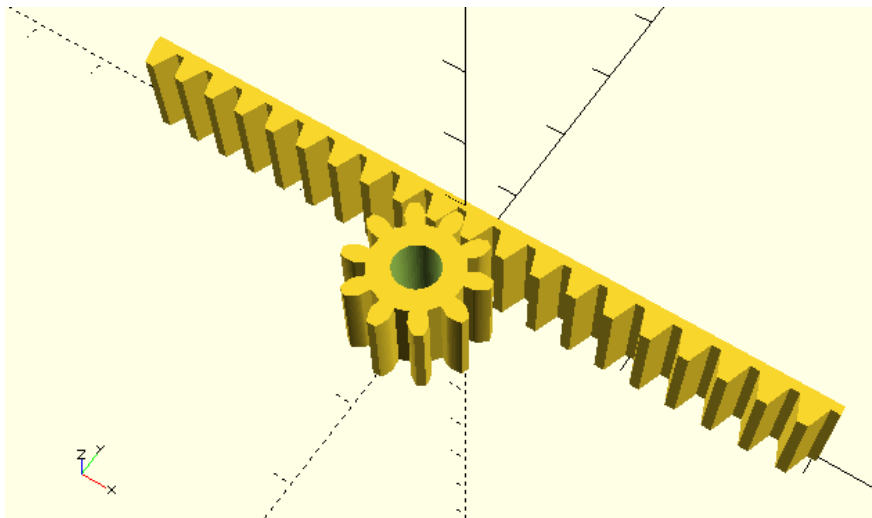
Piñón y cremallera

La librería <http://www.thingiverse.com/thing:172508/#files> permite crear fácilmente modelos basados en piñón y cremallera.

- **rack(4,20,10,1);**
 - Diente (mm)
 - Ancho rack(mm)
 - Alto rack (mm)
 - Alto base (mm)
- **pinion(4,10,10,5);**
 - Diente (mm)
 - Diámetro (mm)

- Alto (mm)
- Diámetro eje (mm)

```
rack(4,20,10,1); //CP (mm/tooth), width (mm), thickness(of base) (mm),
# teeth
// a simple pinion and translation / rotation to make it mesh the rack
translate([0,-8.5,0])
rotate([0,0,360/10/2])
pinion(4,10,10,5);
```



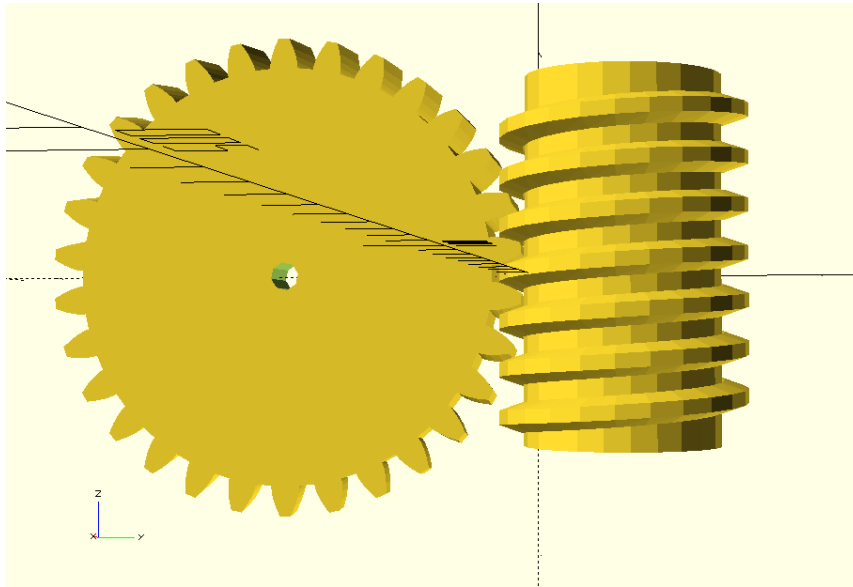
Tornillo sin fin

La librería <http://www.thingiverse.com/thing:8821> facilita la creación de un sistema de tornillo sin fin, se especifican los parámetros de forma general y se aplican al engranaje y al tornillo para que encajen correctamente.



