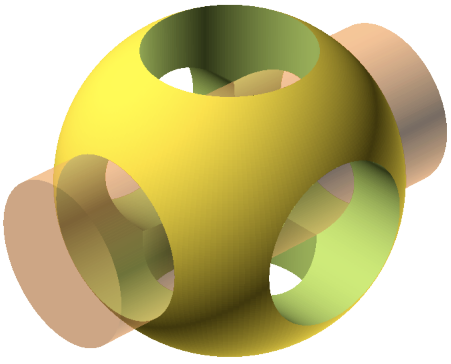
Universidad Nacional Autónoma de México

Facultad de Ingeniería

Laboratorio de Investigación y Desarrollo de Software Libre

Cusrso de Introducción de modelado parametrizable con OpenSCAD

Semestre 2018-1



Programa

Pablo Vivar Colina

Ciudad Universitaria Septiembre de 2017.

Índice

1. Introducción

En el desarrollo de prototipos en ingeniería es importante el desarrollo de modelos 3D para poder visualizar las piezas que se desean manufacturar que si existen errores, éstos puedan ser corregidos. En éste proceso se necesita tiempo y la habilidad de la persona que está desarrollando éstos modelos para lograr corregir éstos errores de manera eficiente.

OpenSCAD es una plataforma de desarrollo de dibujos 2D y 3D en donde los modelos son programados a través de líneas de código,y éstos pueden ser parametrizables.

OpenSCAD es una aplicación libre para crear objetos sólidos de CAD. No es un editor interactivo sino un compilador 3D basado en un lenguaje de descripción textual. Un documento de OpenSCAD especifica primitivas geométricas y define como son modificadas y manipuladas para reproducir un modelo 3D. OpenSCAD está disponible para Windows, Linux y OS X. OpenSCAD realiza geometría constructiva de sólidos (CSG).[**?**]

2. Programa

2.1. Conceptos

1. Objetos

   Los objetos son las piezas de construcción para realizar modelos, creadas en primitivas 2D y 3D, los objetos deben finalizar con punto y coma ';'.[**?**]

2. Acciones

   Las instancias de acción incluye la creación de objetos a partir de primitivas y asignando valores a las variables. Las instancias de acción también finalizan con punto y coma.[**?**]

3. Operadores

   Operadores, o transformaciones, modifica la localización, color u otras propiedades de los objetos. Los operadores usan llaves '' cuando su campo de acción cubre más de una acción. Más de un operador puede ser usado para la misma acción o grupo de acciones. Operadores múltiples son procesados de Derecha a Izquierda, eso quiere decir que el operador más cercano a la acción es procesado primero. Los operadores no terminan con punto coma ';', pero las acciones individuales pueden contenerla.[**?**]

2.2. Sintaxis

2.2.1. Variables

Objetivo: Expliación de las variables en el entorno parametrizable.[**?**]

Las variables de OpenSCAD son creadas en instancia con un nombre o identificador, asignando una expresión y un punto y coma. El rol de los arreglos, encontrado en muchos lenguajes imperativos, es manejado en OpenSCAD via vectores.[**?**]

```
var = 25;
xx = 1.25 * cos(50);
y = 2*xx+var;
logic = true;
MyString = "This is a string";
a_vector = [1,2,3];
rr = a_vector[2];      // member of vector
range1 = [-1.5:0.5:3]; // for() loop range
xx = [0:5];            // alternate for() loop range
```

Figura 1: Diferentes formas de representar variables en OpenSCAD

2.3. 2D

2.3.1. Círculos

Objetivo: Empleo y declaración de círculos.[**?**]

Para crear círculos todos los parámetros excepto la r deben ser nombrados, los círculos se crean en el origen.[**?**]

```
circle(r=radius | d=diameter);
```

Parámetros:

r: es el radio del círculo.
La resolución del círculo está basada en el tamaño usando $fa o $fs.

Para un círculo pequeño de alta resolución, se puede hacer un círculo grande, y reducirlo, o puedes usar el parámetro $fn u otras variables especiales.[**?**]

Nota: Éstos ejemplos exceden la resolución de una impresora 3D.[**?**]

1. d: diámetro del círculo (solo en versiones después de 2014.03).
2. $fa: ángulo mínimo (en grados) de cada fragmento.
3. $fs: largo circunferencial mínimo de cada fragmento.
4. $fn: fragmentos acomodados en 360 grados valores desde 3 a mayores.
5. $fa, $fs y $fn deben ser nombrados.

```
defaults:  circle();
yields:  circle(\$fn = 0, \$fa = 12, \$fs = 2, r = 1);
```

Figura 2: Código base del círculo

Como se puede apreciar en la figura 2 y se ha mencionado en el concepto de objeto, se puede apreciar que para generar un círculo solo hace falta una línea de código, y dentro de sus argumentos pueden ir definidos o no varios campos que le harán modificar sus características.