Tornillo sin fin

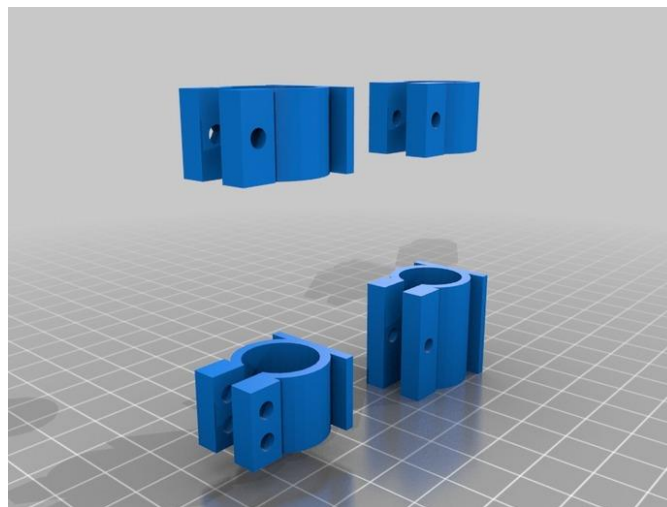
Está librería se basa en **Thread_Library.scad** y **MCAD/involute_gears.scad** (deben estar en las rutas indicadas para que funcione correctamente).



Soportes

Soportes con presión regulable.

<http://www.thingiverse.com/thing:31982/#files>



Soportes para barrillas roscadas y tornillos

<http://www.thingiverse.com/thing:8816>

Anti-backlash

https://www.youtube.com/watch?v=99q_1xiZcrE

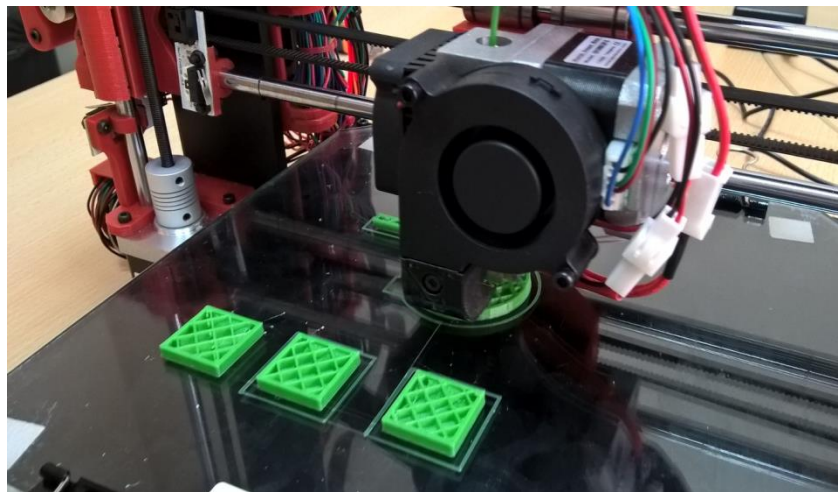
Impresión 3D

Los modelos .stl no suelen ser directamente imprimibles por la mayor parte de las impresoras.

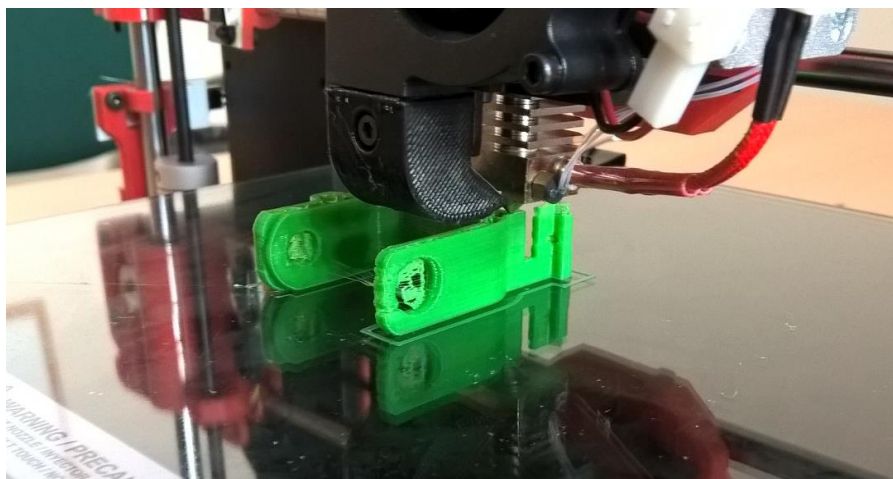
Necesitamos transformar esos modelos 3D en un formato imprimible.