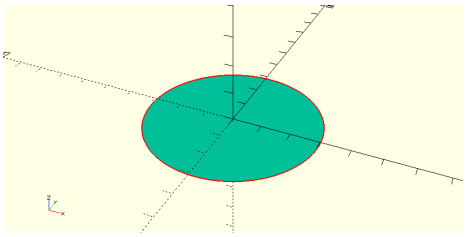


Figura 3: Ejemplo gráfico círculo

En la figura 3 es una representación genérica de un círculo con un radio y definición determinadas, es importante observar que el círculo se genera en el origen del sistema de coordenadas sobre el plano XY y además que es un círculo 2D.

2.3.2. Cuadrado

Objetivo: Empleo y declaración de paralelogramos.

Éste ejemplo sirve para crear un cuadrado o un rectángulo en el primer cuadrante. Los nombres de los argumentos son opcionales.

```
square(size = [x, y], center = true/false);
square(size =  x    , center = true/false);
```

Cuándo el centro es verdadero el cuadrado es centrado en el origen.

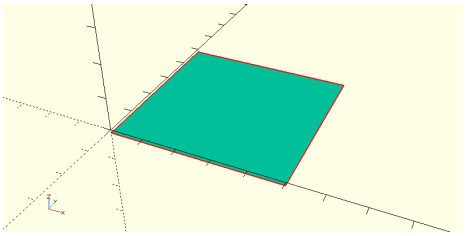


Figura 4: Ejemplo gráfico Cuadrado

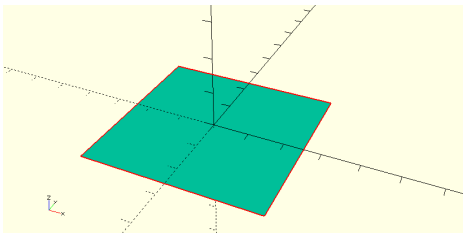


Figura 5: Ejemplo gráfico Cuadrado centrado

2.3.3. Polígonos Regulares

Objetivo: Dibujo de polígonos regulares.[**?**]

Un polígono regular de 3 o más lados puede ser creado usando círculos (circle()) con $fn definiendo el número de lados. El polígono es inscrito dentro del círculo con todos sus lados (y ángulos) igualmente. Una esquina apunta a la dirección "x"positiva.[**?**]

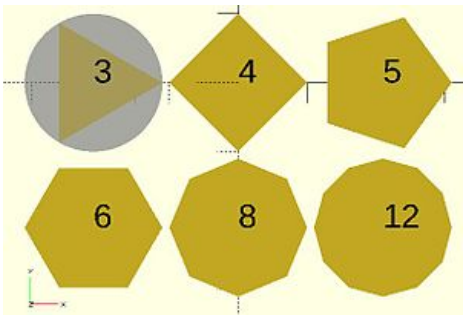


Figura 6: Polígonos regulares

2.3.4. Polígono

Objetivo: Dibujo de polígonos.[**?**]

Para hacer polígonos o una forma de formas múltiples, se genera a partir de coordenadas x,y. Un polígono es el objeto más poderoso 2D. Se puede crear cualquier cosa más que los círculos y los cuadrados pueden y mucho más. Ésto incluye formas irregulares con esquinas cóncavas y convexas. Además se puede colocar agujeros dentro de la figura.[**?**]

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

Parámetros:

1. puntos:la lista de puntos x,y del polígono: Un vector de 2 vectores elementales.
2. trayectorias por defecto: Si no hay una trayectoria en específico, todos los puntos se usarán con el orden de la lista.
3. trayectorias por vector sencillo: para se siga una trayectoria a través de los puntos. Se usan índices desde 0 hasta n-1. Pueden ser usados en un orden diferente, usar todos o una parte de los puntos listados.
4. trayectorias por múltiples vectores: Crea formas primarias y secundarias. Las formas secundarias son sustraídas de las formas primarias (como diferencia). Las formas secundarias pueden ser totalmente o parcialmente-compuestas the la forma primaria.

Una trayectroia cerrada es creada regresando desde el último punto especificado hacia el primero.[**?**]

2.3.5. Texto

Objetivo: Diferentes usos de texto.[**?**]

El módulo de texto crea textos como un dibujo 2D genérico, usando las fuentes instaladas en el sistema local o provistos como una fuente de un archivo por separado.[**?**]

Figura 7: Ejemplo de texto en 2D

2.4. 3D

2.4.1. Esfera

Objetivo: Declaración y uso de esferas.[**?**]

Crea una esfera en un origen de coordenadas de un sistema. El nombre del argumento r es opcional. Para usar d en vez de r, d debe ser nombrada.[**?**]

Parámetros:

1. r: Radio es el radio de la esfera. La resolucipon de la esfera debe ser basada en el tamaño de la esfera y las variables $\$fa$, $\$fs$ y $\$fn$. Para más información en éstas variables especiales consultar: OpenSCAD_User _Manual/Other_Language_Features
2. d Diáetro es el díametro de la esfera. (Nota: d está disponible en las versiones después de la 2014.03. Debaian se encuentra detrás de ésto)
3. $\$fa$ Fragmento de ángulo en grados
4. $\$fs$ Fragmento de tamaño en mm
5. $\$fn$ Resolución

```
default values:  sphere();
yields:   sphere(\$fn = 0, \$fa = 12, \$fs = 2, r = 1);

   // Esto creara una esfera de alta resolucion con 2mm de radio
sphere(2,  \$fn=100);

   // tambien creara una esfera de 2mm con alta resolución,
   //esta en cambio evitara crear pequeños triangulos como pueda en los polos de la esfera
sphere(2, $fa=5, $fs=0.1);
```

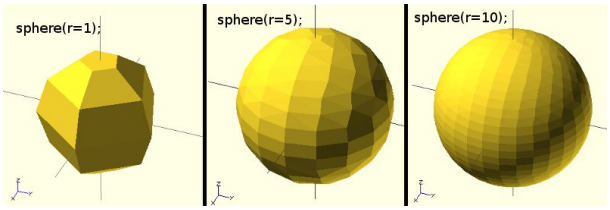


Figura 8: Ejemplo de resolución esferas

2.4.2. Cubo

Objetivo: Declaración y uso de paralepípedos.[**?**]

Crea un cubo en el primer octante. Cuando el centro tiene valor verdadero "true", el cubo es centrado en el origen. Los nombres de los argumentos son opcionales si se dan en el orden mostrado.[**?**]

```
 cube(size = [x,y,z], center = true/false);
cube(size =  x ,     center = true/false);
```

Parámetros:

1. tamaño: Valor sencillo, cubo con todos sus lados, arreglo de 3 valores $[x, y, x]$, cubo con dimensiones x,y y z4.
2. centro: el valor por defecto en falso ("false"), esto es que el cubo se desarrolla en el primer octante, con esquina en $(0, 0, 0)$, cuando es verdadero ("true"), el centro del cubo es centrado en el origen en: $(0, 0, 0)$.

```
default values:  cube();
yields:  cube(size = [1, 1, 1], center = false);
```

Ejemplos:

Códigos equivalentes para el ejemplo de la figura 9:

```
cube(size = 18);
cube(18);
cube([18,18,18]);
.
cube(18,false);
cube([18,18,18],false);
cube([18,18,18],center=false);
cube(size = [18,18,18], center = false);
cube(center = false,size = [18,18,18] );
```

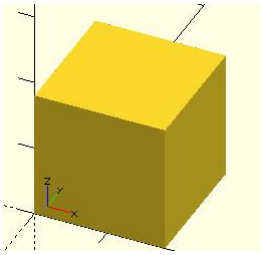


Figura 9: Ejemplo de cubo generado en el primer octante

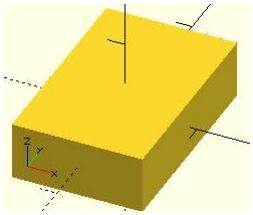


Figura 10: Ejemplo de cubo centrado en el origen

Código equivalente para el ejemplo de la figura 10:

```
cube([18,28,8],true);
box=[18,28,8];cube(box,true);
```

2.4.3. Cilindro

Objetivo: Declaración y uso de cilindros.[**?**]

Crea un cilindro o cono centrado alrededor del eje z. Cuando el centro es verdadero (”true”), es también centrada verticalmente a través del eje z.[**?**]

Los nombres de los parámetros son opcionales si se da el orden mostrado a continuación. Si un parámetro es nombrado, todos los siguientes parámetros también deberán ser nombrados.[**?**]

NOTA: Si r, d, d1, o d2 son usados, deben ser nombrados.[**?**]

```
cylinder(h = height, r1 = BottomRadius, r2 = TopRadius, center = true/false);
```

Parámetros:

1. h: altura del cilindro o cono.