Además de los propios modelos los formatos imprimibles contienen información a cerca de la impresora y la impresión. Por ejemplo:

- Dimensiones de la superficie de impresión
- Calidad de la impresión por capa (mm de material empleado)
- Sistema de relleno de partes huecas (porcentaje de densidad)



- Regulación de velocidad de impresión y temperatura (depende del material)
- Incluir soportes en las zonas que tiene agujeros o van sobre el "aire"



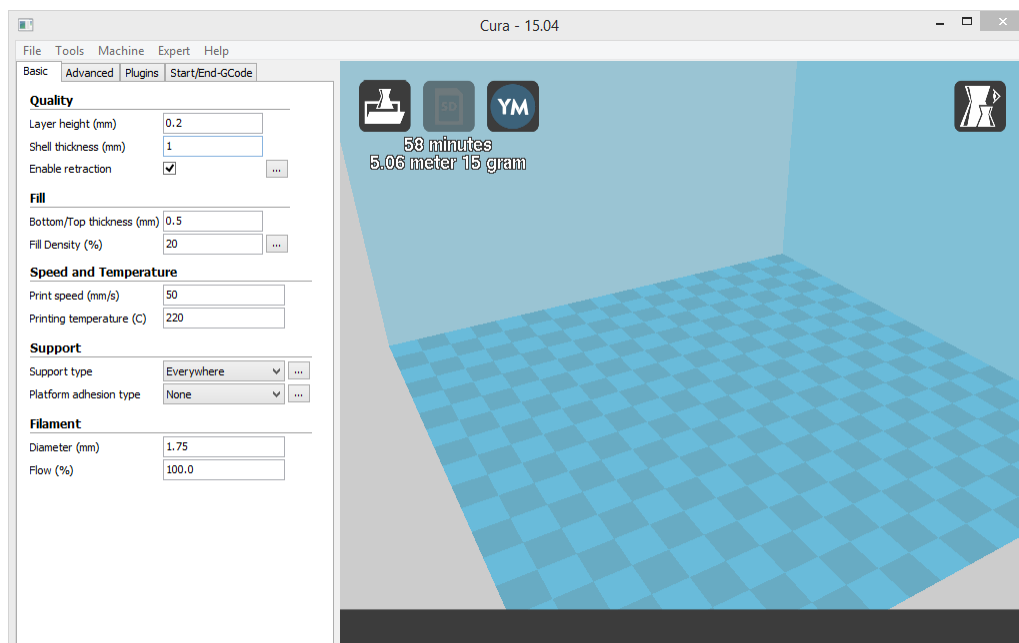
- Características del filamento: diámetro y % de filamento a expulsar.

Hay muchos programas que podemos utilizar para crear un formato imprimible, nosotros utilizaremos **Cura 15.04** (Importante esta versión) configurando las impresiones para una **prusa i3**.



Cura

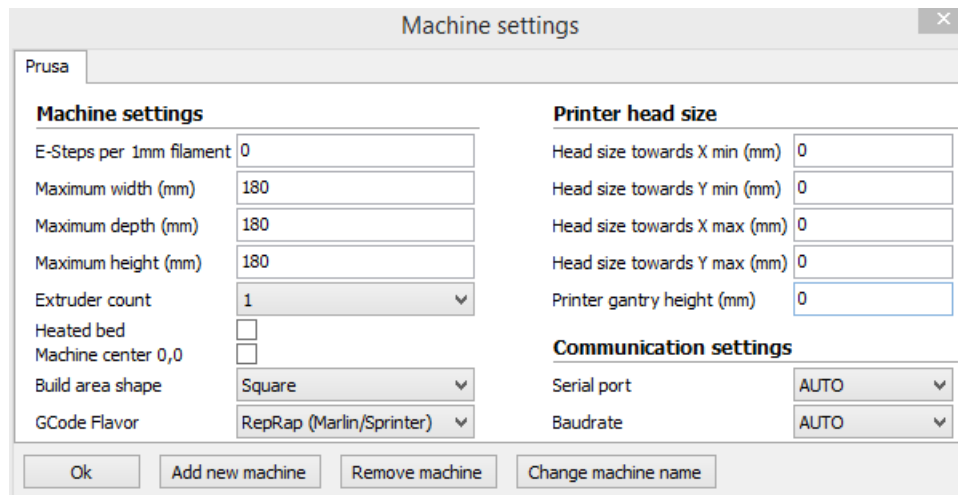
Descarga de programa Cura: <https://ultimaker.com/en/support/software/cura-1504>



Configuración de la impresora

Por defecto el programa no incluye la configuración de la prusa i3, hay que crear un nuevo perfil para la máquina.

Machine -> Add New Machine



The screenshot shows the 'Machine settings' dialog box for a Prusa printer. It is divided into two main sections: 'Machine settings' on the left and 'Printer head size' on the right. Below these are 'Communication settings' and a row of action buttons.

Machine settings	
E-Steps per 1mm filament	0
Maximum width (mm)	180
Maximum depth (mm)	180
Maximum height (mm)	180
Extruder count	1
Heated bed	<input type="checkbox"/>
Machine center 0,0	<input type="checkbox"/>
Build area shape	Square
GCode Flavor	RepRap (Marlin/Sprinter)

Printer head size	
Head size towards X min (mm)	0
Head size towards Y min (mm)	0
Head size towards X max (mm)	0
Head size towards Y max (mm)	0
Printer gantry height (mm)	0

Communication settings	
Serial port	AUTO
Baudrate	AUTO

Buttons: Ok, Add new machine, Remove machine, Change machine name

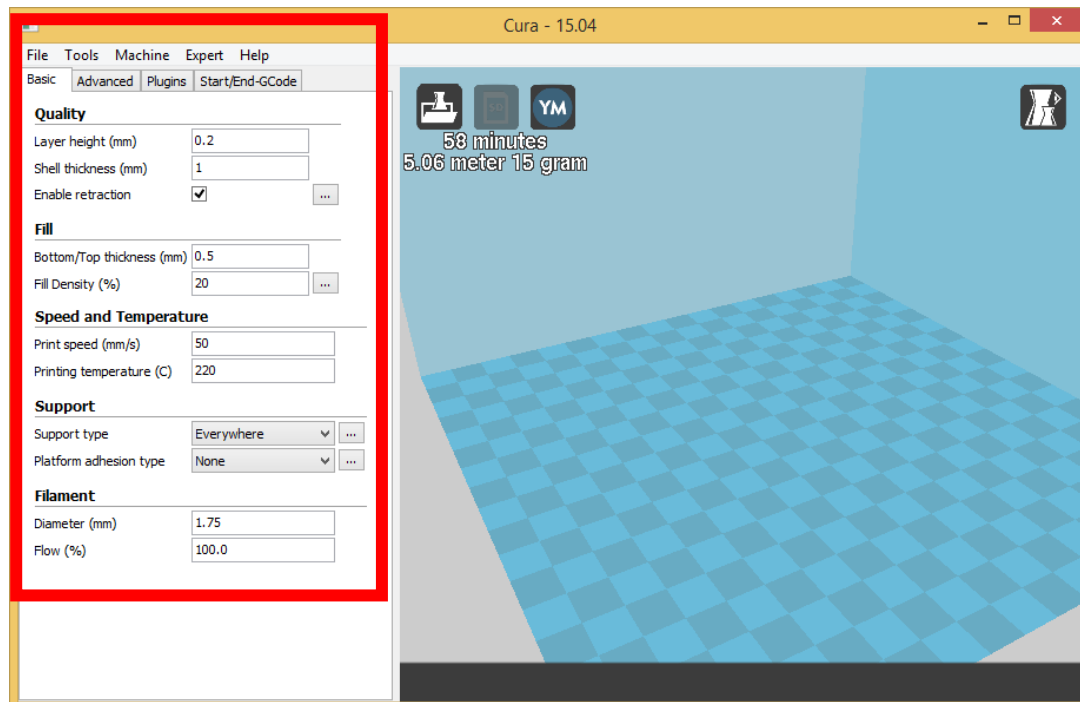
Configuración de la impresión

En la parte derecha se pueden configurar los parámetros básicos de la impresión.