2. r: radio del cilindro. r1 = r2 =r.
3. r1: radio, parte baja del cono.
4. r2: radio, parte alta del cono.
5. d: diámetro del cilindro. r1 = r2 =d/2.
6. d1: diámetro, parte baja del cono. r1 = d1/2.
7. d2: diámetro, parte alta del cono. r2 = d2/2.
   (NOTA: d,d1,d2, requiere 2014.03 o superior. Es sabido que Debian se encuentra debajo de ésto)
8. centro:
   falso ("false") por defecto, z se encuetra en el rango desde 0 hasta h
   verdadero ("true"), z se encuentra en el rango desde -h/2 hasta +h/2
9. $fa: ángulo mínimo (grados) de cada fragmento.
10. $fs: radio circunferencial m+inimo de cada fragmento.
11. $fn: número fijo de fragmentos en 360 grados. Valores desde 3 o más, o sobrecarga de $fa y $fs

    $fa,$fs y $fn deben ser nombradas

```
defaults: cylinder();
yields: cylinder(\$fn = 0, \$fa = 12,
\$fs = 2, h = 1, r1 = 1, r2 = 1, center = false);
```

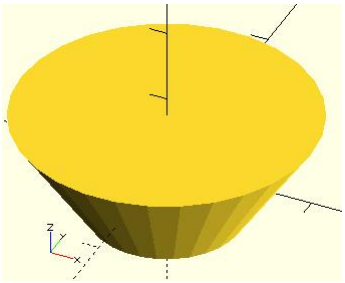


Figura 11: Ejemplo Cono

uso de $fn

Valores más largos de $fn crean superfices mas lisas, más circulares, con el costo con más tiempo de renderizado. Algunos usan valores medianos en el desarrollo para un renderizado más rápido, y al final usan un valor mayor para el renderizado final (F6).

```
translate([10,10,10])
cube([1,1,1]);
```

Figura 12: Ejemplo de "translate"

```
rotate([90,90,90])
cube([1,1,1]);
```

Figura 13: Ejemplo de rotate"

2.5. Transformaciones

2.5.1. Translado

Objetivo: Uso de transformación translado.[**?**]

En la función de translado "translate"se usan 3 argumentos, éstos corresponden al desplazamiento de los objetos en un espacio de coordenadas cartesianas, siendo éstos desplazamientos en el eje "x.[en] el zz en "z".

En la figura 12 se puede apreciar que se genera un cubo de dimensiones de 1 mm en todos sus lados y se ha movido 10 mm en todos los ejes cartesianos.

2.5.2. Rotación

Objetivo: Uso de transformación rotación. [**?**]

En la función de rotación rotate"se usan 3 argumentos, éstos corresponden a un giro en grados de los objetos en un espacio de coordenadas cartesianas, siendo éstos los giros respecto a los ejes "x.[en] el zz en "z".

En la figura 13 se puede apreciar que se genera un cubo de dimensiones de 1 mm en todos sus lados y se ha girado 90 grados respecto a todos los ejes cartesianos.

2.5.3. Escala

Objetivo: Uso de la transformación escala.[**?**]

En la figura 21 se puede apreciar un segmento de código que usa la función *scale*, ésta función utiliza factores de multiplicación en un eje coordenado respecto al argumento que recibe, en éste ejemplo se refiere que en el eje x la figura va a crecer un 150 % y en el eje y va a cambiar en un 50 %.

```
scale([1.5,0.5])circle(d=20);
```

Figura 14: Ejemplo "scale"

```
resize([30,10])circle(d=20);
```

Figura 15: Ejemplo resize"

2.5.4. Redimensionar

Objetivo: Uso de la transformación de redimensión.[**?**]

En la figura 15 se puede apreciar el código de ejemplo de la función *rezise*, a diferencia de la función scale ést función obliga a la figura a ocupar el espacio especificado en sus argumentos, es decir, en el ejemplo se dibuja una circunferencia que se estirará para que en el eje "x"mida 30 mm y en el eje z"mida 10 mm.