Debemos colocar una **densidad de relleno baja** para evitar el excesivo consumo de material en modelos sólidos.

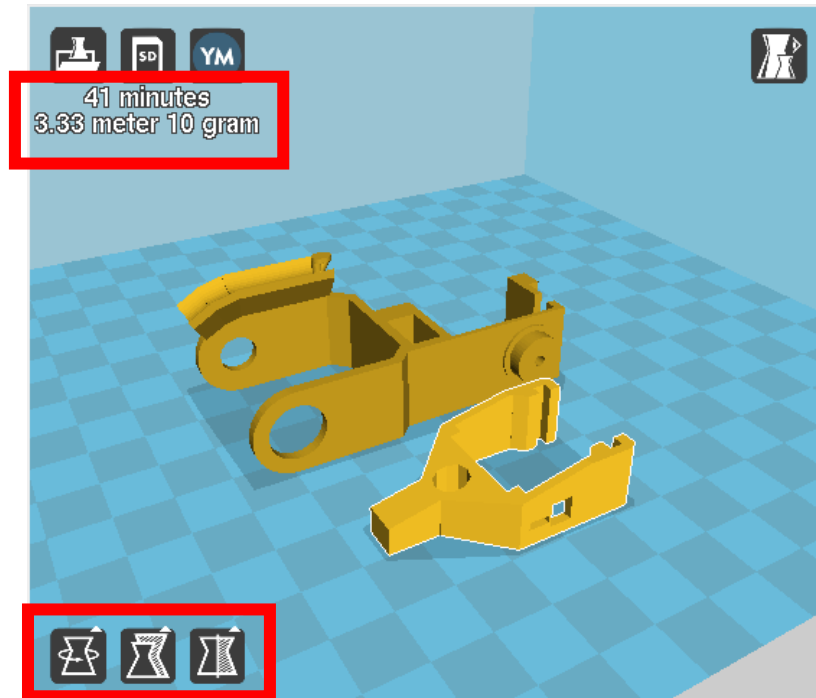
Es muy importante incluir **soportes en todas las partes**, para evitar que las piezas se descompongan durante la impresión, sobre todo en el caso de piezas especialmente complejas

Los **parámetros del filamento** deben ser exactamente los del material que está utilizando nuestra impresora.



Colocación de los modelos

Para colocar piezas basta con arrastrar los modelos .stl sobre la superficie



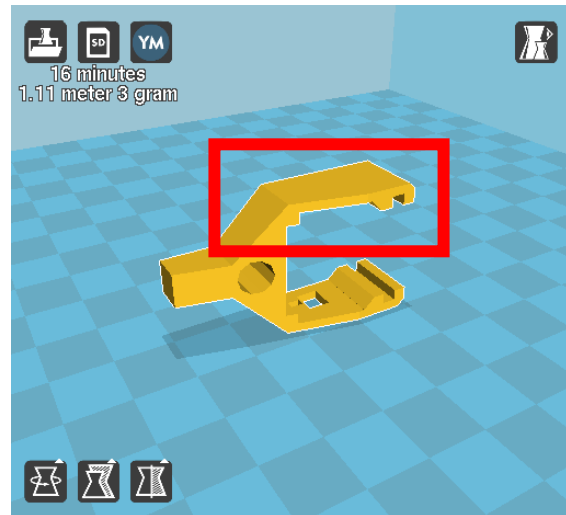
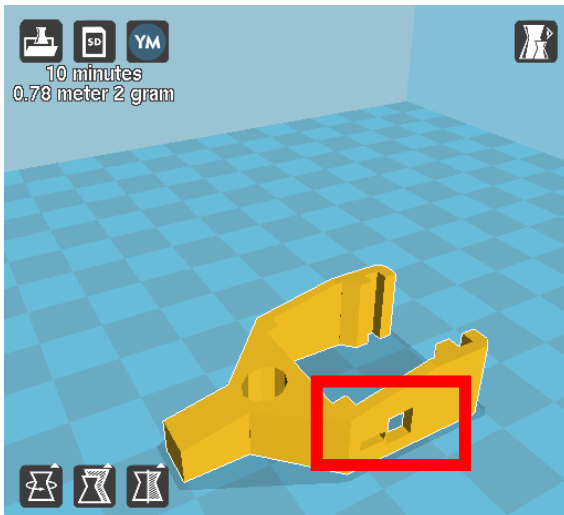
Podemos **mover y girar** los modelos utilizando los botones que aparecen en la parte inferior.

En la parte superior se muestra el **tiempo estimado y el número de gramos de filamento** necesario.

Al cambiar las piezas de posición podemos ver que el tiempo y el filamento necesario varían.

Lo ideal sería colocar las piezas **de la forma más eficiente posible**.

También hay que tener en cuenta que la **calidad de la impresión final** no es siempre la misma en todas las posiciones:



Normalmente la parte que se imprime "en el aire" suele tener peor acabado final (a pesar de usar soportes).

Nunca deben acercarse excesivamente a los **bordes**, por seguridad podemos dejar un margen de 2 o 3 cuadros.

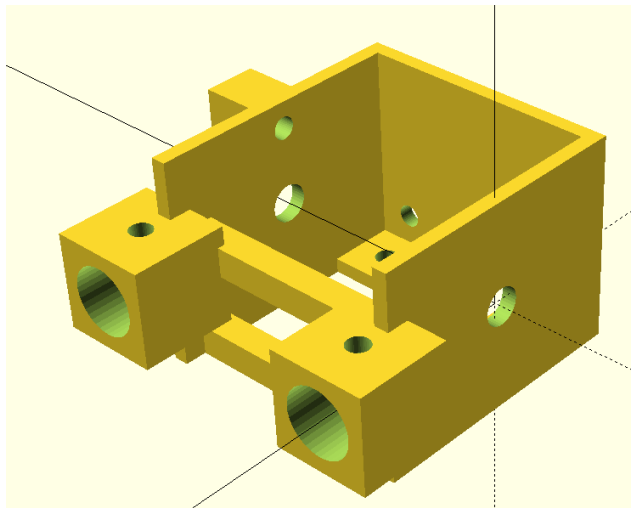
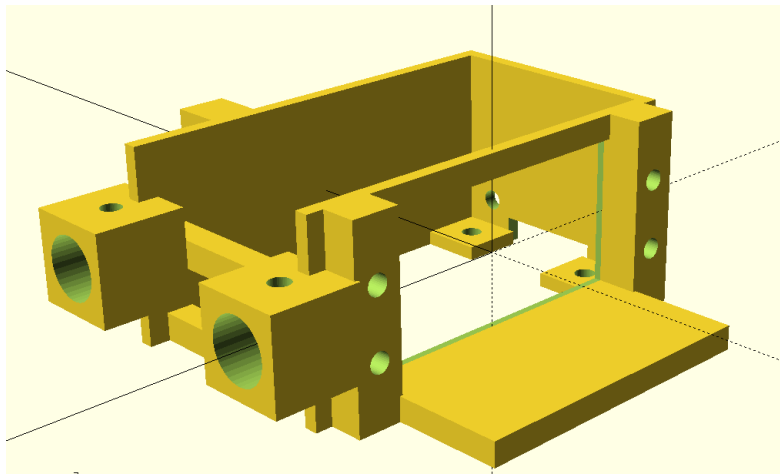
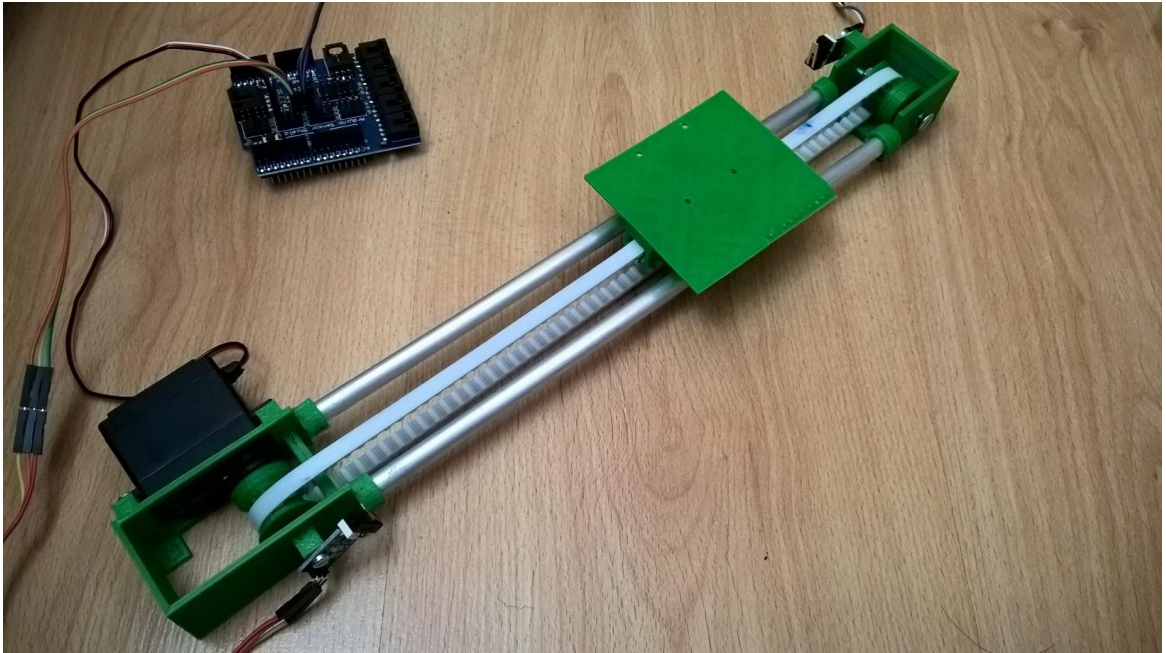
Exportar