2.5.5. Espejo

Objetivo: Uso de la transformación espejo.[**?**]

En la figura 16 se puede apreciar un código de ejemplo de la función *mirror* que sirve para replicar un objeto respecto a un eje coordenado, se puede ver que el objeto en cuestión tiene un translado, por lo tanto el translado
[**?**]

2.6. Operaciones Booleanas

2.6.1. Union

Objetivo: Uso de la operación de Unión.[**?**]

```
mirror([0,1,0])
{
     translate([0,10,0])
     cube([1,1,1],center=true);
}
```

Figura 16: Ejemplo "mirror"

Crea una Unión de sus nodos hijos. Ésto es la suma de todos los hijos.[**?**]

Puede ser usado con objetos tanto 2D y 3D, pero no puede mezclarlos.[**?**]

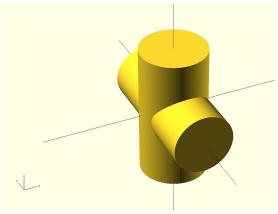


Figura 17: Operación Unión con dos cilindros

Código de referencia que muestra la figura 17:

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

Es importante mencionar que la unión de los nodos hijos es implícita cuando el comando no es usado. Pero es obligatoria, por ejemplo, en diferencia para agrupar a los primeros nodos hijos en uno.

2.6.2. Diferencia

Objetivo: Uso de la operación de Diferencia.[**?**]

Substrae el segundo (o más) nodos hijo del primero. Puede ser usado con objetos en 2D y 3D, pero no se pueden mezclar.[**?**]

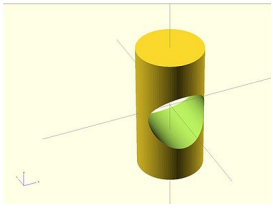


Figura 18: Diferencia

```
Usage example:
```

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

2.6.3. Intersección

Objetivo: Uso de la operación de Intersección.[**?**]

Crea la intersección de todos los nodos hijos. Esto mantiene la sección que se entre cruza (conjunción lógica).[**?**]

Sólo el área que es común o compartida por todos los hijos es retenida.[**?**]

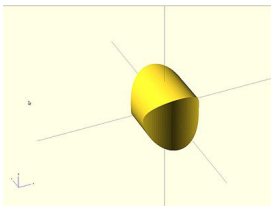


Figura 19: Intersección

Puede usarse en objetos en 2D y 3D pero no se pueden mezclar.[**?**]

2.7. Simplificación

2.7.1. Modulos

Explicación de la creación de módulos.[**?**]

```
module cosa(parametros){
   codigo con parametros a ejecutar;
}
```

La implementación de módulos, resulta ser de mucha utilidad porque con ellos podemos crear objetos que hereden de éstas características múltiples veces.

2.7.2. Include

Objetivo:

Explicación de la inclusión de librerías.[**?**]

```
//funcion en archivo libreria.scad

funcionLibreria(parametros por defecto){
codigo parametrizable a ejecutar;
}

//codigo que renderiza objeto en libreria.scad
funcionLibreria();

include<libreria.scad>

funcionLibreria();
```

La función include es de utilidad porque nos permite retomar funciones realizadas en otros códigos de openscad, es importante mencionar que el hacer uso de include también incluye los modelos a renderizar en el código.

2.7.3. Use

Objetivo:

Explicación de el uso de librerías.[**?**]

La función use sirve para utilizar funciones realizadas en otros códigos de openscad, es importante mencionar que el hacer uso de use no incluye los modelos a renderizar en el código.

2.8. Modificador de simplificación

Ignora el objeto aplicado.[**?**]

```
    difference() {
cube(10, center = true);
translate([0, 0, 5]) {
rotate([0, 90, 0]) {
cylinder(r = 2, h = 20, center = true, $fn = 40);
}
*rotate([90, 0, 0]) {
#cylinder(r = 2, h = 20, center = true, $fn = 40);
}
}
}
```

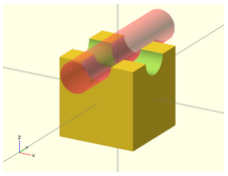


Figura 20: sin el modificador

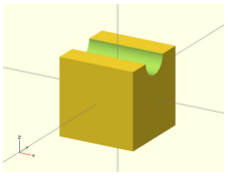


Figura 21: Con el modificador