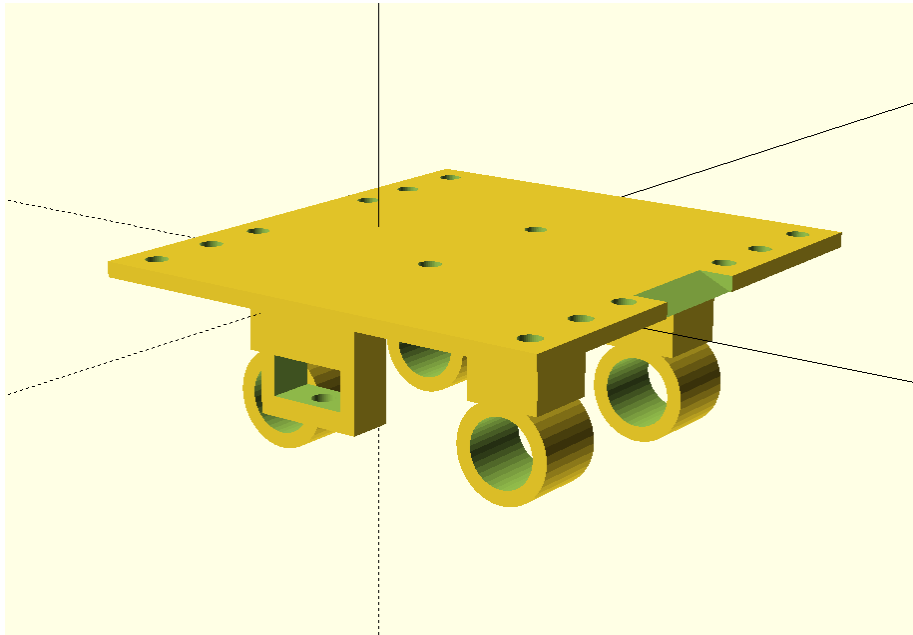
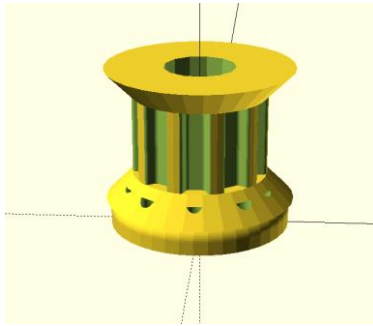
Cura genera ficheros gcode compatibles con la impresora prusa i3. Esos ficheros son los que enviaremos a la máquina para que imprima.

File -> Save GCode

Actuador Lineal

A continuación, vamos a diseñar una pieza de un actuador lineal utilizando OpenScad.





Correa dentada T5



Tubo metal 8 cm de diámetro



Tornillos M3 de 12mm

La lógica del actuador la programaremos durante la siguiente sesión de prácticas.